[redacted]

6115393

Issue Classification: Class 370, Subclass 469

UTILITY SERIAL NUMBER 08/505,083 | PATENT DATE SEP 05 2000 | PATENT NUMBER

SERIAL NUMBER	FILING DATE	CLASS	SUBCLASS	GROUP ART UNIT	EXAMINER
08/505,083	07/21/95	370	469	2759	Hamilton

APPLICANTS

FERDINAND ENGEL, NORTHBOROUGH, MA; KENDALL S. JONES, NEWTON CENTER, MA; MARY ROBERTSON, BEDFORD, MA; DAVID M. THOMPSON, REDMOND, WA; GERARD WHITE, TYNGSBOROUGH, MA.

CONTINUING DATA********

VERIFIED

THIS APPLN IS A DIV OF 07/761,269 09/17/91 ABN
WHICH IS A CIP OF 07/684,695 04/12/91 ABN

FOREIGN APPLICATIONS********

VERIFIED

IF REQUIRED, FOREIGN FILING LICENSE GRANTED 09/29/95 ** SMALL ENTITY **

Foreign priority claimed 35 USC 119 conditions met	Verified and Acknowledged	Examiner's Initials	AS FILED	STATE OR COUNTRY	SHEETS DRWGS.	TOTAL CLAIMS	INDEP. CLAIMS	FILING FEE RECEIVED	ATTORNEY'S DOCKET NO.
☐ yes ☒ no ☐ yes ☐ no			→	MA	38	29	7	$616.00	00124/010002

ADDRESS

ERIC L. PRAHL
FISH AND RICHARDSON
225 FRANKLIN STREET
BOSTON MA 02110-2804

TITLE

NETWORK MONITORING

U.S. DEPT. OF COMM./PAT. & TM—PTO-436L (Rev. 12-94)

PARTS OF APPLICATION FILED SEPARATELY

Allowed Examiner 7-22-99

NOTICE OF ALLOWANCE MAILED	Assistant Examiner	CLAIMS ALLOWED: Total Claims	Print Claim
7-19-99	—	25	1

ISSUE FEE: Amount Due	Date Paid	Primary Examiner	DRAWING: Sheets Drwg.	Figs. Drwg.	Print Fig.
$1605.00	10-26-99 KB	Ajit Patel Primary Examiner	38	48	1, 4

Label Area

PREPARED FOR ISSUE

ISSUE BATCH NUMBER V93

WARNING: The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368. Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

Form PTO-436A (Rev. 6/92)

Formal Drawings (___ shts) set___

(FACE)

U.S.A. SERIES CODE	07/761269	PATENT DATE	PATENT NUMBER

SERIAL NUMBER	FILING DATE	CLASS	SUBCLASS	GROUP ART UNIT	EXAMINER
07/761,269	09/17/91	~~395~~ 340	~~825.52~~ 825.52	~~2404~~ 2604	HORRIK

APPLICANTS

FERDINAND ENGEL, NORTHBOROUGH, MA; KENDALL S. JONES, [illegible]; KARY ROBERTSON, BEDFORD, MA; DAVID M. THOMPSON, REDMOND, WA.

CONTINUING DATA******** 37 CFR 1.60
VERIFIED THIS APPLN IS A CIP OF 07/684,055 04/12/91 ABN

yes MTR

FOREIGN/PCT APPLICATIONS********
VERIFIED

None MTR

FOREIGN FILING LICENSE GRANTED 12/27/91 ***** SMALL ENTITY *****

Foreign priority claimed 35 USC 119 conditions met	Verified and Acknowledged	Examiner's Initials	AS FILED	STATE OR COUNTRY	SHEETS DRWGS.	TOTAL CLAIMS	INDEP. CLAIMS	FILING FEE RECEIVED	ATTORNEY'S DOCKET NO.
☐ yes ☑ no ☐ yes ☑ no	MTR			MA	38	38	9	$525.00	20126/[illegible]

ADDRESS
DAVID L. PETERSON
FISH & RICHARDSON
225 FRANKLIN ST.
BOSTON, MA 02110-2804

TITLE
~~NETWORK MONITORING~~ A Method Of Tracking Node Addresses

U.S. DEPT. of COMM.-Pat. & TM Office—PTO-436L (Rev. 10-78)

Microfiche Appendix
655 microfiche.
14 pages

PARTS OF APPLICATION FILED SEPARATELY

NOTICE OF ALLOWANCE MAILED	PREPARED FOR ISSUE		CLAIMS ALLOWED	
	Assistant Examiner	Docket Clerk	Total Claims	Print Claim
ISSUE FEE			DRAWING	
Amount Due / Date Paid	Primary Examiner		Sheets Drwg. / Figs. Drwg. / Print Fig.	
Label Area	ISSUE CLASSIFICATION: Class / Subclass		ISSUE BATCH NUMBER	

WARNING: The information disclosed herein may be restricted. Unauthorized disclosure may be prohibited by the United States Code Title 35, Sections 122, 181 and 368. Possession outside the U.S. Patent & Trademark Office is restricted to authorized employees and contractors only.

Form PTO-436

APPROVED FOR LICENSE ☐

SEP 24 9146

INITIALS ________

[redacted]

CONTENTS

Entered or Counted			Received or Mailed
RL	1.	Application ✓ papers.	
	2.	Letter Re: Fee + Signature	9/26/91
	3.	Dec, Sm Entity + Surcharge	11/18/91
	4.	prior art	2/24/92
	5.	prior art	3/16/92
	6.	Prior Art	Aug 24 1992
12-27	7.	Restriction (30 days)	1/14/94
	8.	Req. Ext. time (1)	3-14-94
3-26-94	9.	Election	3-14-94
5/25	10.	Rej 3 (mo)	6-2-94 AC
	11.	[illegible]	5-22-94 RW
	12.	Req. Ext. [illegible]	[illegible]
	13.	[illegible]	[illegible]
[illegible]	14.	Final Rej. 3 (months)	8-24-95 [illegible]
	15.	[illegible] (3 mos)	7-25-95
	16.	[illegible]	Sep 13, 1995
	17.		
	18.		
	19.		
	20.		
	21.		
	22.		
	23.		
	24.		
	25.		
	26.		
	27.		
	28.		
	29.		
	30.		
	31.		
	32.		

08505083

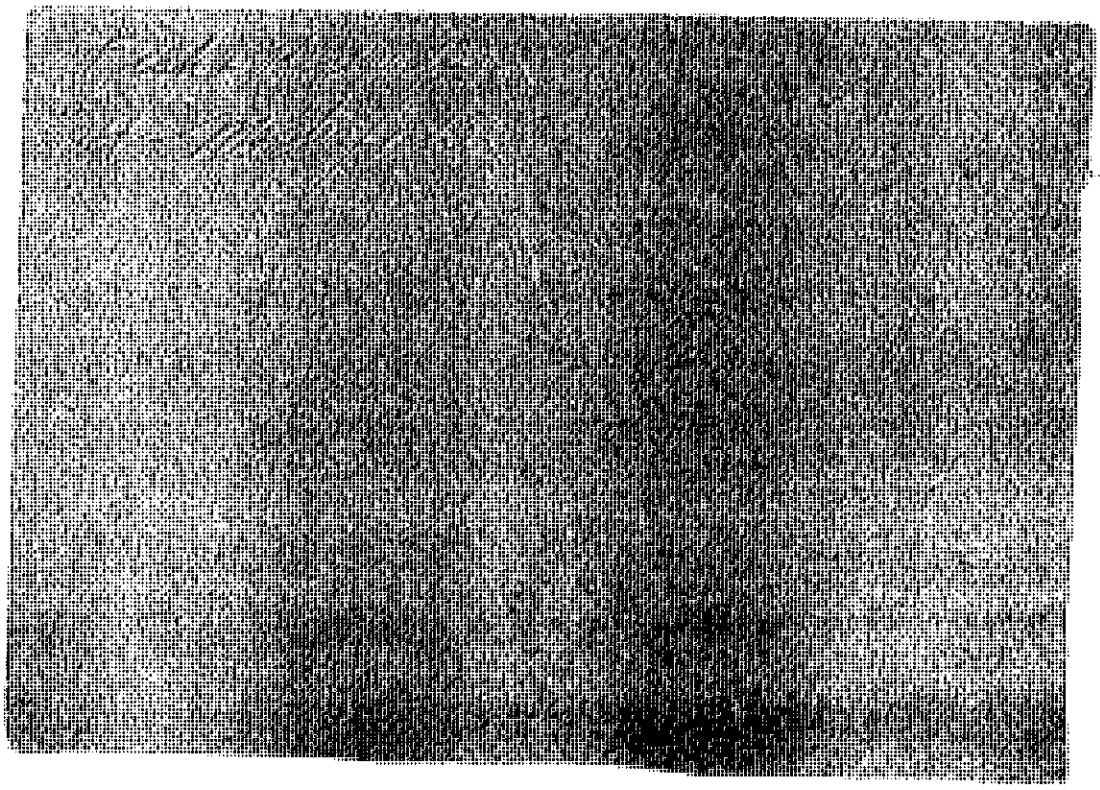
08505083

APPROVED FOR LICENSE ☐

CONTENTS

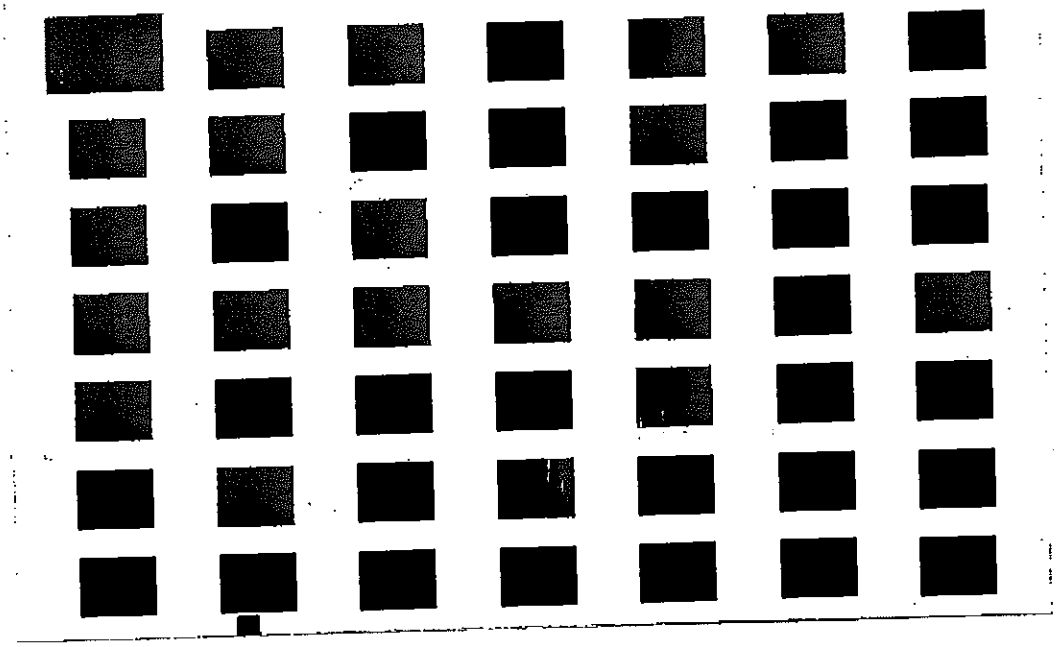
Date Entered or Counted		Date Received or Mailed
	1. Application papers:	
	12. Amdt A/B	July 31 1996
	18. Amdt C	Oct 31, 1996
5-28.	19. Office Actn (Restriction) mailed	5-25-96
	20. Ext of time (1)	7-11-96
	21. Election	7-11-96
10-1	22. Rsp (3mos)	10-8-96
	23. Rsp Ext of time (3)	April 14 1997 7-8-97
	24. Amdt D	April 14 1997 7-8-97
	25. IDS	4/21/97 V.H.
7-21-97	26. Rsp (3mos)	7-21-97 V.H.
	27. Rsp Ext of time (3)	Jan 27 1998
	28. Amdt E & F	Jan 27 1998
4-27	29. Rsp (3mos)	4-27-98 V.H.
	30. Ext of time (1)	Sept 1 1998 8-27-98
	31. Amdt G	Sept 1 1998 8-27-98
11/23	32. Rsp 3mos	Nov. 25, 1998
	33. Rsp Ext of time	May 3 1999
	34. Amdt H	May 3 1999
	35. 20. Interview Summary	7-14-99
7/19 =	36. 21. Examiner Amdt/H	7-19-99 M.L.
	37. 22. Interview Summary	10/14/99
1/21/00	23. Formal Drawings (38 sht(s)) set L	10/26/99
11/14/12	24. Change of Address	7/12-12
	25.	
	26.	
	27.	
	28.	
	29.	
	30.	
	31.	
	32.	

(FRONT)



NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerard White

01



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
/*
 *   rtp_dll_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                   All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/rtp/SCCS/s.rtp_dll_p.c
 *
 *  Date:       8/26/91
 *
 *  Revision:   1.7
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   08-26-91  KR        moved llc parse so that if error, wouldn't inc frames
 *   08-02-91  DPD       always do a get for the destination mac address
 *   06-05-91  KMJ       changed check for min pkt size to > 13
 *   06-05-91  DPD       moved counts for collisions/errors to chip counts
 *   05-31-91  KMJ       forced vars into registers and mimumized global
 *                            var access
 *
 */

static char rtp_dll_p_c [] = "@(#)rtp_dll_p.c     1.7";

#include <stdio.h>

#include <cci_std.h>

#include    "system.h"
```

```
#include    "address.h"
#include  <sys/types.h>
#include    <sys/socket.h>
#include  <bsd43/sys/time.h>
#include    "util.h"
#include    "kuser.h"
#include    "mbuf.h"
#ifdef unix
#include  <sys/cci.h>
#endif
#include    "lanutil.h"
#include  "cpu.h"
#include  "lan_82596.h"
#include    "mtm_if.h"
#include    "mtm_in.h"
#include    "mtm_if_ether.h"
#include    "alarms.h"
#include  "stats.h"
#include  "stats_dll.h"
#include  "em_ctrl.h"
#include  "rtp.h"


/*
 * MAC and LLC Headers and Data Structures
 */
MacAddress          mac_src_addr;
MacAddress          mac_dst_addr;


/*
 * Frame definitions
 */
#define   MINDATA             46
#define   MAXDATA             1500
#define MAX_LLC_DATA      1500-3
```

```
#define MAX_LLC_LENGTH   1518 /* includes mac header and 4 byte FCS */


/*
 * LLC frame masks
 */
#define I_MASK       0x01
#define U_MASK       0x10
#define PF_MASK           0xef
#define IG_MASK           0xfe
#define RESP_BIT     0x01


/*
 * LLC frame command types with p/f = 0
 */

/* information transfer 4 byte header */
#define I_FRAME           0x00
/* supervisory commands/responses 4 byte header */
#define RR_FRAME     0x01
#define REJ_FRAME    0x09
#define RNR_FRAME    0x05
/* unnumbered commands/response 3 byte header */
#define UI_FRAME     0x03
#define SABME_FRAME 0x6f
#define DISC_FRAME  0x43
#define UA_FRAME     0x63
#define DM_FRAME     0x0f
#define FRMR_FRAME  0x87 /* has a additional 5 byte information field */
/* also unnumbered */
#define TEST_FRAME  0xe3
#define XID_FRAME   0xaf /* response is additional 3 bytes long */


/* unused
    LLC_ir_pdu                = #77;
```

```
        LLC_AC0_pdu                 = #67;
        LLC_AC1_pdu                 = #E7;
        LLC_xid_lsdu_length = #03;
        llc_xid_format_id   = #81;
        llc_xid_type1       = #01;
        llc_xid_type2       = #02;
        llc_xid_type3       = #04;
        llc_toggle_bit      = #80;
        llc_seq_pf_mask     = #6F;

        ir_cmd_status_mask  = #0F;
        ir_cmd_ok           = #00;
        ir_cmd_rs           = #01;
        ir_cmd_ue           = #05;
        ir_cmd_pe           = #06;
        ir_cmd_ip           = #07;
        ir_cmd_un           = #09;
        ir_cmd_it           = #0F;
        ir_rsp_status_mask  = #F0;
        ir_rsp_ok           = #00;
        ir_rsp_rs           = #10;
        ir_rsp_ne           = #30;
        ir_rsp_nr           = #40;
        ir_rsp_ue           = #50;
        ir_rsp_ip           = #70;
        ir_rsp_un           = #90;
        ir_rsp_it           = #F0;
*/

/*
 * Header definitions
 */
struct mac_headerstr {
        MacAddress      dst_addr; /*rest per 802.2/3 specs*/
        MacAddress      src_addr;
        u_short         length;
```

```
        };

struct llc_headerstr {
        u_char    dsap;
        u_char    ssap;
        u_char    control;
        u_char    llc_data[MAX_LLC_DATA];  /* actual start of llc data*/
        };

struct snap_headerstr {
        u_char    dsap;
        u_char    ssap;
        u_char    control;
        u_char    org_code[3];   /* protocol id OR origin code */
        u_short   ether_type;          /* there may be a org_code that */
        };                             /* specifices this field to be the EtherType */

struct dll_framestr {
     struct    mac_headerstr  mac_header;     /*rest per 802.2/3 specs*/
     struct    llc_headerstr  llc_header;     /*802.2 specs*/
     u_char    fcs[4];
     };

#define MAC_HEADER_SIZE             14    /*sizeof(mac_headerstr)*/
#define LLC_HEADER_SIZE             3     /*sizeof(llc_headerstr)*/
#define SNAP_LLC_HEADER_SIZE  8     /*sizeof(snap_headerstr)*/
#define SNAP_LLC_HEADER_DIFF  SNAP_LLC_HEADER_SIZE-LLC_HEADER_SIZE
#define NOVELL_HEADER_SIZE          2     /*NOVELL only has a SSAP and DSAP*/


/*
 * Local data structures
 */
extern Uint32       myIpAddr;

static Uint32                   dll_length;
```

```
static Uint32                   llc_protocol;

static StatsAddrEntry           *this_seg_addr_ptr;
static StatsDllSegment          *this_seg_stats_ptr;
static StatsProtocolEntry       *this_seg_protocol_ptr;

static StatsAddrEntry           *src_seg_addr_ptr;
static StatsDllSegment          *src_seg_stats_ptr;
static StatsProtocolEntry       *src_seg_protocol_ptr;

static StatsAddrEntry           *dst_seg_addr_ptr;
static StatsDllSegment          *dst_seg_stats_ptr;
static StatsProtocolEntry       *dst_seg_protocol_ptr;

static StatsAddrEntry           *src_node_addr_ptr;
static StatsDllAddr             *src_node_stats_ptr;
static StatsProtocolEntry       *src_node_protocol_ptr;

static StatsAddrEntry           *dst_node_addr_ptr;
static StatsDllAddr             *dst_node_stats_ptr;
static StatsProtocolEntry       *dst_node_protocol_ptr;

static StatsAddrEntry           *dialog_addr_ptr;
static StatsDialogEntry         *dialog_stats_ptr;


/*
 * Macros for Dll statistics
 * These use the macros defined in stats.h
 */

#define dll_stats_frames { \
     stats_frames (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_bytes { \
     stats_bytes (data_length, DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }
```

```
#define dll_stats_errors { \
     stats_errors (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_bcasts { \
     stats_bcasts (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_mcasts { \
     stats_mcasts (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_enet { \
     stats_enet (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_llc { \
     stats_llc (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_protocol { \
     stats_protocol (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_off_segs { \
     stats_off_segs (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_transits { \
     stats_transits (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_collisions { \
     stats_collisions (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_alignment { \
     stats_alignment (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }

#define dll_stats_runts { \
     stats_runts (DLL_SEGMENT, DLL_NODE, DLL_PAIR, DLL_PROTOCOL); }


/*****************************************************************************
```

```
 *    Data Link Layer Parse Routine
 *
 *    Function  Parse data link layer frame
 *
 *    Input          pointer to the frame, length of frame
 *    Output         objects updated, next layer parse routine called
 */

Uint32 rtp_dll_parse (layer_ptr, data_length)

     char                *layer_ptr;
     Uint32                   data_length;
{
     register Uint32                    adjusted_data_length, i, result;
     register Uint32                    protocol, *uint32_ptr;
     register struct dll_framestr  *frame_ptr;
     register Uint16                    j=0xFFFF;

     register struct ether_arp     *ea;
                                        *wp1, *wp2;
     register struct in_addr       dst_ip_addr;
     register struct mbuf               *mbuf_ptr;
     /* stats_alarm_data can not be put in register so added *uint32_ptr */
     AlarmUserData       stats_alarm_data;

/*
 * Set up local variables
 */

result = GOOD;

frame_ptr = (struct dll_framestr *)layer_ptr;

mac_src_addr = frame_ptr->mac_header.src_addr;
mac_dst_addr = frame_ptr->mac_header.dst_addr;

/*
```

```
 * Test for the broadcast address
 */
i = 0;
while (i<(sizeof(struct ether_addr)/2))
     {
     j &= frame_ptr->mac_header.dst_addr.double_bytes[i];
     i++;
     }
rtp_broadcast = (j == 0xFFFF);


/*
 * Test for a multicast address
 */
if (rtp_broadcast == FALSE)
     if (frame_ptr->mac_header.dst_addr.bytes[0] & 0x01)
          rtp_multicast = TRUE;


/* this test prevents all src & dst error counts if src and or protocol is unknown */
if (data_length > 13)
     {
     if ( (!monCtrl.bcastNodeEnable) && (rtp_broadcast || rtp_multicast) )
          dll_src_node_addr_ptr = src_node_addr_ptr = stats_dll_lookup_addr
(&mac_src_addr);
     else
          dll_src_node_addr_ptr = src_node_addr_ptr = stats_dll_get_addr (&mac_src_addr);


     if (dll_src_node_addr_ptr)
          {
          dll_src_node_stats_ptr = src_node_stats_ptr = (StatsDllAddr *)
src_node_addr_ptr->stats_ptr;
          dll_src_seg_addr_ptr = src_seg_addr_ptr = stats_dll_get_segment
(src_node_addr_ptr->address.segment1);
          if (src_seg_addr_ptr != NULL)
```

```
                dll_src_seg_stats_ptr = src_seg_stats_ptr = (StatsDllSegment *)
src_seg_addr_ptr->stats_ptr;
           else
                dll_src_seg_stats_ptr = src_seg_stats_ptr = NULL;
           }
     else
           {
           dll_src_node_stats_ptr = src_node_stats_ptr = NULL;
           dll_src_seg_addr_ptr = src_seg_addr_ptr   = NULL;
           dll_src_seg_stats_ptr = src_seg_stats_ptr  = NULL;
           }

/*
     if ( (!monCtrl.bcastNodeEnable) && (rtp_broadcast || rtp_multicast) )
           dll_dst_node_addr_ptr = dst_node_addr_ptr = stats_dll_lookup_addr
(&mac_dst_addr);
     else
           dll_dst_node_addr_ptr = dst_node_addr_ptr = stats_dll_get_addr (&mac_dst_addr);

*/
     /*** always do a get on the destination node ***/
     dll_dst_node_addr_ptr = dst_node_addr_ptr = stats_dll_get_addr (&mac_dst_addr);

     if (dll_dst_node_addr_ptr)
           {
           dll_dst_node_stats_ptr = dst_node_stats_ptr = (StatsDllAddr *)
dst_node_addr_ptr->stats_ptr;
           dll_dst_seg_addr_ptr = dst_seg_addr_ptr = stats_dll_get_segment
(dst_node_addr_ptr->address.segment1);
           if (dst_seg_addr_ptr != NULL)
                dll_dst_seg_stats_ptr = dst_seg_stats_ptr = (StatsDllSegment *)
dst_seg_addr_ptr->stats_ptr;
           else
                dll_dst_seg_stats_ptr = dst_seg_stats_ptr = NULL;
           }
     else
           {
```

```
        dll_dst_node_stats_ptr = dst_node_stats_ptr = NULL;
        dll_dst_seg_addr_ptr   = dst_seg_addr_ptr   = NULL;
        dll_dst_seg_stats_ptr  = dst_seg_stats_ptr  = NULL;
        }
    /*
     * Don't allocate a dialog structure for a broadcast or multicast
     */
    if ( (!monCtrl.bcastDialogEnable) && (rtp_broadcast || rtp_multicast) )
        dll_dialog_addr_ptr = dialog_addr_ptr =
            stats_dll_lookup_dialog (&mac_src_addr, &mac_dst_addr);
    else
        dll_dialog_addr_ptr = dialog_addr_ptr =
            stats_dll_get_dialog (&mac_src_addr, &mac_dst_addr);

    if (dialog_addr_ptr != NULL)
        dll_dialog_stats_ptr = dialog_stats_ptr = (StatsDialogEntry *)
dialog_addr_ptr->stats_ptr;
    else
        dll_dialog_stats_ptr = dialog_stats_ptr = NULL;
    }
else
    {
    dll_this_seg_addr_ptr    = this_seg_addr_ptr     = NULL;
    dll_src_seg_addr_ptr     = src_seg_addr_ptr      = NULL;
    dll_dst_seg_addr_ptr     = dst_seg_addr_ptr      = NULL;
    dll_this_seg_stats_ptr   = this_seg_stats_ptr    = NULL;
    dll_src_seg_stats_ptr    = src_seg_stats_ptr     = NULL;
    dll_dst_seg_stats_ptr    = dst_seg_stats_ptr     = NULL;
    dll_this_seg_protocol_ptr     = this_seg_protocol_ptr = NULL;
    dll_src_seg_protocol_ptr = src_seg_protocol_ptr  = NULL;
    dll_dst_seg_protocol_ptr = dst_seg_protocol_ptr  = NULL;
    dll_src_node_stats_ptr   = src_node_stats_ptr    = NULL;
    dll_dst_node_stats_ptr   = dst_node_stats_ptr    = NULL;
    dll_src_node_protocol_ptr     = src_node_protocol_ptr = NULL;
    dll_dst_node_protocol_ptr     = dst_node_protocol_ptr = NULL;
    dll_dialog_stats_ptr     = dialog_stats_ptr      = NULL;
    }    /* end if */
```

```
if (this_seg_addr_ptr = stats_dll_get_segment (mySegmentId))
     dll_this_seg_stats_ptr = this_seg_stats_ptr = (StatsDllSegment *)
this_seg_addr_ptr->stats_ptr;
else
     dll_this_seg_stats_ptr = this_seg_stats_ptr = NULL;

/*
 * Update age timers in the address records and set the 10 second sample
 * rate so that the event manager will calculate rates on the statistics.
 */
stats_update_age_timers;
stats_set_rate_10s;

/*
 * handle BAD frame statistics
 */
mbuf_ptr = dtom(frame_ptr);
if (mbuf_ptr->rcv_status)
     {
     /* receive frame errors detected */
     if (mbuf_ptr->rcv_status & RS_COLLISION)
          {
          /* collision error detected - do not parse or send to MTM_RCV */
/* THESE ARE NOW COUNTED BY THE CHIP
          dll_stats_collisions;
          dll_stats_errors;
*/
          goto mac_bad;
          }
     else if (mbuf_ptr->rcv_status & (RS_CRC_ERR + RS_ALIN_ERR))
          {
          /* CRC and or allignmont error detected - do not parse or send to MTM_RCV */
          dll_stats_alignment;
          dll_stats_errors;
          goto mac_bad;
          }
```

```
     /*
      *else if (mbuf_ptr->rcv_status & RS_CRC_ERR)
      *    {
      *    * CRC error detected - do not parse or send to MTM_RCV *
      *    dll_stats_crc;
      *    dll_stats_errors;
      *    goto mac_bad;
      *    }
      */
     else if (mbuf_ptr->rcv_status & RS_SHORT_ERR)
          {
          /* runt frame detected - do not parse or send to MTM_RCV */
          dll_stats_runts;
          dll_stats_errors;
          goto mac_bad;
          }     /* end if */
     }     /* end if */


/*
 * Determine if the frame is for this station
 */
for (i=sizeof(struct ether_addr); i==0; i--)
     if (frame_ptr->mac_header.dst_addr.bytes[i-1] == ts_addr.ether_addr_octet[i-1])
          {
          rtp_for_this_station = YES;
          }
     else
          {
          rtp_for_this_station = NO;
          break;
          }
/*
 * Determine if the frame is from another monitor
 */
if (src_node_stats_ptr != NULL)
     {
```

```
        if ((rtp_for_this_station == NO) && (src_node_addr_ptr->flags & MibMonitor))
            rtp_from_another_monitor = YES;
        }


    /*
     * Check for ethernet or 802.3/802.2.  Parse 802.2 if necessary.
     * Set the protocol type based on the ethernet type or the dsap.
     */
    dll_length = MAC_HEADER_SIZE;
    adjusted_data_length = data_length - MAC_HEADER_SIZE;

    frame_ptr->mac_header.length = ntohs(frame_ptr->mac_header.length);

    if ( (frame_ptr->mac_header.length >= MAXDATA)
        | ((frame_ptr->mac_header.length == XEROX_PUP)
        && (adjusted_data_length != XEROX_PUP) )
        | ((frame_ptr->mac_header.length == PUP_ADDR_TRANS)
        && (adjusted_data_length != PUP_ADDR_TRANS)))
        {
        /* ethernet frame */
        dll_stats_enet;
        protocol = frame_ptr->mac_header.length;
        }

    else /* 802.2 frame */
        {
        if (frame_ptr->mac_header.length < 3)
            {
            dll_stats_errors;
            goto mac_bad;
            }

        if (frame_ptr->mac_header.length < MINDATA )
            adjusted_data_length = frame_ptr->mac_header.length;

        dll_stats_llc;
```

```
        result = rtp_llc_parse (frame_ptr, adjusted_data_length);
        protocol = llc_protocol;
        }

/*
 * Do some statistics
 */
dll_stats_frames;
dll_stats_bytes;    /* if this frame is truncated this count will be wrong */
dll_stats_off_segs;
dll_stats_transits;

if (rtp_broadcast == TRUE)
        {
        dll_stats_bcasts;
        }

if (rtp_multicast == TRUE)
              {
              dll_stats_mcasts;
              }

/*
 *  Find statistics structure for the protocol distribution statistics
 */
if (src_node_stats_ptr != NULL)
        dll_src_node_protocol_ptr = src_node_protocol_ptr = stats_get_protocol
(&src_node_stats_ptr->protocolQ,protocol);
else
        dll_src_node_protocol_ptr = src_node_protocol_ptr = NULL;

if (dst_node_stats_ptr != NULL)
        dll_dst_node_protocol_ptr = dst_node_protocol_ptr = stats_get_protocol
(&dst_node_stats_ptr->protocolQ,protocol);
else
        dll_dst_node_protocol_ptr = dst_node_protocol_ptr = NULL;
```

```
if (this_seg_stats_ptr != NULL)
     dll_this_seg_protocol_ptr = this_seg_protocol_ptr = stats_get_protocol
(&this_seg_stats_ptr->protocolQ,protocol);
else
     dll_this_seg_protocol_ptr = this_seg_protocol_ptr = NULL;

if (src_seg_stats_ptr != NULL)
     dll_src_seg_protocol_ptr = src_seg_protocol_ptr = stats_get_protocol
(&src_seg_stats_ptr->protocolQ,protocol);
else
     dll_src_seg_protocol_ptr = src_seg_protocol_ptr = NULL;

if (dst_seg_stats_ptr != NULL)
     dll_dst_seg_protocol_ptr = dst_seg_protocol_ptr = stats_get_protocol
(&dst_seg_stats_ptr->protocolQ,protocol);
else
     dll_dst_seg_protocol_ptr = dst_seg_protocol_ptr = NULL;


/*
 * Keep protocol distribution statistics.
 * Pass the protocol as alarm data in case an alarm occurs
 */

stats_alarm_data.length = 4;
/*
 * replaced this
 * bcopy (&protocol, stats_alarm_data.data, 4);
 * with
 */
uint32_ptr = &stats_alarm_data.length;
uint32_ptr++;
*uint32_ptr = protocol;
dll_stats_protocol;
```

```
/*
 * Parse next protocol
 */
switch (protocol)
     {
     case DOD_IP:
          if (result == GOOD)
               result = rtp_ip_parse (layer_ptr + dll_length, data_length - dll_length);
         break;

     case ARP:
          if (result == GOOD)
               {
               ea = (struct ether_arp *)(layer_ptr + dll_length);
               ALIGN_LONG(ea->arp_tpa[0],dst_ip_addr.s_addr);
               if (dst_ip_addr.s_addr == myIpAddr)
                    rtp_for_this_station = TRUE;
               }
          break;

     default:
          break;
     }

/*
 * Determine if this frame is truncated
 */
if (nbuf_ptr->rcv_status & (RS_RES_ERR + RS_TRUNCATED))
     {
     /* truncated frame - can be parsed but should not be sent to MTM_RCV */
     rtp_for_this_station = rtp_from_another_monitor = NO;
     /* increment segment count? */
     }

return (result);

mac_bad:
```

```
/*
 * Determine if this frame is truncated
 */
if (mbuf_ptr->rcv_status & (RS_RES_ERR + RS_TRUNCATED))
     {
     /* truncated frame - can be parsed but should not be sent to MTM_RCV */
     rtp_for_this_station = rtp_from_another_monitor = NO;
     /* increment segment count? */
     }
     return (BAD);

}



/*****************************************************************************
 *
 *   LLC Parse Routine
 *
 *   Function  Parse 802.2 frame
 *
 *   Input          pointer to the frame, length of frame
 *   Output         objects updated
 */

Uint32 rtp_llc_parse (frame_ptr, data_length)

     struct  dll_framestr     *frame_ptr;
     Uint32                        data_length;

{
     register struct snap_headerstr     *snap_ptr;
     register u_char                    type;
     register struct llc_stats_type     *sap_src;
     register struct llc_stats_type     *sap_dst;
     register struct llcs_stats_type    *sap_pair;
```

```
/*
 * Get protocol type
 */
llc_protocol = frame_ptr->llc_header.ssap & IG_MASK; /*clear command/response bit*/

dll_length += LLC_HEADER_SIZE;

switch (llc_protocol)
     {
     case LLC_NULL_LSAP:
          break;

     /* case LLC_GLOBAL: not supported if testing ssap */

     case LLC_MGT:
          break;

     /* case LLC_GROUP_MGT: not supported if testing ssap */

     case LLC_SNA_PATH_CONTROL:
          break;

     case LLC_DOD_IP:
          break;

     case LLC_BSTAP:
          break;

     case LLC_PROWAY_NETMGT:
          break;

     case LLC_NOVELL:
          /* no control byte to parse */
          dll_length--;
          llc_protocol = XNS;
          goto llc_good;
```

```
        case LLC_EIA_RS511:
             break;

        case LLC_PROWAY:
             break;

        case LLC_SNAP:
             snap_ptr = (struct snap_headerstr *) &frame_ptr->llc_header;
             llc_protocol = snap_ptr->ether_type;
             dll_length += SNAP_LLC_HEADER_DIFF;
             break;

        case LLC_ISO:
             break;

        default:
             goto llc_bad;

        }

/*
 * Get frame type
 */
if (frame_ptr->llc_header.control & I_MASK)  /* 0 is Information frame */
     /* supervisory or Unnumbered frame, clear pf bit */
     type = frame_ptr->llc_header.control & PF_MASK;

switch (type)
     {
     case I_FRAME:
          if (llc_protocol != LLC_SNAP)
               dll_length++;
          break;

     case RR_FRAME:
          if (llc_protocol != LLC_SNAP)
```

```
            dll_length++;
        break;

    case REJ_FRAME:
        if (llc_protocol != LLC_SNAP)
            dll_length++;
        break;

    case RNR_FRAME:
        if (llc_protocol != LLC_SNAP)
            dll_length++;
        break;

    /* unnumbered commands/response 3 byte header */
    case UI_FRAME:
        break;

    case SABME_FRAME:
        break;

    case DISC_FRAME:
        break;

    case UA_FRAME:
        break;

    case DM_FRAME:
        break;

    case FRMR_FRAME:    /* has a additional 5 byte information field */
    /* IMPORTANT FRAME TYPE FOR SEEING PROBLEMS DO SOMETHING WITH THE 5 BYTES */
        if (llc_protocol != LLC_SNAP)
            dll_length += 5;
        goto llc_done;

    /* also unnumbered */
    case TEST_FRAME:
```

```
            goto llc_done;

        case XID_FRAME:     /* response is additional 3 bytes long */
            if (frame_ptr->llc_header.ssap & RESP_BIT) /* && (llc_protocol != LLC_SNAP) */
                dll_length += 3;
            goto llc_done;

        default:
            goto llc_done;
        }

llc_good:
    return (GOOD);

llc_done:
    return (DONE);

llc_bad:
    return (BAD);

}
```

```
/*
 *  rtp_icmp_p.c
 *
 *   (description)
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/rtp/SCCS/s.rtp_icmp_p.c
 *
 *  Date:       8/26/91
 *
 *  Revision:   1.5
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   08-kr-91  KR        Fixed udp ptr into source quench data
 *   06-22-91  KR        Stopped parse of middle or last fragment
 *   06-20-91  DPD       Fixed icmp off segment bug.
 *   06-12-91  KR        Added error counting to dest unreach, time exceeded,
 *                            param problem, and param problem option macros
 *
 */

static char rtp_icmp_p_c [] = "@(#)rtp_icmp_p.c   1.5";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include <sys/types.h>
```

```
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "protocols.h"
#include "stats.h"
#include "stats_icmp.h"
#include "rtp.h"
#include "rtp_ip.h"
#include "rtp_udp.h"
#include "rtp_icmp.h"

/*
 * Local Data Structures
 */
/*
StatsAddrEntry      *this_seg_addr_ptr;
StatsIcmpSegment    *this_seg_stats_ptr;
StatsProtocolEntry  *this_seg_protocol_ptr;

StatsAddrEntry      *src_seg_addr_ptr;
StatsIcmpSegment    *src_seg_stats_ptr;

StatsAddrEntry      *dst_seg_addr_ptr;
StatsIcmpSegment    *dst_seg_stats_ptr;

StatsAddrEntry      *src_node_addr_ptr;
StatsIcmpAddr       *src_node_stats_ptr;

StatsAddrEntry      *dst_node_addr_ptr;
StatsIcmpAddr       *dst_node_stats_ptr;

StatsAddrEntry      *dialog_addr_ptr;
StatsDialogEntry    *dialog_stats_ptr;
 */


/*
```

```
 * Macros for ICMP statistics
 * These use the macros defined in stats.h
 */

#define icmp_stats_frames { \
  stats_frames (ICMP_SEGMENT, ICMP_NODE, ICMP_PAIR, ICMP_PROTOCOL); }


#define icmp_stats_bytes { \
  stats_bytes (length, ICMP_SEGMENT, ICMP_NODE, ICMP_PAIR, ICMP_PROTOCOL); }


#define icmp_stats_errors { \
  stats_errors (ICMP_SEGMENT, ICMP_NODE, ICMP_PAIR, ICMP_PROTOCOL); }


#define icmp_stats_off_segs { \
  stats_off_segs (ICMP_SEGMENT, ICMP_NODE, ICMP_PAIR, ICMP_PROTOCOL); }


#define icmp_stats_transits { \
  stats_transits (ICMP_SEGMENT, ICMP_NODE, ICMP_PAIR, ICMP_PROTOCOL); }


#define icmp_stats_echo_reply {\
  stats_for_segs     (echoReply, echoReplyRate, ICMP_SEGMENT, AL_ECHO_REPLY, NULL); \
  stats_for_rcv_node (rcvEchoReply, rcvEchoReplyRate, ICMP_NODE, AL_RCV_ECHO_REPLY, NULL);
\
  stats_for_xmt_node (xmtEchoReply, xmtEchoReplyRate, ICMP_NODE, AL_XMT_ECHO_REPLY, NULL);
}


#define icmp_stats_unreach_net { \
  stats_for_segs     (destUnrNet, destUnrNetRate, ICMP_SEGMENT, AL_UNREACH_NET, NULL); \
  stats_for_rcv_node (rcvDestUnrNet, rcvDestUnrNetRate, ICMP_NODE, AL_RCV_UNREACH_NET,
NULL);\
```

```
  stats_for_xmt_node (xmtDestUnrNet, xmtDestUnrNetRate, ICMP_NODE, AL_XMT_UNREACH_NET,
NULL); }


#define icmp_stats_unreach_host { \
  stats_for_segs      (destUnrHost, destUnrHostRate, ICMP_SEGMENT, AL_UNREACH_HOST,NULL); \

  stats_for_rcv_node (rcvDestUnrHost, rcvDestUnrHostRate,
ICMP_NODE,AL_RCV_UNREACH_HOST,NULL);\
  stats_for_xmt_node (xmtDestUnrHost, xmtDestUnrHostRate, ICMP_NODE,
AL_XMT_UNREACH_HOST,NULL); }


#define icmp_stats_unreach_protocol { \
  stats_for_segs      (destUnrProtocol, destUnrProtocolRate, \
                       ICMP_SEGMENT, AL_UNREACH_PROTOCOL, NULL); \
  stats_for_rcv_node (rcvDestUnrProtocol, rcvDestUnrProtocolRate, \
                       ICMP_NODE, AL_RCV_UNREACH_PROTOCOL, NULL); \
  stats_for_xmt_node (xmtDestUnrProtocol, xmtDestUnrProtocolRate, \
                       ICMP_NODE, AL_XMT_UNREACH_PROTOCOL, NULL); }


#define icmp_stats_unreach_port { \
  stats_for_segs      (destUnrPort, destUnrPortRate, ICMP_SEGMENT, AL_UNREACH_PORT, NULL);
\
  stats_for_rcv_node (rcvDestUnrPort, rcvDestUnrPortRate,
ICMP_NODE,AL_RCV_UNREACH_PORT,NULL);\
  stats_for_xmt_node (xmtDestUnrPort, xmtDestUnrPortRate, ICMP_NODE,
AL_XMT_UNREACH_PORT,NULL); }


#define icmp_stats_unreach_need_frag { \
  stats_for_segs      (destUnrFrgmt, destUnrFrgmtRate, ICMP_SEGMENT, AL_UNREACH_FRAG,
NULL); \
  stats_for_rcv_node
(rcvDestUnrFrgmt,rcvDestUnrFrgmtRate,ICMP_NODE,AL_RCV_UNREACH_FRAG,NULL);\
```

```
  stats_for_xmt_node (xmtDestUnrFrgmt, xmtDestUnrFrgmtRate,ICMP_NODE,
AL_XMT_UNREACH_FRAG,NULL); }


#define icmp_stats_unreach_src_fail { \
  stats_for_segs      (destUnrSrcRoute,destUnrSrcRouteRate, \
                       ICMP_SEGMENT, AL_UNREACH_SRC_FAIL, NULL); \
  stats_for_rcv_node (rcvDestUnrSrcRoute, rcvDestUnrSrcRouteRate, \
                       ICMP_NODE, AL_RCV_UNREACH_SRC_FAIL,NULL);\
  stats_for_xmt_node (xmtDestUnrSrcRoute, xmtDestUnrSrcRouteRate, \
                       ICMP_NODE, AL_XMT_UNREACH_SRC_FAIL,NULL); }

#define icmp_stats_unreach_net_unknown { \
  stats_for_segs      (destUnrNetUnknown, destUnrNetUnknownRate, \
                       ICMP_SEGMENT, AL_UNREACH_NET_UNKNOWN, NULL); \
  stats_for_rcv_node (rcvDestUnrNetUnknown, rcvDestUnrNetUnknownRate, \
                       ICMP_NODE, AL_RCV_UNREACH_NET_UNKNOWN, NULL);\
  stats_for_xmt_node (xmtDestUnrNetUnknown, xmtDestUnrNetUnknownRate, \
                       ICMP_NODE, AL_XMT_UNREACH_NET_UNKNOWN, NULL); }


#define icmp_stats_unreach_host_unknown { \
  stats_for_segs      (destUnrHostUnknown, destUnrHostUnknownRate, \
                       ICMP_SEGMENT, AL_UNREACH_HOST_UNKNOWN, NULL); \
  stats_for_rcv_node (rcvDestUnrHostUnknown, rcvDestUnrHostUnknownRate, \
                       ICMP_NODE, AL_RCV_UNREACH_HOST_UNKNOWN, NULL);\
  stats_for_xmt_node (xmtDestUnrHostUnknown, xmtDestUnrHostUnknownRate, \
                       ICMP_NODE, AL_XMT_UNREACH_HOST_UNKNOWN, NULL); }


#define icmp_stats_unreach_host_isolated { \
     stats_for_segs (destUnrHostIsolated, destUnrHostIsolatedRate, \
                     ICMP_SEGMENT, AL_UNREACH_HOST_ISOLATED, NULL); \
  stats_for_rcv_node (rcvDestUnrHostIsolated, rcvDestUnrHostIsolatedRate, \
                       ICMP_NODE, AL_RCV_UNREACH_HOST_ISOLATED, NULL);\
  stats_for_xmt_node (xmtDestUnrHostIsolated, xmtDestUnrHostIsolatedRate, \
                       ICMP_NODE, AL_XMT_UNREACH_HOST_ISOLATED, NULL); }
```

```
#define icmp_stats_unreach_net_prohibited { \
  stats_for_segs      (destUnrNetProhibited, destUnrNetProhibitedRate, \
                       ICMP_SEGMENT, AL_UNREACH_NET_PROHIBITED, NULL); \
  stats_for_rcv_node (rcvDestUnrNetProhibited, rcvDestUnrNetProhibitedRate, \
                       ICMP_NODE, AL_RCV_UNREACH_NET_PROHIBITED, NULL);\
  stats_for_xmt_node (xmtDestUnrNetProhibited, xmtDestUnrNetProhibitedRate, \
                       ICMP_NODE, AL_XMT_UNREACH_NET_PROHIBITED, NULL); }


#define icmp_stats_unreach_host_prohibited { \
  stats_for_segs      (destUnrHostProhibited, destUnrHostProhibitedRate, \
                       ICMP_SEGMENT, AL_UNREACH_HOST_PROHIBITED, NULL); \
  stats_for_rcv_node (rcvDestUnrHostProhibited, rcvDestUnrHostProhibitedRate, \
                       ICMP_NODE, AL_RCV_UNREACH_HOST_PROHIBITED, NULL);\
  stats_for_xmt_node (xmtDestUnrHostProhibited, xmtDestUnrHostProhibitedRate, \
                       ICMP_NODE, AL_XMT_UNREACH_HOST_PROHIBITED, NULL); }


#define icmp_stats_unreach_net_tos . \
  stats_for_segs      (destUnrNetTos, destUnrNetTosRate, \
                       ICMP_SEGMENT, AL_UNREACH_NET_TOS, NULL); \
  stats_for_rcv_node (rcvDestUnrNetTos, rcvDestUnrNetTosRate, \
                       ICMP_NODE, AL_RCV_UNREACH_NET_TOS, NULL);\
  stats_for_xmt_node (xmtDestUnrNetTos, xmtDestUnrNetTosRate, \
                       ICMP_NODE, AL_XMT_UNREACH_NET_TOS, NULL); }


#define icmp_stats_unreach_host_tos { \
     stats_for_segs (destUnrHostTos, destUnrHostTosRate, \
                     ICMP_SEGMENT, AL_UNREACH_HOST_TOS, NULL); \
  stats_for_rcv_node (rcvDestUnrHostTos, rcvDestUnrHostTosRate, \
                       ICMP_NODE, AL_RCV_UNREACH_HOST_TOS, NULL);\
  stats_for_xmt_node (xmtDestUnrHostTos, xmtDestUnrHostTosRate, \
                       ICMP_NODE, AL_XMT_UNREACH_HOST_TOS, NULL); }
```

```
#define icmp_stats_src_quench { \
  stats_for_segs     (srcQuench, srcQuenchRate, ICMP_SEGMENT, AL_SRC_QUENCH, NULL); \
  stats_for_rcv_node (rcvSrcQuench, rcvSrcQuenchRate, ICMP_NODE, AL_RCV_SRC_QUENCH,NULL);\

  stats_for_xmt_node (xmtSrcQuench, xmtSrcQuenchRate, ICMP_NODE, AL_XMT_SRC_QUENCH,NULL);
}


#define icmp_stats_redirect { \
  stats_for_segs     (redir, redirRate, ICMP_SEGMENT, AL_REDIR, NULL); \
  stats_for_nodes    (redir, redirRate, ICMP_NODE, AL_REDIR,NULL); }


#define icmp_stats_redirect_net { \
  stats_for_segs     (redirNet, redirNetRate, ICMP_SEGMENT, AL_REDIR_NET, NULL); \
  stats_for_rcv_node (rcvRedirNet, rcvRedirNetRate, ICMP_NODE, AL_RCV_REDIR_NET,NULL);\
  stats_for_xmt_node (xmtRedirNet, xmtRedirNetRate, ICMP_NODE, AL_XMT_REDIR_NET,NULL); }


#define icmp_stats_redirect_host { \
  stats_for_segs     (redirHost, redirHostRate, ICMP_SEGMENT, AL_REDIR_HOST, NULL); \
  stats_for_rcv_node (rcvRedirHost, rcvRedirHostRate, ICMP_NODE, AL_RCV_REDIR_HOST,NULL);\

  stats_for_xmt_node (xmtRedirHost, xmtRedirHostRate, ICMP_NODE, AL_XMT_REDIR_HOST,NULL);
}


#define icmp_stats_redirect_net_tos { \
  stats_for_segs     (redirNetTos, redirNetTosRate, ICMP_SEGMENT, AL_REDIR_NET_TOS, NULL);
\
  stats_for_rcv_node (rcvRedirNetTos, rcvRedirNetTosRate,
ICMP_NODE,AL_RCV_REDIR_NET_TOS,NULL);\
  stats_for_xmt_node (xmtRedirNetTos, xmtRedirNetRate, ICMP_NODE, AL_XMT_REDIR_NET,NULL);
}
```

```
#define icmp_stats_redirect_host_tos { \
  stats_for_segs      (redirHostTos, redirHostTosRate, ICMP_SEGMENT, AL_REDIR_HOST_TOS,
NULL); \
  stats_for_rcv_node (rcvRedirHostTos, rcvRedirHostTosRate, \
                      ICMP_NODE, AL_RCV_REDIR_HOST_TOS, NULL);\
  stats_for_xmt_node (xmtRedirHostTos, xmtRedirHostTosRate, \
                      ICMP_NODE, AL_XMT_REDIR_HOST_TOS, NULL); }


#define icmp_stats_echo { \
  stats_for_segs      (echoReq, echoReqRate, ICMP_SEGMENT, AL_ECHO, NULL); \
  stats_for_rcv_node (rcvEchoReq, rcvEchoReqRate, ICMP_NODE,AL_RCV_ECHO, NULL);\
  stats_for_xmt_node (xmtEchoReq, xmtEchoReqRate, ICMP_NODE, AL_XMT_ECHO, NULL); }


#define icmp_stats_time_exceeded { \
  stats_for_segs      (timeExceeded, timeExceededRate, ICMP_SEGMENT, AL_TIME_EXCEEDED,
NULL); \
  stats_for_nodes     (timeExceeded, timeExceededRate, ICMP_NODE, AL_TIME_EXCEEDED, NULL);
\
  icmp_stats_errors; }


#define icmp_stats_time_exceeded_in_trans { \
  stats_for_segs      (timeExceededInTransit, timeExceededInTransitRate, \
                      ICMP_SEGMENT, AL_TIME_EXCEEDED_IN_TRANS, NULL); \
  stats_for_rcv_node (rcvTimeExceededInTransit, rcvTimeExceededInTransitRate, \
                      ICMP_NODE, AL_RCV_TIME_EXCEEDED_IN_TRANS, NULL); \
  stats_for_xmt_node (xmtTimeExceededInTransit, xmtTimeExceededInTransitRate, \
                      ICMP_NODE, AL_XMT_TIME_EXCEEDED_IN_TRANS, NULL); }


#define icmp_stats_time_exceeded_in_reass { \
  stats_for_segs      (timeExceededInReass, timeExceededInReassRate, \
                      ICMP_SEGMENT, AL_TIME_EXCEEDED_IN_REASS, NULL); \
  stats_for_rcv_node (rcvTimeExceededInReass, rcvTimeExceededInReassRate, \
                      ICMP_NODE, AL_RCV_TIME_EXCEEDED_IN_REASS, NULL); \
```

```
  stats_for_xmt_node (xmtTimeExceededInReass, xmtTimeExceededInReassRate, \
                      ICMP_NODE, AL_XMT_TIME_EXCEEDED_IN_REASS, NULL); }


#define icmp_stats_param { \
  stats_for_segs     (paramProblem, paramProblemRate, ICMP_SEGMENT, AL_PARAM, NULL); \
  stats_for_rcv_node (rcvParamProblem, rcvParamProblemRate, ICMP_NODE,AL_RCV_PARAM,
NULL);\
  stats_for_xmt_node (xmtParamProblem, xmtParamProblemRate, ICMP_NODE, AL_XMT_PARAM,
NULL); \
  icmp_stats_errors; }


#define icmp_stats_param_option { \
  stats_for_segs     (paramProblemOption, paramProblemOptionRate, \
                      ICMP_SEGMENT, AL_PARAM_OPTION, NULL); \
  stats_for_rcv_node (rcvParamProblemOption, rcvParamProblemOptionRate, \
                      ICMP_NODE,AL_RCV_PARAM_OPTION, NULL);\
  stats_for_xmt_node (xmtParamProblemOption, xmtParamProblemOptionRate, \
                      ICMP_NODE, AL_XMT_PARAM_OPTION, NULL); \
  icmp_stats_errors; }


#define icmp_stats_timestamp { \
  stats_for_segs     (timestampReq, timestampReqRate, ICMP_SEGMENT, AL_TIMESTAMP, NULL); \
  stats_for_rcv_node (rcvTimestampReq, rcvTimestampReqRate, ICMP_NODE,AL_RCV_TIMESTAMP,
NULL);\
  stats_for_xmt_node (xmtTimestampReq, xmtTimestampReqRate, ICMP_NODE, AL_XMT_TIMESTAMP,
NULL); }


#define icmp_stats_timestamp_reply { \
  stats_for_segs     (timestampReply, timestampReplyRate, \
                      ICMP_SEGMENT, AL_TIMESTAMP_REPLY, NULL); \
  stats_for_rcv_node (rcvTimestampReply, rcvTimestampReplyRate, \
                      ICMP_NODE,AL_RCV_TIMESTAMP_REPLY,  NULL); \
```

```
  stats_for_xmt_node (xmtTimestampReply, xmtTimestampReplyRato, \
                      ICMP_NODE, AL_XMT_TIMESTAMP_REPLY,  NULL); }


#define icmp_stats_mask { \
  stats_for_segs     (addrMaskReq, addrMaskReqRate, ICMP_SEGMENT, AL_MASK, NULL); \
  stats_for_rcv_node (rcvAddrMaskReq, rcvAddrMaskReqRate, ICMP_NODE,AL_RCV_MASK, NULL); \

  stats_for_xmt_node (xmtAddrMaskReq, xmtAddrMaskReqRate, ICMP_NODE, AL_XMT_MASK, NULL); }


#define icmp_stats_mask_reply { \
  stats_for_segs     (addrMaskReply, addrMaskReplyRate, ICMP_SEGMENT, AL_MASK_REPLY,
NULL); \
  stats_for_rcv_node (rcvAddrMaskReply,
rcvAddrMaskReplyRate,ICMP_NODE,AL_RCV_MASK_REPLY,NULL); \
  stats_for_xmt_node (xmtAddrMaskReply, xmtAddrMaskReplyRate, ICMP_NODE,
AL_XMT_MASK_REPLY,NULL);}


/*****************************************************************************
 *
 * Internet Control Message Protocol Parse Routine
 *
 *   Function  Parse ICMP frame
 *
 *   Input          pointer to the ICMP message, length of the message
 *   Output         objects updated
 */

Uint32 rtp_icmp_parse (layer_ptr, length)

     char                        *layer_ptr;
     Uint32                           length;
 {
```

```
register Uint32              i, result;
register struct icmp      *message_ptr;
register struct udphdr    *udp_ptr;
register struct ip        *ip_ptr;
AlarmUserData             stats_alarm_data;

register StatsAddrEntry       *this_seg_addr_ptr;
register StatsIcmpSegment     *this_seg_stats_ptr;
register StatsProtocolEntry   *this_seg_protocol_ptr;

register StatsAddrEntry       *src_seg_addr_ptr;
register StatsIcmpSegment     *src_seg_stats_ptr;

register StatsAddrEntry       *dst_seg_addr_ptr;
register StatsIcmpSegment     *dst_seg_stats_ptr;

register StatsAddrEntry       *src_node_addr_ptr;
register StatsIcmpAddr        *src_node_stats_ptr;

register StatsAddrEntry       *dst_node_addr_ptr;
register StatsIcmpAddr        *dst_node_stats_ptr;

register StatsAddrEntry       *dialog_addr_ptr;
register StatsDialogEntry     *dialog_stats_ptr;

register StatsAddrEntry         *ip_dst_addr_ptr;
register StatsAddrEntry         *ip_src_addr_ptr;

/* If this is a middle or last fragment, there's no header to parse.
 * It's already been done (or will be).
 */
if ((ip_seg_type == MIDDLE_FRAGMENT) || (ip_seg_type == LAST_FRAGMENT))
     {
     goto icmp_done;
     }
```

```
/*
 * Set up local variables
 */
result = GOOD;
message_ptr = (struct icmp *) layer_ptr;


 /*
  * Find all the statistics records
  */
icmp_this_seg_addr_ptr          = NULL;
icmp_src_seg_addr_ptr           = NULL;
icmp_dst_seg_addr_ptr           = NULL;
icmp_this_seg_stats_ptr         = NULL;
icmp_src_seg_stats_ptr          = NULL;
icmp_dst_seg_stats_ptr          = NULL;
icmp_src_node_stats_ptr         = NULL;
icmp_dst_node_stats_ptr         = NULL;
icmp_dialog_stats_ptr           = NULL;


icmp_src_node_addr_ptr = stats_icmp_get_addr (&ip_src_addr);
if (icmp_src_node_addr_ptr != NULL)
     {
     icmp_src_node_stats_ptr = (StatsIcmpAddr *) icmp_src_node_addr_ptr->stats_ptr;
     icmp_src_seg_addr_ptr = stats_iomp_get_segment
(icmp_src_node_addr_ptr->address.segment1);
     }
else
     {
     /**** IF GLOBAL IP POINTERS IMPLEMENTED DON'T NEED TO DO LOOKUP *****/
     if (ip_src_addr_ptr = (StatsAddrEntry *)stats_ip_lookup_addr(&ip_src_addr))
          icmp_src_seg_addr_ptr = stats_icmp_get_segment
(ip_src_addr_ptr->address.segment1);
     }
if (icmp_src_seg_addr_ptr != NULL)
     icmp_src_seg_stats_ptr = (StatsIcmpSegment *) iomp_src_seg_addr_ptr->stats_ptr;
```

```
icmp_dst_node_addr_ptr = stats_icmp_get_addr (&ip_dst_addr);
if (icmp_dst_node_addr_ptr != NULL)
     {
     icmp_dst_node_stats_ptr = (StatsIcmpAddr *) icmp_dst_node_addr_ptr->stats_ptr;
     icmp_dst_seg_addr_ptr  = stats_icmp_get_segment
(icmp_dst_node_addr_ptr->address.segment1);
     }
else
     {
     /**** IF GLOBAL IP POINTERS IMPLEMENTED DON'T NEED TO DO LOOKUP *****/
     if (ip_dst_addr_ptr = (StatsAddrEntry *)stats_ip_lookup_addr(&ip_dst_addr))
          icmp_dst_seg_addr_ptr = stats_icmp_get_segment
(ip_dst_addr_ptr->address.segment1);
     }
if (icmp_dst_seg_addr_ptr != NULL)
     icmp_dst_seg_stats_ptr = (StatsIcmpSegment *) icmp_dst_seg_addr_ptr->stats_ptr;

icmp_this_seg_addr_ptr = stats_icmp_get_segment (mySegmentId);
if (icmp_this_seg_addr_ptr != NULL)
     icmp_this_seg_stats_ptr = (StatsIcmpSegment *) icmp_this_seg_addr_ptr->stats_ptr;


icmp_dialog_addr_ptr = stats_icmp_get_dialog (&ip_src_addr, &ip_dst_addr);
if (icmp_dialog_addr_ptr != NULL)
     icmp_dialog_stats_ptr = (StatsDialogEntry *) icmp_dialog_addr_ptr->stats_ptr;


/*
 * Set up pointers for segment, address, and dialog statistics used by the common macros
 */
this_seg_addr_ptr     = icmp_this_seg_addr_ptr;
this_seg_stats_ptr    = icmp_this_seg_stats_ptr;

src_seg_addr_ptr      = icmp_src_seg_addr_ptr;
src_seg_stats_ptr     = icmp_src_seg_stats_ptr;
```

```
dst_seg_addr_ptr      = icmp_dst_seg_addr_ptr;
dst_seg_stats_ptr     = icmp_dst_seg_stats_ptr;

src_node_addr_ptr     = icmp_src_node_addr_ptr;
src_node_stats_ptr    = icmp_src_node_stats_ptr;

dst_node_addr_ptr     = icmp_dst_node_addr_ptr;
dst_node_stats_ptr    = icmp_dst_node_stats_ptr;

dialog_addr_ptr       = icmp_dialog_addr_ptr;
dialog_stats_ptr      = icmp_dialog_stats_ptr;


/*
 * Update the age timer in each address structure. Set the 10 second sample
 * rate indicator for the event manager so it will calculate rates.
 */
stats_update_age_timers;
stats_set_rate_10s;


/*
 * Do common statistics
 */
icmp_stats_frames;
icmp_stats_bytes;
icmp_stats_off_segs;
icmp_stats_transits;


/*
 * Check the length of the ICMP message
 */
if (length < ICMP_MINLEN)
     goto icmp_bad;
```

```
/*
 * Save the message type in the dialog stats, then process it
 */
if (icmp_dialog_stats_ptr != NULL)
     stats_save_protocol_in_dialog (icmp_dialog_stats_ptr, message_ptr->icmp_type);

switch (message_ptr->icmp_type)
     {
     case ICMP_ECHOREPLY:
          icmp_stats_echo_reply;
          break;    /* ICMP_ECHO_REPLY */


     case ICMP_UNREACH:
          switch (message_ptr->icmp_code)
               {
               /*
                * Check the unreachable code
                */
               case ICMP_UNREACH_NET:
                    icmp_stats_unreach_net;
                    break;
               case ICMP_UNREACH_HOST:
                    icmp_stats_unreach_host;
                    break;
               case ICMP_UNREACH_PROTOCOL:
                    icmp_stats_unreach_protocol;
                    break;
               case ICMP_UNREACH_PORT:
                    icmp_stats_unreach_port;
                    break;
               case ICMP_UNREACH_NEEDFRAG:
                    icmp_stats_unreach_need_frag;
                    break;
               case ICMP_UNREACH_SRCFAIL:
                    icmp_stats_unreach_src_fail;
                    break;
```

```
                case ICMP_UNREACH_NETUNKNOWN:
                     icmp_stats_unreach_net_unknown;
                     break;
                case ICMP_UNREACH_HOSTUNKNOWN:
                     icmp_stats_unreach_host_unknown;
                     break;
                case ICMP_UNREACH_HOSTISOLATED:
                     icmp_stats_unreach_host_isolated;
                     break;
                case ICMP_UNREACH_NETPROHIBITED:
                     icmp_stats_unreach_net_prohibited;
                     break;
                case ICMP_UNREACH_HOSTPROHIBITED:
                     icmp_stats_unreach_host_prohibited;
                     break;
                case ICMP_UNREACH_NETTOS:
                     icmp_stats_unreach_net_tos;
                     break;
                case ICMP_UNREACH_HOSTTOS:
                     icmp_stats_unreach_host_tos;
                     break;

                default:
                     break;
                }
           break;    /* ICMP_UNREACH */


      case ICMP_SRCQUENCH:
           icmp_stats_src_quench;
           if (length > (8 + sizeof(struct ip) + 3) )
                {
                layer_ptr = (char *) &message_ptr->icmp_ip;
                ip_ptr = (struct ip *) layer_ptr;
                layer_ptr = layer_ptr + (ip_ptr->ip_hl << 2);
                udp_ptr = (struct udphdr *) layer_ptr;
                srcQuenchData.ip_src_addr = ALIGN(ip_ptr->ip_src);
```

```
            srcQuenchData.ip_dst_addr = ALIGN(ip_ptr->ip_dst);
            srcQuenchData.protocol = (Uint32) ip_ptr->ip_p;
            srcQuenchData.src_port = (Uint32) udp_ptr->uh_sport;
            srcQuenchData.dst_port = (Uint32) udp_ptr->uh_dport;
            srcQuenchData.data_ptr = (Uint32 *) ip_ptr;

            if (srcQuenchData.protocol == UDP_PROTOCOL)
                 {
                 rtp_udp_src_quench ();
                 }

            if (srcQuenchData.protocol == TCP_PROTOCOL)
                 {
                 rtp_tcp_src_quench ();
                 }
            }
       break;    /* ICMP_SRCQUENCH */


  case ICMP_REDIRECT:
       /*
       icmp_stats_redirect;
       */
       switch (message_ptr->icmp_code)
            {
            case ICMP_REDIRECT_NET:
                 icmp_stats_redirect_net;
                 break;
            case ICMP_REDIRECT_HOST:
                 icmp_stats_redirect_host;
                 break;
            case ICMP_REDIRECT_TOSNET:
                 icmp_stats_redirect_net_tos;
                 break;
            case ICMP_REDIRECT_TOSHOST:
                 icmp_stats_redirect_host_tos;
                 break;
```

```
        }
    break;    /* ICMP_REDIRECT */


case ICMP_ECHO:
    icmp_stats_echo;
    break;    /* ICMP_ECHO */


case ICMP_TIMXCEED:
    /*
    icmp_stats_time_exceeded;
    */
    switch (message_ptr->icmp_code)
        {
        /*
         * Check the time exceeded code
         */
        case ICMP_TIMXCEED_INTRANS:
            icmp_stats_time_exceeded_in_trans;
            break;
        case ICMP_TIMXCEED_REASS:
            icmp_stats_time_exceeded_in_reass;
            break;
        default:
            break;
        }
    break;    /* ICMP_TIMXCEED */


case ICMP_PARAM:
    switch (message_ptr->icmp_code)
        {
        case ICMP_PARAM_OPTION:
```

```
                    icmp_stats_param_option;
                    break;
                    }

               default:
                    {
                    icmp_stats_param;
                    break;
                    }
          break;    /* ICMP_PARAM */


     case ICMP_TSTAMP:
          icmp_stats_timestamp;
          break;    /* ICMP_TSTAMP */


     case ICMP_TSTAMPREPLY:
          icmp_stats_timestamp_reply;
          break;    /* ICMP_TSTAMPREPLY */


     case ICMP_MASKREQ:
          icmp_stats_mask;
          break;    /* ICMP_MASKREQ */


     case ICMP_MASKREPLY:
          icmp_stats_mask_reply;
          break;    /* ICMP_MASKREPLY */

     }


icmp_done:
     return (DONE);
```

```
icmp_bad:
        icmp_stats_errors;
        return (BAD);

}
```

```
/*
 *  rtp_ip_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/rtp/SCCS/s.rtp_ip_p.c
 *
 *  Date:       8/23/91
 *
 *  Revision:   1.9
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO        Description of change. (latest first)
 *   --------  ---        ------------ ---------------------------------
 *
 *   08-15-91  KR         set rtp_net_multicast or rtp_net_broadcast when necessary
 *                             fixed some run over lines
 *   07-16-91  DPD        if duplicate mac, don't write over the mac address
 *   06-19-91  DPD        don't call stats_check_for_duplicate_ip if rtp_broadcast
 *                             or rtp_multicast for destination node
 *   06-18-91  KR         stopped parse if don't frag and non-zero offset
 *   06-18-91  DPD        enabled checks for duplicate ip
 *   06-05-91  DPD        set mib address type when putting mac addr into
 *                             ip addr struct
 *   05-31-91  KMJ        forced vars into registers
 */

static char rtp_ip_p_c [] = "@(#)rtp_ip_p.c  1.9";

#include <stdio.h>
```

```
#include <cci_std.h>

#include "system.h"
#include <sys/types.h>
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "stats.h"
#include "stats_dll.h"
#include "stats_ip.h"
#include "in.h"
#include "rtp.h"
#include "rtp_dll.h"
#include "rtp_ip.h"


/*****************************************************************************
 *
 * IP monitoring
 */


/*
 * Global IP data structures
 */
Uint32                  ip_src_addr;
Uint32                  ip_dst_addr;
Uint32                  ip_seg_type;


/*
 * Local Data Structures
 */
static StatsAddrEntry         *this_seg_addr_ptr;
static StatsIpSegment         *this_seg_stats_ptr;
static StatsProtocolEntry     *this_seg_protocol_ptr;
```

```
static StatsAddrEntry       *src_seg_addr_ptr;
static StatsIpSegment          *src_seg_stats_ptr;
static StatsProtocolEntry   *src_seg_protocol_ptr;

static StatsAddrEntry       *dst_seg_addr_ptr;
static StatsIpSegment          *dst_seg_stats_ptr;
static StatsProtocolEntry   *dst_seg_protocol_ptr;

static StatsAddrEntry       *src_node_addr_ptr;
static StatsIpAddr             *src_node_stats_ptr;
static StatsProtocolEntry   *src_node_protocol_ptr;

static StatsAddrEntry       *dst_node_addr_ptr;
static StatsIpAddr             *dst_node_stats_ptr;
static StatsProtocolEntry   *dst_node_protocol_ptr;

static StatsAddrEntry       *dialog_addr_ptr;
static StatsDialogEntry     *dialog_stats_ptr;


/*
 * Macros for IP statistics
 * These use the macros defined in stats.h
 */
#define ip_stats_frames { \
      stats_frames (IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }

#define ip_stats_bytes { \
     stats_bytes (length, IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }

#define ip_stats_hdr_bytes { \
     stats_hdr_bytes (ip_hdr_length, IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }

#define ip_stats_errors { \
     stats_errors (IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }
```

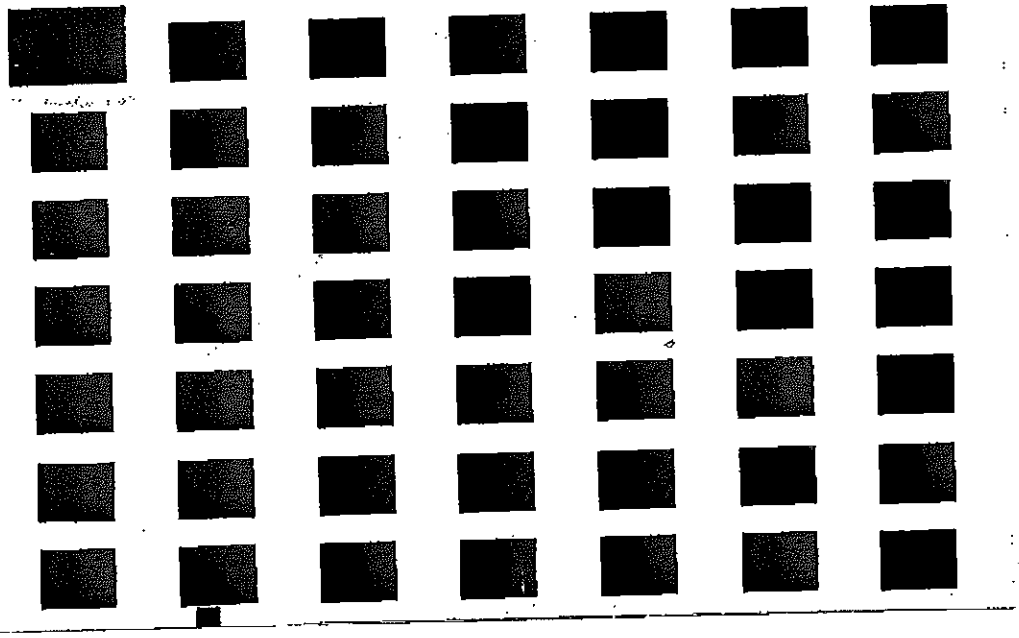
```
#define ip_stats_bcasts { \
     stats_bcasts (IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }

#define ip_stats_mcasts { \
     stats_mcasts (IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }

#define ip_stats_protocol { \
     stats_protocol (IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }

#define ip_stats_off_segs { \
     stats_off_segs (IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }

#define ip_stats_transits { \
     stats_transits (IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }

#define ip_stats_fragments { \
     stats_fragments (IP_SEGMENT, IP_NODE, IP_PAIR, IP_PROTOCOL); }



/*****************************************************************************
 *
 * Do option processing on a datagram
 */

Uint32 rtp_ip_dooptions (datagram_ptr)

     struct ip       *datagram_ptr;
{
     register u_char     i, *cp;
     register Uint32     opt, optlen, cnt, off, code;

cp = (u_char *)(datagram_ptr + 1);
cnt = (datagram_ptr->ip_hl << 2) - sizeof (struct ip);
```

```
/* Process each option */
for (; cnt > 0; cnt -= optlen, cp += optlen)
     {
     opt = cp[IPOPT_OPTVAL];
     if (opt == IPOPT_EOL)
          break;
     if (opt == IPOPT_NOP)
          optlen = 1;
     else
          {
          optlen = cp[IPOPT_OLEN];
          if (optlen <= 0 || optlen > cnt)
               {
               code = &cp[IPOPT_OLEN] - (u_char *)datagram_ptr;
               goto ip_options_bad;
               }
          }
     switch (opt)
          {
          default:
               break;

          case IPOPT_LSRR:
          case IPOPT_SSRR:
               break;
          case IPOPT_RR:
               break;
          case IPOPT_TS:
               break;
          }
     }

ip_options_good:
return (GOOD);

ip_options_bad:
return (BAD);
```

```
}


/*****************************************************************************
 *
 * Internet Protocol Parse Routine
 *
 *   Function  Parse IP frame
 *
 *   Input          pointer to the datagram, length of the datagram
 *   Output         objects updated, next layer parse routine called
 */

Uint32 rtp_ip_parse (layer_ptr, length)

     char                *layer_ptr;
     Uint32                   length;
{
     register Uint32                    i, result, *uint32_ptr;
     register Uint32                    protocol;
     register struct ip            *datagram_ptr;
     register Uint32                    ip_hdr_length;
     AlarmUserData                 stats_alarm_data;

/*
 * Set up local variables
 */

result = GOOD;

ip_seg_type = NOT_A_FRAGMENT;

datagram_ptr = (struct ip *) layer_ptr;

ip_hdr_length = datagram_ptr->ip_hl << 2;
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones, Kary Robertson, David M. Thompson and Gerard White

02



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
protocol = (Uint32)datagram_ptr->ip_p;

rtp_net_broadcast = FALSE;
rtp_net_multicast = FALSE;


/*
 * For alignment reasons, the ip addresses cannot be defined as long within
 * the frame structure.  Since it's easier to handle them as long, convert them.
 */
ip_src_addr = (datagram_ptr->ip_src[0] << 24) + (datagram_ptr->ip_src[1] << 16) +
                  (datagram_ptr->ip_src[2] << 8)  +  datagram_ptr->ip_src[3];

ip_dst_addr = (datagram_ptr->ip_dst[0] << 24) + (datagram_ptr->ip_dst[1] << 16) +
                  (datagram_ptr->ip_dst[2] << 8)  +  datagram_ptr->ip_dst[3];


/*
 * Test for the broadcast address
 */
 if (RTP_IP_CLASSA (ip_dst_addr))
     {
     if ( ((ip_dst_addr & RTP_IP_CLASSA_HOST) == 0) ||
          ((ip_dst_addr & RTP_IP_CLASSA_HOST) == RTP_IP_CLASSA_HOST) )
          rtp_net_broadcast = TRUE;
     }

else if (RTP_IP_CLASSB (ip_dst_addr))
     {
     if ( ((ip_dst_addr & RTP_IP_CLASSB_HOST) == 0) ||
          ((ip_dst_addr & RTP_IP_CLASSB_HOST) == RTP_IP_CLASSB_HOST) )
          rtp_net_broadcast = TRUE;
     }

else if (RTP_IP_CLASSC (ip_dst_addr))
     {
     if ( ((ip_dst_addr & RTP_IP_CLASSC_HOST) == 0) ||
```

```
            ((ip_dst_addr & RTP_IP_CLASSC_HOST) == RTP_IP_CLASSC_HOST) )
            rtp_net_broadcast = TRUE;
      }


/*
 * Test for a multicast address
 * Class D addresses are used for multicast.
 */
else if (RTP_IP_CLASSD (ip_dst_addr))
      rtp_net_multicast = TRUE;


/*
 * Find all statistics records
 */
if (ip_src_node_addr_ptr = src_node_addr_ptr = stats_ip_get_addr (&ip_src_addr))
      {
      ip_src_node_stats_ptr = src_node_stats_ptr =
      (StatsIpAddr *) src_node_addr_ptr->stats_ptr;

      ip_src_seg_addr_ptr = src_seg_addr_ptr  =
      stats_ip_get_segment (src_node_addr_ptr->address.segment1);

      if (src_seg_addr_ptr != NULL)
            ip_src_seg_stats_ptr = src_seg_stats_ptr = (StatsIpSegment *)
src_seg_addr_ptr->stats_ptr; else
            ip_src_seg_stats_ptr = src_seg_stats_ptr = NULL;
      }
else
      {
      ip_src_node_stats_ptr = src_node_stats_ptr = NULL;
      ip_src_seg_addr_ptr = src_seg_addr_ptr  = NULL;
      ip_src_seg_stats_ptr = src_seg_stats_ptr = NULL;
      }
if (ip_dst_node_addr_ptr = dst_node_addr_ptr = stats_ip_get_addr (&ip_dst_addr))
```

```
     {
     ip_dst_node_stats_ptr = dst_node_stats_ptr = (StatsIpAddr *)
dst_node_addr_ptr->stats_ptr;
     ip_dst_seg_addr_ptr = dst_seg_addr_ptr  =
     stats_ip_get_segment (dst_node_addr_ptr->address.segment1);
     if (dst_seg_addr_ptr != NULL)
          ip_dst_seg_stats_ptr = dst_seg_stats_ptr = (StatsIpSegment *)
dst_seg_addr_ptr->stats_ptr;
     else
          ip_dst_seg_stats_ptr = dst_seg_stats_ptr = NULL;
     }
else
     {
     ip_dst_node_stats_ptr = dst_node_stats_ptr = NULL;
     ip_dst_seg_addr_ptr = dst_seg_addr_ptr  = NULL;
     ip_dst_seg_stats_ptr = dst_seg_stats_ptr = NULL;
     }

if (ip_this_seg_addr_ptr = this_seg_addr_ptr = stats_ip_get_segment (mySegmentId))
     ip_this_seg_stats_ptr = this_seg_stats_ptr = (StatsIpSegment *)
this_seg_addr_ptr->stats_ptr;
else
     ip_this_seg_stats_ptr = this_seg_stats_ptr = NULL;

if (ip_dialog_addr_ptr = dialog_addr_ptr = stats_ip_get_dialog (&ip_src_addr,
&ip_dst_addr))
     ip_dialog_stats_ptr = dialog_stats_ptr = (StatsDialogEntry *)
dialog_addr_ptr->stats_ptr;
else
     ip_dialog_stats_ptr = dialog_stats_ptr = NULL;


/*
 * Update the age timer in each address struoture and set the 10 second rate
 * sample period so that the event manager will calculate rates.
 */
stats_update_age_timers;
```

```
stats_set_rate_10s;


/*
 * Set the last mac address seen for the ip address
 */
if (src_node_addr_ptr != NULL)
{
     if (stats_check_for_duplicate_ip
                    (&src_node_addr_ptr->address, &mac_src_addr) == FALSE)
     {
          src_node_addr_ptr->address.addressType |= MibMacAddress1;
          src_node_addr_ptr->address.macAddress1 = mac_src_addr;
     }
}

if (dst_node_addr_ptr != NULL)
{
     if ((rtp_broadcast == FALSE) && (rtp_multicast == FALSE))
     {
          if (stats_check_for_duplicate_ip
                         (&dst_node_addr_ptr->address, &mac_dst_addr) == FALSE)
          {
               dst_node_addr_ptr->address.addressType |= MibMacAddress1;
               dst_node_addr_ptr->address.macAddress1 = mac_dst_addr;
          }
     }
}


/*
 * Do some statistics
 */
ip_stats_frames;
ip_stats_bytes;
ip_stats_hdr_bytes;
ip_stats_off_segs;
```

```
ip_stats_transits;


/*
 * Count broadcasts and multicasts
 */
if (rtp_net_broadcast)
     ip_stats_bcasts;

if (rtp_net_multicast)
     ip_stats_mcasts;


/*
 * Determine if the frame is for this station
 */
if (ip_dst_addr == myIpAddr)
     rtp_for_this_station = YES;



/*
 * Determine if the frame is from another monitor
 */
if (src_node_addr_ptr != NULL)
     {
     if ( (rtp_for_this_station == NO) && (src_node_addr_ptr->flags & MibMonitor) )
          rtp_from_another_monitor = YES;
     }


/*
 *  Find statistics structure for the protocol distribution statistics
 */
if (src_node_stats_ptr != NULL)
     ip_src_node_protocol_ptr = src_node_protocol_ptr =
     stats_get_protocol (&src_node_stats_ptr->protocolQ, protocol);
```

```
else
     ip_src_node_protocol_ptr = src_node_protocol_ptr = NULL;

if (dst_node_stats_ptr != NULL)
     ip_dst_node_protocol_ptr = dst_node_protocol_ptr =
     stats_get_protocol (&dst_node_stats_ptr->protocolQ, protocol);
else
     ip_dst_node_protocol_ptr = dst_node_protocol_ptr = NULL;

if (this_seg_stats_ptr != NULL)
     ip_this_seg_protocol_ptr = this_seg_protocol_ptr =
     stats_get_protocol (&this_seg_stats_ptr->protocolQ, protocol);
else
     ip_this_seg_protocol_ptr = this_seg_protocol_ptr = NULL;

if (src_seg_stats_ptr != NULL)
     ip_src_seg_protocol_ptr = src_seg_protocol_ptr  =
     stats_get_protocol (&src_seg_stats_ptr->protocolQ, protocol);
else
     ip_src_seg_protocol_ptr = src_seg_protocol_ptr  = NULL;

if (dst_seg_stats_ptr != NULL)
     ip_dst_seg_protocol_ptr = dst_seg_protocol_ptr  =
     stats_get_protocol (&dst_seg_stats_ptr->protocolQ, protocol);
else
     ip_dst_seg_protocol_ptr = dst_seg_protocol_ptr  = NULL;


/*
 * Keep protocol distribution statistics.
 * Pass the protocol as alarm data in case an alarm occurs
 */
stats_alarm_data.length = 4;
/*
 * replaced
 * bcopy (&protocol, stats_alarm_data.data, 4);
 * with
```

```
 */
uint32_ptr = (Uint32 *)stats_alarm_data.data;
*uint32_ptr = protocol;
ip_stats_protocol;


/*
 * Check the header length for the following problems:
 *        1.   hdr length < minimum length
 *        2.    hdr length is ok but frame is truncated to < minimum length
 *        3,4. hdr length in frame is inconsistent with total frame length field of hdr
 */
if ( (ip_hdr_length < sizeof(struct ip) ) ||
     ( (ip_hdr_length > length) && (length < sizeof(struct ip)) ) ||
      (ip_hdr_length > datagram_ptr->ip_len) ||
      (datagram_ptr->ip_len < ip_hdr_length) )
          {
          ip_stats_errors;
          goto ip_bad;
          }

/*
 * Process the options
 */
if (ip_hdr_length > sizeof(struct ip))
     rtp_ip_dooptions (datagram_ptr);

/*
 * Check for fragmentation
 *
 * DF: 0 = may fragment, 1 = don't fragment
 * MF: 0 = last fragment, 1 = more fragments
 * If offset or IP_MF are set, the message is fragmented
 * Set ip_seg_type appropriately for higher layer parse routines
 * so they know whether to parse a protocol header.
 */
```

```
if ( (datagram_ptr->ip_off & 0x1fff) || (datagram_ptr->ip_off & IP_MF) )
     {
     if ( (datagram_ptr->ip_off & 0x1FFF) == 0 )
          ip_seg_type = FIRST_FRAGMENT;

     else if (datagram_ptr->ip_off & IP_MF)
          ip_seg_type = MIDDLE_FRAGMENT;

     else
          ip_seg_type = LAST_FRAGMENT;

     ip_stats_fragments;
     }

if ( (datagram_ptr->ip_off & 0x1fff) && (datagram_ptr->ip_off & IP_DF) )
     {
     ip_stats_errors;
     goto ip_bad;
     }


/*
 * Parse the next protocol
 */
switch (protocol)
     {
     case ICMP_PROTOCOL:
          result = rtp_icmp_parse (layer_ptr + ip_hdr_length,
                                   datagram_ptr->ip_len - ip_hdr_length);

          break;

     case UDP_PROTOCOL:
          result = rtp_udp_parse (layer_ptr + ip_hdr_length,
                                  datagram_ptr->ip_len - ip_hdr_length);

          break;

     case TCP_PROTOCOL:
```

```
            result = rtp_tcp_parse (layer_ptr + ip_hdr_length,
                                    datagram_ptr->ip_len - ip_hdr_length);
            break;

      default:
            break;
      }

return (result);


ip_bad:
return (BAD);

ip_done:
return (DONE);
}
```

```
/*
 *  rtp_nfs_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/rtp/SCCS/s.rtp_nfs_p.c
 *
 *  Date:       8/26/91
 *
 *  Revision:   1.14
 *
 *
 *  Changes:
 *
 *   MM-DD-YY   WHO        Description of change. (latest first)
 *   --------   ---        ---------------------------------------------
 *
 *   08-15-91   KR         changed mount parse to check for non-zero return code in rsp
 *   07-22-91   KR         changed port references to protocol
 *                              added check for NFS_OUTSTANDING status when searching for
 *
xid
 *   06-24-91   KR         moved off_seg count to within call checks
 *   06-21-91   DPD        fixed off segment bug
 *   06-18-91   KR         changed read and write byte macros to use byte_count instead
 *                              length from rpcInfo
 *                              added flow control counts
 *
 *   06-15-91   KR         added unlink of file system on unsuccessful mount
 *   06-10-91   KR         added checks for bad cred and verf lengths
 *   06-10-91   KR         file system to unmount was being set up from server instead of
 *                              client file system link.  Although dequeue would have
 *
failed,
 *                              active file system link would've been freed, resulting in
 *
```

```
 *                              a utlb miss when reused.
 *    06-06-91  KR          Added unlink of file system on unmount
 */

static char rtp_nfs_p_c [] = "@(#)rtp_nfs_p.c     1.14";

#include <stdio.h>

#include <cci_std.h>

#include <sys/types.h>
#include "system.h"
#include "util.h"
#include "kuser.h"
#include "protocols.h"
#include "stats.h"
#include "stats_rpc.h"
#include "stats_nfs.h"
#include "rtp.h"
#include "rtp_ip.h"
#include "rtp_rpc.h"

/*
 * NFS Definitions
 */

enum nfs_op {
     NFS_NULL        = 0,
     NFS_GETATTR          = 1,
     NFS_SETATTR          = 2,
     NFS_ROOT        = 3,
     NFS_LOOKUP           = 4,
     NFS_READLINK    = 5,
     NFS_READ        = 6,
     NFS_WRITECACHE  = 7,
     NFS_WRITE       = 8,
     NFS_CREATE           = 9,
```

```
        NFS_REMOVE          = 10,
        NFS_RENAME          = 11,
        NFS_LINK        = 12,
        NFS_SYMLINK         = 13,
        NFS_MKDIR       = 14,
        NFS_RMDIR       = 15,
        NFS_READDIR         = 16,
        NFS_STATFS          = 17
        };

    enum nfs_stat {
        NFS_OK                  = 0,
        NFSERR_PERM             = 1,
        NFSERR_NOENT        = 2,
        NFSERR_IO           = 5,
        NFSERR_NXIO             = 6,
        NFSERR_ACCES        = 13,
        NFSERR_EXIST        = 17,
        NFSERR_NODEV        = 19,
        NFSERR_NOTDIR       = 20,
        NFSERR_ISDIR        = 21,
        NFSERR_FBIG             = 27,
        NFSERR_NOSPC        = 28,
        NFSERR_ROFS             = 30,
        NFSERR_NAMETOOLONG  = 63,
        NFSERR_NOTEMPTY         = 66,
        NFSERR_DQUOT        = 69,
        NFSERR_STALE        = 70,
        NFSERR_WFLUSH       = 99
        };

    enum nfs_ftype {
        NFNON                   = 0,
        NFREG                   = 1,
        NFDIR                   = 2,
```

```
        NFBLK                   = 3,
        NFCHR                   = 4,
        NFLNK                   = 5
        };

struct nfs_fhandle {
        Byte        fhandle[NFS_FHSIZE];
        };

struct nfs_timeval {
        Uint32          seconds;
        Uint32          useconds;
        };

struct nfs_fattr {
        enum nfs_ftype      type;
        Byte                    mode[4];
        Byte                    nlink[4];
        Byte                    uid[4];
        Byte                    gid[4];
        Byte                    size[4];
        Byte                    blocksize[4];
        Byte                    rdev[4];
        Byte                    blocks[4];
        Byte                    fsid[4];
        Byte                    fileid[4];
        struct nfs_timeval  atime;
        struct nfs_timeval  mtime;
        struct nfs_timeval  ctime;
        };

struct nfs_sattr {
        Uint32                  mode;
        Uint32                  uid;
        Uint32                  gid;
        Uint32                  size;
```

```
        struct nfs_timeval  atime;
        struct nfs_timeval  mtime;
        };

struct nfs_filename {
        Byte        length[4];
        Byte        filename[NFS_MAXNAMLEN];
        };

struct nfs_path {
        Byte        length[4];
        Byte        path[NFS_MAXPATHLEN]
        };

struct nfs_attrstat {
             struct nfs_fattr    attributes;          /* if stat = NFS_OK */
             };

struct nfs_dirop_args {
        struct nfs_fhandle      dir;
        struct nfs_filename name;
        };

struct nfs_dirop_result {
        struct nfs_fhandle      file;                /* if stat = NFS_OK */
        struct nfs_fattr        attributes;
        };

struct nfs_read_args {
        struct nfs_fhandle      file;
        Byte                    offset[4];
        Byte                    count[4];
        Byte                    total_count[4];
        };

struct nfs_write_args {
        struct nfs_fhandle      file;
```

```
        Byte                    beginoffset[4];
        Byte                    offset[4];
        Byte                    totalcount[4];
        Byte                    write_length[4];
        };

struct nfs_read_result {
        Byte                    status[4];
        struct nfs_fattr        attributes;
        Byte                    read_length[4];
        };

/*
 * Mount protocol definitions
 */
enum mount_op {
        MOUNTPROC_NULL          = 0,
        MOUNTPROC_MNT           = 1,
        MOUNTPROC_DUMP          = 2,
        MOUNTPROC_UMNT          = 3,
        MOUNTPROC_UMNTALL       = 4,
        MOUNTPROC_EXPORT        = 5
        };

struct mount_args {
        Byte      length[4];
        Byte      dirpath;
        };

struct mount_reply {
        Byte                  status[4];
        struct nfs_fhandle    handle;
        };


/*
 * Local NFS data structures
```

```
 */
static StatsAddrEntry       *this_seg_addr_ptr;
static StatsNfsSegment      *this_seg_stats_ptr;

static StatsAddrEntry       *src_seg_addr_ptr;
static StatsNfsSegment      *src_seg_stats_ptr;

static StatsAddrEntry       *dst_seg_addr_ptr;
static StatsNfsSegment      *dst_seg_stats_ptr;

static StatsAddrEntry          *src_node_addr_ptr;
static StatsNfsAddr            *src_node_stats_ptr;

static StatsAddrEntry          *dst_node_addr_ptr;
static StatsNfsAddr            *dst_node_stats_ptr;

static StatsAddrEntry          *dialog_addr_ptr;
static StatsNfsDialogEntry     *dialog_stats_ptr;

static StatsAddrEntry          *ip_src_addr_ptr;
static StatsAddrEntry          *ip_dst_addr_ptr;


/*
 * Macros for NFS statistics
 * These use the macros defined in stats.h
 */

#define nfs_stats_ops { \
     stats_for_segs      (ops, opRate, NFS_SEGMENT, AL_OPS, 0); \
     stats_for_nodes          (ops, opRate, NFS_NODE, AL_OPS, 0); \
     stats_for_dialog    (ops, opRate, NFS_PAIR, AL_OPS, 0); }


#define nfs_stats_read_ops { \
     stats_for_segs      (readOps, readOpRate, NFS_SEGMENT, AL_READ_OPS, 0); \
     stats_for_nodes          (readOps, readOpRate, NFS_NODE, AL_READ_OPS, 0); \
```

```
     stats_for_dialog    (readOps, readOpRate, NFS_PAIR, AL_READ_OPS, 0); }


#define nfs_stats_write_ops { \
     stats_for_segs      (writeOps, writeOpRate, NFS_SEGMENT, AL_WRITE_OPS, 0); \
     stats_for_nodes          (writeOps, writeOpRate, NFS_NODE, AL_WRITE_OPS, 0); \
     stats_for_dialog    (writeOps, writeOpRate, NFS_PAIR, AL_WRITE_OPS, 0); }


#define nfs_stats_write_cache_ops { \
     stats_for_segs      (writeCacheOps, writeCacheOpRate, NFS_SEGMENT,
AL_WRITE_CACHE_OPS, 0); \
     stats_for_nodes          (writeCacheOps, writeCacheOpRate, NFS_NODE,
AL_WRITE_CACHE_OPS, 0); \
     stats_for_dialog    (writeCacheOps, writeCacheOpRate, NFS_PAIR, AL_WRITE_CACHE_OPS,
0); }


#define nfs_stats_bytes { \
     stats_bytes_for_segs (length, bytes, byteRate, NFS_SEGMENT, AL_BYTES, 0); \
     stats_bytes_for_nodes(length, bytes, byteRate, NFS_NODE, AL_BYTES, 0); }


#define nfs_stats_read_bytes { \
     stats_bytes_for_segs (byte_count, readBytes, readByteRate, \
                           NFS_SEGMENT, AL_READ_BYTES, 0); \
     stats_bytes_for_nodes(byte_count, readBytes, readByteRate, \
                           NFS_NODE, AL_READ_BYTES, 0); \
     stats_bytes_for_dialog(byte_count, readBytes, readByteRate, \
                           NFS_PAIR, AL_READ_BYTES, 0); }


#define nfs_stats_write_bytes { \
     stats_bytes_for_segs (byte_count, writeBytes, writeByteRate, \
                           NFS_SEGMENT, AL_WRITE_BYTES, 0); \
     stats_bytes_for_nodes(byte_count, writeBytes, writeByteRate, \
                           NFS_NODE, AL_WRITE_BYTES, 0); \
```

```
    stats_bytes_for_dialog(byte_count, writeBytes, writeByteRate, \
                           NFS_PAIR, AL_WRITE_BYTES, 0); }


#define nfs_stats_off_seg \
    stats_off_segs (NFS_SEGMENT, NFS_NODE, NFS_PAIR, 0)


#define nfs_stats_rexmts { \
    stats_for_segs      (rexmts, rexmtRate, NFS_SEGMENT, AL_REXMTS, 0); \
    stats_for_nodes         (rexmts, rexmtRate, NFS_NODE, AL_REXMTS, 0); \
    stats_for_dialog    (rexmts, rexmtRate, NFS_PAIR, AL_REXMTS, 0); }


#define nfs_stats_flow_ctrls { \
    stats_for_segs      (flowCtrls, flowCtrlRate, NFS_SEGMENT, AL_FLOW_CTRLS, 0); \
    stats_for_nodes         (flowCtrls, flowCtrlRate, NFS_NODE, AL_FLOW_CTRLS, 0); \
    stats_for_dialog    (flowCtrls, flowCtrlRate, NFS_PAIR, AL_FLOW_CTRLS, 0); }


#define nfs_stats_errors { \
    stats_for_segs      (errors, errorRate, NFS_SEGMENT, AL_ERRORS, 0); \
    stats_for_nodes         (errors, errorRate, NFS_NODE, AL_ERRORS, 0); }

#define nfs_stats_rpc_mismatch { \
    stats_for_segs (rpcStats.versionMismatch, rpcStats.versionMismatchRate, \
                    NFS_SEGMENT, AL_RPC_VERSION, 0); \
    stats_for_nodes     (rpcStats.versionMismatch, rpcStats.versionMismatchRate, \
                    NFS_NODE, AL_RPC_VERSION, 0); \
    stats_for_dialog(rpcMismatch, rpcMismatchRate, NFS_PAIR, AL_RPC_VERSION, 0); }

#define nfs_stats_rpc_auth_bad_cred { \
    stats_for_segs (rpcStats.authBadCred, rpcStats.authBadCredRate, \
                    NFS_SEGMENT, AL_RPC_BAD_CRED, 0); \
    stats_for_nodes     (rpcStats.authBadCred, rpcStats.authBadCredRate, \
                    NFS_NODE, AL_RPC_BAD_CRED, 0); \
    stats_for_dialog(rpcAuthFail, rpcAuthFailRate, NFS_PAIR, AL_RPC_BAD_CRED, 0); }
```

```
#define nfs_stats_rpc_auth_rejected_cred { \
     stats_for_segs    (rpcStats.authRejectedCred, rpcStats.authRejectedCredRate, \
                        NFS_SEGMENT, AL_RPC_REJECT_CRED, 0); \
     stats_for_nodes       (rpcStats.authRejectedCred, rpcStats.authRejectedCredRate, \
                        NFS_NODE, AL_RPC_REJECT_CRED, 0); \
     stats_for_dialog  (rpcAuthFail, rpcAuthFailRate, NFS_PAIR, AL_RPC_REJECT_CRED, 0); }


#define nfs_stats_rpc_auth_bad_verf { \
     stats_for_segs    (rpcStats.authBadVerf, rpcStats.authBadVerfRate, \
                        NFS_SEGMENT, AL_RPC_BAD_VERF, 0); \
     stats_for_nodes       (rpcStats.authBadVerf, rpcStats.authBadVerfRate, \
                        NFS_NODE, AL_RPC_BAD_VERF, 0); \
     stats_for_dialog  (rpcAuthFail, rpcAuthFailRate, NFS_PAIR, AL_RPC_BAD_VERF, 0); }

#define nfs_stats_rpc_auth_rejected_verf { \
     stats_for_segs    (rpcStats.authRejectedVerf, rpcStats.authRejectedVerfRate, \
                        NFS_SEGMENT, AL_RPC_REJECT_VERF, 0); \
     stats_for_nodes       (rpcStats.authRejectedVerf, rpcStats.authRejectedVerfRate, \
                        NFS_NODE, AL_RPC_REJECT_VERF, 0); \
     stats_for_dialog  (rpcAuthFail, rpcAuthFailRate, NFS_PAIR, AL_RPC_REJECT_VERF, 0); }


#define nfs_stats_rpc_auth_too_weak { \
     stats_for_segs    (rpcStats.authTooWeak, rpcStats.authTooWeakRate, \
                        NFS_SEGMENT, AL_RPC_AUTH_WEAK, 0); \
     stats_for_nodes       (rpcStats.authTooWeak, rpcStats.authTooWeakRate, \
                        NFS_NODE, AL_RPC_AUTH_WEAK, 0); \
     stats_for_dialog  (rpcAuthFail, rpcAuthFailRate, NFS_PAIR, AL_RPC_AUTH_WEAK, 0); }

#define nfs_stats_rpc_auth_other { \
     stats_for_segs    (rpcStats.authOther, rpcStats.authOtherRate, \
                        NFS_SEGMENT, AL_RPC_AUTH_OTHER, 0); \
     stats_for_nodes       (rpcStats.authOther, rpcStats.authOtherRate, \
                        NFS_NODE, AL_RPC_AUTH_OTHER, 0); \
     stats_for_dialog  (rpcAuthFail, rpcAuthFailRate, NFS_PAIR, AL_RPC_AUTH_OTHER, 0); }
```

```
#define nfs_stats_rpc_prog_unavail { \
     stats_for_segs    (rpcStats.progUnavail, rpcStats.progUnavailRate, \
                        NFS_SEGMENT, AL_RPC_PROG_UNAVAIL, 0); \
     stats_for_nodes       (rpcStats.progUnavail, rpcStats.progUnavailRate, \
                        NFS_NODE, AL_RPC_PROG_UNAVAIL, 0); \
     stats_for_dialog  (rpcProgramFail, rpcProgramFailRate, NFS_PAIR, AL_RPC_PROG_UNAVAIL,
0); }

#define nfs_stats_rpc_prog_mismatch { \
     stats_for_segs    (rpcStats.progVersionMismatch, rpcStats.progVersionMismatchRate, \
                        NFS_SEGMENT, AL_RPC_PROG_VERSION, 0); \
     stats_for_nodes       (rpcStats.progVersionMismatch,
rpcStats.progVersionMismatchRate, \
                        NFS_NODE, AL_RPC_PROG_VERSION, 0); \
     stats_for_dialog  (rpcProgramFail, rpcProgramFailRate, NFS_PAIR, AL_RPC_PROG_VERSION,
0); }

#define nfs_stats_rpc_proc_unavail { \
     stats_for_segs    (rpcStats.procUnavail, rpcStats.procUnavailRate, \
                        NFS_SEGMENT, AL_RPC_PROC_UNAVAIL, 0); \
     stats_for_nodes       (rpcStats.procUnavail, rpcStats.procUnavailRate, \
                        NFS_NODE, AL_RPC_PROC_UNAVAIL, 0); \
     stats_for_dialog  (rpcProgramFail, rpcProgramFailRate, NFS_PAIR, AL_RPC_PROC_UNAVAIL,
0); }

#define nfs_stats_rpc_proc_garbage_args { \
     stats_for_segs    (rpcStats.procGarbageArgs, rpcStats.procGarbageArgsRate, \
                        NFS_SEGMENT, AL_RPC_PROC_GARBAGE, 0); \
     stats_for_nodes       (rpcStats.procGarbageArgs, rpcStats.procGarbageArgsRate, \
                        NFS_NODE, AL_RPC_PROC_GARBAGE, 0); \
     stats_for_dialog  (rpcProgramFail, rpcProgramFailRate, NFS_PAIR, AL_RPC_PROC_GARBAGE,
0); }

#define nfs_stats_mount_fail { \
     stats_for_segs    (mountFailures, mountFailureRate, NFS_SEGMENT, AL_MOUNT_FAILURES,
0); \
```

```
     stats_for_nodes       (mountFailures, mountFailureRate, NFS_NODE, AL_MOUNT_FAILURES,
0); }

#define nfs_stats_err_perm { \
     stats_for_segs    (nfsErrPerm, nfsErrPermRate, NFS_SEGMENT, AL_NFS_ERR_PERM, 0); \
     stats_for_nodes       (nfsErrPerm, nfsErrPermRate, NFS_NODE, AL_NFS_ERR_PERM, 0); \
     stats_for_dialog  (nfsAccessErr, nfsAccessErrRate, NFS_PAIR, AL_NFS_ERR_PERM, 0); }

#define nfs_stats_err_no_ent { \
     stats_for_segs    (nfsErrNoEnt, nfsErrNoEntRate, NFS_SEGMENT, AL_NFS_ERR_NO_ENT, 0); \

     stats_for_nodes       (nfsErrNoEnt, nfsErrNoEntRate, NFS_NODE, AL_NFS_ERR_NO_ENT, 0);
\
     stats_for_dialog  (nfsUserErr, nfsUserErrRate, NFS_PAIR, AL_NFS_ERR_NO_ENT, 0); }

#define nfs_stats_err_io { \
     stats_for_segs    (nfsErrIO, nfsErrIORate, NFS_SEGMENT, AL_NFS_ERR_IO, 0); \
     stats_for_nodes       (nfsErrIO, nfsErrIORate, NFS_NODE, AL_NFS_ERR_IO, 0); \
     stats_for_dialog  (nfsHardErr, nfsHardErrRate, NFS_PAIR, AL_NFS_ERR_IO, 0); }

#define nfs_stats_err_nxio { \
     stats_for_segs    (nfsErrNXIO, nfsErrNXIORate, NFS_SEGMENT, AL_NFS_ERR_NXIO, 0); \
     stats_for_nodes       (nfsErrNXIO, nfsErrNXIORate, NFS_NODE, AL_NFS_ERR_NXIO, 0); \
     stats_for_dialog  (nfsUserErr, nfsUserErrRate, NFS_PAIR, AL_NFS_ERR_NXIO, 0); }

#define nfs_stats_err_access { \
     stats_for_segs    (nfsErrAccess, nfsErrAccessRate, NFS_SEGMENT, AL_NFS_ERR_ACCESS, 0);
\
     stats_for_nodes       (nfsErrAccess, nfsErrAccessRate, NFS_NODE, AL_NFS_ERR_ACCESS,
0); \
     stats_for_dialog  (nfsAccessErr, nfsAccessErrRate, NFS_PAIR, AL_NFS_ERR_ACCESS, 0); }


#define nfs_stats_err_exist { \
     stats_for_segs    (nfsErrExist, nfsErrExistRate, NFS_SEGMENT, AL_NFS_ERR_EXIST, 0); \
```

```
     stats_for_nodes       (nfsErrExist, nfsErrExistRate, NFS_NODE, AL_NFS_ERR_EXIST, 0);
\
     stats_for_dialog  (nfsUserErr, nfsUserErrRate, NFS_PAIR, AL_NFS_ERR_EXIST, 0); }

#define nfs_stats_err_no_dev { \
     stats_for_segs    (nfsErrNoDev, nfsErrNoDevRate, NFS_SEGMENT, AL_NFS_ERR_NO_DEV, 0); \

     stats_for_nodes       (nfsErrNoDev, nfsErrNoDevRate, NFS_NODE, AL_NFS_ERR_NO_DEV, 0);
\
     stats_for_dialog  (nfsUserErr, nfsUserErrRate, NFS_PAIR, AL_NFS_ERR_NO_DEV, 0); }

#define nfs_stats_err_not_dir { \
     stats_for_segs    (nfsErrNotDir, nfsErrNotDirRate, NFS_SEGMENT, AL_NFS_ERR_NOT_DIR,
0); \
     stats_for_nodes       (nfsErrNotDir, nfsErrNotDirRate, NFS_NODE, AL_NFS_ERR_NOT_DIR,
0); \
     stats_for_dialog  (nfsUserErr, nfsUserErrRate, NFS_PAIR, AL_NFS_ERR_NOT_DIR, 0); }

#define nfs_stats_err_is_dir { \
     stats_for_segs    (nfsErrIsDir, nfsErrIsDirRate, NFS_SEGMENT, AL_NFS_ERR_IS_DIR, 0); \

     stats_for_nodes       (nfsErrIsDir, nfsErrIsDirRate, NFS_NODE, AL_NFS_ERR_IS_DIR, 0);
\
     stats_for_dialog  (nfsUserErr, nfsUserErrRate, NFS_PAIR, AL_NFS_ERR_IS_DIR, 0); }

#define nfs_stats_err_fbig { \
     stats_for_segs    (nfsErrFBig, nfsErrFBigRate, NFS_SEGMENT, AL_NFS_ERR_FBIG, 0); \
     stats_for_nodes       (nfsErrFBig, nfsErrFBigRate, NFS_NODE, AL_NFS_ERR_FBIG, 0); \
     stats_for_dialog  (nfsResourceErr, nfsResourceErrRate, NFS_PAIR, AL_NFS_ERR_FBIG, 0);
}

#define nfs_stats_err_no_space { \
     stats_for_segs    (nfsErrNoSpace, nfsErrNoSpaceRate, NFS_SEGMENT, AL_NFS_ERR_NO_SPACE,
0); \
     stats_for_nodes       (nfsErrNoSpace, nfsErrNoSpaceRate, NFS_NODE,
AL_NFS_ERR_NO_SPACE, 0); \
```

```
     stats_for_dialog  (nfsResourceErr, nfsResourceErrRate, NFS_PAIR, AL_NFS_ERR_NO_SPACE,
0); }

#define nfs_stats_err_rofs { \
     stats_for_segs    (nfsErrROFS, nfsErrROFSRate, NFS_SEGMENT, AL_NFS_ERR_ROFS, 0); \
     stats_for_nodes        (nfsErrROFS, nfsErrROFSRate, NFS_NODE, AL_NFS_ERR_ROFS, 0); \
     stats_for_dialog  (nfsAccessErr, nfsAccessErrRate, NFS_PAIR, AL_NFS_ERR_ROFS, 0); }

#define nfs_stats_err_name_too_long { \
     stats_for_segs    (nfsErrNameTooLong, nfsErrNameTooLongRate, \
                        NFS_SEGMENT, AL_NFS_ERR_NAME_TOO_LONG, 0); \
     stats_for_nodes        (nfsErrNameTooLong, nfsErrNameTooLongRate, \
                        NFS_NODE, AL_NFS_ERR_NAME_TOO_LONG, 0); \
     stats_for_dialog  (nfsUserErr, nfsUserErrRate, NFS_PAIR, AL_NFS_ERR_NAME_TOO_LONG,
0); }

#define nfs_stats_err_not_empty { \
     stats_for_segs    (nfsErrNotEmpty, nfsErrNotEmptyRate, NFS_SEGMENT,
AL_NFS_ERR_NOT_EMPTY, 0);\
     stats_for_nodes        (nfsErrNotEmpty, nfsErrNotEmptyRate, NFS_NODE,
AL_NFS_ERR_NOT_EMPTY, 0);\
     stats_for_dialog  (nfsUserErr, nfsUserErrRate, NFS_PAIR, AL_NFS_ERR_NOT_EMPTY, 0); }


#define nfs_stats_err_dquota { \
     stats_for_segs    (nfsErrDQuota, nfsErrDQuotaRate, NFS_SEGMENT, AL_NFS_ERR_DQUOTA, 0);
\
     stats_for_nodes        (nfsErrDQuota, nfsErrDQuotaRate, NFS_NODE, AL_NFS_ERR_DQUOTA,
0); \
     stats_for_dialog  (nfsResourceErr, nfsResourceErrRate, NFS_PAIR, AL_NFS_ERR_DQUOTA,
0); }

#define nfs_stats_err_stale { \
     stats_for_segs    (nfsErrStale, nfsErrStaleRate, NFS_SEGMENT, AL_NFS_ERR_STALE, 0); \

     stats_for_nodes        (nfsErrStale, nfsErrStaleRate, NFS_NODE, AL_NFS_ERR_STALE, 0);
\
```

```
     stats_for_dialog  (nfsAccessErr, nfsAccessErrRate, NFS_PAIR, AL_NFS_ERR_STALE, 0); }

#define nfs_stats_err_wflush { \
     stats_for_segs    (nfsErrWFlush, nfsErrWFlushRate, NFS_SEGMENT, AL_NFS_ERR_WFLUSH, 0);
\
     stats_for_nodes        (nfsErrWFlush, nfsErrWFlushRate, NFS_NODE, AL_NFS_ERR_WFLUSH,
0); \
     stats_for_dialog  (nfsResourceErr, nfsResourceErrRate, NFS_PAIR, AL_NFS_ERR_WFLUSH,
0); }


/*****************************************************************
 *
 * NFS parse routine
 */

Uint32 rtp_nfs_parse (layer_ptr, length)

     char                                *layer_ptr;
     Uint32                              length;
{

     register Bool                       xid_found, reply_ok;
     register Uint32                          i, result, data_length, byte_count,
mount_status;
     register Uint32                          p_index, p_last, p_past;

     register RpcInfo                    *rpc_info_ptr;
     register StatsNfsState              *nfs_state_ptr;
     register struct nfs_write_args      *nfs_write_ptr;
     register struct nfs_read_result     *nfs_read_result_ptr;
     register struct mount_args          *mount_ptr;
     register struct mount_reply         *mount_reply_ptr;
     register StateNfsFileSystem         *nfs_file_system_ptr;
     register StatsNfsFileSystemLink     *file_system_link_ptr;
```

```
    register Uint32                         nfs_status;

    register StatsAddrEntry             *ip_src_addr_ptr;
    register StatsAddrEntry             *ip_dst_addr_ptr;

    AlarmUserData                       stats_alarm_data;

/*
 * Set up local variables
 */
result = GOOD;
rpc_info_ptr = &rpcInfo;

/*
 * Find statistics records
 */
if (rpc_info_ptr->direction == RPC_CALL)
    {
    nfs_src_node_addr_ptr = src_node_addr_ptr = stats_nfs_get_client(&ip_src_addr);
    nfs_dst_node_addr_ptr = dst_node_addr_ptr = stats_nfs_get_server(&ip_dst_addr);
    nfs_dialog_addr_ptr   = dialog_addr_ptr = stats_nfs_get_dialog(&ip_dst_addr,
&ip_src_addr);
    }
else
    {
    nfs_src_node_addr_ptr = src_node_addr_ptr = stats_nfs_get_server(&ip_src_addr);
    nfs_dst_node_addr_ptr = dst_node_addr_ptr = stats_nfs_get_client(&ip_dst_addr);
    nfs_dialog_addr_ptr   = dialog_addr_ptr   = stats_nfs_get_dialog(&ip_src_addr,
&ip_dst_addr);
    }

if (src_node_addr_ptr != NULL)
    {
    nfs_src_node_stats_ptr = src_node_stats_ptr =
        (StatsNfsAddr*)nfs_src_node_addr_ptr->stats_ptr;
    nfs_src_seg_addr_ptr  = src_seg_addr_ptr =
        stats_nfs_get_segment (nfs_src_node_addr_ptr->address.segment1);
```

```
    }
else
    {
    nfs_src_node_stats_ptr = src_node_stats_ptr = NULL;
    /**** IF GLOBAL IP POINTERS IMPLEMENTED DON'T NEED TO DO LOOKUP *****/
    if (ip_src_addr_ptr = (StatsAddrEntry *)stats_ip_lookup_addr(&ip_src_addr))
        nfs_src_seg_addr_ptr = src_seg_addr_ptr =
            stats_nfs_get_segment (ip_src_addr_ptr->address.segment1);
    else
        nfs_src_seg_addr_ptr = src_seg_addr_ptr = NULL;
    }
if (nfs_src_seg_addr_ptr != NULL)
    nfs_src_seg_stats_ptr = src_seg_stats_ptr =
        (StatsNfsSegment *) nfs_src_seg_addr_ptr->stats_ptr;
else
    nfs_src_seg_stats_ptr = src_seg_stats_ptr = NULL;


if (dst_node_addr_ptr != NULL)
    {
    nfs_dst_node_stats_ptr = dst_node_stats_ptr =
        (StatsNfsAddr *) nfs_dst_node_addr_ptr->stats_ptr;
    nfs_dst_seg_addr_ptr = dst_seg_addr_ptr =
        stats_nfs_get_segment (nfs_dst_node_addr_ptr->address.segment1);
    }
else
    {
    nfs_dst_node_stats_ptr = dst_node_stats_ptr = NULL;
    /**** IF GLOBAL IP POINTERS IMPLEMENTED DON'T NEED TO DO LOOKUP *****/
    if (ip_dst_addr_ptr = (StatsAddrEntry *)stats_ip_lookup_addr(&ip_dst_addr))
        nfs_dst_seg_addr_ptr = dst_seg_addr_ptr =
            stats_nfs_get_segment (ip_dst_addr_ptr->address.segment1);
    else
        nfs_dst_seg_addr_ptr = dst_seg_addr_ptr = NULL;
    }
if (nfs_dst_seg_addr_ptr != NULL)
    nfs_dst_seg_stats_ptr = dst_seg_stats_ptr =
```

```
            (StatsNfsSegment *) nfs_dst_seg_addr_ptr->stats_ptr;
else
     nfs_dst_seg_stats_ptr = dst_seg_stats_ptr = NULL;


nfs_this_seg_addr_ptr = this_seg_addr_ptr = stats_nfs_get_segment (mySegmentId);
if (this_seg_addr_ptr != NULL)
     nfs_this_seg_stats_ptr = this_seg_stats_ptr =
          (StatsNfsSegment *) nfs_this_seg_addr_ptr->stats_ptr;
else
     nfs_this_seg_stats_ptr = this_seg_stats_ptr = NULL;


if (dialog_addr_ptr != NULL)
     nfs_dialog_stats_ptr = dialog_stats_ptr =
          (StatsNfsDialogEntry *) nfs_dialog_addr_ptr->stats_ptr;
else
     nfs_dialog_stats_ptr = dialog_stats_ptr = NULL;

if (dialog_stats_ptr != NULL)
     nfs_state_ptr  = (StatsNfsState *) dialog_stats_ptr->state_ptr;
else
     nfs_state_ptr = NULL;

/*
 * Update the age timer in each address structure and set the 10 second rate
 * sample period so that the event manager will calculate rates.
 */
stats_update_age_timers;
stats_set_rate_10s;

/*
 * Do some statistics
 */
nfs_stats_bytes;

/*
```

```
 * If cred or verf length is bad, count error and get out
 * Should this be moved so that some processing takes place (e.g. dequeue of file system)?

 */
if ( (rpc_info_ptr->cred_length > RPC_AUTH_LENGTH) || (rpc_info_ptr->verf_length >
RPC_AUTH_LENGTH) )
     {
     goto nfs_bad;
     }

/*
 * Set up pointer to the nfs history
 * Get index to last recorded event
 * Note that histx points to the next entry to store history into.
 *
 * p_index = current history index to store into
 * p_last = last history index written to
 */
xid_found = FALSE;
if (nfs_state_ptr != NULL)
     {
     p_index             = nfs_state_ptr->histx;
     p_last              = (nfs_state_ptr->histx == 0) ? NFS_HISTORY_ENTRIES - 1 :
                           nfs_state_ptr->histx - 1;
     p_past              = p_last;

     /*
      * Check for a retransmission and save this event in history for the dialog if it is
a call.
      */
     if (rpc_info_ptr->direction == RPC_CALL)
          {
          for (i=0; i<NFS_HISTORY_ENTRIES; i++)
               {
               if ( (nfs_state_ptr->history[p_past].xid == rpc_info_ptr->xid) &&
                    (nfs_state_ptr->history[p_past].status == NFS_OUTSTANDING) )
                    {
```

```
                    nfs_stats_rexmts;
                    break;
                    }
               p_past = (p_past == 0) ? NFS_HISTORY_ENTRIES - 1 : p_past - 1;
               }

          nfs_state_ptr->history[p_index].xid          = rpc_info_ptr->xid;
          nfs_state_ptr->history[p_index].call_proc    = rpc_info_ptr->call_proc;
          nfs_state_ptr->history[p_index].status       = NFS_OUTSTANDING;
          nfs_state_ptr->history[p_index].ptr              = NULL;
          nfs_state_ptr->histx                             = ((p_index + 1) ==
NFS_HISTORY_ENTRIES) ? 0 :
  p_index + 1;
          }


     /*
      * Set the status in the history entry if this is a reply.
      * If the xid of this reply matches that of a call in the history, mark
      * the called NFS_ANSWERED and remember the call by setting p_past.
      */
     else
          {
          for (i=0; i<NFS_HISTORY_ENTRIES; i++)
               {
               if ( (nfs_state_ptr->history[p_past].xid == rpc_info_ptr->xid) &&
                    (nfs_state_ptr->history[p_past].status == NFS_OUTSTANDING) )
                    {
                    nfs_state_ptr->history[p_past].status = NFS_ANSWERED;
                    xid_found = TRUE;
                    break;
                    }
               p_past = (p_past == 0) ? NFS_HISTORY_ENTRIES - 1 : p_past - 1;
               }
          }
     }
```

```
/*
 * If this is the mount protocol, handle it.
 */
if (rpc_info_ptr->protocol == MOUNT_PORT)
     {
     /*
      * Determine if this is a call or a reply.
      */
     if (rpc_info_ptr->direction == RPC_CALL)
          {
          nfs_stats_ops;
          nfs_stats_off_seg;
          switch (rpc_info_ptr->call_proc)
               {
               /*
                * If mounting a file, make sure it's in the statsNfsFileSystemQ and
                * linked to the server, then link it to the client mounting it by doing
                * a get.  Remember the file system so that it can it can be unlinked if
the
                * mount request fails.
                *
                */
               case MOUNTPROC_MNT:
                    mount_ptr = (struct mount_args *) rpc_info_ptr->user_ptr;
                    byte_count = ALIGN(mount_ptr->length);
                    if ( (mount_ptr != NULL) && (byte_count != 0) && (nfs_state_ptr !=
NULL) )
                              {
                              nfs_state_ptr->history[p_index].ptr =
                                   (Uint32 *) stats_nfs_get_file_system
(dst_node_addr_ptr,
                                   src_node_addr_ptr, byte_count, (NfsFileSystem *)
&mount_ptr->dirpath);
                              }
                    break;
```

```
                /*
                 * If unmounting a file, lookup the file and remember it in the
                 * history so that it can be unlinked if necessary when the response comes
in.
                 */
                case MOUNTPROC_UMNT:
                     mount_ptr = (struct mount_args *) rpc_info_ptr->user_ptr;
                     byte_count = ALIGN(mount_ptr->length);
                     if ( (mount_ptr != NULL) && (byte_count != 0) && (nfs_state_ptr !=
NULL) )
                          {
                          nfs_state_ptr->history[p_index].ptr =
                               (Uint32 *) stats_nfs_lookup_file_system (dst_node_addr_ptr,

                               src_node_addr_ptr, byte_count, (NfsFileSystem *)
&mount_ptr->dirpath);
                          }
                     break;


                case MOUNTPROC_UMNTALL:
                case MOUNTPROC_NULL:
                case MOUNTPROC_DUMP:
                case MOUNTPROC_EXPORT:
                     break;

                default:
                     break;
                } /* call_proc */
           } /* RPC_CALL */


      /*
       * It's a mount reply
       */
```

```
        else
            {
            reply_ok = FALSE;
            switch (rpc_info_ptr->reply_status)
                 {
                 case RPC_MSG_ACCEPTED:
                      switch (rpc_info_ptr->reply_code)
                           {
                           case RPC_SUCCESS:
                                /*
                                 * Check to make sure the mount code is ok
                                 */
                                reply_ok = TRUE;
                                if (rpc_info_ptr->user_length > 0 )
                                     {
                                     mount_reply_ptr = (struct mount_reply *)
rpc_info_ptr->user_ptr;
                                     mount_status = ALIGN(mount_reply_ptr->status);
                                     if (mount_status != 0)
                                          {
                                          reply_ok = FALSE;
                                          }
                                     }
                                break;
                           case RPC_PROG_UNAVAIL:
                                nfs_stats_rpc_prog_unavail;
                                break;
                           case RPC_PROG_MISMATCH:
                                nfs_stats_rpc_prog_mismatch;
                                break;
                           case RPC_PROC_UNAVAIL:
                                nfs_stats_rpc_proc_unavail;
                                break;
                           case RPC_GARBAGE_ARGS:
                                nfs_stats_rpc_proc_garbage_args;
                                break;
                           case RPC_SYSTEM_ERR:
```

```
                default:
                    break;
                } /* reply_code */
            break;    /* RPC_MSG_ACCEPTED */

        case RPC_MSG_DENIED:
            switch (rpc_info_ptr->reply_code)
                {
                case RPC_MISMATCH:
                    nfs_stats_rpc_mismatch;
                    break;
                case RPC_AUTH_ERROR:
                    switch (rpc_info_ptr->reply_auth_status)
                        {
                        case RPC_AUTH_BADCRED:
                            nfs_stats_rpc_auth_bad_cred;
                            break;
                        case RPC_AUTH_REJECTEDCRED:
                            nfs_stats_rpc_auth_rejected_cred;
                            break;
                        case RPC_AUTH_BADVERF:
                            nfs_stats_rpc_auth_bad_Verf;
                            break;
                        case RPC_AUTH_REJECTEDVERF:
                            nfs_stats_rpc_auth_rejected_verf;
                            break;
                        case RPC_AUTH_TOOWEAK:
                            nfs_stats_rpc_auth_too_weak;
                            break;
                        default:
                            break;
                        } /* reply_code */
                    break;
                default:
                    break;
                }
            break;    /* RPC_MSG_DENIED */
```

```
                default:
                     nfs_stats_mount_fail;
                     goto nfs_bad;
                     break;
                } /* reply_status */

     if (xid_found)
          {
          if (reply_ok)
               {
               /*
                * If this is an umount response, get rid of the client's link to the file

                * system being unmounted.
                */
               if (nfs_state_ptr->history[p_past].call_proc == MOUNTPROC_UMNT)
                    {
                    if ((nfs_state_ptr->history[p_past].ptr != NULL) &&
(dst_node_stats_ptr != NULL))
                         {
                         file_system_link_ptr =
                              stats_nfs_lookup_file_system_link (dst_node_stats_ptr,
                                   (StatsNfsFileSystem *)
nfs_state_ptr->history[p_past].ptr);

                         if (file_system_link_ptr != NULL)
                              {
                              FBRemqm (&dst_node_stats_ptr->fileSystemQ,
                                        (PFBQentry_type) file_system_link_ptr);
                              stats_deallocate (file_system_link_ptr,
sizeof(StatsNfsFileSystemLink));
                              }
                         }
                    } /* MOUNTPROC_MNT */

               /*
```

```
                * If this is an unmount all response, get rid of all the links to the file
                * systems for this client.
                */
               else if (nfs_state_ptr->history[p_past].call_proc == MOUNTPROC_UMNTALL)
                    {
                    if (dst_node_stats_ptr != NULL)
                         {
                         file_system_link_ptr =
                              (StatsNfsFileSystemLink *)
dst_node_stats_ptr->fileSystemQ.pFlink;

                         while (file_system_link_ptr != NULL)
                              {
                              if (file_system_link_ptr->file_system_ptr->serverIpAddr ==
ip_src_addr)
                                   {
                                   FBRemqm (&src_node_stats_ptr->fileSystemQ,
                                        (PFBQentry_type) file_system_link_ptr);
                                   stats_deallocate
                                        (file_system_link_ptr,
sizeof(StatsNfsFileSystemLink));
                                   }
                              file_system_link_ptr =
                                   (StatsNfsFileSystemLink *)
file_system_link_ptr->link.pFlink;
                              }
                         }
                    }

               /*
                * If this is a reply to a successful mount, mark the file system in use.
                */
               else if (nfs_state_ptr->history[p_past].call_proc == MOUNTPROC_MNT)
                    {
                    if (nfs_state_ptr->history[p_past].ptr != NULL)
                         {
```

```
                    nfs_file_system_ptr =
                        (StatsNfsFileSystem *)nfs_state_ptr->history[p_past].ptr;
                    nfs_file_system_ptr->status = NFS_FILE_IN_USE;
                    }
                } /* MOUNTPROC_UMNTALL */


            } /* reply_ok */

        else
            {
            /*
             * If this is a reply to an unsuccessful mount, get rid of the
             * client's link to the file system.  Check to see if the file
             * system should be discarded.  If so, unlink it
             * from the server and dequeue it from statsNfsFileSystemQ.
             * Count the number of failed mount requests.
             */
            if (nfs_state_ptr->history[p_past].call_proc == MOUNTPROC_MNT)
                {
                nfs_stats_mount_fail;
                nfs_file_system_ptr = (StatsNfsFileSystem *)
nfs_state_ptr->history[p_past].ptr;

                if (nfs_file_system_ptr != NULL)
                    {
                    if (dst_node_stats_ptr != NULL)
                        {
                        file_system_link_ptr =
                            stats_nfs_lookup_file_system_link (dst_node_stats_ptr,

                                nfs_file_system_ptr);

                        if (file_system_link_ptr != NULL)
                            {
                            FBRemqm (&dst_node_stats_ptr->fileSystemQ,
                                     (PFBQentry_type) file_system_link_ptr);
```

```
                                stats_deallocate (file_system_link_ptr,
                                                  sizeof(StatsNfsFileSystemLink));
                                }
                            }

                        if (nfs_file_system_ptr->status != NFS_FILE_IN_USE)
                            {
                            if (src_node_stats_ptr != NULL)
                                {
                                file_system_link_ptr =
                                    stats_nfs_lookup_file_system_link
(src_node_stats_ptr,

nfs_file_system_ptr);

                                if (file_system_link_ptr != NULL)
                                    {
                                    FBRemqm (&src_node_stats_ptr->fileSystemQ,
                                             (PFBQentry_type)

file_system_link_ptr);

                                    stats_deallocate (file_system_link_ptr,

sizeof(StatsNfsFileSystemLink));

                                    }
                                }

                             stats_nfs_deallocate_file_system (src_node_addr_ptr,

dst_node_addr_ptr,

nfs_file_system_ptr);
                            }
                        }

                    } /* MOUNTPROC_MNT */

                goto nfs_bad;
                } /* not reply_ok */
```

```
          } /* xid and nfs_state_ptr */
     } /* REPLY */
} /* MOUNT_PORT */


/*
 * If this is the nfs protocol handle it.
 */
else if (rpc_info_ptr->protocol == NFS_PORT)
     {
     /*
      * Determine if this is a call or a reply.
      */
     if (rpc_info_ptr->direction == RPC_CALL)
          {
          nfs_stats_ops;
          nfs_stats_off_seg;
          switch (rpc_info_ptr->call_proc)
               {
               case NFS_READ:
                    nfs_stats_read_ops;
                    break;

               case NFS_WRITECACHE:
                    nfs_stats_write_cache_ops;
                    nfs_write_ptr = (struct nfs_write_args *) rpc_info_ptr->user_ptr;
                    byte_count = ALIGN(nfs_write_ptr->write_length);
                    nfs_stats_write_bytes;
                    break;

               case NFS_WRITE:
                    nfs_stats_write_ops;
                    nfs_write_ptr = (struct nfs_write_args *) rpc_info_ptr->user_ptr;
                    byte_count = ALIGN(nfs_write_ptr->write_length);
                    nfs_stats_write_bytes;
                    break;
```

```
                case NFS_NULL:
                case NFS_GETATTR:
                case NFS_SETATTR:
                case NFS_ROOT:
                case NFS_LOOKUP:
                case NFS_READLINK:
                case NFS_CREATE:
                case NFS_REMOVE:
                case NFS_RENAME:
                case NFS_LINK:
                case NFS_SYMLINK:
                case NFS_MKDIR:
                case NFS_RMDIR:
                case NFS_READDIR:
                case NFS_STATFS:
                     break;

                default:
                     break;
                }
           goto nfs_good;
           }

      else
           {
           /*
            * It's a reply
            */
           switch (rpc_info_ptr->reply_status)
                {
                case RPC_MSG_ACCEPTED:
                     switch (rpc_info_ptr->reply_code)
                          {
                          case RPC_SUCCESS:
                               break;
                          case RPC_PROG_UNAVAIL:
                               nfs_state_errors;
```

```
                    nfs_stats_rpc_prog_unavail;
                    goto nfs_bad;
                    break;
               case RPC_PROG_MISMATCH:
                    nfs_stats_errors;
                    nfs_stats_rpc_prog_mismatch;
                    goto nfs_bad;
                    break;
               case RPC_PROC_UNAVAIL:
                    nfs_stats_rpc_proc_unavail;
                    goto nfs_bad;
                    break;
               case RPC_GARBAGE_ARGS:
                    nfs_stats_rpc_proc_garbage_args;
                    goto nfs_bad;
                    break;
               case RPC_SYSTEM_ERR:
               default:
                    goto nfs_bad;
                    break;
               }
          break;    /* RPC_MSG_ACCEPTED */

     case RPC_MSG_DENIED:
          switch (rpc_info_ptr->reply_code)
               {
               case RPC_MISMATCH:
                    nfs_stats_rpc_mismatch;
                    break;

               case RPC_AUTH_ERROR:
                    switch (rpc_info_ptr->reply_auth_status)
                         {
                         case RPC_AUTH_BADCRED:
                              nfs_stats_rpc_auth_bad_cred;
                              break;
                         case RPC_AUTH_REJECTEDCRED:
```

```
                                        nfs_stats_rpc_auth_rejected_cred;
                                        break;
                                   case RPC_AUTH_BADVERF:
                                        nfs_stats_rpc_auth_bad_verf;
                                        break;
                                   case RPC_AUTH_REJECTEDVERF:
                                        nfs_stats_rpc_auth_rejected_verf;
                                        break;
                                   case RPC_AUTH_TOOWEAK:
                                        nfs_stats_rpc_auth_too_weak;
                                        break;
                                   default:
                                        break;
                                   }
                              break;

                         default:
                              break;
                         }
                    goto nfs_bad;
                    break;    /* RPC_MSG_DENIED */

               default:
                    goto nfs_bad;
                    break;
               }
          }

     nfs_status = ALIGN(rpc_info_ptr->user_ptr);
     switch (nfs_status)
          {
          case NFS_OK:
               if (xid_found)
                    {
                    if (nfs_state_ptr->history[p_past].call_proc == NFS_READ)
                         {
```

```
                        nfs_read_result_ptr = (struct nfs_read_result *)
rpc_info_ptr->user_ptr;
                        byte_count = ALIGN(nfs_read_result_ptr->read_length);
                        nfs_stats_read_bytes;
                        }
                    }
                goto nfs_good;
                break;

            case NFSERR_PERM:
                nfs_stats_err_perm;
                break;
            case NFSERR_NOENT:
                nfs_stats_err_no_ent;
                break;
            case NFSERR_IO:
                nfs_stats_err_io;
                break;
            case NFSERR_NXIO:
                nfs_stats_err_nxio;
                break;
            case NFSERR_ACCES:
                nfs_stats_err_access;
                break;
            case NFSERR_EXIST:
                nfs_stats_err_exist;
                break;
            case NFSERR_NODEV:
                nfs_stats_err_no_dev;
                break;
            case NFSERR_NOTDIR:
                nfs_stats_err_not_dir;
                break;
            case NFSERR_ISDIR:
                nfs_stats_err_is_dir;
                break;
            case NFSERR_FBIG:
```

```
                nfs_stats_err_fbig;
                break;
            case NFSERR_NOSPC:
                nfs_stats_err_no_space;
                break;
            case NFSERR_ROFS:
                nfs_stats_err_rofs;
                break;
            case NFSERR_NAMETOOLONG:
                nfs_stats_err_name_too_long;
                break;
            case NFSERR_NOTEMPTY:
                nfs_stats_err_not_empty;
                break;
            case NFSERR_DQUOT:
                nfs_stats_err_dquota;
                break;
            case NFSERR_STALE:
                nfs_stats_err_stale;
                break;
            case NFSERR_WFLUSH:
                nfs_stats_err_wflush;
                break;

            default:
                break;
            }
        goto nfs_bad;
        }

    return (result);

nfs_good:
    return (GOOD);

nfs_done:
```

```
return (DONE);

nfs_bad:
nfs_stats_errors;
return (BAD);
}


/*****************************************************************************
 *
 * Process an icmp source quench
 */

void rtp_nfs_src_quench ()
{
register SrcQuenchData   *srcQuenchData_ptr;

srcQuenchData_ptr = &srcQuenchData;

state_nfs_lookup_ptrs (&srcQuenchData_ptr->ip_src_addr, srcQuenchData_ptr->src_port,
                       &srcQuenchData_ptr->ip_dst_addr, srcQuenchData_ptr->dst_port);


nfs_stats_flow_ctrls;
}
```

```
/*
 *  rtp_pmap_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/rtp/SCCS/s.rtp_pmap_p.c
 *
 *  Date:       8/8/91
 *
 *  Revision:   1.1
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   06-18-91  KR        Created
 */

static char rtp_pmap_p_c [] = "@(#)rtp_pmap_p.c   1.1";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "util.h"
#include "kuser.h"
#include "protocols.h"
#include "stats.h"
#include "stats_ip.h"
```

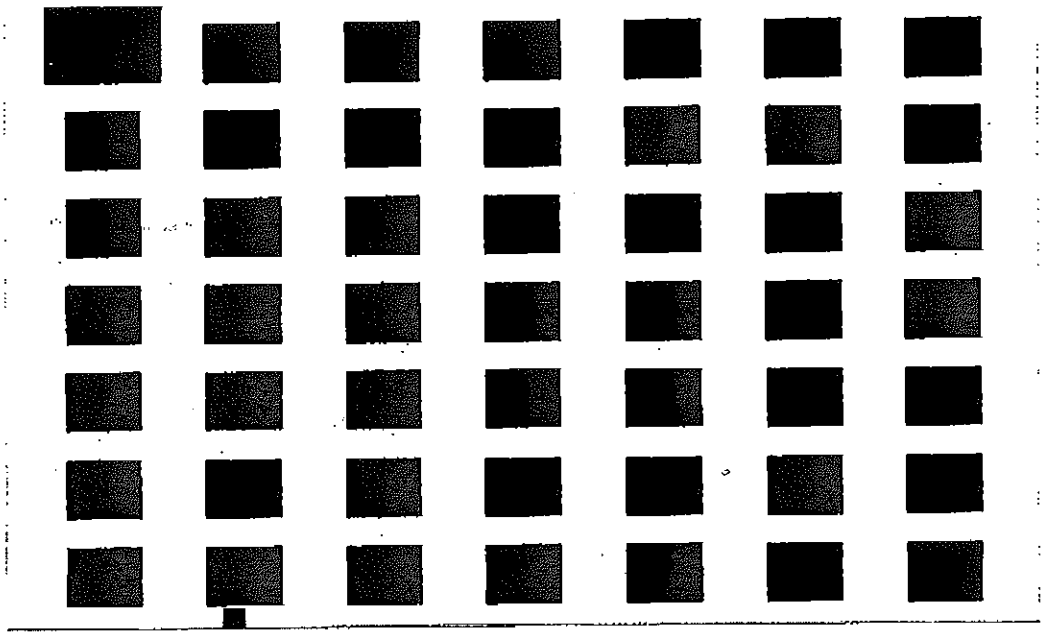
```
#include "stats_pmap.h"
#include "rtp.h"
#include "rtp_rpc.h"
#include "rtp_pmap.h"


/*
 * Pmap protocol definitions
 */
#define PMAP_NULL          0
#define PMAP_SET           1
#define PMAP_UNSET         2
#define PMAP_GETPORT       3
#define PMAP_DUMP          4
#define PMAP_CALLIT        5


/*
 * Mapping as it appears in the pmap protocol message
 */
struct pmap_msg_mapping_type {
     Byte      program_number[4];
     Byte      version[4];
     Byte      transport_protocol[4];
     Byte      port[4];       /* ignored in the request */
     };

struct pmap_msg_reply_type {
     Byte reply[4];
     };


/*
 * Pmap data structures
 */
PmapData            pmapData;
StatsPmapping       *dbg_mapping_ptr;   /* For debugging purposes */
```

```
/*****************************************************************
 *
 * Parse the port map protocol in order to associate ports with
 * particular protocols.
 */
Uint32 rtp_pmap_parse (layer_ptr, length)

     char                                          *layer_ptr;
     Uint32                                             length;
{
     register RpcInfo                              *rpc_info_ptr;
     register PmapData                             *pmap_data_ptr;
     register struct pmap_msg_mapping_type    *mapping_ptr;
     register Bool                                      xid_found;
     register StatsPmapping                        *pmapping_ptr;
     register struct pmap_msg_reply_type           *pmap_reply_ptr;
     register Uint32                                    i, histx, program_number, version,
protocol;
     register Uint32                                    pmap_reply;


rpc_info_ptr  = &rpcInfo;
pmap_data_ptr = &pmapData;


 switch (rpc_info_ptr->direction)
     {
     case RPC_CALL:
          /*
           * Save this in the history so that the reply can be recognized when it
           * comes in.  If this is a get, set or unset, save the mapping too.
           */
          mapping_ptr          = (struct pmap_msg_mapping_type *) rpc_info_ptr->user_ptr;
```

```
          histx                = pmap_data_ptr->histx;
          pmap_data_ptr->histx = ((histx + 1) == PMAP_HISTORY_ENTRIES) ? 0 : histx + 1;

          pmap_data_ptr->history[histx].status = PMAP_OUTSTANDING;
          pmap_data_ptr->history[histx].pmap_proc          = rpc_info_ptr->call_proc;
          pmap_data_ptr->history[histx].ip_addr_ptr       = ip_sro_node_addr_ptr;
          pmap_data_ptr->history[histx].xid                = rpc_info_ptr->xid;

          switch (rpc_info_ptr->call_proc)
               {
               case PMAP_GETPORT:
               case PMAP_SET:
               case PMAP_UNSET:
                    pmap_data_ptr->history[histx].program_number =
ALIGN(mapping_ptr->program_number);
                    pmap_data_ptr->history[histx].version        =
ALIGN(mapping_ptr->version);
                    pmap_data_ptr->history[histx].transport_protocol=
                         ALIGN(mapping_ptr->transport_protocol);
                    break;

               default:
                    break;
               }

          break;

     case RPC_REPLY:
          /*
           * Walk back through the history entries.  If the destination ip address
structure
           * and xid of this reply matches that of a call in the history, mark it
PMAP_ANSWERED.
           * If the reply is for a get, set, or unset, process it.
           */
          histx = (pmap_data_ptr->histx == 0) ? PMAP_HISTORY_ENTRIES - 1 :
pmap_data_ptr->histx - 1;
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerard White

03



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David N. Thompson and
Gerald White

```
            xid_found = FALSE;
            i = 0;
            while (i != PMAP_HISTORY_ENTRIES)
                 {
                 if ( (pmap_data_ptr->history[histx].xid == rpc_info_ptr->xid) &&
                      (pmap_data_ptr->history[histx].ip_addr_ptr == ip_dst_node_addr_ptr)
&&
                      (pmap_data_ptr->history[histx].status == PMAP_OUTSTANDING) )
                     {
                     pmap_data_ptr->history[histx].status = PMAP_ANSWERED;
                     xid_found = TRUE;
                     break;
                     }
                 i++;
                 histx = (histx == 0) ? PMAP_HISTORY_ENTRIES - 1 : histx - 1;
                 }

            if (xid_found)
                 {
                 /*
                  * If there was really no address structure for the node that sent the
request,
                  * don't process any further.  Need both the ip addr and xid for
identification.
                  */
                  if (pmap_data_ptr->history[histx].ip_addr_ptr == NULL)
                      goto pmap_done;

                 /*
                  * If this is an unsuccessful reply, just exit.
                  */
                 if ( (rpc_info_ptr->reply_status != RPC_MSG_ACCEPTED) ||
                      (rpc_info_ptr->reply_code != RPC_SUCCESS) )
                      goto pmap_bad;

                 /*
                  * The reply for a get is a port number or 0 if unsuccessful.
```

```
                * The reply for a set or unset is a boolean, TRUE is successful, FALSE
otherwise.
                * FALSE == 0.  If unsuccessful, just exit.
                */
               pmap_reply_ptr = (struot pmap_msg_reply_type *) rpc_info_ptr->user_ptr;
               pmap_reply = ALIGN(pmap_reply_ptr->reply);
               if (pmap_reply == 0)
                    goto pmap_bad;

               /*
                * If this is a successful reply to a get request, then if the program
number
                * is not in the table of those to be monitored or if there's no ip address

                * structure, exit.  Otherwise, look up the mapping in the ip structure.
                * If it's already recorded, don't do anything.  If it isn't, try to get
                * a port mapping structure.  Do this for the node replying to the get
request.
                */
               if (pmap_data_ptr->history[histx].pmap_proc == PMAP_GETPORT)
                    {
                    program_number = pmap_data_ptr->history[histx].program_number;
                    version        = pmap_data_ptr->history[histx].version;
                    protocol       = stats_program(program_number, version);

                    if (protocol == 0)
                         goto pmap_done;

                    if (ip_src_node_addr_ptr == NULL)
                         goto pmap_done;

                    if (stats_pmap_lookup (ip_src_node_addr_ptr, pmap_reply,

pmap_data_ptr->history[histx].transport_protocol) )
                              goto pmap_good;

                    pmapping_ptr = stats_pmap_get (ip_src_node_addr_ptr, pmap_reply,
```

```
pmap_data_ptr->history[histx].transport_protocol);
                    if (pmapping_ptr)
                         {
                         pmapping_ptr->port      = pmap_reply;
                         pmapping_ptr->protocol = protocol;
                         pmapping_ptr->program  = program_number;
                         pmapping_ptr->version  = version;
                         pmapping_ptr->transport_protocol
                              = pmap_data_ptr->history[histx].transport_protocol;
                         }
                    goto pmap_good;
                    }

               /*
                * If this is a successful reply to a set request, do the same as for the
                * get request above, but do it for the node that sourced the set request.
                */
               if (pmap_data_ptr->history[histx].pmap_proc == PMAP_SET)
                    {
                    program_number = pmap_data_ptr->history[histx].program_number;
                    version        = pmap_data_ptr->history[histx].version;
                    protocol       = stats_program(program_number, version);

                    if (protocol == 0)
                         goto pmap_done;

                    if (ip_dst_node_addr_ptr == NULL)
                         goto pmap_done;

                    if (stats_pmap_lookup (ip_dst_node_addr_ptr, pmap_reply,
                                           pmap_data_ptr->history[histx].transport_protocol)

)                        goto pmap_good;

                    pmapping_ptr = stats_pmap_get (ip_dst_node_addr_ptr, pmap_reply,
```

```
pmap_data_ptr->history[histx].transport_protocol);
                    if (pmapping_ptr)
                         {
                         pmapping_ptr->port      = pmap_reply;
                         pmapping_ptr->protocol = protocol;
                         pmapping_ptr->program  = program_number;
                         pmapping_ptr->version  = version;
                         pmapping_ptr->transport_protocol
                              = pmap_data_ptr->history[histx].transport_protocol;
                         }
                    goto pmap_good;
                    }


               /*
                * If this is a successful reply to an unset request, remove the mapping
                * from the ip address that sourced the unset request.
                */
               if (pmap_data_ptr->history[histx].pmap_proc == PMAP_UNSET)
                    {
                    stats_pmap_deallocate (ip_dst_node_addr_ptr, pmap_reply,

pmap_data_ptr->history[histx].transport_protocol);
                    }
               }
          break;

     default:
          goto pmap_bad;
          break;
     }


pmap_good:
     return (GOOD);
```

```
pmap_bad:
     return (BAD);

pmap_done:
     return (DONE);
}
```

```
/*
 *  rtp_rpc_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/rtp/SCCS/s.rtp_rpc_p.c
 *
 *  Date:       8/26/91
 *
 *  Revision:   1.7
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   08-26-91  KR        Needed to check verf length before accessing
 *   07-23-91  KR        Added port mapper call
 *   06-10-91  KR        Added check for bad length on verf and cred
 *   05-14-91  KR        Created
 */

static char rtp_rpc_p_c [] = "@(#)rtp_rpc_p.c     1.7";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "util.h"
#include "kuser.h"
```

```
#include "protocols.h"
#include "rtp.h"
#include "stats.h"
#include "stats_rpc.h"
#include "rtp_rpc.h"


/*
 * Global structures
 */
RpcInfo         rpcInfo;


/*****************************************************************
 *
 * Remote Procedure Call parse routine
 */
Uint32 rtp_rpc_parse (layer_ptr, length, protocol, transfer_protocol)

     char                        *layer_ptr;
     Uint32                           length;
     Uint32                           protocol;
     Uint32                           transfer_protocol;
{
     register RpcInfo            *rpc_info_ptr;
     register struct rpc_msg     *rpc_frame_ptr;
     Uint32                           result, residual;


result = GOOD;
rpc_frame_ptr = (struct rpc_msg *) layer_ptr;
rpc_info_ptr = &rpcInfo;
bzero (rpc_info_ptr, sizeof(RpcInfo));
rpc_info_ptr->protocol = protocol;
rpc_info_ptr->transfer_protocol = transfer_protocol;
```

```
/*
 * Align all 32-bit integers to avoid a bus error
 */
rpc_info_ptr->xid       = ALIGN(rpc_frame_ptr->xid);
rpc_info_ptr->direction = ALIGN(rpc_frame_ptr->direction);


/*
 * Set up all field in the rpcInfo structure.  The next level parse
 * routine will actually handle most of the fields.
 */
 switch (rpc_info_ptr->direction)
     {
     case RPC_CALL:
          rpc_info_ptr->call_version    = ALIGN(rpc_frame_ptr->rpc.call.rpcvers);
          rpc_info_ptr->call_prog       = ALIGN(rpc_frame_ptr->rpc.call.prog);
          rpc_info_ptr->call_vers       = ALIGN(rpc_frame_ptr->rpc.call.vers);
          rpc_info_ptr->call_proc       = ALIGN(rpc_frame_ptr->rpc.call.proc);
          rpc_info_ptr->cred_length     = ALIGN(rpc_frame_ptr->rpc.call.cred.length);
          rpc_info_ptr->cred_ptr        = &rpc_frame_ptr->rpc.call.cred;

          /*
           * Check the field length to see if it's too big, and, if so,
           * pass on to next layer for counting, but don't continue to
           * parse
           */
          if (rpc_info_ptr->cred_length > RPC_AUTH_LENGTH)
               goto rpc_hand_off;


          XDM_RESIDUAL(rpc_info_ptr->cred_length);
          rpc_info_ptr->verf_ptr = (struct rpc_opaque_auth *)
               (rpc_frame_ptr->rpc.call.cred.body + rpc_info_ptr->cred_length + residual);


          rpc_info_ptr->verf_length = ALIGN(rpc_info_ptr->verf_ptr->length);
          /*
```

```
                * Check the field length to see if it's too big, and, if so,
                * pass on to next layer for counting, but don't continue to
                * parse
                */
               if (rpc_info_ptr->verf_length > RPC_AUTH_LENGTH)
                    goto rpc_hand_off;


               XDM_RESIDUAL(rpc_info_ptr->verf_length);
               rpc_info_ptr->user_ptr = (Byte *)
                    (rpc_info_ptr->verf_ptr->body + rpc_info_ptr->verf_length + residual);
               rpc_info_ptr->user_length =
                    length - ( (Uint32)rpc_info_ptr->user_ptr - (Uint32)rpc_frame_ptr );

               break;

          case RPC_REPLY:
               rpc_info_ptr->reply_status = ALIGN(rpc_frame_ptr->rpc.reply.stat);
               switch (rpc_info_ptr->reply_status)
                    {
                    case RPC_MSG_ACCEPTED:
                         rpc_info_ptr->verf_ptr = &rpc_frame_ptr->rpc.reply.reply.accept.verf;

                         rpc_info_ptr->verf_length = ALIGN(rpc_info_ptr->verf_ptr->length);
                    /*
                     * Check the field length to see if it's too big, and, if so,
                     * pass on to next layer for counting, but don't continue to
                     * parse
                     */
                         if (rpc_info_ptr->cred_length > RPC_AUTH_LENGTH)
                              goto rpc_hand_off;

                         XDM_RESIDUAL(rpc_info_ptr->verf_length);
                         rpc_info_ptr->user_ptr = (Byte*)
                              (rpc_info_ptr->verf_ptr->body + rpc_info_ptr->verf_length +
residual);
```

```
                    rpc_info_ptr->reply_code = ALIGN(rpc_info_ptr->user_ptr);
                    rpc_info_ptr->user_ptr += 4;

                    if (rpc_info_ptr->reply_code == RPC_PROG_MISMATCH)
                         {
                         rpc_info_ptr->reply_version_low = ALIGN(rpc_info_ptr->user_ptr);

                         rpc_info_ptr->user_ptr += 4;
                         rpc_info_ptr->reply_version_high = ALIGN(rpc_info_ptr->user_ptr);

                         rpc_info_ptr->user_ptr += 4;
                         }
                    break;

                    case RPC_MSG_DENIED:
                         rpc_info_ptr->reply_code =
                              ALIGN(rpc_frame_ptr->rpc.reply.reply.reject.stat);
                         rpc_info_ptr->user_ptr = (Byte *)
                              &rpc_frame_ptr->rpc.reply.reply.reject.reject_data;
                         if (rpc_info_ptr->reply_code == RPC_MISMATCH)
                              {
                              rpc_info_ptr->reply_version_low =

ALIGN(rpc_frame_ptr->rpc.reply.reply.reject.reject_data.versions.low);
                              rpc_info_ptr->reply_version_high =

ALIGN(rpc_frame_ptr->rpc.reply.reply.reject.reject_data.versions.high);
                              rpc_info_ptr->user_ptr += 8;
                              }
                         else if (rpc_info_ptr->reply_code == RPC_AUTH_ERROR)
                              rpc_info_ptr->reply_auth_status =

ALIGN(rpc_frame_ptr->rpc.reply.reply.reject.reject_data.stat);
                              rpc_info_ptr->user_ptr += 4;
                         break;

                    default:
```

```
                    break;
                }

            rpc_info_ptr->user_length =
                length - ( (UInt32)rpc_info_ptr->user_ptr - (UInt32)rpc_frame_ptr );

            break;

        default:
            goto rpc_bad;
            break;
        }


    /*
     * Hand off the message
     */
    rpc_hand_off:

    switch (protocol)
        {
        case MAPPER_PORT:
            result = rtp_pmap_parse (layer_ptr, length);
            break;

        case NFS_PORT:
        case MOUNT_PORT:
            result = rtp_nfs_parse (layer_ptr, length, protocol, transfer_protocol);
            break;

        default:
            break;
        }

    return (result);
```

```
rpc_good:
     return (GOOD);

rpc_bad:
     return (BAD);

rpc_done:
     return (DONE);
}
```

```
/*
 *  rtp_tcp_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                           All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/rtp/SCCS/s.rtp_tcp_p.c
 *
 *  Date:       8/8/91
 *
 *  Revision:   1.13
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   07-22-91  KR        lookup protocol via stats_port_to_protocol and called
rtp_rpc_parse if
 *                            fixed some line overruns
 *                            nfs or port mapper
 *   06-27-91  KR        changed tcp_stats_off_seg macro to test stats ptr prior to
stats_count
 *   06-20-91  KR        changed look_for_data to call set_seq_numbers on entering data
state
 *                            changed process_data to check last_length_sent for 0
 *                            and to return if data_length == 0
 *
 *   06-19-91  KR        added dialog counting of connection retries
 *   06-18-91  KR        changed off seg count to check seg addr records instead of node
 *   06-14-91  KR        moved set of tcp_protocol up so that stats_tcp_get_dialog could
 *                            use it to set applicationProtocol
 *
 *   06-12-91  KR        added error macro call to rxmt, out of order, after close,
```

```
 *                              after window, and rst macros
 *                              added missing calls to stats_for_seg and stats_bytes_for_seg
 *
 *    06-07-91  KR          moved failed connection macro to stats_tcp.h
 *                              added call to stats_well_known_port
 *    06-04-91  KR          moved active connection macros to stats_tcp.h
 *    06-03-91  KMJ         added tcp_stats_reverse_successful_connections macro
 *                              and forced vars into registers
 *    06-03-91  DPD         Fixed on/seg/transit count bug
 */

static char rtp_tcp_p_c [] = "@(#)rtp_tcp_p.c      1.13";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include <sys/types.h>
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "in.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_tcp.h"
#include "rtp.h"
#include "rtp_dll.h"
#include "rtp_ip.h"
#include "rtp_tcp.h"


/*****************************************************************
 *
 * TCP monitoring
 */
```

```
/*
 * The following constants are used by tcp_state to look back into the
 * state history.  They shouldn't be set greater than MAX_TCP_HISTORY.
 * If they are, the state machine will look at an entry more than once
 * and may come to the wrong conclusion.
 */
#define MAX_LOOK_FOR_REXMT  10    /* entries to search for rexmit */


/*
 * TCP Events
 */
#define EV_UNKNOWN  0
#define EV_SYN      1
#define EV_RST      2
#define EV_ACK      3
#define EV_FIN      4
#define EV_DATA     5

/*
 * TCP States
 */
#define ST_UNKNOWN      0
#define ST_CONNECTING   1
#define ST_DATA         2
#define ST_CLOSING      3
#define ST_CLOSED       4
#define ST_INACTIVE      5


/*
 * Global TCP data structures
 */
struct tcphdr  *tcp_ptr;
u_char              tcp_flags;
```

```
Uint32              tcp_src_port;
Uint32              tcp_dst_port;
Uint32              tcp_seq;
Uint32              tcp_ack;
Uint32              tcp_window;
Uint32              tcp_protocol;


/*
 * Global TCP data structures
 */


/*
 * Local TCP data structures
 */
static StatsAddrEntry       *this_seg_addr_ptr;
static StatsTcpSegment      *this_seg_stats_ptr;
static StatsProtocolEntry   *this_seg_protocol_ptr;

static StatsAddrEntry       *src_seg_addr_ptr;
static StatsTcpSegment      *src_seg_stats_ptr;
static StatsProtocolEntry   *src_seg_protocol_ptr;

static StatsAddrEntry       *dst_seg_addr_ptr;
static StatsTcpSegment      *dst_seg_stats_ptr;
static StatsProtocolEntry   *dst_seg_protocol_ptr;

static StatsAddrEntry       *src_node_addr_ptr;
static StatsTcpAddr         *src_node_state_ptr;
static StatsProtocolEntry   *src_node_protocol_ptr;

static StatsAddrEntry       *dst_node_addr_ptr;
static StatsTcpAddr         *dst_node_stats_ptr;
static StatsProtocolEntry   *dst_node_protocol_ptr;
```

```
static StatsAddrEntry       *src_socket_addr_ptr;
static StatsTcpSocket       *src_socket_stats_ptr;

static StatsAddrEntry       *dst_socket_addr_ptr;
static StatsTcpSocket       *dst_socket_stats_ptr;

static StatsAddrEntry       *dialog_addr_ptr;
static StatsDialogEntry     *dialog_stats_ptr;

static StatsAddrEntry          *ip_dst_addr_ptr;
static StatsAddrEntry          *ip_src_addr_ptr;

static TcpConnectionStateType   *connection_state_ptr;
static TcpSocketStateType          *src_state_ptr;
static TcpSocketStateType       *dst_state_ptr;


/*
 * Macros for TCP statistics
 * These use the macros defined in stats.h
 */

#define tcp_stats_frames { \
     stats_for_segs        (frames, frameRate, TCP_SEGMENT, AL_FRAMES, 0); \
     stats_for_nodes       (common.frames, common.frameRate, TCP_NODE, AL_FRAMES, 0); \
     stats_for_rcv_node    (common.rcvFrames, common.rcvFrameRate, TCP_NODE, AL_RCV_FRAMES,
0); \
     stats_for_xmt_node    (common.xmtFrames, common.xmtFrameRate, TCP_NODE, AL_XMT_FRAMES,
0); \
     stats_for_sockets     (common.frames, common.frameRate, TCP_SOCKET, AL_FRAMES, 0); \
     stats_for_rcv_socket  (common.rcvFrames, common.rcvFrameRate, \
                                 TCP_SOCKET, AL_RCV_FRAMES, 0); \
     stats_for_xmt_socket  (common.xmtFrames, common.xmtFrameRate, \
                                 TCP_SOCKET, AL_XMT_FRAMES, 0); \
     stats_for_dialog      (packets, packetRate, TCP_PAIR, AL_FRAMES, 0); }
```

```
#define tcp_stats_bytes { \
     stats_bytes_for_segs        (length, bytes, byteRate, TCP_SEGMENT, AL_BYTES, 0); \
     stats_bytes_for_nodes       (length, common.bytes, common.byteRate, TCP_NODE,
AL_BYTES, 0); \
     stats_rcv_bytes_for_node    (length, common.rcvBytes, common.rcvByteRate, \
                                  TCP_NODE, AL_RCV_BYTES, 0); \
     stats_xmt_bytes_for_node    (length, common.xmtBytes, common.xmtByteRate, \
                                  TCP_NODE, AL_XMT_BYTES, 0); \
     stats_bytes_for_sockets     (length, common.bytes, common.byteRate, \
                                  TCP_SOCKET, AL_BYTES, 0); \
     stats_rcv_bytes_for_socket (length, common.rcvBytes, common.rcvByteRate,\
                                  TCP_SOCKET, AL_RCV_BYTES, 0); \
     stats_xmt_bytes_for_socket (length, common.xmtBytes, common.xmtByteRate,\
                                  TCP_SOCKET, AL_XMT_BYTES, 0); \
     stats_bytes_for_dialog      (length, bytes, byteRate, TCP_PAIR, AL_BYTES, 0); }

#define tcp_stats_errors { \
     stats_for_segs         (errors, errorRate, TCP_SEGMENT, AL_ERRORS, 0); \
     stats_for_nodes        (common.errors, common.errorRate, TCP_NODE, AL_ERRORS, 0); \
     stats_for_rcv_node     (common.rcvErrors, common.rcvErrorRate, TCP_NODE, AL_RCV_ERRORS,
0); \
     stats_for_xmt_node     (common.xmtErrors, common.xmtErrorRate, TCP_NODE, AL_XMT_ERRORS,
0); \
     stats_for_dialog       (errors, errorRate, TCP_PAIR, AL_ERRORS, 0); \
     stats_for_rcv_socket   (common.rcvFrames ,common.rcvFrameRate, \
                                  TCP_SOCKET, AL_RCV_FRAMES, 0); \
     stats_for_xmt_socket   (common.xmtFrames, common.xmtFrameRate, \
                                  TCP_SOCKET, AL_XMT_FRAMES, 0); \
     stats_for_sockets      (common.errors, common.errorRate, TCP_SOCKET, AL_ERRORS, 0); }

#define tcp_stats_off_segs { \
     if ( (src_seg_addr_ptr != NULL) && (dst_seg_addr_ptr != NULL) ) { \
          if ( src_seg_addr_ptr->address.segment1 != dst_seg_addr_ptr->address.segment1) {
\
               stats_for_rcv_node (common.rcvOffSegs, common.rcvOffSegRate, \
                                  TCP_NODE, AL_RCV_OFF_SEG, 0); \
               stats_for_xmt_node (common.xmtOffSegs, common.xmtOffSegRate, \
```

```
                                        TCP_NODE, AL_XMT_OFF_SEG, 0); \
                if (src_seg_stats_ptr != NULL) \
                     stats_count (&src_seg_stats_ptr->xmtOffSegs, \
                                  &src_seg_stats_ptr->xmtOffSegRate, \
                                  src_seg_addr_ptr, TCP_SEGMENT, AL_XMT_OFF_SEG, 0); \
                if (dst_seg_stats_ptr != NULL) \
                     stats_count (&dst_seg_stats_ptr->rcvOffSegs, \
                                  &dst_seg_stats_ptr->rcvOffSegRate, \
                                  dst_seg_addr_ptr, TCP_SEGMENT, AL_RCV_OFF_SEG, 0); \
                stats_for_rcv_socket (common.rcvOffSegs, common.rcvOffSegRate, \
                                      TCP_SOCKET, AL_RCV_OFF_SEG, 0); \
                stats_for_xmt_socket (common.xmtOffSegs, common.xmtOffSegRate, \
                                      TCP_SOCKET, AL_XMT_OFF_SEG, 0); \

            ) \
        } \
}

#define tcp_stats_transits { \
     stats_transits (TCP_SEGMENT, TCP_NODE, TCP_PAIR, TCP_APPL_PROTOCOL); }

#define tcp_stats_flow_ctrls { \
     stats_for_segs       (flowCtrls, flowCtrlRate, TCP_SEGMENT, AL_FLOW_CTRLS, 0); \
     stats_for_nodes      (common.flowCtrls, common.flowCtrlRate, TCP_NODE, AL_FLOW_CTRLS,
0); \
     stats_for_sockets    (common.flowCtrls, common.flowCtrlRate, TCP_SOCKET,
AL_FLOW_CTRLS, 0); \
     stats_for_dialog     (flowCtrls, flowCtrlRate, TCP_PAIR, AL_FLOW_CTRLS, 0); }

#define tcp_stats_hdr_bytes { \
     stats_bytes_for_segs      (tcp_hdr_length, hdrBytes, hdrByteRate, \
                                TCP_SEGMENT, AL_HDR_BYTES, 0); \
     stats_rcv_bytes_for_node (tcp_hdr_length, common.rcvHdrBytes, common.rcvHdrByteRate,
\                              TCP_NODE, AL_RCV_HDR_BYTES, 0); \
     stats_xmt_bytes_for_node (tcp_hdr_length, common.xmtHdrBytes, common.xmtHdrByteRate,
\                              TCP_NODE, AL_XMT_HDR_BYTES, 0); \
```

```
     stats_rcv_bytes_for_socket (tcp_hdr_length, common.rcvHdrBytes,
common.rcvHdrByteRate, \
                                TCP_SOCKET, AL_RCV_HDR_BYTES, 0); \
     stats_xmt_bytes_for_socket (tcp_hdr_length, common.xmtHdrBytes,
common.xmtHdrByteRate, \
                                TCP_SOCKET, AL_XMT_HDR_BYTES, 0); }

#define tcp_stats_fragments { \
     stats_for_segs       (frgmts, frgmtRate, TCP_SEGMENT, AL_FRAGMENTS, 0); \
     stats_for_nodes          (common.frgmts, common.frgmtRate, TCP_NODE, AL_FRAGMENTS,
0); \
     stats_for_rcv_node   (common.rcvFrgmts, common.rcvFrgmtRate, TCP_NODE,
AL_RCV_FRAGMENTS, 0);\
     stats_for_xmt_node   (common.xmtFrgmts, common.rcvFrgmtRate, TCP_NODE,
AL_XMT_FRAGMENTS, 0);\
     stats_for_sockets    (common.frgmts, common.frgmtRate, TCP_SOCKET, AL_FRAGMENTS, 0);
\
     stats_for_rcv_socket (common.rcvFrgmts, common.rcvFrgmtRate, \
                                TCP_SOCKET, AL_RCV_FRAGMENTS, 0); \
     stats_for_xmt_socket (common.xmtFrgmts, common.xmtFrgmtRate, \
                                TCP_SOCKET, AL_XMT_FRAGMENTS, 0); \
     state_for_dialog     (fragments, fragmentRate, TCP_PAIR, AL_FRAGMENTS, 0); }

#define tcp_stats_rexmt  { \
     stats_for_segs       (rexmts, rexmtRate, TCP_SEGMENT, AL_REXMTS, 0); \
     stats_for_nodes          (common.rexmts, common.rexmtRate, TCP_NODE, AL_REXMTS, 0); \
     stats_for_rcv_node   (common.rcvRexmts, common.rcvRexmtRate, TCP_NODE, AL_RCV_REXMTS,
0); \
     stats_for_xmt_node   (common.xmtRexmts, common.xmtRexmtRate, TCP_NODE, AL_XMT_REXMTS,
0); \
     stats_for_sockets    (common.rexmts, common.rexmtRate, TCP_SOCKET, AL_FRAGMENTS, 0);
\
     stats_for_rcv_socket (common.rcvRexmts, common.rcvRexmtRate, TCP_SOCKET,
AL_RCV_REXMTS, 0); \
     stats_for_xmt_socket (common.xmtRexmts, common.xmtRexmtRate, TCP_SOCKET,
AL_XMT_REXMTS, 0); \
```

```
    stats_for_dialog     (rexmts, rexmtRate, TCP_PAIR, AL_REXMTS, 0); \
    tcp_stats_errors; }


#define tcp_stats_rexmt_bytes { \
    stats_bytes_for_segs       (data_length, rexmtBytes, rexmtByteRate, \
                                    TCP_SEGMENT, AL_REXMT_BYTES, 0); \
    stats_rcv_bytes_for_node   (data_length, common.rcvRexmtBytes,
common.rcvRexmtByteRate, \
                                    TCP_NODE, AL_RCV_REXMT_BYTES, 0); \
    stats_xmt_bytes_for_node   (data_length, common.xmtRexmtBytes,
common.xmtRexmtByteRate, \
                                    TCP_NODE, AL_XMT_REXMT_BYTES, 0); \
    stats_rcv_bytes_for_socket (data_length, common.rcvRexmtBytes,
common.rcvRexmtByteRate, \
                                    TCP_SOCKET, AL_RCV_REXMT_BYTES, 0); \
    stats_xmt_bytes_for_socket (data_length, common.xmtRexmtBytes,
common.xmtRexmtByteRate, \
                                    TCP_SOCKET, AL_XMT_REXMT_BYTES, 0); }


#define tcp_st .ts_keep_alives { \
    stats_for_segs       (keepAlives, keepAliveRate, TCP_SEGMENT, AL_KEEP_ALIVES, 0); \
    stats_for_rcv_node  (common.rcvKeepAlives, common.rcvKeepAliveRate, \
                            TCP_NODE, AL_RCV_KEEP_ALIVES, 0); \
    stats_for_xmt_node  (common.xmtKeepAlives, common.xmtKeepAliveRate, \
                            TCP_NODE, AL_XMT_KEEP_ALIVES, 0); \
    stats_for_rcv_socket(common.rcvKeepAlives, common.rcvKeepAliveRate, \
                            TCP_SOCKET, AL_RCV_KEEP_ALIVES, 0); \
    stats_for_xmt_socket(common.xmtKeepAlives, common.xmtKeepAliveRate, \
                            TCP_SOCKET, AL_XMT_KEEP_ALIVES, 0); }


#define tcp_stats_window_probes { \
    stats_for_segs       (windowProbes, windowProbeRate, TCP_SEGMENT, AL_WINDOW_PROBES,
0); \
    stats_for_rcv_node   (common.rcvWindowProbes, common.rcvWindowProbeRate, \
```

```
                                TCP_NODE, AL_RCV_WINDOW_PROBES, 0); \
    stats_for_xmt_node    (common.xmtWindowProbes, common.xmtWindowProbeRate, \
                                TCP_NODE, AL_XMT_WINDOW_PROBES, 0); \
    stats_for_rcv_socket (common.rcvWindowProbes, common.rcvWindowProbeRate, \
                                TCP_SOCKET, AL_RCV_WINDOW_PROBES, 0); \
    stats_for_xmt_socket (common.xmtWindowProbes, common.xmtWindowProbeRate, \
                                TCP_SOCKET, AL_XMT_WINDOW_PROBES, 0); }


#define tcp_stats_out_of_order { \
    stats_for_segs        (outOfOrder, outOfOrderRate, TCP_SEGMENT, AL_OUT_OF_ORDER, 0); \

    stats_for_rcv_node   (common.rcvOutOfOrder, common.rcvOutOfOrderRate, \
                                TCP_NODE, AL_RCV_OUT_OF_ORDER, 0); \
    stats_for_xmt_node   (common.xmtOutOfOrder, common.xmtOutOfOrderRate, \
                                TCP_NODE, AL_XMT_OUT_OF_ORDER, 0); \
    stats_for_rcv_socket(common.rcvOutOfOrder, common.rcvOutOfOrderRate, \
                                TCP_SOCKET, AL_RCV_OUT_OF_ORDER,0);\
    stats_for_xmt_socket(common.xmtOutOfOrder, common.xmtOutOfOrderRate, \
                                TCP_SOCKET, AL_XMT_OUT_OF_ORDER,0); \
    tcp_stats_errors; }


#define tcp_stats_after_window { \
    stats_for_segs        (afterWindow, afterWindowRate, TCP_SEGMENT, AL_AFTER_WINDOW, 0);
\
    stats_for_rcv_node    (common.rcvAfterWindow, common.rcvAfterWindowRate, \
                                TCP_NODE, AL_RCV_AFTER_WINDOW, 0);\
    stats_for_xmt_node    (common.xmtAfterWindow, common.xmtAfterWindowRate, \
                                TCP_NODE, AL_XMT_AFTER_WINDOW, 0); \
    stats_for_rcv_socket (common.rcvAfterWindow, common.rcvAfterWindowRate, \
                                TCP_SOCKET, AL_RCV_AFTER_WINDOW, 0);\
    stats_for_xmt_socket (common.xmtAfterWindow, common.xmtAfterWindowRate, \
                                TCP_SOCKET, AL_XMT_AFTER_WINDOW, 0); \
    tcp_stats_errors; }
```

```
#define tcp_stats_after_window_bytes { \
     stats_rcv_bytes_for_node    (extraBytes, common.rcvAfterWindowBytes, \
                                    common.rcvAfterWindowByteRate, \
                                  TCP_NODE, AL_RCV_AFTER_WINDOW_BYTES, 0); \
     stats_xmt_bytes_for_node    (extraBytes, common.xmtAfterWindowBytes, \
                                    common.xmtAfterWindowByteRate, \
                                  TCP_NODE, AL_XMT_AFTER_WINDOW_BYTES, 0); \
     stats_rcv_bytes_for_socket (extraBytes, common.rcvAfterWindowBytes,\
                                    common.rcvAfterWindowByteRate, \
                                  TCP_SOCKET, AL_RCV_AFTER_WINDOW_BYTES, 0); \
     stats_xmt_bytes_for_socket (extraBytes, common.xmtAfterWindowBytes, \
                                    common.xmtAfterWindowByteRate, \
                                  TCP_SOCKET, AL_XMT_AFTER_WINDOW_BYTES, 0); }

#define tcp_stats_after_close { \
     stats_for_segs      (afterClose, afterCloseRate, TCP_SEGMENT, AL_AFTER_CLOSE, 0); \
     stats_for_rcv_node   (common.rcvAfterClose, common.rcvAfterCloseRate, \
                           TCP_NODE, AL_RCV_AFTER_CLOSE, 0);\
     stats_for_xmt_node   (common.xmtAfterClose, common.xmtAfterCloseRate, \
                           TCP_NODE, AL_XMT_AFTER_CLOSE, 0); \
     stats_for_rcv_socket (common.rcvAfterClose, common.rcvAfterCloseRate, \
                           TCP_SOCKET, AL_RCV_AFTER_CLOSE, 0);\
     stats_for_xmt_socket (common.xmtAfterClose, common.xmtAfterCloseRate, \
                           TCP_SOCKET, AL_XMT_AFTER_CLOSE, 0); \

     tcp_stats_errors; }

#define tcp_stats_urg { \
     stats_for_segs      (urgs, urgRate, TCP_SEGMENT, AL_URG, 0); \
     stats_for_rcv_node   (common.rcvUrgs, common.rcvUrgRate, TCP_NODE, AL_RCV_URG, 0); \

     stats_for_xmt_node   (common.xmtUrgs, common.xmtUrgRate, TCP_NODE, AL_XMT_URG, 0); \

     stats_for_rcv_socket (common.rcvUrgs, common.rcvUrgRate, TCP_SOCKET, AL_RCV_URG, 0);
\
     stats_for_xmt_socket (common.xmtUrgs, common.xmtUrgRate, TCP_SOCKET, AL_XMT_URG, 0);
}
```

```
#define tcp_stats_rsts { \
     stats_for_segs       (rsts, rstRate, TCP_SEGMENT, AL_RST, 0); \
     stats_for_rcv_node   (common.rcvRsts, common.rcvRstRate, TCP_NODE, AL_RCV_RST, 0); \

     stats_for_xmt_node   (common.xmtRsts, common.xmtRstRate, TCP_NODE, AL_XMT_RST, 0); \

     stats_for_rcv_socket (common.rcvRsts, common.rcvRstRate, TCP_SOCKET, AL_RCV_RST, 0);
\
     stats_for_xmt_socket (common.xmtRsts, common.xmtRstRate, TCP_SOCKET, AL_XMT_RST, 0);
\
     tcp_stats_errors; }

#define tcp_stats_seg_successful_connections { \
     stats_for_segs       (successfulConnections, successfulConnectionRate, \
                           TCP_SEGMENT, AL_SUCCESSFUL_CONNECTIONS, 0); }

#define tcp_stats_successful_connections { \
     stats_for_rcv_node   (common.rcvSuccessfulConnections,
common.rcvSuccessfulConnectionRate, \
                           TCP_NODE, AL_RCV_SUCCESSFUL_CONNECTIONS, 0); \
     stats_for_xmt_node   (common.xmtSuccessfulConnections,
common.xmtSuccessfulConnectionRate, \
                           TCP_NODE, AL_XMT_SUCCESSFUL_CONNECTIONS, 0); \
     stats_for_rcv_socket (common.rcvSuccessfulConnections,
common.rcvSuccessfulConneotionRate, \
                           TCP_SOCKET, AL_RCV_SUCCESSFUL_CONNECTIONS, 0); \
     stats_for_xmt_socket (common.xmtSuccessfulConnections,
common.xmtSuccessfulConnectionRate, \
                           TCP_SOCKET, AL_XMT_SUCCESSFUL_CONNECTIONS, 0); }

#define tcp_stats_reverse_successful_connections { \
     stats_for_rcv_node   (common.xmtSuccessfulConnections,
common.xmtSuccessfulConnectionRate, \
                           TCP_NODE, AL_XMT_SUCCESSFUL_CONNECTIONS, 0); \
     stats_for_xmt_node   (common.rcvSuccessfulConnections,
common.rcvSuccessfulConnectionRate, \
```

```
                                TCP_NODE, AL_RCV_SUCCESSFUL_CONNECTIONS, 0); \
     stats_for_rcv_socket (common.xmtSuccessfulConnections,
common.xmtSuccessfulConnectionRate, \
                                TCP_SOCKET, AL_XMT_SUCCESSFUL_CONNECTIONS, 0); \
     stats_for_xmt_socket (common.rcvSuccessfulConnections,
common.rcvSuccessfulConnectionRate, \
                                TCP_SOCKET, AL_RCV_SUCCESSFUL_CONNECTIONS, 0); }

#define tcp_stats_connection_retries { \
     stats_for_segs       (connectionRetries, connectionRetryRate, \
                                TCP_SEGMENT, AL_CONNECTION_RETRIES, 0); \
     stats_for_rcv_node   (common.rcvConnectionRetries, common.rcvConnectionRetryRate, \
                                TCP_NODE, AL_RCV_CONNECTION_RETRIES, 0); \
     stats_for_xmt_node   (common.xmtConnectionRetries, common.xmtConnectionRetryRate, \
                                TCP_NODE, AL_XMT_CONNECTION_RETRIES, 0); \
     stats_for_rcv_socket (common.rcvConnectionRetries, common.rcvConnectionRetryRate, \
                                TCP_NODE, AL_RCV_CONNECTION_RETRIES, 0); \
     stats_for_xmt_socket (common.xmtConnectionRetries, common.xmtConnectionRetryRate, \
                                TCP_NODE, AL_XMT_CONNECTION_RETRIES, 0); \
     if (dialog_stats_ptr != NULL) \
          dialog_stats_ptr->transport.connectionRetries++; }


#define tcp_stats_protocol { \
     stats_protocol (TCP_SEGMENT, TCP_NODE, TCP_PAIR, tcp_protocol); }


/*****************************************************************************
 *
 * Process the TCP options
 */

Uint32 tcp_dooptions (options_length)

     Uint32          options_length;
{
     register u_char     *cp;
```

```
        register int        opt, optlen, cnt;
        register u_short    t_maxseg;

    cp = (u_char *) (&tcp_ptr->th_urp + 2);
    cnt = options_length;


    /*
     * Process each option
     */
    for (; cnt > 0; cnt -= optlen, cp += optlen)
        {
        opt = cp[0];
        if (opt == TCPOPT_EOL)
            break;
        if (opt == TCPOPT_NOP)
            optlen = 1;
        else
            {
            optlen = cp[1];
            if (optlen <= 0)
                break;
            }
        switch (opt)
            {
            default:
                break;

            case TCPOPT_MAXSEG:
                if (optlen != 4)
                    continue;
                if (!(tcp_flags & TH_SYN))
                    continue;
                t_maxseg = *(u_short *)(cp + 2);
                t_maxseg = ntohs((u_short)t_maxseg);
                break;
            }
```

```
    }

tcp_options_good:
return (GOOD);

tcp_options_bad:
return (BAD);
}


/*****************************************************************
 *
 * TCP parse routine.  See pages 65-76 of the
 * protocol specification dated September, 1981.
 */

Uint32 rtp_tcp_parse (layer_ptr, length)

     char                *layer_ptr;
     Uint32              length;
{
     register Uint32              result, protocol, tcp_hdr_length, data_length,
                                  options_length, initiator, *uint32_ptr;

     AlarmUserData           stats_alarm_data;

/*
 * Set up local variables
 */

result = GOOD;

if ((ip_seg_type == MIDDLE_FRAGMENT) || (ip_seg_type == LAST_FRAGMENT))
     {
     /* No header to parse.  It's already been done (or will be). */
     /* Should count fragments here... */
     goto tcp_done;
```

```
    }

tcp_ptr = (struct tcphdr *) layer_ptr;
tcp_hdr_length = tcp_ptr->th_off << 2;        /* Number of 32-bit words in hdr */
tcp_flags = tcp_ptr->th_flags;

/*
 * Convert fields to local representation.
 * Don't bother with checksum or urgent ptr.
 */
tcp_src_port = ntohs (tcp_ptr->th_sport);
tcp_dst_port = ntohs (tcp_ptr->th_dport);
tcp_window   = ntohs (tcp_ptr->th_win);

/*
 * For alignment reasons, the tcp sequence number and ack number
 * as well as the ip source and destination addresses cannot be
 * defined as longs within the frame structure.  The ip addresses
 * have already been converted.  Convert the seq and ack numbers now.
 */
tcp_seq = (tcp_ptr->th_seq[0] << 24) + (tcp_ptr->th_seq[1] << 16) +
          (tcp_ptr->th_seq[2] << 8)  + tcp_ptr->th_seq[3];

tcp_ack = (tcp_ptr->th_ack[0] << 24) + (tcp_ptr->th_ack[1] << 16) +
          (tcp_ptr->th_ack[2] << 8)  + tcp_ptr->th_ack[3];


/*
 * Determine protocol.  If neither port is known, lump these stats in with the other
 * unknown port stats.
 */
tcp_protocol = stats_port_to_protocol (ip_dst_node_addr_ptr, tcp_ptr->th_dport,
TCP_PROTOCOL);
if (tcp_protocol == UNKNOWN_PORT)
    tcp_protocol = stats_port_to_protocol (ip_src_node_addr_ptr, tcp_ptr->th_sport,
TCP_PROTOCOL);
protocol = tcp_protocol;
```

```
tcp_connection_state_ptr = connection_state_ptr    = NULL;

if (tcp_src_node_addr_ptr = src_node_addr_ptr = stats_tcp_get_addr (&ip_src_addr))
     {
     tcp_src_node_stats_ptr = src_node_stats_ptr = (StatsTcpAddr *)
src_node_addr_ptr->stats_ptr;
     tcp_src_seg_addr_ptr    = src_seg_addr_ptr    =
                                   stats_tcp_get_segment
(src_node_addr_ptr->address.segment1);
     }
else
     {
     tcp_src_node_stats_ptr = src_node_stats_ptr = NULL;
     /**** IF GLOBAL IP POINTERS IMPLEMENTED DON'T NEED TO DO LOOKUP *****/
     if (ip_src_addr_ptr = (StatsAddrEntry *)stats_ip_lookup_addr(&ip_src_addr))
          tcp_src_seg_addr_ptr = src_seg_addr_ptr =
                                      stats_tcp_get_segment
(ip_src_addr_ptr->address.segment1);
     else
          tcp_src_seg_addr_ptr = src_seg_addr_ptr = NULL;
     }
if (src_seg_addr_ptr != NULL)
     tcp_src_seg_stats_ptr = src_seg_stats_ptr = (StatsTcpSegment *)
src_seg_addr_ptr->stats_ptr;
else
     tcp_src_seg_stats_ptr = src_seg_stats_ptr = NULL;

if (tcp_dst_node_addr_ptr = dst_node_addr_ptr = stats_tcp_get_addr (&ip_dst_addr))
     {
     tcp_dst_node_stats_ptr = dst_node_stats_ptr = (StatsTcpAddr *)
dst_node_addr_ptr->stats_ptr;
     tcp_dst_seg_addr_ptr = dst_seg_addr_ptr =
                                  stats_tcp_get_segment
(dst_node_addr_ptr->address.segment1);
     }
else
```

```
     {
     tcp_dst_node_stats_ptr = dst_node_stats_ptr = NULL;
     /**** IF GLOBAL IP POINTERS IMPLEMENTED DON'T NEED TO DO LOOKUP *****/
     if (ip_dst_addr_ptr = (StatsAddrEntry *)stats_ip_lookup_addr(&ip_dst_addr))
          tcp_dst_seg_addr_ptr = dst_seg_addr_ptr =
                                        stats_tcp_get_segment
(ip_dst_addr_ptr->address.segment1);
     else
          tcp_dst_seg_addr_ptr = dst_seg_addr_ptr = NULL;
     }
if (dst_seg_addr_ptr != NULL)
     tcp_dst_seg_stats_ptr = dst_seg_stats_ptr = (StatsTcpSegment *)
dst_seg_addr_ptr->stats_ptr;
else
     tcp_dst_seg_stats_ptr = dst_seg_stats_ptr = NULL;

if (tcp_src_socket_addr_ptr = src_socket_addr_ptr =
                                   stats_tcp_get_socket (&ip_src_addr, tcp_src_port))
     tcp_src_socket_stats_ptr = src_socket_stats_ptr =
                                   (StatsTcpSocket *) src_socket_addr_ptr->stats_ptr;
else
     tcp_src_socket_stats_ptr = src_socket_stats_ptr = NULL;

if (tcp_dst_socket_addr_ptr = dst_socket_addr_ptr =
                                   stats_tcp_get_socket (&ip_dst_addr, tcp_dst_port))
     tcp_dst_socket_stats_ptr = dst_socket_stats_ptr =
                                   (StatsTcpSocket *) dst_socket_addr_ptr->stats_ptr;
else
     tcp_dst_socket_stats_ptr = dst_socket_stats_ptr = NULL;

if (tcp_this_seg_addr_ptr = this_seg_addr_ptr = stats_tcp_get_segment (mySegmentId))
     tcp_this_seg_stats_ptr = this_seg_stats_ptr = (StatsTcpSegment
*)this_seg_addr_ptr->stats_ptr;
else
     tcp_this_seg_stats_ptr = this_seg_stats_ptr = NULL;

if (tcp_dialog_addr_ptr = dialog_addr_ptr =
```

```
    stats_tcp_get_dialog (&ip_src_addr, tcp_src_port, &ip_dst_addr, tcp_dst_port))
        tcp_dialog_stats_ptr = dialog_stats_ptr = (StatsDialogEntry *)
dialog_addr_ptr->stats_ptr;
else
        tcp_dialog_stats_ptr = dialog_stats_ptr = NULL;


/*
 * Update the age timer in each address structure.
 * Set the 10 second rate sample period in the statistics structures so that the
 * event manager will calculate rates.
 */
stats_update_age_timers;
stats_update_socket_age_timers;
stats_set_rate_10s;
stats_set_socket_rate_10s;


/*
 * Set the last mac address seen
 */
if (src_node_addr_ptr != NULL)
    src_node_addr_ptr->address.macAddress1 = mac_src_addr;
if (dst_node_addr_ptr != NULL)
    dst_node_addr_ptr->address.macAddress1 = mac_dst_addr;
if (src_socket_addr_ptr != NULL)
    src_socket_addr_ptr->address.macAddress1 = mac_src_addr;
if (dst_socket_addr_ptr != NULL)
    dst_socket_addr_ptr->address.macAddress1 = mac_dst_addr;


/*
 * Do some stats
 */
tcp_stats_frames;
tcp_stats_bytes;
```

```
tcp_stats_hdr_bytes;
tcp_stats_off_segs;
tcp_stats_transits;

if (ip_seg_type != NOT_A_FRAGMENT)
     {
     tcp_stats_fragments;
     }


/*
 * Find statistics structures for the protocol distribution statistics
 */

if (src_node_stats_ptr != NULL)
     tcp_src_node_protocol_ptr = src_node_protocol_ptr =
          stats_get_protocol (&src_node_stats_ptr->protocolQ, tcp_protocol);
else
     tcp_src_node_protocol_ptr = src_node_protocol_ptr = NULL;

if (dst_node_stats_ptr != NULL)
     tcp_dst_node_protocol_ptr = dst_node_protocol_ptr =
          stats_get_protocol (&dst_node_stats_ptr->protocolQ, tcp_protocol);
else
     tcp_dst_node_protocol_ptr = dst_node_protocol_ptr = NULL;

if (this_seg_stats_ptr != NULL)
     tcp_this_seg_protocol_ptr = this_seg_protocol_ptr =
          stats_get_protocol (&this_seg_stats_ptr->protocolQ, tcp_protocol);
else
     tcp_this_seg_protocol_ptr = this_seg_protocol_ptr = NULL;

if (src_seg_stats_ptr != NULL)
     tcp_src_seg_protocol_ptr = src_seg_protocol_ptr  =
          stats_get_protocol (&src_seg_stats_ptr->protocolQ, tcp_protocol);
else
     tcp_src_seg_protocol_ptr = src_seg_protocol_ptr  = NULL;
```

```
if (dst_seg_stats_ptr != NULL)
     tcp_dst_seg_protocol_ptr = dst_seg_protocol_ptr  =
          stats_get_protocol (&dst_seg_stats_ptr->protocolQ, tcp_protocol);
else
     tcp_dst_seg_protocol_ptr = dst_seg_protocol_ptr  = NULL;



/*
 * Keep protocol distribution statistics.
 * Pass the protocol as alarm data in case an alarm occurs.
 */
stats_alarm_data.length = 4;
uint32_ptr = (Uint32 *)stats_alarm_data.data;
*uint32_ptr = tcp_protocol;
tcp_stats_protocol;


/*
 * Check the length and header length for the following problems:
 *   1. length < minimum length (frame truncated, can't parse)
 *   2. hdr length < minimum length
 *   3. hdr length > length
 *   4. hdr length is inconsistent with length
 */
if ( (length < sizeof(struct tcphdr)) ||
      (tcp_hdr_length > length) ||
      ( (tcp_hdr_length > sizeof(struct tcphdr)) &&
        (length < tcp_hdr_length) ) )
     {
     tcp_stats_errors;
     goto tcp_bad;
     }

/*
 * Go through the options
```

```
 */
if (tcp_hdr_length > sizeof(struct tcphdr))
     tcp_dooptions (tcp_hdr_length - sizeof(struct tcphdr));


/* Count frames that have urgent data */
if (tcp_flags & TH_URG)
     {
     tcp_stats_urg;
     }


/*
 * Advance past the tcp header
 */
data_length = length - tcp_hdr_length;



/*
 * Set up pointers to tcp state information and pass through the state machine
 */
if (dialog_stats_ptr != NULL)
     {
     tcp_connection_state_ptr = connection_state_ptr =
          (TcpConnectionStateType *) dialog_stats_ptr->transport.state_ptr;

     if ((ip_src_addr == dialog_addr_ptr->address.netAddress1.u.ipAddress) &&
         (tcp_src_port == dialog_addr_ptr->address.port1))
          {
          initiator = 1;
          tcp_src_state_ptr = src_state_ptr = &connection_state_ptr->tcp1;
          tcp_dst_state_ptr = dst_state_ptr = &connection_state_ptr->tcp2;
          }
     else
          {
          initiator = 2;
```

```
            tcp_src_state_ptr = src_state_ptr = &connection_state_ptr->tcp2;
            tcp_dst_state_ptr = dst_state_ptr = &connection_state_ptr->tcp1;
            }

        /*
         * Set the event code and call the state trakker.
         */

        src_state_ptr->current_event = EV_UNKNOWN;

        if (tcp_flags & TH_ACK)
            src_state_ptr->current_event = EV_ACK;

        if (data_length > 0)
            src_state_ptr->current_event = EV_DATA;

        if (tcp_flags & TH_SYN)
            src_state_ptr->current_event = EV_SYN;

        if (tcp_flags & TH_FIN)
            src_state_ptr->current_event = EV_FIN;

        if (tcp_flags & TH_RST)
            {
            tcp_stats_rsts;
            src_state_ptr->current_event = EV_RST;
            }

        result = rtp_tcp_state (data_length, initiator);

        if (result != GOOD)
            return (result);


        }
    /*
     * Parse the next layer
```

```
 */
switch (tcp_protocol)
     {
     case FTP_PORT:
          /* result = rtp_ftp_parse (layer_ptr + tcp_hdr_length, data_length,
TCP_PROTOCOL); */
          break;

     case TELNET_PORT:
          /* result = rtp_telnet_parse (layer_ptr + tcp_hdr_length, data_length,
TCP_PROTOCOL); */
          break;

     case SMTP_PORT:
          /* result = rtp_smtp_parse (layer_ptr + tcp_hdr_length, data_length,
TCP_PROTOCOL); */
          break;

     case MOUNT_PORT:
     case NFS_PORT:
     case MAPPER_PORT:
          result = rtp_rpc_parse (layer_ptr + tcp_hdr_length, data_length, TCP_PROTOCOL);

          break;

     default:
          break;
     }

return (result);

tcp_good:
return (GOOD);

tcp_done:
return (DONE);
```

```
tcp_bad:
return (BAD);
}


/*****************************************************************************
 *
 * If the RST is being sent in response to a SYN, the RST is valid if
 * the ACK field acknowledges the SYN.  In all other cases, the RST is
 * valid if its sequence number is in the window.  This routine checks
 * for a valid reset and returns GOOD, BAD, or NOT_KNOWN.
 */
Uint32 valid_reset ()
{
return (GOOD);
}


/*****************************************************************************
 *
 * Set all seq numbers for the first time based on the ones in the frame.
 */
Uint32 set_initial_seq_numbers (data_length)

     Uint32          data_length;
{
src_state_ptr->max_seq_sent = tcp_seq;
src_state_ptr->last_length_sent = data_length;
src_state_ptr->max_ack_sent = tcp_ack;
src_state_ptr->last_window_sent = tcp_window;
src_state_ptr->max_window_sent = tcp_window;
src_state_ptr->min_window_sent = tcp_window;
src_state_ptr->snd_wl1 = tcp_seq;
src_state_ptr->snd_wl2 = tcp_ack;
return (GOOD);
```

```
}


/*****************************************************************
 *
 * Look back into the state history and try to make a decision about
 * the connection state based on the activity. If each node has
 * sent at least 2 ack or data frames, assume they're in data state.
 */
Uint32 look_for_data (data_length, p_last)

     Uint32    data_length, p_last;

{
     register Uint32     i, node1, node2, syn_count, syn_initiator, p_past;

p_past = p_last;
if (src_state_ptr->indicator == 1)
     {
     node1 = 1;
     node2 = 0;
     }
else
     {
     node1 = 0;
     node2 = 1;
     }

i = 0;
while (i<TCP_MAX_HISTORY)
     {
     i++;
     switch (connection_state_ptr->history[p_past].event)
          {
     case EV_ACK:
     case EV_DATA:
```

```
                if (connection_state_ptr->history[p_past].initiator == 1)
                node1++;
                else
                node2++;
                if ( (node1 > 1) && (node2 > 1) )
                goto done_looking_for_data;
                 break;

           /*
            * Don't go to data state if there's a syn, fin or rst in there.
            */
           case EV_SYN:
           case EV_FIN:
           case EV_RST:
                goto done_looking_for_data;
                break;

      default:
                break;
      }
      p_past = (p_past == 0) ? TCP_MAX_HISTORY - 1 : p_past - 1;
      }

done_looking_for_data:

if ( (node1 > 1) && (node2 > 1) )
      {
      src_state_ptr->current_state = ST_DATA;
      dst_state_ptr->current_state = ST_DATA;
      dialog_stats_ptr->transport.state = ConnectionStateData;
      tcp_stats_active_connections;
      tcp_stats_seq_successful_connections;
     set_initial_seq_numbers (data_length);

      /* Try to determine which side initiated the connection */
      syn_initiator = 0;
      syn_count = 0;
```

```
        p_past = p_last;

        i = 0;
        while (i<TCP_MAX_HISTORY)
        {
        i++;
        switch (connection_state_ptr->history[p_past].event)
                  {
                  case EV_SYN:
                       if (syn_count > 0 )
                            {
                            if (syn_initiator !=
connection_state_ptr->history[p_past].initiator)
                                 {
                                 syn_count = 2;
                            syn_initiator = connection_state_ptr->history[p_past].initiator;

                                 break;
                                 }
                            }
                       else
                            {
                            syn_count = 1;
                       syn_initiator = connection_state_ptr->history[p_past].initiator;
                            }

                  break;

                  default:
                       break;
                  }

        p_past = (p_past == 0) ? TCP_MAX_HISTORY - 1 : p_past - 1;
             }

        if (syn_count > 1)
             {
```

```
            dialog_stats_ptr->transport.initiator = syn_initiator;
            if (src_state_ptr->indicator == syn_initiator)
                 {
                 tcp_stats_successful_connections;
                 }
            else
                 /*
                  * Reverse so that initiator gets the right statistic adjusted.
                  */
                 {
                 tcp_stats_reverse_successful_connections;
                 }
            }
       else
            {
            dialog_stats_ptr->transport.initiator = ConnectionInitiatorUnknown;
            }
       }

  return (GOOD);
  }


  /*************************************************************************
   *
   * Look back into the state history and see if this frame is a rexmt.
   * Do this by finding data transmitted by the same node.  If the seq
   * number in the state history is >= to that of this transmission,
   * this is either out of order or it's a retransmission.  Note
   * that the sequence number space range
   * is 0 - 2**32.  Even though this applies to bytes, it won't wrap
   * in the space of a small number of transmissions.  If it did, this
   * routine could be modified to check acks from the other node.
   */

  Uint32 look_for_rexmt (data_length, p_last)
```

```
Uint32          data_length;
Uint32          p_last;

{
     register Uint32          i, p_past;

p_past = p_last;

i = 0;
while (i<MAX_LOOK_FOR_REXMT)
  {
  i++;
  if ( (connection_state_ptr->history[p_past].data_length > 0) &&
       (connection_state_ptr->history[p_past].initiator == src_state_ptr->indicator) &&
       (connection_state_ptr->history[p_past].seq >= tcp_seq) )
          {
          tcp_stats_rexmt;
         tcp_stats_rexmt_bytes;
        goto done_looking;
        }
  p_past = (p_past == 0) ? TCP_MAX_HISTORY -1 : p_past - 1;
  }

done_looking:

return (GOOD);
}



/*************************************************************************
 *
 * Examine data
 */
Uint32 process_data (data_length, p_last)
```

```
    Uint32    data_length, p_last;

{
    register Uint32     extraBytes;

if (data_length == 0)
    return (GOOD);

/*
 * Remember the largest amount of data the peer has sent into the window.
 */
if (data_length > src_state_ptr->max_data_sent)
    src_state_ptr->max_data_sent = data_length;

/*
 * If this seq <  or = max seq sent, and if data was sent, consider this a
 * retransmission.  Additional checking should actually be done for out or
 * order vs. rexmt and for overlapping data.
 */
if ( (SEQ_LEQ (tcp_seq, src_state_ptr->max_seq_sent))&&
    (src_state_ptr->last_length_sent != 0) )
        {
        tcp_stats_rexmt;
       tcp_stats_rexmt_bytes;
      }

/*
 * Remember the data transfer with the highest sequence number and its length.
 */
else
    {
    src_state_ptr->max_seq_sent = tcp_seq;
    src_state_ptr->last_length_sent = data_length;
    }

/*
```

```
 * The sender shouldn't send more than the window offered by its peer.
 * The window size was set when the peer sent a window update.
 * Note that if we missed a window update, this count will be in error.
 */
if (data_length > dst_state_ptr->last_window_sent)
     {
     extraBytes = data_length - dst_state_ptr->last_window_sent;
     tcp_stats_after_window;
     tcp_stats_after_window_bytes;
     return (GOOD);
     }
}

/*********************************************************************
 *
 * Process window and the ack sequence number
 */
Uint32 process_ack ()

{
if ((tcp_flags & TH_ACK) == 0)
     return;

/*
 * If the sequence number of the last window update <= this one and
 * if the ack sent in the last window update <= this one
 * and if this window > last one, (Should this check be done?)
 * update the window information.
 */
if ( (SEQ_LT(dst_state_ptr->snd_wl1, tcp_seq) || dst_state_ptr->snd_wl1 == tcp_seq &&
     (SEQ_LT(dst_state_ptr->snd_wl2, tcp_ack) || dst_state_ptr->snd_wl2 == tcp_ack &&
      tcp_window > dst_state_ptr->last_window_sent)) )
     {

     /* To keep track of pure window updates:
      * if (data_length == 0 &&
```

```
      *  dst_state_ptr->snd_wl2 == tcp_ack && tcp_window >
dst_state_ptr->last_window_sent)
      * tcp_stats_window_update;
      */

     dst_state_ptr->last_window_sent = tcp_window;
     dst_state_ptr->snd_wl1 = tcp_seq;
     dst_state_ptr->snd_wl2 = tcp_ack;
     if (tcp_window > dst_state_ptr->max_window_sent)
          dst_state_ptr->max_window_sent = tcp_window;
     if (tcp_window < dst_state_ptr->min_window_sent)
          dst_state_ptr->min_window_sent = tcp_window;

     /*
      * Set the last advertised receive window in the dialog stats
      */
     if (dst_state_ptr->indicator == 1)
          dialog_stats_ptr->transport.addr1_window = tcp_window;
     else
          dialog_stats_ptr->transport.addr2_window = tcp_window;

     }

     if ( SEQ_LT (src_state_ptr->max_ack_sent, tcp_ack) )
          src_state_ptr->max_ack_sent = tcp_ack;
}



/*****************************************************************************
 *
 *        TCP State Tracker
 *
 * Objectives of the TCP State Tracker:
 *
 * 1) Keep history of events and state transitions per connection
```

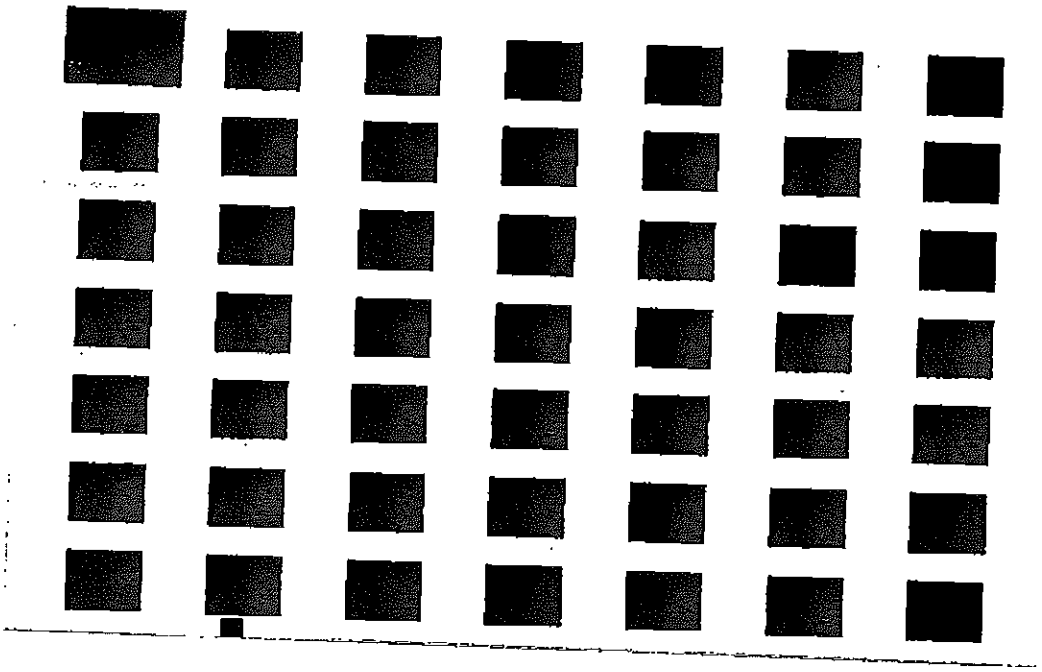
```
 * 2) Detect data transfer state so that seq tracking can begin
 * 3) Count inconsistencies but maintain tracking while falling into
 *     an appropriate state (e.g. unknown).
 *
 */

static unsigned tcp_state_table [7][6] =
{
/*
                                        ST_INACTIVE
                                      ST_CLOSED |
                         ST_CLOSING     |     |
                        ST_DATA    |    |     |
              ST_CONNECTING    |   |    |     |
             ST_UNKNOWN    |   |   |    |     |
STATES                |   |   |   |    |     |
-------------------------------------------------------------------------
EVENTS
*/
/*EV_UNKNOWN*/  {  0,  0,  0,  0,  4,   5 },
/*EV_SYN*/        {  1,  2,  3,  3,  6,   1 },
/*EV_RST*/        { 11, 11, 11, 11,  7,  11 },
/*EV_ACK*/        { 21, 21, 23, 24,  4,  22 },
/*EV_FIN*/        { 31, 31, 31, 32,  4,  31 },
/*EV_DATA*/       { 41, 41, 43, 44,  4,  22 },
/*EV_CLOSETIME*/{  0,  0,  0, 54,  0,   0 },
};


Uint32 rtp_tcp_state (data_length, initiator)
     Uint32                         data_length, initiator;

{    register Uint32                 action, result, p_index, p_last;
```

```
result = GOOD;

action = tcp_state_table[src_state_ptr->current_event][src_state_ptr->current_state];


/*
 * Get indices to last recorded events and current events
 * Note that histx points to the next entry to store history into.
 *
 * p_index = current pair history index
 * p_last = last pair history index
 */
p_index = connection_state_ptr->histx;
p_last = (connection_state_ptr->histx == 0) ?
    TCP_MAX_HISTORY - 1 : connection_state_ptr->histx - 1;


/*
 * Save this event in history for the connection.
 */
connection_state_ptr->history[p_index].state = src_state_ptr->current_state;
connection_state_ptr->history[p_index].event = src_state_ptr->current_event;
connection_state_ptr->history[p_index].data_length = data_length;
connection_state_ptr->history[p_index].seq = tcp_seq;
connection_state_ptr->history[p_index].ack = tcp_ack;
connection_state_ptr->history[p_index].flags = tcp_flags;
connection_state_ptr->history[p_index].initiator = initiator;
connection_state_ptr->histx = ((p_index + 1) == TCP_MAX_HISTORY) ?
                                        0 : p_index + 1;

switch (action)
  {

  case 0;
     /* Unknown event */
     break;
```

```
case 1:
case_1:
  /*
   * syn detected
   *
   * Assume the connection is starting up.
   * Ignore the syn if the segment contains rst.
   * Initialize rcv seq number variables, enter connecting state,
   * and process other fields of the segment.
   */
  src_state_ptr->current_state = ST_CONNECTING;
   dialog_stats_ptr->transport.state = ConnectionStateConnecting;
  set_initial_seq_numbers (data_length);

  /* Check for ack and update window info.
   * Process urg.
   */

  /* Data may be sent on syn.  If we haven't seen anything from the dst
   * (e.g. it  has not yet sent a syn), we can't know what the dst's rcv
   * window is, so we should not check the amount of data sent against the
   * dst's rcv window.
   * if (dst_state_ptr->current_state == ST_UNKNOWN)
   *     {}
   */

  break;


case 2:
  /* Duplicate syn */
  top_stats_connection_retries;
  goto case_1;
  break;
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerard White

04



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
case 3:
  /* Out of order */
   if (dialog_stats_ptr->transport.state == ConnectionStateData)
        tcp_stats_decr_active_connections;
  src_state_ptr->current_state = ST_UNKNOWN;
   dialog_stats_ptr->transport.state = ConnectionStateUnknown;
  tcp_stats_out_of_order;
  break;


case 4:
  src_state_ptr->current_state = ST_UNKNOWN;
   dialog_stats_ptr->transport.state = ConnectionStateUnknown;
   tcp_stats_after_close;
   break;


case 5:
  src_state_ptr->current_state = ST_UNKNOWN;
   dialog_stats_ptr->transport.state = ConnectionStateUnknown;
   break;


case 6:
  src_state_ptr->current_state = ST_CONNECTING;
   dialog_stats_ptr->transport.state = ConnectionStateConnecting;
   tcp_stats_after_close;
   break;


case 7:
   tcp_stats_after_close;


case 11:
  /* Reset detected */
  switch ( result = valid_reset() )
```

```
    {
    case GOOD:
        if (dialog_stats_ptr->transport.state == ConnectionStateData)
            {
            tcp_stats_decr_active_connections;
            }
        if (dialog_stats_ptr->transport.state == ConnectionStateConnecting)
            {
            tcp_stats_failed_connections;
            }
      src_state_ptr->current_state = ST_CLOSED;
      dst_state_ptr->current_state = ST_CLOSED;
        dialog_stats_ptr->transport.state = ConnectionStateClosed;
        dialog_stats_ptr->transport.closeReason = ConnectionCloseRst;
      break;

    case BAD:
      break;

    case NOT_KNOWN:
      break;
    }

  break;


case 21:
  look_for_data (data_length, p_last);
  break;


case 22:
  src_state_ptr->current_state = ST_UNKNOWN;
   dialog_stats_ptr->transport.state = ConnectionStateUnknown;
  look_for_data (data_length, p_last);
  break;
```

```
case 23:
  /*
   * State = DATA, ack detected
   */
   process_ack ();
  break;


case 24:
  /* State = closing, ack detected */
   process_ack ();
  break;


/*
* NOTE: Sending the fin bit with a retransmit of the last unacked data
* packet is characteristic of the TCP implementation from FTP, which
* assumes that the cost per packet > cost per byte.
*/

 case 31:
   /* State = unknown, fin detected */
    if (dialog_stats_ptr->transport.state == ConnectionStateData)
          {
          tcp_stats_decr_active_connections;
          }
    src_state_ptr->current_state = ST_CLOSING;
    dialog_stats_ptr->transport.state = ConnectionStateClosing;
   break;


 case 32:
   /* State = closing, fin detected */
   /* Duplicate fin */
   break;
```

```
case 41:
  /* State = unknown, data detected */
  look_for_data (data_length, p_last);
  break;


case 43:
  /* State = data, data detected */
   process_ack ();
   process_data (data_length, p_last);
  break;


case 44:
  /* State = closing, data detected */
  tcp_stats_out_of_order;
  break;


case 54:
   src_state_ptr->current_state = ST_CLOSED;
   dialog_stats_ptr->transport.state = ConnectionStateClosed;
  break;


default:
   stats_mon_db_problem;
  break;


 }

done:

return (result);
```

```
}

/*****************************************************************************
*
* Process an icmp source quench
*/

void rtp_tcp_src_quench ()

{
register SrcQuenchData   *srcQuenchData_ptr;

srcQuenchData_ptr = &srcQuenchData;

stats_tcp_lookup_ptrs(&srcQuenchData_ptr->ip_src_addr, srcQuenchData_ptr->src_port,
                      &srcQuenchData_ptr->ip_dst_addr, srcQuenchData_ptr->dst_port);

tcp_stats_flow_ctrls;

if ( (srcQuenchData_ptr->src_port == MOUNT_PORT) ||
     (srcQuenchData_ptr->dst_port == MOUNT_PORT) ||
      (srcQuenchData_ptr->src_port == NFS_PORT) ||
      (srcQuenchData_ptr->dst_port == NFS_PORT) )
      {
      rtp_nfs_src_quench ();
      }
}
```

```
/*
 *  rtp_udp_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                   All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/rtp/SCCS/s.rtp_udp_p.c
 *
 *  Date:       8/26/91
 *
 *  Revision:   1.10
 *
 *
 *  Changes:
 *
 *    MM-DD-YY  WHO       Description of change. (latest first)
 *    --------  ---       ---------------------------------------------
 *
 *    08-26-91  KR        changed src quench to use stats_port_to_protocol
 *    07-23-91  KR        call stats_port_to_protocol to get protocol and call
rtp_rpc_parse if
 *                             protocol is nfs or port mapper
 *  06-27-91    KR      changed udp_stats_off_seg macro to test stats ptr prior to
stats_count
 *    06-25-91  KR        Added check for NULL around access of ip_src_addr_ptr and
ip_dst_addr_ptr
 *    06-18-91  KR        Changed off seg macro to use seg addr records for checks intead
of nodes
 *                             Made ptrs static
 *                             Added rtp_udp_src_quench
 *    06-14-91  KR        Moved set of udp_protocol up so that stats_tcp_get_dialog could
 *                             use it to set applicationProtocol field
 *    06-07-91  KR        Added call to stats_well_known_port
```

```
*    06-03-91  DPD        Fixed on/seg/transit count bug
*
*/

static char rtp_udp_p_c [] = "@(#)rtp_udp_p.c      1.10";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include <sys/types.h>
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "in.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_udp.h"
#include "rtp.h"
#include "rtp_dll.h"
#include "rtp_ip.h"
#include "rtp_udp.h"


/*****************************************************************
 *
 * UDP monitoring
 */


/*
 * Global UDP data structures
 */
struct udphdr  *udp_ptr;
Uint32          udp_src_port;
Uint32          udp_dst_port;
```

```
Uint32              udp_protocol;


/*
 * Local UDP data structures
 */

static StatsAddrEntry       *this_seg_addr_ptr;
static StatsUdpSegment      *this_seg_stats_ptr;
static StatsProtocolEntry   *this_seg_protocol_ptr;

static StatsAddrEntry       *src_seg_addr_ptr;
static StatsUdpSegment      *src_seg_stats_ptr;
static StatsProtocolEntry   *src_seg_protocol_ptr;

static StatsAddrEntry       *dst_seg_addr_ptr;
static StatsUdpSegment      *dst_seg_stats_ptr;
static StatsProtocolEntry   *dst_seg_protocol_ptr;

static StatsAddrEntry       *src_node_addr_ptr;
static StatsUdpAddr         *src_node_stats_ptr;
static StatsProtocolEntry   *src_node_protocol_ptr;

static StatsAddrEntry       *dst_node_addr_ptr;
static StatsUdpAddr         *dst_node_stats_ptr;
static StatsProtocolEntry   *dst_node_protocol_ptr;

static StatsAddrEntry       *src_socket_addr_ptr;
static StatsUdpSocket       *src_socket_stats_ptr;

static StatsAddrEntry       *dst_socket_addr_ptr;
static StatsUdpSocket       *dst_socket_stats_ptr;

static StatsAddrEntry       *dialog_addr_ptr;
static StatsDialogEntry     *dialog_stats_ptr;
```

```
static StatsAddrEntry          *ip_src_addr_ptr;
static StatsAddrEntry          *ip_dst_addr_ptr;

/*
 * Macros for UDP statistics
 * These use the macros defined in stats.h
 */
#define udp_stats_frames { \
     stats_for_segs        (frames, frameRate, UDP_SEGMENT, AL_FRAMES, 0); \
     stats_for_nodes       (common.frames, common.frameRate, UDP_NODE, AL_FRAMES, 0); \
     stats_for_rcv_node    (common.rcvFrames, common.rcvFrameRate, UDP_NODE, AL_RCV_FRAMES,
0); \
     stats_for_xmt_node    (common.xmtFrames, common.xmtFrameRate, UDP_NODE, AL_XMT_FRAMES,
0); \
     stats_for_sockets     (common.frames, common.frameRate, UDP_SOCKET, AL_FRAMES, 0); \
     stats_for_rcv_socket  (common.rcvFrames, common.rcvFrameRate, UDP_SOCKET,
AL_RCV_FRAMES, 0); \
     stats_for_xmt_socket  (common.xmtFrames, common.xmtFrameRate, UDP_SOCKET,
AL_XMT_FRAMES, 0); \
     stats_for_dialog      (packets, packetRate, UDP_PAIR, AL_FRAMES, 0); }

#define udp_stats_bytes { \
     stats_bytes_for_segs        (length, bytes, byteRate, UDP_SEGMENT, AL_BYTES, 0); \
     stats_bytes_for_nodes       (length, common.bytes, common.byteRate, UDP_NODE,
AL_BYTES, 0); \
     stats_rcv_bytes_for_node    (length, common.rcvBytes, common.rcvByteRate, \
                                  UDP_NODE, AL_RCV_BYTES, 0); \
     stats_xmt_bytes_for_node    (length, common.xmtBytes, common.xmtByteRate, \
                                  UDP_NODE, AL_XMT_BYTES, 0); \
     stats_bytes_for_sockets     (length, common.bytes, common.byteRate, \
                                  UDP_SOCKET, AL_BYTES, 0); \
     stats_rcv_bytes_for_socket  (length, common.rcvBytes, common.rcvByteRate, \
                                  UDP_SOCKET, AL_RCV_BYTES, 0); \
     stats_xmt_bytes_for_socket  (length, common.xmtBytes, common.xmtByteRate, \
                                  UDP_SOCKET, AL_XMT_BYTES, 0); \
     stats_bytes_for_dialog      (length, bytes, byteRate, UDP_PAIR, AL_BYTES, 0); }
```

```
#define udp_stats_errors { \
     stats_for_segs        (errors, errorRate, UDP_SEGMENT, AL_ERRORS, 0); \
     stats_for_nodes       (common.errors, common.errorRate, UDP_NODE, AL_ERRORS, 0); \
     stats_for_rcv_node    (common.rcvErrors, common.rcvErrorRate, UDP_NODE, AL_RCV_ERRORS,
0); \
     stats_for_xmt_node    (common.xmtErrors, common.xmtErrorRate, UDP_NODE, AL_XMT_ERRORS,
0); \
     stats_for_sockets     (common.errors, common.errorRate, UDP_SOCKET, AL_ERRORS, 0); \
     stats_for_rcv_socket (common.rcvFrames, common.rcvFrameRate, UDP_SOCKET,
AL_RCV_FRAMES, 0); \
     stats_for_xmt_socket (common.xmtFrames, common.xmtFrameRate, UDP_SOCKET,
AL_XMT_FRAMES, 0); \
     stats_for_dialog      (errors, errorRate, UDP_PAIR, AL_ERRORS, 0); }

#define udp_stats_off_segs { \
     if ( (src_seg_addr_ptr != NULL) && (dst_seg_addr_ptr != NULL) ) { \
          if ( src_seg_addr_ptr->address.segment1 != dst_seg_addr_ptr->address.segment1) {
\
               stats_for_rcv_node (common.rcvOffSegs, common.rcvOffSegRate, \
                                   UDP_NODE, AL_RCV_OFF_SEG, 0); \
               stats_for_xmt_node (common.xmtOffSegs, common.xmtOffSegRate, \
                                   UDP_NODE, AL_XMT_OFF_SEG, 0); \
               if (src_seg_stats_ptr != NULL) \
                    stats_count (&src_seg_stats_ptr->xmtOffSegs, \
                                 &src_seg_stats_ptr->xmtOffSegRate, \
                                 src_seg_addr_ptr, UDP_SEGMENT, AL_XMT_OFF_SEG, 0); \
               if (dst_seg_stats_ptr != NULL) \
                    stats_count (&dst_seg_stats_ptr->rcvOffSegs, \
                                 &dst_seg_stats_ptr->rcvOffSegRate, \
                                 dst_seg_addr_ptr, UDP_SEGMENT, AL_RCV_OFF_SEG, 0); \
               stats_for_rcv_socket (common.rcvOffSegs, common.rcvOffSegRate, \
                                     UDP_SOCKET, AL_RCV_OFF_SEG, 0); \
               stats_for_xmt_socket (common.xmtOffSegs, common.xmtOffSegRate, \
                                     UDP_SOCKET, AL_XMT_OFF_SEG, 0); \

          } \
     } \
}
```

```
#define udp_stats_transits { \
     stats_transits (UDP_SEGMENT, UDP_NODE, UDP_PAIR, UDP_APPL_PROTOCOL); }


#define udp_stats_flow_ctrls { \
     stats_for_segs        (flowCtrls, flowCtrlRate, UDP_SEGMENT, AL_FLOW_CTRLS, 0); \
     stats_for_nodes       (common.flowCtrls, common.flowCtrlRate, UDP_NODE, AL_FLOW_CTRLS,
0); \
     stats_for_sockets     (common.flowCtrls, common.flowCtrlRate, UDP_SOCKET,
AL_FLOW_CTRLS, 0); \
     stats_for_dialog      (flowCtrls, flowCtrlRate, UDP_PAIR, AL_FLOW_CTRLS, 0); }


#define udp_stats_fragments { \
     stats_for_segs        (frgmts, frgmtRate, UDP_SEGMENT, AL_FRAGMENTS, 0); \
     stats_for_nodes           (common.frgmts, common.frgmtRate, UDP_NODE, AL_FRAGMENTS,
0); \
     stats_for_rcv_node    (common.rcvFrgmts, common.rcvFrgmtRate, UDP_NODE,
AL_RCV_FRAGMENTS, 0); \
     stats_for_xmt_node    (common.xmtFrgmts, common.rcvFrgmtRate, UDP_NODE,
AL_XMT_FRAGMENTS, 0); \
     stats_for_sockets     (common.frgmts, common.frgmtRate, UDP_SOCKET, AL_FRAGMENTS, 0);
\
     stats_for_rcv_socket (common.rcvFrgnts, common.rcvFrgmtRate, UDP_SOCKET,
AL_RCV_FRAGMENTS,0);\
     stats_for_xmt_socket (common.xmtFrgnts, common.xmtFrgmtRate, UDP_SOCKET,
AL_XMT_FRAGMENTS,0);\
     stats_for_dialog      (fragments, fragmentRate, UDP_PAIR, AL_FRAGMENTS, 0); }

#define udp_stats_protocol { \
     stats_protocol (UDP_SEGMENT, UDP_NODE, UDP_PAIR, udp_protocol); }


/*****************************************************************
```

```
*
* UDP parse routine
*/

Uint32 rtp_udp_parse (layer_ptr, length)

     char                *layer_ptr;
     Uint32              length;
{
     Uint32                   result, protocol, data_length, initiator;
     AlarmUserData       stats_alarm_data;


/*
 * Set up local variables
 */

result = GOOD;

if ((ip_seg_type == MIDDLE_FRAGMENT) || (ip_seg_type == LAST_FRAGMENT))
     {
     /* No header to parse.  It's already been done (or will be). */
     /* Should count fragments here... */
     goto udp_done;
     }

udp_ptr = (struct udphdr *) layer_ptr;

/*
 * Convert fields to local representation.
 * Don't bother with checksum or urgent ptr.
 */
udp_src_port = ntohs (udp_ptr->uh_sport);
udp_dst_port = ntohs (udp_ptr->uh_dport);
data_length = ntohs (udp_ptr->uh_ulen) - sizeof(struct udphdr);

/*
```

```
 * Determine protocol.  If neither port is known, lump these stats in with the
 * other unknown port stats.
 */
udp_protocol = stats_port_to_protocol (ip_dst_node_addr_ptr, udp_dst_port, UDP_PROTOCOL);

if (udp_protocol == UNKNOWN_PORT)
     udp_protocol = stats_port_to_protocol (ip_src_node_addr_ptr, udp_src_port,
UDP_PROTOCOL);
protocol = udp_protocol;


/*
 * Find all statistics records
 */
udp_this_seg_addr_ptr          = NULL;
udp_src_seg_addr_ptr           = NULL;
udp_dst_seg_addr_ptr           = NULL;
udp_this_seg_stats_ptr         = NULL;
udp_src_seg_stats_ptr          = NULL;
udp_dst_seg_stats_ptr          = NULL;
udp_this_seg_protocol_ptr      = NULL;
udp_src_seg_protocol_ptr = NULL;
udp_dst_seg_protocol_ptr = NULL;
udp_src_node_stats_ptr         = NULL;
udp_dst_node_stats_ptr         = NULL;
udp_src_socket_stats_ptr = NULL;
udp_dst_socket_stats_ptr = NULL;
udp_src_node_protocol_ptr      = NULL;
udp_dst_node_protocol_ptr      = NULL;
udp_dialog_stats_ptr           = NULL;

udp_src_node_addr_ptr = stats_udp_get_addr (&ip_src_addr);
if (udp_src_node_addr_ptr != NULL)
     {
     udp_src_node_stats_ptr = (StatsUdpAddr *) udp_src_node_addr_ptr->stats_ptr;
     udp_src_seg_addr_ptr = stats_udp_get_segment
(udp_src_node_addr_ptr->address.segment1);
```

```
    }
else
    {
    /* get segment id from ip structure */
    if (ip_src_addr_ptr = (StatsAddrEntry *)stats_ip_lookup_addr (&ip_src_addr))
        udp_src_seg_addr_ptr = stats_udp_get_segment
(ip_src_addr_ptr->address.segment1);
    }
if (udp_src_seg_addr_ptr != NULL)
    udp_src_seg_stats_ptr = (StatsUdpSegment *) udp_src_seg_addr_ptr->stats_ptr;

udp_dst_node_addr_ptr = stats_udp_get_addr (&ip_dst_addr);
if (udp_dst_node_addr_ptr != NULL)
    {
    udp_dst_node_stats_ptr = (StatsUdpAddr *) udp_dst_node_addr_ptr->stats_ptr;
    udp_dst_seg_addr_ptr = stats_udp_get_segment
(udp_dst_node_addr_ptr->address.segment1);
    }
else
    {
    /* get segment id from ip structure */
    if (ip_dst_addr_ptr = (StatsAddrEntry *)stats_ip_lookup_addr (&ip_dst_addr))
        udp_dst_seg_addr_ptr = stats_udp_get_segment
(ip_dst_addr_ptr->address.segment1);
    }
if (udp_dst_seg_addr_ptr != NULL)
    udp_dst_seg_stats_ptr = (StatsUdpSegment *) udp_dst_seg_addr_ptr->stats_ptr;

udp_src_socket_addr_ptr = stats_udp_get_socket (&ip_src_addr, udp_src_port);
if (udp_src_socket_addr_ptr != NULL)
    udp_src_socket_stats_ptr = (StatsUdpSocket *) udp_src_socket_addr_ptr->stats_ptr;

udp_dst_socket_addr_ptr = stats_udp_get_socket (&ip_dst_addr, udp_dst_port);
if (udp_dst_socket_addr_ptr != NULL)
    udp_dst_socket_stats_ptr = (StatsUdpSocket *) udp_dst_socket_addr_ptr->stats_ptr;

udp_this_seg_addr_ptr = stats_udp_get_segment (mySegmentId);
```

```
if (udp_this_seg_addr_ptr != NULL)
     udp_this_seg_stats_ptr = (StatsUdpSegment *) udp_this_seg_addr_ptr->stats_ptr;

/***** FIX FOR UDP DIALOGS:                                                    *****

 *****    dialogs are now kept as well-known socket + ip addr (port = 0)     *****
 *****
*****/
if (udp_protocol == udp_ptr->uh_dport)
{
     udp_dialog_addr_ptr =
          stats_udp_get_dialog (&ip_src_addr, 0, &ip_dst_addr, udp_dst_port);
}
else
{
     if (udp_protocol == udp_ptr->uh_sport)
          udp_dialog_addr_ptr =
               stats_udp_get_dialog (&ip_src_addr, udp_src_port, &ip_dst_addr, 0);
     else
          udp_dialog_addr_ptr = NULL;
}

if (udp_dialog_addr_ptr != NULL)
     udp_dialog_stats_ptr = (StatsDialogEntry *) udp_dialog_addr_ptr->stats_ptr;



/*
 * Set up pointers for segment, address, and dialog statistics used by the common macros
 */
this_seg_addr_ptr       = udp_this_seg_addr_ptr;
this_seg_stats_ptr      = udp_this_seg_stats_ptr;

src_seg_addr_ptr        = udp_src_seg_addr_ptr;
src_seg_stats_ptr       = udp_src_seg_stats_ptr;
```

```
dst_seg_addr_ptr          = udp_dst_seg_addr_ptr;
dst_seg_stats_ptr         = udp_dst_seg_stats_ptr;

src_node_addr_ptr         = udp_src_node_addr_ptr;
src_node_stats_ptr        = udp_src_node_stats_ptr;

dst_node_addr_ptr         = udp_dst_node_addr_ptr;
dst_node_stats_ptr        = udp_dst_node_stats_ptr;

src_socket_addr_ptr       = udp_src_socket_addr_ptr;
src_socket_stats_ptr      = udp_src_socket_stats_ptr;

dst_socket_addr_ptr       = udp_dst_socket_addr_ptr;
dst_socket_stats_ptr      = udp_dst_socket_stats_ptr;

dialog_addr_ptr           = udp_dialog_addr_ptr;
dialog_stats_ptr          = udp_dialog_stats_ptr;

/*
 * Update the age timer in each address structure.
 * Set the 10 second rate sample period in the statistics structures so that the
 * event manager will calculate rates.
 */
stats_update_age_timers;
stats_update_socket_age_timers;
stats_set_rate_10s;
stats_set_socket_rate_10s;


/*
 * Set the last mac address seen
 */
if (udp_src_node_addr_ptr != NULL)
     udp_src_node_addr_ptr->address.macAddress1 = mac_src_addr;
if (udp_dst_node_addr_ptr != NULL)
     udp_dst_node_addr_ptr->address.macAddress1 = mac_dst_addr;
if (udp_src_socket_addr_ptr != NULL)
```

```
    udp_src_socket_addr_ptr->address.macAddress1 = mac_src_addr;
if (udp_dst_socket_addr_ptr != NULL)
    udp_dst_socket_addr_ptr->address.macAddress1 = mac_dst_addr;


/*
 * Do some stats
 */
udp_stats_frames;
udp_stats_bytes;
udp_stats_off_segs;
udp_stats_transits;

if (ip_seg_type != NOT_A_FRAGMENT)
    {
    udp_stats_fragments;
    }


/*
 * Find statistics structures for the protocol distribution statistics
 */
if (udp_src_node_stats_ptr != NULL)
    udp_src_node_protocol_ptr =
        stats_get_protocol (&udp_src_node_stats_ptr->protocolQ, udp_protocol);
if (udp_dst_node_stats_ptr != NULL)
    udp_dst_node_protocol_ptr =
        stats_get_protocol (&udp_dst_node_stats_ptr->protocolQ, udp_protocol);
if (udp_this_seg_stats_ptr != NULL)
    udp_this_seg_protocol_ptr =
        stats_get_protocol (&udp_this_seg_stats_ptr->protocolQ, udp_protocol);
if (udp_src_seg_stats_ptr != NULL)
    udp_src_seg_protocol_ptr  =
        stats_get_protocol (&udp_src_seg_stats_ptr->protocolQ, udp_protocol);
if (udp_dst_seg_stats_ptr != NULL)
    udp_dst_seg_protocol_ptr  =
```

```
        stats_get_protocol (&udp_dst_seg_stats_ptr->protocolQ, udp_protocol);


/*
 * Set up pointers for the protocol distribution statistics used by the common macros
 */

this_seg_protocol_ptr = udp_this_seg_protocol_ptr;
src_seg_protocol_ptr  = udp_src_seg_protocol_ptr;
dst_seg_protocol_ptr  = udp_dst_seg_protocol_ptr;
src_node_protocol_ptr = udp_src_node_protocol_ptr;
dst_node_protocol_ptr = udp_dst_node_protocol_ptr;


/*
 * Keep protocol distribution statistics.
 * Pass the protocol as alarm data in case an alarm occurs.
 */
stats_alarm_data.length = 4;
bcopy (&udp_protocol, stats_alarm_data.data, 4);
udp_stats_protocol;


/*
 * Check the header length
 */
if (length < sizeof(struct udphdr))
     {
     udp_stats_errors;
     goto udp_bad;
     }


/*
 * Parse the next layer
 */
```

```
switch (udp_protocol)
     {
     case MAPPER_PORT:
     case MOUNT_PORT:
     case NFS_PORT:
          result = rtp_rpc_parse (layer_ptr + sizeof(struct udphdr), data_length,
                                  udp_protocol, UDP_PROTOCOL);

          break;

     default:
          break;
     }

return (result);

udp_good:
return (GOOD);

udp_done:
return (DONE);

udp_bad:
return (BAD);
}

/*****************************************************************
 *
 * Process an icmp source quench
 * If it's for mount or nfs, pass it up to be counted.
 */

void rtp_udp_src_quench ()
{
register SrcQuenchData    *srcQuenchData_ptr;
register Uint32                protocol;
```

```
srcQuenchData_ptr = &srcQuenchData;

stats_udp_lookup_ptrs(&srcQuenchData.ip_src_addr, srcQuenchData.src_port,
                      &srcQuenchData.ip_dst_addr, srcQuenchData.dst_port);

udp_stats_flow_ctrls;

protocol = stats_port_to_protocol (ip_src_addr_ptr, srcQuenchData.src_port, UDP_PROTOCOL);

if (protocol == UNKNOWN_PORT)
    protocol = stats_port_to_protocol (ip_dst_addr_ptr, srcQuenchData.dst_port,
UDP_PROTOCOL);

if ( (protocol == MOUNT_PORT) || (protocol == NFS_PORT) )
          {
          rtp_nfs_src_quench ();
          }
}
```

```
/*
 *
 *  rtp.c - Real Time Parse task implementation.
 *
 *      Copyright (c) 1990 Concord Communications Inc.
 *                        All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/rtp/SCCS/s.rtp.c
 *
 *  Date:       3/26/91
 *
 *  Revision:   1.12
 *
 *
 *  Changes:
 *
 *   MM-DD-YY   WHO         Description of change.
 *
 *   03-25-91   KR          changed not to free mbuf if handed off to em
 *   01-28-91   KR          removed UNIX switch
 *   01-21-91   KR          added rtp_broadcast and rtp_multicast
 *   01-18-91   KMJ         removed SENSOR_BOX conditionals, and added
 *                               frame_ptr to rtp local variables (ifdef UNIX)
 *
 *   01-14-91   KR          Set stats_start_time at start of parse
 *   01-07-91   KR          Scheduled EM if frame is from another monitor
 *   10-29-90   KMJ         Put under source control
 *                  KHJ          created
 *
 *
 */

static char rtp_c [] = "@(#)rtp.c  1.10";

#include "cci_std.h"
#include "system.h"
#include <sys/types.h>
#include <bsd43/sys/time.h>
```

```
#include <sys/cci.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#include "lanutil.h"
#include "rtp.h"
#include "stats.h"
#include "ntm_rcv.h"

#define MAX_FRAMES_PER_RTP_TASK_SCHED 10


/*
 * Global Variables
 */

Bool      rtp_for_this_station;
Bool      rtp_from_another_monitor;
Bool      rtp_broadcast;
Bool      rtp_multicast;


/*****************************************************************************
 *
 *   rtp_init ()
 *
 *        Real Time Parser Task initialization routine.
 *        Called from the kernel task upon startup.
 *
 */

Uint32 rtp_init ()

{
  Uint32  a;

  if ( (a = rtp_dll_init() ) == BAD )
    goto end_rtp_init;
```

```
  if ( (a = rtp_arp_init() ) == BAD )
    goto end_rtp_init;
  if ( (a = rtp_ip_init() ) == BAD )
    goto end_rtp_init;
  if ( (a = rtp_icmp_init() ) == BAD )
    goto end_rtp_init;
  if ( (a = rtp_tcp_init() ) == BAD )
    goto end_rtp_init;
  if ( (a = rtp_udp_init() ) == BAD )
    goto end_rtp_init;
  if ( (a = rtp_ftp_init() ) == BAD )
    goto end_rtp_init;
  if ( (a = rtp_telnet_init() ) == BAD )
    goto end_rtp_init;
  if ( (a = rtp_smtp_init() ) == BAD )
    goto end_rtp_init;
  if ( (a = rtp_nfs_init() ) == BAD )
    goto end_rtp_init;

end_rtp_init:
  return (a);
}

/*****************************************************************************
 *
 *        rtp ()
 *
 *        Real Time Parse routine.
 *        Called from the kernel main loop (scheduler) when
 *        there are frames queued on the lan_rcv_frame_q.
 *
 */

void rtp ()

{
  Uint32                a, data_length;
```

```
Uint32                  i;
ITM_Type                *itm;
struct rtp_frame_type *frame_ptr;

for (i=0; i<MAX_FRAMES_PER_RTP_TASK_SCHED; i++)
{
  /*
   * Get the frame to be parsed off the lan_rcv_frame_q
   */
  mbuf_frame_ptr = (struct mbuf *)frame_remqh(lan_rcv_frame_q_ptr);
  if (mbuf_frame_ptr == NULL)
    return;

  rtp_for_this_station = NO;
   rtp_from_another_monitor = NO;
   rtp_broadcast = NO;
   rtp_multicast = NO;

   mon_gettimeofday (&stats_start_time, 0);

   /*
    * Parse the frame
    */
   frame_ptr = (struct rtp_frame_type *) mtod(mbuf_frame_ptr, char *);
   data_length = mbuf_frame_ptr->total_len;
   a = rtp_dll_parse (frame_ptr, data_length);

/* Pass the frame to another task on an ITM or recover its
 * resources via the return_frame routine.
 *
 * NOTE:
 * When tracing is added, the trace task may need to be scheduled
 * and the frame passed to it as well as to the EM or the MTM task.
 * If this occurs, the frame may need to be copied, since it should
 * not appear on two task queues.
 */
```

```
/*
 * If the from is from another monitor, schedule the Event Manager
 * Task and pass the frame to it.  Note that rtp_from_another_monitor
 * and rtp_for_this station should not both be set, since monitor's
 * don't talk to each other.  The check appears below as a safeguard.
 */
    if (rtp_from_another_monitor)
         {
         if (rtp_for_this_station == NO)
              {
              /* Build ITM and queue it to the EM_task */
              /*
               * COMMENT OUT UNTIL AUTO TOPOLOGY IS IMPLEMENTED
               * itm = get_itm();
               * if (itm)
               *    {
               *    itm->type = PARSE_EVENT;
               *    itm->mBufPtr = mbuf_frame_ptr;
               *    send_itm(itm, EM_task);
               *    }
               * else
               *    {
          *    frame_return (mbuf_frame_ptr);
               *    }
               */

               /* The following line comes out when the above goes in */
               frame_return (mbuf_frame_ptr);
              }
         }

/*
 * If the frame is for the monitor, schedule the Message Transfer
 * Module task and pass it the frame.
 */
    if (rtp_for_this_station)
    {
```

```
        /* Build ITM and queue it to the MTM_RCV_task */
        itm = get_itm();
        if (itm)
        {
          itm->type = RCV_LAN_WS_MSG;
          itm->mBufPtr = mbuf_frame_ptr;
          send_itm(itm, MTM_RCV_task);
        }
        else
        {
          dropped_MTM_RCV_frames++;
          frame_return (mbuf_frame_ptr);
        }

      }
       if ( (rtp_for_this_station) || (rtp_from_another_monitor))
            {}
      else
       {
            frame_return (mbuf_frame_ptr);
       }
    }
  }
```

```
/*
 *  stats.h
 *
 *   [description]
 *
 *
 *        Copyright (c) 1990 Concord Communications Inc.
 *                   All rights reserved.
 *
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/include/SCCS/s.stats.h
 *
 *  Date:       8/9/91
 *
 *  Revision:   1.75
 *
 *
 *  Changes:
 *
 *   MM-DD-YY          WHO      Description of change.
 *
 *  07-30-91          DPD      added high to StatsBucketRate
 *  07-16-91          DPD      changed stats_check_for_duplicate_ip to return Uint32
 *   07-22-91         KR         added port mapper stuff, including stats_program and
 *                                    stats_port_to_protocol, and protocol_specific_ptr to
 *                                    StatsAddrEntry
 *
 *   07-16-91         DPD        changed stats_check_for_duplicate_ip to return Uint32
 *   07-08-91         KR         fixed off seg macro yet again
 *   06-18-91         KR         changed off seg macro use seg addr instead of node addr for
checks
 *   06-12-91         KR         added UNKNOWN_SEGMENT_ID
 *   06-10-91         DPD        added alarmsSent to StatsMonitorType
                                    and removed from StatsDialogEntry
 *
 *   06-05-91         DPD        added statsRatesNotDone and bit assignments
 *   06-04-91         KR         fixed input param for stats_deor_count_for_nodes
                                    added statsNfsFileSystemQ
 *
 *   05-31-91         DPD        added EM_RATE_TIME_MIN
```

```
*    05-31-91        KR        removed rpc stuff
*    05-31-91        DPD       added support for nfs
*    05-28-91        DPD       changed EM_RATE_TIME to 60 seconds
*    05-16-91        DPD       added stats_check_for_duplicate_ip import
*    05-15-91        DPD       added imports for non-opaque addr pair routines
*    05-10-91        KR        added StatsRpc and Nfs queues
*    04-29-91        DPD       added imports for statsGetSegMostActive() and
*                                   statsGetAddrMostActive()
*
*    04-26-91        DPD       removed stats mib macros and replaced with routines
*    04-23-91        DPD       added STATS_DLL_NODE and STATS_IP_NODE
*    04-12-91        DPD       added stats_mon_chip_dropped macro
*    03-28-91        KR        removed UNIX conditional around parse control
*    03-26-91        KR        added UNIX conditional around parse control
*    03-20-91        KR        added stats_well_known_port declaration
*                                   added startTime and lastTime to StatsAddrEntry
*                                   removed lastTime and startTime from StatsDialogEntry
*                                   added stats_update_age_timers
*                                   added mySegmentId
*                                   added socket macros
*                                   added statsTcpAddrQ and statsUdpAddrQ
*
*    03-12-91        DPD       removed imports of stats_dll_get_stats and
stats_ip_get_stats
*    02-28-91        KR        added monitor stats
*    02-26-91        KR        added MAX_LAYER_PROTOCOLS
*    02-21-91        KR        added stats_ptr to StatsDialogEntry
*                                   deleted MAX_TCP_HISTORY
*
*    02-20-91        DPD       added default parse control for addresses.
*    02-19-91        DPD       modified getMibRatePerS to support low rates.
*    02-18-91        DPD       put low back in
*    02-15-91        DPD       removed low, low_thld and sample period from all tables.
*    02-07-91        KR        added icmp queues, error codes
*   02-05-91         DPD      moved Dll macros to stats_dll.h and moved dialog, protocol,
*                                   and sorts for "most active" in
*
*    01-31-91        KR        fixed stats_error macro
*    01-29-91        KR        added common macros, added prototyping, deleted unused
constants
*    01-28-91        DPD       fixed bug in GetMacAddrIndexFromOid
```

```
 *    01-26-91        DPD         added import of statsCalcRates
 *    01-25-91        DPD         added GetMacAddrsFromOid macro
 *    01-24-91        DPD         added structures for alarms
 *    01-23-91        KR          completed dialogs
 *    01-21-91        KR          added dialogs
 *                                     removed rate_type and added seconds_start_time to
StatsAddrEntry
 *    01-16-91        DPD         added GetMacAddrIndexFromOid macro
 *    01-10-91        KR          added link to StatsProtocolEntry and StatsDialogEntry and
 *                                     StatsUdpTcpDialogEntry
 *                                     changed StatsProtocolEntry frame and frame_rate types
 *
 *    01-10-91        DPD         added bucket start time and count to StatsBucketRate.
 *                                     and modified "getMib" rate and bucket macros.
 *
 *    01-08-91        DPD         removed alarm related #defines.
 *    01-07-91        KR          added DeviceType and MibAddress in StatsAddrEntry
 *    01-03-91        DPD         added imports of stats rate routines
 *    12-26-90        KR          split stats.h according to protocol type
 *    12-06-90        KMJ         corrected ifdef stats_h usage
 *    10-29-90        KMJ         Put under source control
 *                         KR          initial
 *
 */

#ifndef stats_h
#define stats_h


#include "cci_std.h"
#include "address.h"
#include "kuser.h"
#include "util.h"
#include "protocols.h"
#include "mib_defs.h"
#include "mib_monitor.h"
#include "snmpd.h"
#include "alarms.h"
```

```
#include "bsd43/sys/time.h"
#include "seg_dflts.h"
#include "node_dflts.h"

/*
 * Common Statistics Definitions
 */
#define UNKNOWN_SEGMENT_ID          0


/*
 * Error codes for mon_panic
 */
#define STATS_PTRS_NOT_EQUAL  0



/*
 * OID Definitions
 */
#define SEGMENT                 1
#define ADDRESS                 2

                                5
#define SEG_END
#define ADDR_END            7
#define NFS_SEG_END         6       /* nfs */
#define NFS_ADDR_END        14      /* nfs */

#define STATS_DLL_NODE          1       /* dll node request */
#define STATS_IP_NODE           2       /* ip node request */
#define STATS_NFS_SERVER 3      /* nfs server request */

#define STATS_DLL_LAYER         1       /* dll layer request */
#define STATS_IP_LAYER          2       /* ip layer request */
#define STATS_ICMP_LAYER 3      /* icmp layer request */
#define STATS_TCP_LAYER         4       /* tcp layer request */
#define STATS_UDP_LAYER         5       /* udp layer request */
```

```
#define STATS_NFS_LAYER        6          /* nfs layer request */

#define PARSE_CONTROL_ADDR_DEFAULT 0

#define PARSE_CONTROL_DEFAULT       MibParseDll + MibParseArp + MibParseIp + MibParseUdp +
\                                           MibParseTcp + MibParseFtp + MibParseTelnet +
MibParseSmtp + \                            MibParseNfs + MibParseIcmp


#define MAX_LAYER_PROTOCOLS                 300
#define MAX_SEG                               1
#define MAX_UDP_ADDR                          0
#define MAX_UDP_PAIRS                         0
#define MAX_TCP_SOCKETS                      64
#define MAX_TCP_CONNECTIONS                  64
#define MAX_CONNECT_HISTORIES       10
#define MAX_CONNECT_HISTORY_ENTRIES          10
#define EM_AGE_TIME                         300  /* seconds */
#define EM_RATE_TIME                         60  /* seconds */
#define EM_RATE_TIME_MIN            10   /* seconds */

Import struct timeval     start_time;
Import struct timeval     end_time;

/* end of to be deleted */


Import MibTimeOfDay       mon_startup_time;

Import Uint32             stats_tiny_size;
Import Uint32             stats_very_small_size;
Import Uint32             stats_small_size;
Import Uint32             stats_medium_size;
Import Uint32             stats_large_size;
```

```
Import Uint32           stats_very_large_size;

Import Qhead_type       statsTinyQ;
Import Qhead_type       statsVerySmallQ;
Import Qhead_type       statsSmallQ;
Import Qhead_type       statsMediumQ;
Import Qhead_type       statsLargeQ;
Import Qhead_type       statsVeryLargeQ;


Import Uint32           stats_max_q_count;
Import Uint32           stats_q_count;
Import Uint32           statsRatesNotDone;

#define STATS_DLL_ADDR_RATE         Bit0
#define STATS_DLL_PAIR_RATE         Bit1
#define STATS_IP_ADDR_RATE          Bit2
#define STATS_IP_PAIR_RATE          Bit3
#define STATS_ICMP_ADDR_RATE  Bit4
#define STATS_ICMP_PAIR_RATE  Bit5
#define STATS_TCP_ADDR_RATE         Bit6
#define STATS_TCP_SOCKET_RATE Bit7
#define STATS_TCP_PAIR_RATE         Bit8
#define STATS_UDP_ADDR_RATE         Bit9
#define STATS_UDP_SOCKET_RATE Bit10
#define STATS_UDP_PAIR_RATE         Bit11
#define STATS_NFS_SERVER_RATE Bit12
#define STATS_NFS_CLIENT_RATE Bit13
#define STATS_NFS_PAIR_RATE         Bit14


/*
 * Statistics Constants
 */
#define STATS_BUCKETS_PER_RATE              12
#define STATS_BUCKET_WIDTH                  300       /* seconds */
```

```
#define STATS_PROTOCOLS_PER_DIALOG      10


/*
 * Statistics Types
 */
typedef struct stats_count32_type
     {
     Uint32          count;
     Uint32          running_count;
     Uint32          high_thld;
     } StatsCount32;


typedef struct stats_meter_type
     {
     Uint32          current;
     Uint32          high;
     Uint32          low;
     Uint32          high_thld;
     } StatsMeter;


typedef struct stats_average_meter_type
     {
     Uint32          current;
     Uint32          high;
     Uint32          low;
     Uint32          high_thld;
     } StatsAverageMeter;


typedef struct stats_percent_type
     {
     Uint32          current;
     Uint32          high;
     Uint32          low;
```

```
    Uint32          high_thld;
    } StatsPercent;


typedef struct stats_rate_value
    {
    Uint32          count;
    Uint32          time;
    } StatsRateValue;


typedef struct stats_rolling_rate_type
    {
    StatsRateValue current;
    StatsRateValue high;
    StatsRateValue low;
    Uint32                  high_thld;
    Uint32                  count;
    Uint32                  type;
    } StatsRollingRate;


typedef StatsRollingRate StatsRatePerS;
typedef StatsRollingRate StatsRatePerH;


typedef struct stats_bucket_rate_type
    {
    Uint32                  index;
    Uint32                  rate[STATS_BUCKETS_PER_RATE];       /* 12 5-min count buckets */
    StatsRateValue max_rate[STATS_BUCKETS_PER_RATE];  /* 12 5-min max rate buckets */
    Uint32                  count;                              /* partial count
over the bucket */
    Uint32                  bucket_time;                        /* start time for the
count */
    StatsRateValue  high;                                       /* high rate over the current
bucket */
```

```
    } StatsBucketRate;


/*
 * Address Table Entry
 *
 * Portions of this type are set from the MibParseControl type
 */
typedef struct stats_addr_entry_type {
     FBQentry_type                          link;
     struct stats_addr_entry_type  *hash_link;
     MibAddress                          address;
     Uint32                                flags;
     MibDeviceType                       type;
     Uint32                           parse_control;
     Uint32                                em_control;
     Uint32                          seconds_start_time;
     Uint32                                startTime;
     Uint32                                lastTime;
     Uint32                                *stats_ptr;
     Uint32                                *protocol_specific_ptr;  /* Used by ip for
port mappings */
     } StatsAddrEntry;


/*
 * Values for em_control field of StatsAddrEntry
 */
#define STATS_RATE_10S              Bit0
#define STATS_RATE_60S              Bit1
#define STATS_RATE_HOUR             Bit2


/*
 * Structures used in alarm processing
 */
typedef struct {
```

```
    StatsCount32        value;
} AlarmCount32;

typedef struct {
    StatsRollingRate    value;
} AlarmRollingRate;

typedef struct {
    Uint32              length;
    Byte                data[MAX_ALARM_DATA];
} AlarmUserData;


/*
 * Protocol entry definitions
 */
typedef struct stats_protocol_entry_type
    {
    struct stats_protocol_entry_type   *link;
    Uint32                              protocol;
    StatsCount32                        frames;
    StatsRatePerS                       frameRate;
    } StatsProtocolEntry;


/*
 * Dialog entry definitions
 */

typedef struct stats_transport_type {
    Uint32                  initiator;
    Uint32                  connectionRetries;
    Uint32                  transportProtocol;
    Uint32                  applicationProtocol;
    Uint32                  addr1_window;
    Uint32                  addr2_window;
    Uint32                  state;
```

```
    Uint32                      closeReason;
    Uint32                      *state_ptr;
    } StatsTransportType;


typedef struct stats_dialog_entry_type
    {
    Uint32                          protocolEntries;
    Uint32                          protocols[STATS_PROTOCOLS_PER_DIALOG];
    StatsCount32                packets;
    StatsRatePerS               packetRate;
    StatsCount32                bytes;
    StatsRatePerS               byteRate;
    StatsCount32                errors;
    StatsRatePerS               errorRate;
    StatsCount32                fragments;
    StatsRatePerS               fragmentRate;
    StatsCount32                rexmts;
    StatsRatePerS               rexmtRate;
    StatsCount32                flowCtrls;
    StatsRatePerS               flowCtrlRate;
    StatsTransportType          transport;

    } StatsDialogEntry;


typedef struct dialog_link_type {
    FBQentry_type               link;
    StatsAddrEntry              *dialog_addr_ptr;
    } StatsDialogLink;


/*
 * Monitor Statistics
 */
```

```
typedef struct stats_monitor_type
     {
     StatsCount32          chipDropped;
     StatsRatePerS         chipDroppedRate;
     StatsCount32          dllDropped;
     StatsRatePerS         dllDroppedRate;
     StatsCount32          ipDropped;
     StatsRatePerS         ipDroppedRate;
     StatsCount32          icmpDropped;
     StatsRatePerS         icmpDroppedRate;
     StatsCount32          tcpDropped;
     StatsRatePerS         tcpDroppedRate;
     StatsCount32          udpDropped;
     StatsRatePerS         udpDroppedRate;
     StatsCount32          arpDropped;
     StatsRatePerS         arpDroppedRate;
     StatsCount32          nfsDropped;
     StatsRatePerS         nfsDroppedRate;
     StatsCount32          dbProblem;
     StatsCount32          cpuUtilization;
     StatsCount32          memoryUtilization;
     StatsCount32          discardodAlarms;
     StatsCount32          alarmsSent;
     } StatsMonitorType;


/* These structures are used by the MostActive routines */
typedef struct {
     Uint32                val[MIB_MAX_MOST_ACTIVE];
} Count_;

typedef struct {
     Uint32                val[MIB_MAX_MOST_ACTIVE];
} Time_;

typedef struct {
```

```
    StatsAddrEntry *addr[HIB_MAX_MOST_ACTIVE];
} Addr_;

typedef struct stats_well_known_ports_type {
    Uint32          count;
    Uint32          port[WELL_KNOWN_PORTS_COUNT];
    } StatsWellKnownPorts;

typedef struct stats_programs_entry_type {
    Uint32          program_number;
    Uint32          version;
    Uint32          protocol;
    } StatsProgramEntry;

typedef struct stats_programs_type {
    Uint32                      count;
    StatsProgramEntry   entry[PROGRAM_COUNT];
    } StatsPrograms;

/*
 * Global Routines
 */
Import Uint32                   stats_init ();
Import PQentry_type             stats_allocate (Uint32);
Import void                         stats_deallocate ();
Import StatsProtocolEntry       *stats_lookup_protocol (Qhead_type *, Uint32);
Import StatsProtocolEntry       *stats_get_protocol (Qhead_type *, Uint32);
Import void                         stats_save_protocol_in_dialog (StatsDialogEntry *,
Uint32);
Import Bool                         stats_well_known_port (Uint32);
Import Uint32                   stats_program (Uint32, Uint32);
Import Uint32                   stats_port_to_protocol (StatsAddrEntry *, Uint32, Uint32);

Import void stats_count             (StatsCount32 *, StatsRollingRate *,
                                        StatsAddrEntry *, Uint32, Uint32, Byte *);

Import void stats_decr              (StatsCount32 *, StatsRollingRate *,
                                        StatsAddrEntry *, Uint32, Uint32, Byte *);
```

```
Import void stats_count_bytes (StatsCount32 *, StatsRollingRate *, Uint32,
                               StatsAddrEntry *, Uint32, Uint32, Byte *);


Import void                    statsRates (Uint32);
Import void                         statsCalcRates(StatsRatePerS *, StatsBucketRate *,
                                                   Uint32, StatsAddrEntry *, Uint32,

Uint32,                                            Uint32, AlarmUserData *);

Import void                         statsBubble (Count_ *, Time_ *, Addr_ *, Uint32);
Import void                         statsSort (Uint32, Uint32, StatsAddrEntry *,
                                               Count_ *, Time_ *, Addr_ *, Uint32);

Import VarBind                 *statsGetDialogsO (OID *, StatsAddrEntry *,
                                                   StatsDialogLink *, Uint32,

Uint32);
Import VarBind                 *statsGetProtocolO (OID *, StatsProtocolEntry *,
                                                   Uint32, StatsRatePerS *);

Import VarBind                 *statsGetSegMostActiveO  (OID *, Uint32, FBQhead_type *);
Import VarBind                 *statsGetAddrMostActiveO (OID *, Uint32, FBQhead_type *);

Import VarBind                 *make_vb_from_int (OID *, Uint32);
Import VarBind                 *make_vb_from_rate (OID *, StatsRateValue *);
Import VarBind                 *make_vb_from_rates (OID *, StatsRateValue *, StatsRateValue
*);

Import void                         stats_new_node_alarm (MibAddress *);

Import void                         statsGetMibCount32 (StatsCount32 *, MibCount32 *);
Import void                         statsGetMibRatePerS(StatsRatePerS *, MibRatePerS *);
Import void                         statsGetMibCountRate          (StatsCount32 *,
                                                                   MibCount32 *,
                                                                   StatsRatePerS *,
                                                                   MibRatePerS *);


Import void                         statsGetMibShortCountRate     (StatsCount32 *,
                                                                   MibShortCount32 *,

                                                                   StatsRatePerS *,
```

```
                                                                MibShortRatePerS
*);
Import void             statsGetMibShortCountBucket   (StatsCount32 *,
                                                       MibShortCount32 *,
                                                       StatsBucketRate *,
                                                       MibBucketRate *,
                                                       StatsRatePerS *);

Import VarBind          *statsGetAddrPair     (OID *, StatsAddrEntry *,
                                                 StatsDialogEntry *,
Uint32);
Import Uint32           statsSetAddrPair      (OID *, StatsDialogEntry *,
                                                 ObjectSyntax *, Uint32);
Import VarBind          *statsGetAddrPairThld (OID *, StatsAddrEntry *,
                                                 StatsDialogEntry *,
Uint32);
Import Uint32           statsSetAddrPairThld  (OID *, StatsDialogEntry *,
                                                 ObjectSyntax *, Uint32);

Import Uint32           stats_check_for_duplicate_ip (MibAddress *, MacAddress *);
Import VarBind          *statsGetNodes (OID *, OID *, Uint32, VarEntry *, Uint32);


/*
 * Global Data Structures
 */
Import struct timeval   stats_start_time;
Import Uint32           mySegmentId;
Import StatsAddrEntry   statsMonitorAddr;
Import StatsMonitorType statsMonitor;
```

```
Import FBQhead_type             statsDllSegQ;
Import FBQhead_type             statsIpSegQ;
Import FBQhead_type             statsIcmpSegQ;
Import FBQhead_type             statsUdpSegQ;
Import FBQhead_type             statsTcpSegQ;
Import FBQhead_type             statsNfsSegQ;

Import FBQhead_type             statsMacAddrQ;
Import FBQhead_type             statsIpAddrQ;
Import FBQhead_type             statsIcmpAddrQ;
Import FBQhead_type             statsUdpAddrQ;
Import FBQhead_type             statsTcpAddrQ;
Import FBQhead_type             statsNfsClientQ;

Import FBQhead_type             statsUdpSocketQ;
Import FBQhead_type             statsTcpSocketQ;
Import FBQhead_type             statsNfsServerQ;

Import FBQhead_type             statsDllDialogQ;
Import FBQhead_type             statsIpDialogQ;
Import FBQhead_type             statsIcmpDialogQ;
Import FBQhead_type             statsUdpDialogQ;
Import FBQhead_type             statsTcpDialogQ;
Import FBQhead_type             statsNfsDialogQ;

Import FBQhead_type             statsNfsFileSystemQ;

Import MibAddress               statsDfltAddr;
Import unsigned char            *statsOctetPtr;
Import MibSegmentDefaults       mibSegDefaults;
Import MibNodeDefaults          mibNodeDefaults;

Import VarBind                  *statsVbPtr;

Import struct tm                monitor_gmt;
Import struct timeval           monitor_ticks;
```

```
Import StatsWellKnownPorts      statsWellKnownPorts;
Import StatsPrograms            statsPrograms;

/*
 * Statistics Macros
 */

/* min */
#define min(x, y) \
     x < y ? x : y


/*****************************************************************
 *
 * Common macros for updating statistics
 */


/*
 * Macro to hit the age timer in each address structure
 */
#define stats_update_age_timers \
{\
     if (this_seg_addr_ptr != NULL) \
          this_seg_addr_ptr->lastTime = stats_start_time.tv_sec; \
     if (src_seg_addr_ptr != NULL) \
          src_seg_addr_ptr->lastTime = stats_start_time.tv_sec; \
     if (dst_seg_addr_ptr != NULL) \
          dst_seg_addr_ptr->lastTime = stats_start_time.tv_sec; \
     if (src_node_addr_ptr != NULL) \
          src_node_addr_ptr->lastTime = stats_start_time.tv_sec; \
     if (dst_node_addr_ptr != NULL) \
          dst_node_addr_ptr->lastTime = stats_start_time.tv_sec; \
     if (dialog_addr_ptr != NULL) \
          dialog_addr_ptr->lastTime = stats_start_time.tv_sec; \

}
```

```
#define stats_update_socket_age_timers \
{\
     if (src_socket_addr_ptr != NULL) \
          src_socket_addr_ptr->lastTime = stats_start_time.tv_sec; \
     if (dst_socket_addr_ptr != NULL) \
          dst_socket_addr_ptr->lastTime = stats_start_time.tv_sec; \
}


/*
 * Macro for setting the 10 second sample rate indicator used by the event
 * manager.  On seeing this set, the event manager calculates rates for
 * statistics in the structure.
 */
#define stats_set_rate_10s \
{\
     if ( (src_node_addr_ptr != NULL) && (src_node_stats_ptr != NULL) ) \
          { \
          if ((src_node_addr_ptr->em_control & STATS_RATE_10S) == 0) \
               { \
               src_node_addr_ptr->em_control += STATS_RATE_10S; \
               src_node_addr_ptr->seconds_start_time = stats_start_time.tv_sec; \
               } \
          } \
     if ( (dst_node_addr_ptr != NULL) && (dst_node_stats_ptr != NULL) ) \
          { \
          if ((dst_node_addr_ptr->em_control & STATS_RATE_10S) == 0) \
               { \
               dst_node_addr_ptr->em_control += STATS_RATE_10S; \
               dst_node_addr_ptr->seconds_start_time = stats_start_time.tv_sec; \
               } \
          } \
     if ( (this_seg_addr_ptr != NULL) && (this_seg_stats_ptr != NULL) ) \
          { \
          if ((this_seg_addr_ptr->em_control & STATS_RATE_10S) == 0) \
               { \
```

```
            this_seg_addr_ptr->em_control += STATS_RATE_10S; \
            this_seg_addr_ptr->seconds_start_time = stats_start_time.tv_sec; \
            } \
        } \
    if ( (dst_seg_addr_ptr != NULL) && \
         (dst_seg_stats_ptr != NULL) && \
         (dst_seg_addr_ptr != this_seg_addr_ptr) ) \
        { \
        if ((dst_seg_addr_ptr->em_control & STATS_RATE_10S) == 0) \
            { \
            dst_seg_addr_ptr->em_control += STATS_RATE_10S; \
            dst_seg_addr_ptr->seconds_start_time = stats_start_time.tv_sec; \
            } \
        } \
    if ( (src_seg_addr_ptr != NULL) && \
         (src_seg_stats_ptr != NULL) && \
         (src_seg_addr_ptr != this_seg_addr_ptr) ) \
        { \
        if ((src_seg_addr_ptr->em_control & STATS_RATE_10S) == 0) \
            { \
            src_seg_addr_ptr->em_control += STATS_RATE_10S; \
            src_seg_addr_ptr->seconds_start_time = stats_start_time.tv_sec; \
            } \
        } \
    if ( (dialog_addr_ptr != NULL) && (dialog_stats_ptr != NULL) ) \
        { \
        if ((dialog_addr_ptr->em_control & STATS_RATE_10S) == 0 ) \
            { \
            dialog_addr_ptr->em_control += STATS_RATE_10S; \
            dialog_addr_ptr->seconds_start_time = stats_start_time.tv_sec; \
            } \
        } \
}

#define stats_set_socket_rate_10s \
{\
```

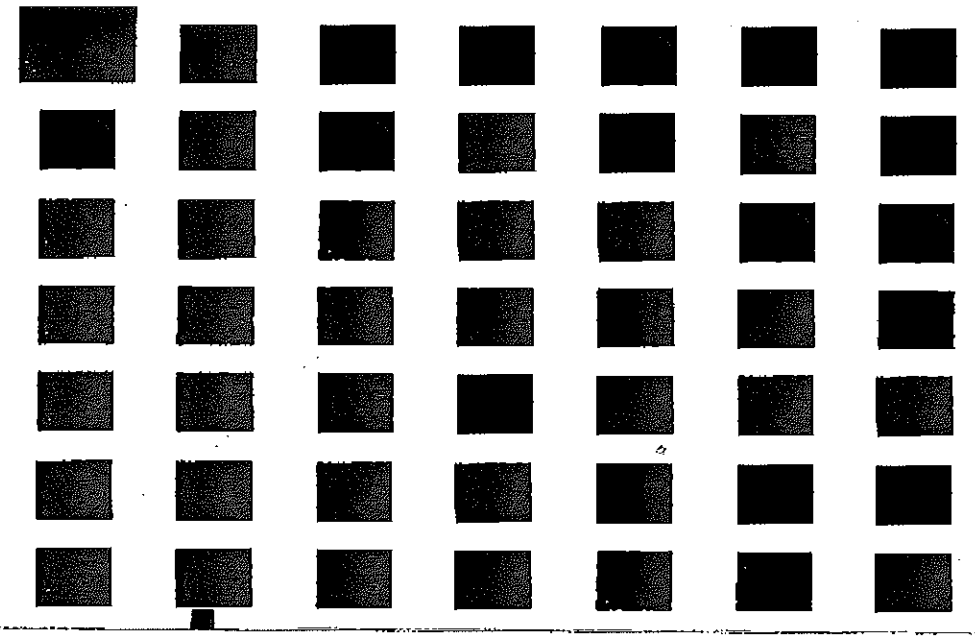
```
    if ( (src_socket_addr_ptr != NULL) && (src_socket_stats_ptr != NULL) ) \
         { \
         if ((src_socket_addr_ptr->em_control & STATS_RATE_10S) == 0) \
              { \
              src_socket_addr_ptr->em_control += STATS_RATE_10S; \
              src_socket_addr_ptr->seconds_start_time = stats_start_time.tv_sec; \
              } \
         } \
     if ( (dst_socket_addr_ptr != NULL) && (dst_socket_stats_ptr != NULL) ) \
         { \
         if ((dst_socket_addr_ptr->em_control & STATS_RATE_10S) == 0) \
              { \
              dst_socket_addr_ptr->em_control += STATS_RATE_10S; \
              dst_socket_addr_ptr->seconds_start_time = stats_start_time.tv_sec; \
              } \
         } \
}


/*
 * Macro to update rcv statistics for a node
 */
#define stats_for_rcv_node(rcvStat, rcvStatRate, NODE, ALARM, ALARM_DATA_PTR) \
{ \
     if (dst_node_stats_ptr != NULL) \
          stats_count (&dst_node_stats_ptr->rcvStat, &dst_node_stats_ptr->rcvStatRate, \
                       dst_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \

}


/*
 * Macro to update xmt statistics for a node
 */
#define stats_for_xmt_node(xmtStat, xmtStatRate, NODE, ALARM, ALARM_DATA_PTR) \
{ \
     if (src_node_stats_ptr != NULL) \
```

```
        stats_count (&src_node_stats_ptr->xmtStat, &src_node_stats_ptr->xmtStatRate, \
                     src_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \
}

/*
 * Macro to update rcv byte statistics for a node
 */
#define stats_rcv_bytes_for_node(byte_count, rcvStat, rcvStatRate, NODE, ALARM,
ALARM_DATA_PTR) \
{ \
     if (dst_node_stats_ptr != NULL) \
          stats_count_bytes (&dst_node_stats_ptr->rcvStat, \
                             &dst_node_stats_ptr->rcvStatRate, byte_count,
dst_node_addr_ptr, \
                             NODE, ALARM, ALARM_DATA_PTR); \
}

/*
 * Macro to update xmt byte statistics for a node
 */
#define stats_xmt_bytes_for_node(byte_count, xmtStat, xmtStatRate, NODE, ALARM,
ALARM_DATA_PTR) \
{ \
     if (src_node_stats_ptr != NULL) \
          stats_count_bytes (&src_node_stats_ptr->xmtStat, \
                             &src_node_stats_ptr->xmtStatRate, byte_count,
src_node_addr_ptr, \
                             NODE, ALARM, ALARM_DATA_PTR); \
}

/*
 * Macro to update the same statistic for both nodes
 */
#define stats_for_nodes(stat, statRate, NODE, ALARM, ALARM_DATA_PTR) \
```

```
{ \
     if (src_node_stats_ptr != NULL) \
          stats_count (&src_node_stats_ptr->stat, &src_node_stats_ptr->statRate, \
                       src_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \
     if (dst_node_stats_ptr != NULL) \
          stats_count (&dst_node_stats_ptr->stat, &dst_node_stats_ptr->statRate, \
                       dst_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \

}


/*
 * Macro to increment a node statistic with no rate
 */
#define stats_count_for_nodes(stat, NODE, ALARM, ALARM_DATA_PTR) \
{ \
     if (src_node_stats_ptr != NULL) \
          stats_count (&src_node_stats_ptr->stat, NULL, \
                       src_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \
     if (dst_node_stats_ptr != NULL) \
          stats_count (&dst_node_stats_ptr->stat, NULL, \
                       dst_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \

}


/*
 * Macro to decrement a node statistic with no rate
 */
#define stats_decr_count_for_nodes(stat, NODE, ALARM, ALARM_DATA_PTR) \
{ \
     if (src_node_stats_ptr != NULL) \
          stats_decr (&src_node_stats_ptr->stat, NULL, \
                      src_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \
     if (dst_node_stats_ptr != NULL) \
          stats_decr (&dst_node_stats_ptr->stat, NULL, \
                      dst_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \

}
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerard White

05



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
/*
 * Macro to update the same byte statistic for both nodes
 */
#define stats_bytes_for_nodes(byte_count, stat, statRate, NODE, ALARM, ALARM_DATA_PTR) \
{ \
     if (src_node_stats_ptr != NULL) \
          stats_count_bytes (&src_node_stats_ptr->stat, &src_node_stats_ptr->statRate,
byte_count,\
                             src_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \
     if (dst_node_stats_ptr != NULL) \
          stats_count_bytes(&dst_node_stats_ptr->stat, &dst_node_stats_ptr->statRate,
byte_count, \
                             dst_node_addr_ptr, NODE, ALARM, ALARM_DATA_PTR); \
}



/*
 * Macro to update rcv statistics for a socket
 */
#define stats_for_rcv_socket(rcvStat, rcvStatRate, SOCKET, ALARM, ALARM_DATA_PTR) \
{ \
     if (dst_socket_stats_ptr != NULL) \
          stats_count (&dst_socket_stats_ptr->rcvStat, &dst_socket_stats_ptr->rcvStatRate,
\
                       dst_socket_addr_ptr, SOCKET, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to update xmt statistics for a socket
 */
#define stats_for_xmt_socket(xmtStat, xmtStatRate, SOCKET, ALARM, ALARM_DATA_PTR) \
{ \
     if (src_socket_stats_ptr != NULL) \
```

```
        stats_count (&src_socket_stats_ptr->xmtStat, &src_socket_stats_ptr->xmtStatRate,
\
                        src_socket_addr_ptr, SOCKET, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to update rcv byte statistics for a socket
 */
#define
stats_rcv_bytes_for_socket(byte_count,rcvStat,rcvStatRate,SOCKET,ALARM,ALARM_DATA_PTR) \
{ \
     if (dst_socket_stats_ptr != NULL) \
          stats_count_bytes (&dst_socket_stats_ptr->rcvStat, \
                             &dst_socket_stats_ptr->rcvStatRate, byte_count,
dst_socket_addr_ptr, \
                             SOCKET, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to update xmt byte statistics for a socket
 */
#define
stats_xmt_bytes_for_socket(byte_count,xmtStat,xmtStatRate,SOCKET,ALARM,ALARM_DATA_PTR) \
{ \
     if (src_socket_stats_ptr != NULL) \
          stats_count_bytes (&src_socket_stats_ptr->xmtStat, \
                             &src_socket_stats_ptr->xmtStatRate, byte_count,
src_socket_addr_ptr, \
                             SOCKET, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to update the same statistic for both sockets
 */
```

```
#define stats_for_sockets(stat, statRate, SOCKET, ALARM, ALARM_DATA_PTR) \
{ \
     if (src_socket_stats_ptr != NULL) \
          stats_count (&src_socket_stats_ptr->stat, &src_socket_stats_ptr->statRate, \
                       src_socket_addr_ptr, SOCKET, ALARM, ALARM_DATA_PTR); \
     if (dst_socket_stats_ptr != NULL) \
          stats_count (&dst_socket_stats_ptr->stat, &dst_socket_stats_ptr->statRate, \
                       dst_socket_addr_ptr, SOCKET, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to update the same byte statistic for both sockets
 */
#define stats_bytes_for_sockets(byte_count, stat, statRate, SOCKET, ALARM, ALARM_DATA_PTR)
\
{ \
     if (src_socket_stats_ptr != NULL) \

stats_count_bytes(&src_socket_stats_ptr->stat,&src_socket_stats_ptr->statRate,byte_count,\

                               src_socket_addr_ptr, SOCKET, ALARM, ALARM_DATA_PTR); \
     if (dst_socket_stats_ptr != NULL) \

stats_count_bytes(&dst_socket_stats_ptr->stat,&dst_socket_stats_ptr->statRate,byte_count,\

                               dst_socket_addr_ptr, SOCKET, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to update statistic for segments
 */
#define stats_for_segs(stat, statRate, SEGMENT, ALARM, ALARM_DATA_PTR) \
{ \
     if (this_seg_stats_ptr != NULL) \
```

```
        stats_count (&this_seg_stats_ptr->stat, &this_seg_stats_ptr->statRate, \
                     this_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
\
    if ( (src_seg_stats_ptr != NULL) && \
         (src_seg_addr_ptr != this_seg_addr_ptr) ) \
        stats_count (&src_seg_stats_ptr->stat, &src_seg_stats_ptr->statRate, \
                     src_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
\
    if ( (dst_seg_stats_ptr != NULL) && \
         (dst_seg_addr_ptr != this_seg_addr_ptr) ) \
        stats_count (&dst_seg_stats_ptr->stat, &dst_seg_stats_ptr->statRate, \
                     dst_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to decrement a segment statistic
 */
#define stats_decr_for_segs(stat, statRate, SEGMENT, ALARM, ALARM_DATA_PTR) \
{ \
    if (this_seg_stats_ptr != NULL) \
        stats_decr  (&this_seg_stats_ptr->stat, &this_seg_stats_ptr->statRate, \
                     this_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
\
    if ( (src_seg_stats_ptr != NULL) && \
         (src_seg_addr_ptr != this_seg_addr_ptr) ) \
        stats_decr  (&src_seg_stats_ptr->stat, &src_seg_stats_ptr->statRate, \
                     src_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
\
    if ( (dst_seg_stats_ptr != NULL) && \
         (dst_seg_addr_ptr != this_seg_addr_ptr) ) \
        stats_decr  (&dst_seg_stats_ptr->stat, &dst_seg_stats_ptr->statRate, \
                     dst_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
}


/*
```

```
 * Macro to increment a segment statistic that doesn't have a rate
 */
#define stats_count_for_segs(stat, SEGMENT, ALARM, ALARM_DATA_PTR) \
{ \
     if (this_seg_stats_ptr != NULL) \
          stats_count (&this_seg_stats_ptr->stat, NULL, \
                       this_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
\
     if ( (src_seg_stats_ptr != NULL) && \
          (src_seg_addr_ptr != this_seg_addr_ptr) ) \
          stats_count (&src_seg_stats_ptr->stat, NULL, \
                       src_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
\
     if ( (dst_seg_stats_ptr != NULL) && \
          (dst_seg_addr_ptr != this_seg_addr_ptr) ) \
          stats_count (&dst_seg_stats_ptr->stat, NULL, \
                       dst_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to decrement a segment statistic that doesn't have a rate
 */
#define stats_decr_count_for_segs(stat, SEGMENT, ALARM, ALARM_DATA_PTR) \
{ \
     if (this_seg_stats_ptr != NULL) \
          stats_decr  (&this_seg_stats_ptr->stat, NULL, \
                       this_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
\
     if ( (src_seg_stats_ptr != NULL) && \
          (src_seg_addr_ptr != this_seg_addr_ptr) ) \
          stats_decr  (&src_seg_stats_ptr->stat, NULL, \
                       src_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
\
     if ( (dst_seg_stats_ptr != NULL) && \
          (dst_seg_addr_ptr != this_seg_addr_ptr) ) \
          stats_decr  (&dst_seg_stats_ptr->stat, NULL, \
```

```
                        dst_seg_addr_ptr, SEGMENT, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to update byte statistic for segments
 */
#define stats_bytes_for_segs(byte_count, stat, statRate, SEGMENT, ALARM, ALARM_DATA_PTR) \

{ \
     if (this_seg_stats_ptr != NULL) { \
               stats_count_bytes(&this_seg_stats_ptr->stat, \
                                 &this_seg_stats_ptr->statRate, byte_count,
this_seg_addr_ptr,\
                                 SEGMENT, ALARM, ALARM_DATA_PTR); \
     } \
     if ( (src_seg_stats_ptr != NULL) && \
          (src_seg_addr_ptr != this_seg_addr_ptr) ) { \
              stats_count_bytes(&src_seg_stats_ptr->stat, \
                                 &src_seg_stats_ptr->statRate, byte_count,
src_seg_addr_ptr, \
                                 SEGMENT, ALARM, ALARM_DATA_PTR); \
     } \
     if ( (dst_seg_stats_ptr != NULL) && \
          (dst_seg_addr_ptr != this_seg_addr_ptr) ) { \
              stats_count_bytes(&dst_seg_stats_ptr->stat, \
                                 &dst_seg_stats_ptr->statRate, byte_count,
dst_seg_addr_ptr, \
                                 SEGMENT, ALARM, ALARM_DATA_PTR); \
     } \
}


/*
 * Macro to update statistic for a dialog
 */
#define stats_for_dialog(stat, statRate, PAIR, ALARM, ALARM_DATA_PTR) \
```

```
{ \
     if (dialog_stats_ptr != NULL) \
               stats_count (&dialog_stats_ptr->stat, &dialog_stats_ptr->statRate, \
                              dialog_addr_ptr, PAIR, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to update byte statistic for a dialog
 */
#define stats_bytes_for_dialog(byte_count, stat, statRate, PAIR, ALARM, ALARM_DATA_PTR) \

{ \
     if (dialog_stats_ptr != NULL) \
               stats_count_bytes(&dialog_stats_ptr->stat, \
                                 &dialog_stats_ptr->statRate, byte_count,
dialog_addr_ptr, \
                                 PAIR, ALARM, ALARM_DATA_PTR); \
}


/*
 * Macro to update frame statistics
 */
#define stats_frames(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     stats_for_segs     (frames, frameRate, SEGMENT, AL_FRAMES, 0); \
     stats_for_nodes    (frames, frameRate, NODE, AL_FRAMES, 0); \
     stats_for_rcv_node (rcvFrames, rcvFrameRate, NODE, AL_RCV_FRAMES, 0); \
     stats_for_xmt_node (xmtFrames, xmtFrameRate, NODE, AL_XMT_FRAMES, 0); \
     stats_for_dialog   (packets, packetRate, PAIR, AL_FRAMES, 0); \
}


/*
 * Macro to update byte statistics
```

```
 */
#define stats_bytes(byte_count, SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     stats_bytes_for_segs      (byte_count, bytes, byteRate, SEGMENT, AL_BYTES, 0); \
     stats_bytes_for_nodes     (byte_count, bytes, byteRate, NODE, AL_BYTES, 0); \
     stats_rcv_bytes_for_node (byte_count, rcvBytes, rcvByteRate, NODE, AL_RCV_BYTES, 0);
\
     stats_xmt_bytes_for_node (byte_count, xmtBytes, xmtByteRate, NODE, AL_XMT_BYTES, 0);
\
     stats_bytes_for_dialog    (byte_count, bytes, byteRate, PAIR, AL_BYTES, 0); \
}


/*
 * Macro to update hdr byte statistics
 */
#define stats_hdr_bytes(byte_count, SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     stats_bytes_for_segs      (byte_count, hdrBytes, hdrByteRate, SEGMENT, AL_HDR_BYTES,
0); \
     stats_rcv_bytes_for_node (byte_count,
rcvHdrBytes,rcvHdrByteRate,NODE,AL_RCV_HDR_BYTES,0); \
     stats_xmt_bytes_for_node (byte_count,
xmtHdrBytes,xmtHdrByteRate,NODE,AL_XMT_HDR_BYTES,0); \
     }


/*
 * Macro to count errors
 */
#define stats_errors(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     stats_for_segs      (errors, errorRate, SEGMENT, AL_ERRORS, 0); \
     stats_for_nodes     (errors, errorRate, NODE, AL_ERRORS, 0); \
     stats_for_rcv_node (rcvErrors, rcvErrorRate, NODE, AL_RCV_ERRORS, 0); \
     stats_for_xmt_node (xmtErrors, xmtErrorRate, NODE, AL_XMT_ERRORS, 0); \
     stats_for_dialog    (errors, errorRate, PAIR, AL_ERRORS, 0); \
```

```
}


/*
 * Macros to count off segment statistics
 */
#define stats_off_segs(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     if ( (src_seg_addr_ptr != NULL) && (dst_seg_addr_ptr != NULL) ) { \
          if ( src_seg_addr_ptr->address.segment1 != dst_seg_addr_ptr->address.segment1) {
\
               stats_for_rcv_node (rcvOffSegs, rcvOffSegRate, NODE, AL_RCV_OFF_SEG, 0); \

               stats_for_xmt_node (xmtOffSegs, xmtOffSegRate, NODE, AL_XMT_OFF_SEG, 0); \

               if (src_seg_stats_ptr != NULL) \
                    stats_count (&src_seg_stats_ptr->xmtOffSegs, \
                                 &src_seg_stats_ptr->xmtOffSegRate, \
                                 src_seg_addr_ptr, SEGMENT, AL_XMT_OFF_SEG, 0); \
               if (dst_seg_stats_ptr != NULL) \
                    stats_count (&dst_seg_stats_ptr->rcvOffSegs, \
                                 &dst_seg_stats_ptr->rcvOffSegRate, \
                                 dst_seg_addr_ptr, SEGMENT, AL_RCV_OFF_SEG, 0); \

          } \
     } \
}

#define stats_socket_off_segs(SOCKET) \
{ \
     if ( (src_seg_addr_ptr != NULL) && (dst_seg_addr_ptr != NULL) ) { \
          if ( src_seg_addr_ptr->address.segment1 != dst_seg_addr_ptr->address.segment1) {
\
               stats_for_rcv_socket (rcvOffSegs, rcvOffSegRate, SOCKET, AL_RCV_OFF_SEG,
0); \
               stats_for_xmt_socket (xmtOffSegs, xmtOffSegRate, SOCKET, AL_XMT_OFF_SEG,
0); \
```

```
        } \
    } \
}


/*
 * Macro to count transits
 */
#define stats_transits(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
    if ( (this_seg_addr_ptr != NULL) && \
         (src_seg_addr_ptr != NULL) && \
         (dst_seg_addr_ptr != NULL) ) { \
        if ((this_seg_addr_ptr->address.segment1 != \
             src_seg_addr_ptr->address.segment1) && \
            (this_seg_addr_ptr->address.segment1 != \
             dst_seg_addr_ptr->address.segment1) ) { \
                if (this_seg_stats_ptr != NULL) { \
                    stats_count (&this_seg_stats_ptr->transits, \
                                 &this_seg_stats_ptr->transitRate, \
                                 this_seg_addr_ptr, SEGMENT, AL_TRANSIT, 0); \
                } \
            } \
        } \
}


/*
 * Macro to count broadcasts
 */
#define stats_bcasts(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
    stats_for_segs     (bcasts, bcastRate, SEGMENT, AL_BCAST, 0); \
    stats_for_xmt_node (xmtBcasts, xmtBcastRate, NODE, AL_BCAST, 0); \
}
```

```
/*
 * Macro to count multicasts
 */
#define stats_mcasts(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     stats_for_segs     (mcasts, mcastRate, SEGMENT, AL_MCAST, 0); \
     stats_for_xmt_node (xmtMcasts, xmtMcastRate, NODE, AL_MCAST, 0); \
}


/*
 * Macro to count collisions
 */
#define stats_collisions(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     if (this_seg_stats_ptr != NULL) \
          stats_count (&this_seg_stats_ptr->collisions,
&this_seg_stats_ptr->collisionRate, \
                       this_seg_addr_ptr, SEGMENT, AL_COLLISION, 0); \
}


/*
 * Macro to count alignment errors
 */
#define stats_alignment(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     if (this_seg_stats_ptr != NULL) \
          stats_count (&this_seg_stats_ptr->alignmentErrors, \
                       &this_seg_stats_ptr->alignmentErrorRate, this_seg_addr_ptr, \
                       SEGMENT, AL_ALIGNMENT, 0); \
}


/*
 * Macro to count ethernet frames
 */
```

```
#define stats_enet(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     stats_for_nodes (enetFrames, enetFrameRate, NODE, AL_ENET_FRAMES, 0); \
     stats_for_segs (enetFrames, enetFrameRate, SEGMENT, AL_ENET_FRAMES, 0); \
}


/*
 * Macro to count 802.2 frames
 */
#define stats_llc(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     stats_for_nodes (llcFrames, llcFrameRate, NODE, AL_LLC_FRAMES, 0); \
     stats_for_segs (llcFrames, llcFrameRate, SEGMENT, AL_LLC_FRAMES, 0); \
}


/*
 * Macro to count runt frames
 */
#define stats_runts(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     stats_for_nodes (runtFrames, runtFrameRate, NODE, AL_RUNT_FRAMES, 0); \
     stats_for_segs (runtFrames, runtFrameRate, SEGMENT, AL_RUNT_FRAMES, 0); \
}


/*
 * Macro to count fragments
 */
#define stats_fragments(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     stats_for_segs     (frgnts, frgmtRate, SEGMENT, AL_FRAGMENTS, 0); \
     stats_for_nodes    (frgnts, frgmtRate, NODE, AL_FRAGMENTS, 0); \
     stats_for_rcv_node (rcvFrgmts, rcvFrgmtRate, NODE, AL_RCV_FRAGMENTS, 0); \
     stats_for_xmt_node (xmtFrgmts, xmtFrgmtRate, NODE, AL_XMT_FRAGMENTS, 0); \
     stats_for_dialog   (fragments, fragmentRate, PAIR, AL_FRAGMENTS, 0); \
```

```
}


/*
 * Macro to count protocol distribution statistics
 */
#define stats_protocol(SEGMENT, NODE, PAIR, PROTOCOL) \
{ \
     if (src_node_protocol_ptr != NULL) { \
          stats_count (&src_node_protocol_ptr->frames, &src_node_protocol_ptr->frameRate,
\
                       src_node_addr_ptr, PROTOCOL, AL_FRAMES, (Byte
*)&stats_alarm_data); \
          } \
     if (dst_node_protocol_ptr != NULL) { \
          stats_count (&dst_node_protocol_ptr->frames, &dst_node_protocol_ptr->frameRate,
\
                       dst_node_addr_ptr, PROTOCOL, AL_FRAMES, (Byte
*)&stats_alarm_data); \
          } \
     if ( (src_seg_protocol_ptr != NULL) && \
           (src_seg_protocol_ptr != this_seg_protocol_ptr) ) { \
          stats_count (&src_seg_protocol_ptr->frames, &src_seg_protocol_ptr->frameRate,\
                       src_seg_addr_ptr, PROTOCOL, AL_FRAMES, (Byte
*)&stats_alarm_data); \
          } \
     if (this_seg_protocol_ptr != NULL) { \
          stats_count(&this_seg_protocol_ptr->frames,&this_seg_protocol_ptr->frameRate,\
                       this_seg_addr_ptr, PROTOCOL, AL_FRAMES, (Byte
*)&stats_alarm_data); \
          } \
     if ( (dst_seg_protocol_ptr != NULL) && \
           (dst_seg_protocol_ptr != this_seg_protocol_ptr) ) { \
          stats_count (&dst_seg_protocol_ptr->frames, &dst_seg_protocol_ptr->frameRate,\
                       dst_seg_addr_ptr, PROTOCOL, AL_FRAMES, (Byte
*)&stats_alarm_data); \
          } \
```

```
    stats_save_protocol_in_dia    (dialog_stats_ptr, protocol); \
}


/*
 * Macros for Monitor Statistics
 */

#define stats_mon_dll_dropped \
    stats_count (&statsMonitor.dllDropped, 0, &statsMonitorAddr, \
                 MONITOR_ALARM, AL_DLL_DROPPED, 0)

#define stats_mon_ip_dropped \
    stats_count (&statsMonitor.ipDropped, 0, &statsMonitorAddr, \
                 MONITOR_ALARM, AL_IP_DROPPED, 0)

#define stats_mon_icmp_dropped \
    stats_count (&statsMonitor.icmpDropped, 0, &statsMonitorAddr, \
                 MONITOR_ALARM, AL_ICMP_DROPPED, 0)

#define stats_mon_tcp_dropped \
    stats_count (&statsMonitor.tcpDropped, 0, &statsMonitorAddr, \
                 MONITOR_ALARM, AL_TCP_DROPPED, 0)

#define stats_mon_udp_dropped \
    stats_count (&statsMonitor.udpDropped, 0, &statsMonitorAddr, \
                 MONITOR_ALARM, AL_UDP_DROPPED, 0)

#define stats_mon_arp_dropped \
    stats_count (&statsMonitor.arpDropped, 0, &statsMonitorAddr, \
                 MONITOR_ALARM, AL_ARP_DROPPED, 0)

#define stats_mon_nfs_dropped \
    stats_count (&statsMonitor.nfsDropped, 0, &statsMonitorAddr, \
                 MONITOR_ALARM, AL_NFS_DROPPED, 0)
```

```
#define stats_mon_db_problem \
     stats_count (&statsMonitor.dbProblem, 0, &statsMonitorAddr, \
                  MONITOR_ALARM, AL_DB_PROBLEM, 0)

#define stats_mon_cpu_utilization \
     stats_count (&statsMonitor.cpuUtilization, 0, &statsMonitorAddr, \
                  MONITOR_ALARM, AL_CPU_UTILIZATION, 0)

#define stats_mon_discarded_alarms \
     stats_count (&statsMonitor.discardedAlarms, 0, &statsMonitorAddr, \
                  MONITOR_ALARM, AL_DISCARDED_ALARMS, 0)

#define stats_mon_cpu_utilization \
     stats_count (&statsMonitor.cpuUtilization, 0, &statsMonitorAddr, \
                  MONITOR_ALARM, AL_CPU_UTILIZATION, 0)

#define stats_mon_chip_dropped \
     stats_count_bytes (&statsMonitor.chipDropped, &statsMonitor.chipDroppedRate, \
                  driver_counts->dropped_packets, &statsMonitorAddr, \
                  MONITOR_ALARM, AL_CHIP_DROPPED, 0)

#endif /* stats_h */
```

```
/*
 *  stats_dll.h - [description]
 *
 *
 *        Copyright (c) 1990 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/include/SCCS/s.stats_dll.h
 *
 *  Date:       6/5/91
 *
 *  Revision:   1.30
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO        Description of change, (latest first)
 *   --------  ---        ---------------------------------------------
 *
 *   04-09-91  DPD        added imports for thld gets/sets
 *   03-16-91  KR         added statsDllAgeAddr
 *   03-12-91  DPD        added import of stats_dll_get_stats
 *   02-28-91  DPD        general clean-up.
 *   02-25-91  DPD        changed import of statsDllDialogRate.
 *   02-05-91  DPD        moved Dll macros from stats.h
 *   01-29-91  KR         moved common macros to stats.h, deleted unused constants
 *  01-26-91    DPD      added import of statsDllDialogRate
 *   01-25-91  KR         added on seg, off seg, and transit changes
 *   01-24-91  KR         added runtFrames, enetFrames, and lloFranes
 *  01-24-91    DPD      dll_count_protocol now passes protocol in user data
 *   01-21-91  KR         added dialogs
 *   01-18-91  KR         changed field names to match those in mib include files
 *                             changed macros to set em_control and time
 *   01-11-91  KR         added protocol table support
 *   01-08-91  DPD        removed #defines related to alarms to alarms.h
 *   01-07-91  KR         removed StatsMacAddrEntry
```

```
 *   01-03-91  DPD       added imports of dll rate routines
 *                  KR         initial
 *
 */

#ifndef stats_dll_h
#define stats_dll_h


#include <cci_std.h>
#include "address.h"
#include "kuser.h"
#include "snmpd.h"
#include "alarms.h"
#include "stats.h"



#define MAX_MAC_ADDR                    64
#define MAX_DLL_PAIRS                   64
#define MAC_HASH_TABLE_SIZE             16
#define DLL_DIALOG_HASH_TABLE_SIZE 16



/*
 * Dll Statistics
 */


/*
 * Segment  Statistics
 */
typedef struct stats_dll_segment_type
     {
     StatsCount32        frames;
     StatsRatePerS       frameRate;
```

```
StatsBucketRate         frameBuckets;

StatsCount32        bytes;
StatsRatePerS       byteRate;
StatsBucketRate         byteBuckets;

StatsCount32        errors;
StatsRatePerS       errorRate;
StatsBucketRate         errorBuckets;

StatsCount32        rcvOffSegs;
StatsRatePerS       rcvOffSegRate;
StatsBucketRate         rcvOffSegBuckets;

StatsCount32        xmtOffSegs;
StatsRatePerS       xmtOffSegRate;
StatsBucketRate         xmtOffSegBuckets;

StatsCount32        transits;
StatsRatePerS       transitRate;
StatsBucketRate         transitBuckets;

StatsCount32        bcasts;
StatsRatePerS       bcastRate;
StatsBucketRate         bcastBuckets;

StatsCount32        mcasts;
StatsRatePerS       mcastRate;
StatsBucketRate         mcastBuckets;

StatsCount32        collisions;
StatsRatePerS       collisionRate;

StatsCount32        alignmentErrors;
StatsRatePerS       alignmentErrorRate;

StatsCount32        enetFrames;
```

```
        StatsRatePerS       enetFrameRate;

        StatsCount32        llcFrames;
        StatsRatePerS       llcFrameRate;

        StatsCount32        runtFrames;
        StatsRatePerS       runtFrameRate;

        Qhead_type              protocolQ;

        } StatsDllSegment;


/*
 * Address  Statistics
 */
typedef struct stats_dll_addr_type
        {
        StatsCount32                frames;
        StatsRatePerS               frameRate;
        StatsBucketRate                 frameBuckets;
        StatsCount32                rcvFrames;
        StatsRatePerS               rcvFrameRate;
        StatsCount32                xmtFrames;
        StatsRatePerS               xmtFrameRate;

        StatsCount32                bytes;
        StatsRatePerS               byteRate;
        StatsBucketRate                 byteBuckets;
        StatsCount32                rcvBytes;
        StatsRatePerS               rcvByteRate;
        StatsCount32                xmtBytes;
        StatsRatePerS               xmtByteRate;

        StatsCount32                errors;
        StatsRatePerS               errorRate;
        StatsBucketRate                 errorBuckets;
```

```
StatsCount32            rcvErrors;
StatsRatePerS           rcvErrorRate;
StatsCount32            xmtErrors;
StatsRatePerS           xmtErrorRate;

StatsCount32            rcvOffSegs;
StatsRatePerS           rcvOffSegRate;
StatsBucketRate             rcvOffSegBuckets;

StatsCount32            xmtOffSegs;
StatsRatePerS           xmtOffSegRate;
StatsBucketRate             xmtOffSegBuckets;

StatsCount32            xmtBcasts;
StatsRatePerS           xmtBcastRate;
StatsBucketRate             xmtBcastBuckets;

StatsCount32            xmtMcasts;
StatsRatePerS           xmtMcastRate;
StatsBucketRate             xmtMcastBuckets;

StatsCount32            enetFrames;
StatsRatePerS           enetFrameRate;

StatsCount32            llcFrames;
StatsRatePerS           llcFrameRate;

StatsCount32            runtFrames;
StatsRatePerS           runtFrameRate;

Qhead_type                  protocolQ;
FBQhead_type            dialogQ;        /* Queue of StatsDialogLink types */

} StatsDllAddr;
```

```
/*
 * Global Data Structures
 */
Import StatsAddrEntry          *dll_this_seg_addr_ptr;
Import StatsDllSegment         *dll_this_seg_stats_ptr;
Import StatsProtocolEntry      *dll_this_seg_protocol_ptr;

Import StatsAddrEntry          *dll_src_seg_addr_ptr, *dll_dst_seg_addr_ptr;
Import StatsDllSegment         *dll_src_seg_stats_ptr, *dll_dst_seg_stats_ptr;
Import StatsProtocolEntry      *dll_src_seg_protocol_ptr, *dll_dst_seg_protocol_ptr;

Import StatsAddrEntry          *dll_src_node_addr_ptr, *dll_dst_node_addr_ptr;
Import StatsDllAddr            *dll_src_node_stats_ptr, *dll_dst_node_stats_ptr;

Import StatsAddrEntry          *dll_dialog_addr_ptr;
Import StatsDialogEntry        *dll_dialog_stats_ptr;

Import StatsProtocolEntry      *dll_src_node_protocol_ptr, *dll_dst_node_protocol_ptr;


Import StatsAddrEntry          *mac_hash_table[MAC_HASH_TABLE_SIZE];
Import StatsAddrEntry          *dialog_hash_table[DLL_DIALOG_HASH_TABLE_SIZE];

FBQentry_type                  *statsNextDllAddrEntry;
FBQentry_type                  *statsNextDllPairEntry;

Import VarBind                 *(*statsGetDllSeg[5]) ();
Import VarBind                 *(*statsGetDllSegO[5]) ();
Import VarBind                 *(*statsGetDllAddr[7]) ();
Import VarBind                 *(*statsGetDllAddrO[7]) ();

Import Uint32                  (*statsSetDllSeg[5]) ();
Import Uint32                  (*statsSetDllAddr[7]) ();

/*
 * Local data structures
```

```
 */
Import Uint32                   mac_hash;
Import StatsAddrEntry           *mac_hash_link;
Import StatsAddrEntry           *previous_mac_hash_link;
Import Uint32                   dialog_hash;
Import StatsAddrEntry           *dialog_hash_link;
Import StatsAddrEntry           *previous_dialog_hash_link;


/*
 * Global Routines
 */
Import Uint32               stats_dll_init ();
Import StatsAddrEntry       *stats_dll_get_segment    (Uint32);
Import StatsAddrEntry       *stats_dll_lookup_segment (Uint32);
Import StatsAddrEntry       *stats_dll_allocate_addr (MacAddress*);
Import StatsAddrEntry       *stats_dll_get_addr      (MacAddress *);
Import StatsAddrEntry       *stats_dll_lookup_addr   (MacAddress *);
Import StatsProtocolEntry *stats_dll_get_protocol       (StatsDllAddr *, Uint32);
Import StatsProtocolEntry *stats_dll_lookup_protocol (StatsDllAddr *, Uint32);
Import StatsAddrEntry       *stats_dll_get_dialog    (MacAddress *, MacAddress *);
Import StatsAddrEntry      *stats_dll_lookup_dialog (MacAddress *, MacAddress *);

Import StatsAddrEntry       *stats_dll_get_parse      (MacAddress *);

Import VarBind             *statsGetDll (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind             *statsGetDll0 (OID *, OID *, Uint32, VarEntry *, Uint32);

Import VarBind             *getDllSegSum           (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind             *getDllSegSum0          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind             *getDllSegVal           (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind             *getDllSegVal0          (OID *, OID *, Uint32, VarEntry *,
Uint32);
```

```
Import VarBind          *getDllSegProtocol      (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllSegProtocol0     (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllSegMostActive    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllSegMostActive0   (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllSegThld          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllSegThld0              (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import VarBind          *getDllAddrSum          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrSum0         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrVal          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrVal0         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrProtocol     (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrProtocol0    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrMostActive   (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrMostActive0  (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrPair         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrPair0        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrThld              (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrThld0        (OID *, OID *, Uint32, VarEntry *,
Uint32);
```

```
Import VarBind          *getDllAddrPairThld    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getDllAddrPairThldO   (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import Uint32           setUllSegSum           (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllSegVal           (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllSegProtocol      (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllSegMostActive    (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllSegThld          (OID *, OID *, Uint32, ObjectSyntax *);

Import Uint32           setDllAddrSum          (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllAddrVal          (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllAddrProtocol     (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllAddrMostActive   (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllAddrPair         (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllAddrThld         (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setDllAddrPairThld     (OID *, OID *, Uint32, ObjectSyntax *);

Import void                 setDllAddrDflts     (MibDllAddrDefaults *, StatsDllAddr *);

Import void                 setDllSegDflts      (MibDllSegDefaults *, StatsDllSegment
*);
Import void                 setDllAddrPairDflts (MibDllAddrPairDefaults *,
StatsDialogEntry *);

Import void             statsDllSegRate    (Uint32);
Import void             statsDllAddrRate   (Uint32);
Import void             statsDllDialogRate (Uint32);

Import void                 statsDllAgeDialog (StatsAddrEntry *);
Import void                 statsDllAgeAddr   (StatsAddrEntry *);

Import Uint32           stats_dll_get_stats (StatsAddrEntry *);


/*
```

```
 * Dll macros
*/

/*   GetMacAddrFromOid - gets a mac address from the end of the OID */
#define GetMacAddrFromOid(source, dest) \
for (i = 0; i < 6; i++) { \
dest.bytes[i] = source->oid_ptr[source->length - 6 + i]; }


/*  GetMacAddrsFromOid - gets a mac address pair from the end of the OID */
#define GetMacAddrsFromOid(source, dest1, dest2) \
for (i = 0; i < 6; i++) { \
dest1.bytes[i] = source->oid_ptr[source->length - 12 + i]; \
dest2.bytes[i] = source->oid_ptr[source->length - 6 + i]; } \


/*  GetMacAddrIndexFromOid - gets a mac address from the end of the OID
                             and the index which follows                  */
#define GetMacAddrIndexFromOid(source, dest, indx) \
 for (i = 0; i < 6; i++) { \
 dest.bytes[i] = source->oid_ptr[source->length - 7 + i]; } \
 indx = source->oid_ptr[source->length - 1]


#endif /* stats_dll_h */
```

```
/*
 *  stats_icmp.h
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/include/SCCS/s.stats_icmp.h
 *
 *  Date:       6/5/91
 *
 *  Revision:   1.10
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   05-06-91  DPD       added statsNextIcmpAddrEntry
 *   04-10-91  DPD       added imports for thld gets/sets
 *   03-16-91  KR        Added statsIcmpAgeAddr
 *   03-12-91  DPD       Added imports for gets and sets
 *   03-11-91  DPD       Cleaned up StatsIcmpAddr
 *   02-06-91  KR        Created
 *
 */

#ifndef stats_icmp_h
#define stats_icmp_h

#include "cci_std.h"
#include "address.h"
#include "util.h"
```

```
#include "kuser.h"
#include "stats.h"


/*
 * Following are to be deleted
 */
#define MAX_ICMP_ADDR                   16
#define MAX_ICMP_PAIRS                  16


#define ICMP_HASH_TABLE_SIZE        16
#define ICMP_DIALOG_HASH_TABLE_SIZE     16

/*
 * ICMP Statistics
 */


/*
 * Segment  Statistics
 */
typedef struct stats_icmp_segment_type
     {
     StatsCount32         frames;
     StatsRatePerS        frameRate;
     StatsBucketRate           frameBuckets;

     StatsCount32         bytes;
     StatsRatePerS        byteRate:
     StatsBucketRate           byteBuckets;

     StatsCount32         errors;
     StatsRatePerS        errorRate;
     StatsBucketRate           errorBuckets;

     StatsCount32         revOffSegs;
```

```
StatsRatePerS       rcvOffSegRate;
StatsBucketRate          rcvOffSegBuckets;

StatsCount32        xmtOffSegs;
StatsRatePerS       xmtOffSegRate;
StatsBucketRate          xmtOffSegBuckets;

StatsCount32        transits;
StatsRatePerS       transitRate;
StatsBucketRate          transitBuckets;

StatsCount32        echoReq;
StatsRatePerS       echoReqRate;
StatsCount32        echoReply;
StatsRatePerS       echoReplyRate;

StatsCount32        destUnrNet;
StatsRatePerS       destUnrNetRate;
StatsCount32        destUnrHost;
StatsRatePerS       destUnrHostRate;
StatsCount32        destUnrProtocol;
StatsRatePerS       destUnrProtocolRate;
StatsCount32        destUnrPort;
StatsRatePerS       destUnrPortRate;
StatsCount32        destUnrFrgmt;
StatsRatePerS       destUnrFrgmtRate;
StatsCount32        destUnrSrcRoute;
StatsRatePerS       destUnrSrcRouteRate;
StatsCount32        destUnrNetUnknown;
StatsRatePerS       destUnrNetUnknownRate;
StatsCount32        destUnrHostUnknown;
StatsRatePerS       destUnrHostUnknownRate;
StatsCount32        destUnrHostIsolated;
StatsRatePerS       destUnrHostIsolatedRate;
StatsCount32        destUnrNetProhibited;
StatsRatePerS       destUnrNetProhibitedRate;
StatsCount32        destUnrHostProhibited;
```

```
        StatsRatePerS       destUnrHostProhibitedRate;
        StatsCount32        destUnrNetTos;
        StatsRatePerS       destUnrNetTosRate;
        StatsCount32        destUnrHostTos;
        StatsRatePerS       destUnrHostTosRate;

        StatsCount32        srcQuench;
        StatsRatePerS       srcQuenchRate;

        StatsCount32        redirNet;
        StatsRatePerS       redirNetRate;
        StatsCount32        redirHost;
        StatsRatePerS       redirHostRate;
        StatsCount32        redirNetTos;
        StatsRatePerS       redirNetTosRate;
        StatsCount32        redirHostTos;
        StatsRatePerS       redirHostTosRate;

        StatsCount32        timeExceededInTransit;
        StatsRatePerS       timeExceededInTransitRate;
        StatsCount32        timeExceededInReass;
        StatsRatePerS       timeExceededInReassRate;

        StatsCount32        paramProblem;
        StatsRatePerS       paramProblemRate;
        StatsCount32        paramProblemOption;
        StatsRatePerS       paramProblemOptionRate;

        StatsCount32        timestampReq;
        StatsRatePerS       timestampReqRate;
        StatsCount32        timestampReply;
        StatsRatePerS       timestampReplyRate;
                                                    /* information req/reply no
longer used */
        StatsCount32        addrMaskReq;
        StatsRatePerS       addrMaskReqRate;
        StatsCount32        addrMaskReply;
```

```
    StatsRatePerS       addrMaskReplyRate;

    } StatsIcmpSegment;


/*
 * Address  Statistics
 */
typedef struct stats_icmp_addr_type
    {
    StatsCount32                    frames;
    StatsRatePerS                   frameRate;
    StatsBucketRate                      frameBuckets;
    StatsCount32                    rcvFrames;
    StatsRatePerS                   rcvFrameRate;
    StatsCount32                    xmtFrames;
    StatsRatePerS                   xmtFrameRate;

    StatsCount32                    bytes;
    StatsRatePerS                   byteRate;
    StatsBucketRate                      byteBuckets;
    StatsCount32                    rcvBytes;
    StatsRatePerS                   rcvByteRate;
    StatsCount32                    xmtBytes;
    StatsRatePerS                   xmtByteRate;

    StatsCount32                    errors;
    StatsRatePerS                   errorRate;
    StatsBucketRate                      errorBuckets;
    StatsCount32                    rcvErrors;
    StatsRatePerS                   rcvErrorRate;
    StatsCount32                    xmtErrors;
    StatsRatePerS                   xmtErrorRate;

    StatsCount32                    rcvOffSegs;
    StatsRatePerS                   rcvOffSegRate;
```

```
 StatsBucketRate                 rcvOffSegBuckets;

 StatsCount32                xmtOffSegs;
 StatsRatePerS               xmtOffSegRate;
 StatsBucketRate                 xmtOffSegBuckets;

StatsCount32                 rcvEchoReq;
 StatsRatePerS               rcvEchoReqRate;
 StatsCount32                xmtEchoReq;
 StatsRatePerS               xmtEchoReqRate;
StatsCount32                 echoReply;
 StatsRatePerS               echoReplyRate;
 StatsCount32                rcvEchoReply;
 StatsRatePerS               rcvEchoReplyRate;
 StatsCount32                xmtEchoReply;
 StatsRatePerS               xmtEchoReplyRate;

 StatsCount32                rcvDestUnrNet;
 StatsRatePerS               rcvDestUnrNetRate;
 StatsCount32                rcvDestUnrHost;
 StatsRatePerS               rcvDestUnrHostRate;
 StatsCount32                rcvDestUnrProtocol;
 StatsRatePerS               rcvDestUnrProtocolRate;
 StatsCount32                rcvDestUnrPort;
 StatsRatePerS               rcvDestUnrPortRate;
 StatsCount32                rcvDestUnrFrgmt;
 StatsRatePerS               rcvDestUnrFrgmtRate;
 StatsCount32                rcvDestUnrSrcRoute;
 StatsRatePerS               rcvDestUnrSroRouteRate;
 StatsCount32                rcvDestUnrNetUnknown;
 StatsRatePerS               rcvDestUnrNetUnknownRate;
 StatsCount32                rcvDestUnrHostUnknown;
 StatsRatePerS               rcvDestUnrHostUnknownRate;
 StatsCount32                rcvDestUnrHostIsolated;
 StatsRatePerS               rcvDestUnrHostIsolatedRate;
 StatsCount32                rcvDestUnrNetProhibited;
 StatsRatePerS               rcvDestUnrNetProhibitedRate;
```

```
StatsCount32            rcvDestUnrHostProhibited;
StatsRatePerS           rcvDestUnrHostProhibitedRate;
StatsCount32            rcvDestUnrNetTos;
StatsRatePerS           rcvDestUnrNetTosRate;
StatsCount32            rcvDestUnrHostTos;
StatsRatePerS           rcvDestUnrHostTosRate;

StatsCount32            xmtDestUnrNet;
StatsRatePerS           xmtDestUnrNetRate;
StatsCount32            xmtDestUnrHost;
StatsRatePerS           xmtDestUnrHostRate;
StatsCount32            xmtDestUnrProtocol;
StatsRatePerS           xmtDestUnrProtocolRate;
StatsCount32            xmtDestUnrPort;
StatsRatePerS           xmtDestUnrPortRate;
StatsCount32            xmtDestUnrFrgmt;
StatsRatePerS           xmtDestUnrFrgmtRate;
StatsCount32            xmtDestUnrSrcRoute;
StatsRatePerS           xmtDestUnrSrcRouteRate;
StatsCount32            xmtDestUnrNetUnknown;
StatsRatePerS           xmtDestUnrNetUnknownRate;
StatsCount32            xmtDestUnrHostUnknown;
StatsRatePerS           xmtDestUnrHostUnknownRate;
StatsCount32            xmtDestUnrHostIsolated;
StatsRatePerS           xmtDestUnrHostIsolatedRate;
StatsCount32            xmtDestUnrNetProhibited;
StatsRatePerS           xmtDestUnrNetProhibitedRate;
StatsCount32            xmtDestUnrHostProhibited;
StatsRatePerS           xmtDestUnrHostProhibitedRate;
StatsCount32            xmtDestUnrNetTos;
StatsRatePerS           xmtDestUnrNetTosRate;
StatsCount32            xmtDestUnrHostTos;
StatsRatePerS           xmtDestUnrHostTosRate;

StatsCount32            rcvSrcQuench;
StatsRatePerS           rcvSrcQuenchRate;
StatsCount32            xmtSrcQuench;
```

```
StatsRatePerS           xmtSrcQuenchRate;

StatsCount32            rcvRedirNet;
StatsRatePerS           rcvRedirNetRate;
StatsCount32            rcvRedirHost;
StatsRatePerS           rcvRedirHostRate;
StatsCount32            rcvRedirNetTos;
StatsRatePerS           rcvRedirNetTosRate;
StatsCount32            rcvRedirHostTos;
StatsRatePerS           rcvRedirHostTosRate;
StatsCount32            xmtRedirNet;
StatsRatePerS           xmtRedirNetRate;
StatsCount32            xmtRedirHost;
StatsRatePerS           xmtRedirHostRate;
StatsCount32            xmtRedirNetTos;
StatsRatePerS           xmtRedirNetTosRate;
StatsCount32            xmtRedirHostTos;
StatsRatePerS           xmtRedirHostTosRate;

StatsCount32            rcvTimeExceededInTransit;
StatsRatePerS           rcvTimeExceededInTransitRate;
StatsCount32            rcvTimeExceededInReass;
StatsRatePerS           rcvTimeExceededInReassRate;
StatsCount32            xmtTimeExceededInTransit;
StatsRatePerS           xmtTimeExceededInTransitRate;
StatsCount32            xmtTimeExceededInReass;
StatsRatePerS           xmtTimeExceededInReassRate;

StatsCount32            rcvParamProblem;
StatsRatePerS           rcvParamProblemRate;
StatsCount32            rcvParamProblemOption;
StatsRatePerS           rcvParamProblemOptionRate;
StatsCount32            xmtParamProblem;
StatsRatePerS           xmtParamProblemRate;
StatsCount32            xmtParamProblemOption;
StatsRatePerS           xmtParamProblemOptionRate;
```

```
        StatsCount32                    rcvTimestampReq;

        StatsRatePerS                   rcvTimestampReqRate;
        StatsCount32                    xmtTimestampReq;
        StatsRatePerS                   xmtTimestampReqRate;
        StatsCount32                    rcvTimestampReply;
        StatsRatePerS                   rcvTimestampReplyRate;
        StatsCount32                    xmtTimestampReply;
        StatsRatePerS                   xmtTimestampReplyRate;

        StatsCount32                    rcvAddrMaskReq;
        StatsRatePerS                   rcvAddrMaskReqRate;
        StatsCount32                    xmtAddrMaskReq;
        StatsRatePerS                   xmtAddrMaskReqRate;
        StatsCount32                    rcvAddrMaskReply;
        StatsRatePerS                   rcvAddrMaskReplyRate;
        StatsCount32                    xmtAddrMaskReply;
        StatsRatePerS                   xmtAddrMaskReplyRate;

        FBQhead_type                    dialogQ;        /* Queue of StatsDialogLink types */

        } StatsIcmpAddr;



    /*
     * Global Data Structures
     */
    Import StatsAddrEntry           *icmp_this_seg_addr_ptr;
    Import StatsIcmpSegment         *icmp_this_seg_stats_ptr;

    Import StatsAddrEntry           *icmp_src_seg_addr_ptr, *icmp_dst_seg_addr_ptr;
    Import StatsIcmpSegment         *icmp_src_seg_stats_ptr, *icmp_dst_seg_stats_ptr;

    Import StatsAddrEntry           *icmp_src_node_addr_ptr, *icmp_dst_node_addr_ptr;
    Import StatsIcmpAddr            *icmp_src_node_stats_ptr, *icmp_dst_node_stats_ptr;
```

```
Import StatsAddrEntry          *icmp_dialog_addr_ptr;
Import StatsDialogEntry        *icmp_dialog_stats_ptr;

Import FBQentry_type           *statsNextIcmpAddrEntry;
Import FBQentry_type           *statsNextIcmpPairEntry;

Import VarBind                 *(*statsGetIcmpSeg[5]) ();
Import VarBind                 *(*statsGetIcmpSegO[5]) ();
Import VarBind                 *(*statsGetIcmpAddr[7]) ();
Import VarBind                 *(*statsGetIcmpAddrO[7]) ();

Import Uint32                  (*statsSetIcmpSeg[5]) ();
Import Uint32                  (*statsSetIcmpAddr[7]) ();

Import StatsAddrEntry          *icmp_hash_table[ICMP_HASH_TABLE_SIZE];
Import StatsAddrEntry          *icmp_dialog_hash_table[ICMP_DIALOG_HASH_TABLE_SIZE];


/*
 * Local data structures
 */
Import Uint32                  icmp_hash;
Import StatsAddrEntry          *icmp_hash_link;
Import StatsAddrEntry          *icmp_previous_hash_link;
Import Uint32                  icmp_dialog_hash;
Import StatsAddrEntry          *icmp_dialog_hash_link;
Import StatsAddrEntry          *icmp_previous_dialog_hash_link;


/*
 * Global Routines
 */
Import Uint32              stats_icmp_init ();
Import StatsAddrEntry      *stats_icmp_get_segment    (Uint32);
Import StatsAddrEntry      *stats_icmp_lookup_segment (Uint32);
Import StatsAddrEntry      *stats_icmp_allocate_addr (Uint32 *);
```

```
Import StatsAddrEntry      *stats_icmp_get_addr      (Uint32 *);
Import StatsAddrEntry      *stats_icmp_lookup_addr   (Uint32 *);
Import StatsAddrEntry      *stats_icmp_get_dialog    (Uint32 *, Uint32 *);
Import StatsAddrEntry     *stats_icmp_lookup_dialog (Uint32 *, Uint32 *);


Import VarBind           *statsGetIcmp (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind           *statsGetIcmpO (OID *, OID *, Uint32, VarEntry *, Uint32);

Import VarBind           *getIcmpSegSum          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpSegSumO         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpSegVal          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpSegValO         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpSegProtocol     (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpSegProtocolO    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpSegMostActive   (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpSegMostActiveO  (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpSegThld              (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpSegThldO        (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import VarBind           *getIcmpAddrSum         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpAddrSumO        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind           *getIcmpAddrVal         (OID *, OID *, Uint32, VarEntry *,
Uint32);
```

```
Import VarBind          *getIcmpAddrVal0        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrProtocol    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrProtocol0   (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrMostActive  (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrMostActive0 (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrPair        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrPair0       (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrThld        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrThld0       (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrPairThld    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIcmpAddrPairThld0   (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import Uint32           setIcmpSegSum           (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpSegVal           (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpSegProtocol      (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpSegMostActive    (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpSegThld          (OID *, OID *, Uint32, ObjectSyntax *);

Import Uint32           setIcmpAddrSum          (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpAddrVal          (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpAddrProtocol     (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpAddrMostActive   (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpAddrPair         (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpAddrThld         (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIcmpAddrPairThld     (OID *, OID *, Uint32, ObjectSyntax *);
```

```
Import void                 setIcmpAddrDflts     (MibIcmpAddrDefaults *, StatsIcmpAddr
*);
Import void                 setIcmpSegDflts      (MibIcmpSegDefaults *, StatsIcmpSegment
*);
Import void                 setIcmpAddrPairDflts (MibIcmpAddrPairDefaults *,
StatsDialogEntry *);

Import void             statsIcmpSegRate  (Uint32);
Import void             statsIcmpAddrRate (Uint32);
Import void             statsIcmpDialogRate (Uint32);

Import void                 statsIcmpAgeDialog (StatsAddrEntry *);
Import void                 statsIcmpAgeAddr   (StatsAddrEntry *);

Import Uint32           stats_icmp_get_stats (StatsAddrEntry *);


#endif /* stats_icmp_h */
```

```
/*
 *  stats_ip.h - [description]
 *
 *
 *       Copyright (c) 1990 Concord Communications Inc.
 *                  All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/include/SCCS/s.stats_ip.h
 *
 *  Date:       6/5/91
 *
 *  Revision:   1.19
 *
 *
 *  Changes:
 *
 *   MM-DD-YY    WHO     Description of change.
 *
 *   04-10-91  DPD       added imports for thld gets/sets
 *   03-16-91  KR        added statsIpAgeAddr
 *   03-13-91  DPD       fixed bug in 'GetIpAddr' macros
 *   03-12-91  DPD       added import of stats_ip_get_stats
 *   03-11-91  DPD       added flowCtrls and frgmt buckets to StatsIpAddr
 *   02-28-91  DPD       general clean-up
 *   02-05-91  DPD       added IP macros
 *   01-31-91  KR        added real stuff
 *  01-08-91    DPD     removed #defines related to alarms to alarms.h
 *   01-07-91  KR        removed StatsIpAddrEntry
 *                  KR        initial
 *
 */

#ifndef stats_ip_h
#define stats_ip_h
```

```
#include "cci_std.h"
#include "address.h"
#include "util.h"
#include "kuser.h"
#include "stats.h"


/*
 * Following are to be deleted
 */
#define MAX_IP_ADDR                     32
#define MAX_IP_PAIRS                    32


#define IP_HASH_TABLE_SIZE              16
#define IP_DIALOG_HASH_TABLE_SIZE  16

/*
 * IP Statistics
 */


/*
 * Segment  Statistics
 */
typedef struct stats_ip_segment_type
     {
     StatsCount32         frames;
     StatsRatePerS        frameRate;
     StatsBucketRate           frameBuckets;

     StatsCount32         bytes;
     StatsRatePerS        byteRate;
     StatsBucketRate           byteBuckets;

     StatsCount32         errors;
     StatsRatePerS        errorRate;
```

```
StatsBucketRate         errorBuckets;

StatsCount32        rcvOffSegs;
StatsRatePerS       rcvOffSegRate;
StatsBucketRate         rcvOffSegBuckets;

StatsCount32        xmtOffSegs;
StatsRatePerS       xmtOffSegRate;
StatsBucketRate         xmtOffSegBuckets;

StatsCount32        transits;
StatsRatePerS       transitRate;
StatsBucketRate         transitBuckets;

StatsCount32        flowCtrls;
StatsRatePerS       flowCtrlRate;
StatsBucketRate         flowCtrlBuckets;

StatsCount32        bcasts;
StatsRatePerS       bcastRate;
StatsBucketRate         bcastBuckets;

StatsCount32        mcasts;
StatsRatePerS       mcastRate;
StatsBucketRate         mcastBuckets;

StatsCount32        frgmts;
StatsRatePerS       frgmtRate;
StatsBucketRate         frgmtBuckets;

StatsCount32        hdrBytes;
StatsRatePerS       hdrByteRate;

Qhead_type              protocolQ;

} StatsIpSegment;
```

```
/*
 * Address  Statistics
 */
typedef struct stats_ip_addr_type
     {
     StatsCount32                    frames;
     StatsRatePerS                   frameRate;
     StatsBucketRate                      frameBuckets;
     StatsCount32                    rcvFrames;
     StatsRatePerS                   rcvFrameRate;
     StatsCount32                    xmtFrames;
     StatsRatePerS                   xmtFrameRate;

     StatsCount32                    bytes;
     StatsRatePerS                   byteRate;
     StatsBucketRate                      byteBuckets;
     StatsCount32                    rcvBytes;
     StatsRatePerS                   rcvbyteRate;
     StatsCount32                    xmtBytes;
     StatsRatePerS                   xmtByteRate;

     StatsCount32                    errors;
     StatsRatePerS                   errorRate;
     StatsBucketRate                      errorBuckets;
     StatsCount32                    rcvErrors;
     StatsRatePerS                   rcvErrorRate;
     StatsCount32                    xmtErrors;
     StatsRatePerS                   xmtErrorRate;

     StatsCount32                    rcvOffSegs;
     StatsRatePerS                   rcvOffSegRate;
     StatsBucketRate                      rcvOffSegBuckets;

     StatsCount32                    xmtOffSegs;
     StatsRatePerS                   xmtOffSegRate;
```

```
    StatsBucketRate                 xmtOffSegBuckets;

    StatsCount32                flowCtrls;
    StatsRatePerS               flowCtrlRate;
    StatsBucketRate                 flowCtrlBuckets;

    StatsCount32                xmtBcasts;
    StatsRatePerS               xmtBcastRate;
    StatsBucketRate                 xmtBcastBuckets;

    StatsCount32                xmtMcasts;
    StatsRatePerS               xmtMcastRate;
    StatsBucketRate                 xmtMcastBuckets;

    StatsCount32                rcvHdrBytes;
    StatsRatePerS               rcvHdrByteRate;
    StatsCount32                xmtHdrBytes;
    StatsRatePerS               xmtHdrByteRate;

    StatsCount32                frgmts;
    StatsRatePerS               frgmtRate;
    StatsBucketRate                 frgmtBuckets;
    StatsCount32                rcvFrgmts;
    StatsRatePerS               rcvFrgmtRate;
    StatsCount32                xmtFrgmts;
    StatsRatePerS               xmtFrgmtRate;

    Qhead_type                      protocolQ;
    FBQhead_type                dialogQ;        /* Queue of StatsDialogLink types */

    } StatsIpAddr;


/*
 * Global Data Structures
```

```
 */
Import StatsAddrEntry          *ip_this_seg_addr_ptr;
Import StatsIpSegment          *ip_this_seg_stats_ptr;
Import StatsProtocolEntry      *ip_this_seg_protocol_ptr;

Import StatsAddrEntry          *ip_src_seg_addr_ptr, *ip_dst_seg_addr_ptr;
Import StatsIpSegment          *ip_src_seg_stats_ptr, *ip_dst_seg_stats_ptr;
Import StatsProtocolEntry      *ip_src_seg_protocol_ptr, *ip_dst_seg_protocol_ptr;

Import StatsAddrEntry          *ip_src_node_addr_ptr, *ip_dst_node_addr_ptr;
Import StatsIpAddr             *ip_src_node_stats_ptr, *ip_dst_node_stats_ptr;

Import StatsAddrEntry          *ip_dialog_addr_ptr;
Import StatsDialogEntry        *ip_dialog_stats_ptr;

Import StatsProtocolEntry      *ip_src_node_protocol_ptr, *ip_dst_node_protocol_ptr;

Import StatsAddrEntry          *ip_hash_table[IP_HASH_TABLE_SIZE];
Import StatsAddrEntry          *ip_dialog_hash_table[IP_DIALOG_HASH_TABLE_SIZE];

Import FBQentry_type           *statsNextIpAddrEntry;
Import FBQentry_type           *statsNextIpPairEntry;

Import VarBind                 *(*statsGetIpSeg[5]) ();
Import VarBind                 *(*statsGetIpSegO[5]) ();
Import VarBind                 *(*statsGetIpAddr[7]) ();
Import VarBind                 *(*statsGetIpAddrO[7]) ();

Import Uint32                  (*statsSetIpSeg[5]) ();
Import Uint32                  (*statsSetIpAddr[7]) ();


/*
 * Local data structures
 */
Import Uint32                  ip_hash;
```

```
Import StatsAddrEntry          *ip_hash_link;
Import StatsAddrEntry          *ip_previous_hash_link;
Import Uint32                  ip_dialog_hash;
Import StatsAddrEntry          *ip_dialog_hash_link;
Import StatsAddrEntry          *ip_previous_dialog_hash_link;


/*
 * Global Routines
 */
Import Uint32                stats_ip_init ();
Import StatsAddrEntry        *stats_ip_get_segment    (Uint32);
Import StatsAddrEntry        *stats_ip_lookup_segment (Uint32);
Import StatsAddrEntry        *stats_ip_allocate_addr (Uint32 *);
Import StatsAddrEntry        *stats_ip_get_addr      (Uint32 *);
Import StatsAddrEntry        *stats_ip_lookup_addr   (Uint32 *);
Import StatsProtocolEntry *stats_ip_get_protocol   (StatsIpAddr *, Uint32);
Import StatsProtocolEntry *stats_ip_lookup_protocol (StatsIpAddr *, Uint32);
Import StatsAddrEntry        *stats_ip_get_dialog    (Uint32 *, Uint32 *);
Import StatsAddrEntry       *stats_ip_lookup_dialog (Uint32 *, Uint32 *);

Import StatsAddrEntry        *stats_ip_get_parse      (Uint32 *);

Import VarBind             *statsGetIp (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind             *statsGetIp0 (OID *, OID *, Uint32, VarEntry *, Uint32);

Import VarBind             *getIpSegSum          (OID *, OID *, Uint32, VarEntry *, Uint32);

Import VarBind             *getIpSegSum0         (OID *, OID *, Uint32, VarEntry *, Uint32);

Import VarBind             *getIpSegVal          (OID *, OID *, Uint32, VarEntry *, Uint32);

Import VarBind             *getIpSegVal0         (OID *, OID *, Uint32, VarEntry *, Uint32);

Import VarBind             *getIpSegProtocol     (OID *, OID *, Uint32, VarEntry *, Uint32);
```

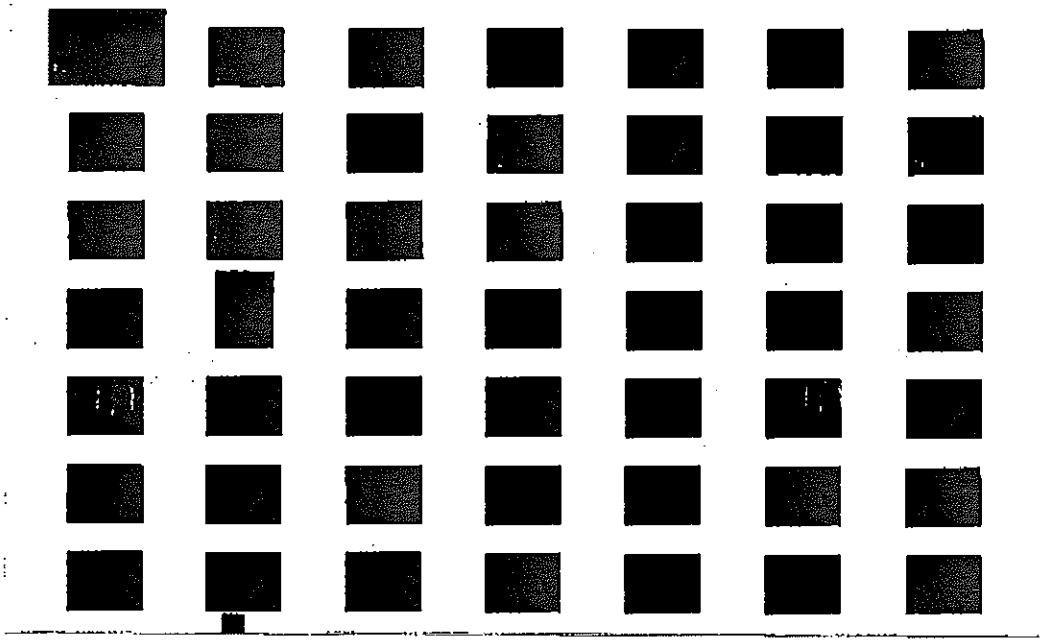
```
Import VarBind          *getIpSegProtocol0    (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpSegMostActive   (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpSegMostActive0  (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpSegThld         (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpSegThld0        (OID *, OID *, Uint32, VarEntry *, Uint32);

Import VarBind          *getIpAddrSum         (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrSum0        (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrVal         (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrVal0        (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrProtocol    (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrProtocol0   (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrMostActive  (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrMostActive0 (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrPair        (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrPair0       (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrThld        (OID *, OID *, Uint32, VarEntry *, Uint32);
Import VarBind          *getIpAddrThld0            (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind          *getIpAddrPairThld    (OID *, OID *, Uint32, VarEntry *, Uint32);
```

```
Import VarBind          *getIpAddrPairThldO   (OID *, OID *, Uint32, VarEntry *, Uint32);


Import Uint32           setIpSegSum           (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpSegVal           (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpSegProtocol      (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpSegMostActive    (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpSegThld          (OID *, OID *, Uint32, ObjectSyntax *);

Import Uint32           setIpAddrSum          (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpAddrVal          (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpAddrProtocol     (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpAddrMostActive   (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpAddrPair         (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpAddrThld         (OID *, OID *, Uint32, ObjectSyntax *);
Import Uint32           setIpAddrPairThld     (OID *, OID *, Uint32, ObjectSyntax *);

Import void                  setIpAddrDflts     (MibIpAddrDefaults *, StatsIpAddr *);
Import void                  setIpSegDflts      (MibIpSegDefaults *, StatsIpSegment *);
Import void                  setIpAddrPairDflts (MibIpAddrPairDefaults *,
StatsDialogEntry *);

Import void             statsIpSegRate  (Uint32);
Import void             statsIpAddrRate (Uint32);
Import void             statsIpDialogRate (Uint32);

Import void                  statsIpAgeDialog (StatsAddrEntry *);
Import void                  statsIpAgeAddr   (StatsAddrEntry *);

Import Uint32           stats_ip_get_stats (StatsAddrEntry *);


/*
 * IP Macro's
 */

/*  GetIpAddrFromOid - gets a ip address from the end of the OID */
```

```
#define GetIpAddrFromOid(source, dest) \
dest = (source->oid_ptr[source->length - 1]) + \
(source->oid_ptr[source->length - 2] << 8) + \
(source->oid_ptr[source->length - 3] << 16) + \
(source->oid_ptr[source->length - 4] << 24);

/*  GetIpAddrsFromOid - gets a ip address from the end of the OID */
#define GetIpAddrsFromOid(source, dest1, dest2) \
dest1 = (source->oid_ptr[source->length - 5]) + \
(source->oid_ptr[source->length - 6] << 8) + \
(source->oid_ptr[source->length - 7] << 16) + \
(source->oid_ptr[source->length - 8] << 24); \
dest2 = (source->oid_ptr[source->length - 1]) + \
(source->oid_ptr[source->length - 2] << 8) + \
(source->oid_ptr[source->length - 3] << 16) + \
(source->oid_ptr[source->length - 4] << 24);

/*  GetIpAddrIndexFromOid - gets a ip address from the end of the OID */
#define GetIpAddrIndexFromOid(source, dest, indx) \
dest = (source->oid_ptr[source->length - 2]) + \
(source->oid_ptr[source->length - 3] << 8) + \
(source->oid_ptr[source->length - 4] << 16) + \
(source->oid_ptr[source->length - 5] << 24); \
indx = source->oid_ptr[source->length - 1]


#endif /* stats_ip_h */
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerard White

06



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
/*
 *  stats_nfs.h
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/include/SCCS/s.stats_nfs.h
 *
 *  Date:       8/8/91
 *
 *  Revision:   1.15
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   07-25-91  KR        Moved NFS_VERSION to protocols.h
 *   06-18-91  KR        Added flowCtrls to dialog
 *                            Added stats_nfs_lookup_ptrs
 *
 *   06-11-91  KR        Added status field to file system type
 *                            and stats_nfs_deallocate_file_system
 *
 *   06-07-91  KR        Moved program numbers to stats.h
 *   06-06-91  KR        Added ptr to history
 *   06-05-91  KR        Added dialogs
 *   06-04-91  KR        Added file system defs and file hash externs
 *   06-02-91  KR        Added state stuff
 *   05-31-91  DPD       Added support for snmp get/set and rates
 *   05-08-91  KR        Created
 */

#ifndef stats_nfs_h
```

```
#define stats_nfs_h


#include "cci_std.h"
#include "address.h"
#include "util.h"
#include "kuser.h"
#include "stats.h"

#define NFS_FHSIZE              32
#define NFS_MAXDATA             8192
#define NFS_MAXNAMLEN           255
#define NFS_MAXPATHLEN          1024
#define NFS_COOKIESIZE          4

#define STATS_NFS_FILE_SYSTEM_LENGTH    64



#define NFS_SERVER_HASH_TABLE_SIZE 48
#define NFS_CLIENT_HASH_TABLE_SIZE 48
#define NFS_DIALOG_HASH_TABLE_SIZE 48
#define NFS_FILE_HASH_TABLE_SIZE   48

#define NFS_HISTORY_ENTRIES             8

#define NFS_ANSWERED                    0
#define NFS_OUTSTANDING                 1

#define NFS_FILE_IN_USE                 1
#define NFS_FILE_NEW                    0



/*
 * Following are to be deleted
 */
```

```
#define NFS_MAX_ADDR                    32
#define NFS_MAX_PAIRS                   32
/* end of to be deleted */


/*
 * NFS Segment Statistics
 */
typedef struct stats_nfs_segment_type
     {
     StatsCount32        ops;
     StatsRatePerS       opRate;
     StatsBucketRate          opBuckets;
     StatsCount32        readOps;
     StatsRatePerS       readOpRate;
     StatsCount32        writeOps;
     StatsRatePerS       writeOpRate;
     StatsCount32        writeCacheOps;
     StatsRatePerS       writeCacheOpRate;

     StatsCount32        bytes;
     StatsRatePerS       byteRate;
     StatsBucketRate          byteBuckets;
     StatsCount32        readBytes;
     StatsRatePerS       readByteRate;
     StatsCount32        writeBytes;
     StatsRatePerS       writeByteRate;

     StatsCount32        rcvOffSegs;
     StatsRatePerS       rcvOffSegRate;
     StatsBucketRate          rcvOffSegBuckets;

     StatsCount32        xmtOffSegs;
     StatsRatePerS       xmtOffSegRate;
     StatsBucketRate          xmtOffSegBuckets;
```

```
StatsCount32        transits;
StatsRatePerS       transitRate;
StatsBucketRate         transitBuckets;

StatsCount32        flowCtrls;
StatsRatePerS       flowCtrlRate;
StatsBucketRate         flowCtrlBuckets;

StatsCount32        rexmts;
StatsRatePerS       rexmtRate;
StatsBucketRate         rexmtBuckets;

StatsCount32        frgmts;
StatsRatePerS       frgmtRate;

StatsCount32        errors;
StatsRatePerS       errorRate;
StatsBucketRate         errorBuckets;

StatsRpc            rpcStats;

StatsCount32        mountFailures;
StatsRatePerS       mountFailureRate;

StatsCount32        nfsErrPerm;
StatsRatePerS       nfsErrPermRate;
StatsCount32        nfsErrNoEnt;
StatsRatePerS       nfsErrNoEntRate;
StatsCount32        nfsErrIO;
StatsRatePerS       nfsErrIORate;
StatsCount32        nfsErrNXIO;
StatsRatePerS       nfsErrNXIORate;
StatsCount32        nfsErrAccess;
StatsRatePerS       nfsErrAccessRate;
StatsCount32        nfsErrExist;
StatsRatePerS       nfsErrExistRate;
StatsCount32        nfsErrNoDev;
```

```
        StatsRatePerS       nfsErrNoDevRate;
        StatsCount32        nfsErrNotDir;
        StatsRatePerS       nfsErrNotDirRate;
        StatsCount32        nfsErrIsDir;
        StatsRatePerS       nfsErrIsDirRate;
        StatsCount32        nfsErrFBig;
        StatsRatePerS       nfsErrFBigRate;
        StatsCount32        nfsErrNoSpace;
        StatsRatePerS       nfsErrNoSpaceRate;
        StatsCount32        nfsErrROFS;
        StatsRatePerS       nfsErrROFSRate;
        StatsCount32        nfsErrNameTooLong;
        StatsRatePerS       nfsErrNameTooLongRate;
        StatsCount32        nfsErrNotEmpty;
        StatsRatePerS       nfsErrNotEmptyRate;
        StatsCount32        nfsErrDQuota;
        StatsRatePerS       nfsErrDQuotaRate;
        StatsCount32        nfsErrStale;
        StatsRatePerS       nfsErrStaleRate;
        StatsCount32        nfsErrWFlush;
        StatsRatePerS       nfsErrWFlushRate;

        FBQhead_type        fileSystemQ;

        } StatsNfsSegment;


/*
 * NFS Address Statistics
 */
typedef struct stats_nfs_addr_type
        {
        Uint32                  nfsLowVersion;
        Uint32                  nfsHighVersion;
        Uint32                  rpcLowVersion;
        Uint32                  rpcHighVersion;
```

```
Uint32                  mountLowVersion;
Uint32                  mountHighVersion;

StatsCount32        ops;
StatsRatePerS       opRate;
StatsBucketRate         opBuckets;
StatsCount32        readOps;
StatsRatePerS       readOpRate;
StatsCount32        writeOps;
StatsRatePerS       writeOpRate;
StatsCount32        writeCacheOps;
StatsRatePerS       writeCacheOpRate;

StatsCount32        bytes;
StatsRatePerS       byteRate;
StatsBucketRate         byteBuckets;
StatsCount32        readBytes;
StatsRatePerS       readByteRate;
StatsCount32        writeBytes;
StatsRatePerS       writeByteRate;

StatsCount32        errors;
StatsRatePerS       errorRate;
StatsBucketRate         errorBuckets;

StatsCount32        rcvOffSegs;
StatsRatePerS       rcvOffSegRate;
StatsBucketRate         rcvOffSegBuckets;

StatsCount32        xmtOffSegs;
StatsRatePerS       xmtOffSegRate;
StatsBucketRate         xmtOffSegBuckets;

StatsCount32        flowCtrls;
StatsRatePerS       flowCtrlRate;
StatsBucketRate     flowCtrlBuckets;
```

```
StatsCount32        rcvFrgmts;
 StatsRatePerS       rcvFrgmtRate;
 StatsCount32        xmtFrgmts;
 StatsRatePerS       xmtFrgmtRate;

 StatsCount32        rexmts;
 StatsRatePerS       rexmtRate;
 StatsBucketRate     rexmtBuckets;

 StatsRpc            rpcStats;

 StatsCount32        mountFailures;
 StatsRatePerS       mountFailureRate;

 StatsCount32        nfsErrPerm;
 StatsRatePerS       nfsErrPermRate;
 StatsCount32        nfsErrNoEnt;
 StatsRatePerS       nfsErrNoEntRate;
 StatsCount32        nfsErrIO;
 StatsRatePerS       nfsErrIORate;
 StatsCount32        nfsErrNXIO;
 StatsRatePerS       nfsErrNXIORate;
 StatsCount32        nfsErrAccess;
 StatsRatePerS       nfsErrAccessRate;
 StatsCount32        nfsErrExist;
 StatsRatePerS       nfsErrExistRate;
 StatsCount32        nfsErrNoDev;
 StatsRatePerS       nfsErrNoDevRate;
 StatsCount32        nfsErrNotDir;
 StatsRatePerS       nfsErrNotDirRate;
 StatsCount32        nfsErrIsDir;
 StatsRatePerS       nfsErrIsDirRate;
 StatsCount32        nfsErrFBig;
 StatsRatePerS       nfsErrFBigRate;
 StatsCount32        nfsErrNoSpace;
 StatsRatePerS       nfsErrNoSpaceRate;
 StatsCount32        nfsErrROFS;
```

```
    StatsRatePerS       nfsErrROFSRate;
    StatsCount32        nfsErrNameTooLong;
    StatsRatePerS       nfsErrNameTooLongRate;
    StatsCount32        nfsErrNotEmpty;
    StatsRatePerS       nfsErrNotEmptyRate;
    StatsCount32        nfsErrDQuota;
    StatsRatePerS       nfsErrDQuotaRate;
    StatsCount32        nfsErrStale;
    StatsRatePerS       nfsErrStaleRate;
    StatsCount32        nfsErrWFlush;
    StatsRatePerS       nfsErrWFlushRate;

    FBQhead_type        dialogQ;        /* Queue of StatsDialogLink types */
    FBQhead_type        fileSystemQ;    /* Queue of StatsNfsFileSystemLink types */

    } StatsNfsAddr;


typedef struct nfs_history_entry
    {
    Uint32          xid;
    Uint32          call_proc;
    Uint32          status;         /* NFS_ANSWERED or NFS_OUTSTANDING */
    Uint32          *ptr;           /* If unmount, ptr to StatsNfsFileSystemLink to unlink
on reply */
    } StatsNfsHistoryEntry;


typedef struct stats_nfs_state
    {
    Uint32                      histx;
    StatsNfsHistoryEntry    history[NFS_HISTORY_ENTRIES];
    } StatsNfsState;
```

```
typedef struct stats_nfs_dialog_entry_type {
     StatsCount32        ops;
     StatsRatePerS       opRate;
     StatsCount32        readOps;
     StatsRatePerS       readOpRate;
     StatsCount32        writeOps;
     StatsRatePerS       writeOpRate;
     StatsCount32        writeCacheOps;
     StatsRatePerS       writeCacheOpRate;
     StatsCount32        readBytes;
     StatsRatePerS       readByteRate;
     StatsCount32        writeBytes;
     StatsRatePerS       writeByteRate;

     StatsCount32        rexmts;
     StatsRatePerS       rexmtRate;
     StatsCount32        flowCtrls;
     StatsRatePerS       flowCtrlRate;

     StatsCount32        rpcProgramFail;          /* prog unavail or mismatch; */
     StatsRatePerS       rpcProgramFailRate; /* proc unavail or bad args */

     StatsCount32        rpcMismatch;        /* wrong version number of rpc */
     StatsRatePerS       rpcMismatchRate;

     StatsCount32        rpcAuthFail;        /* bad or rejected cred or verf; or auth too
weak */
     StatsRatePerS       rpcAuthFailRate;

     StatsCount32        nfsAccessErr;       /* not owner, no access, read only, or stale
handle */
     StatsRatePerS       nfsAccessErrRate;

     StatsCount32        nfsHardErr;             /* hard error, (e.g. disk error), */
     StatsRatePerS       nfsHardErrRate;
```

```
    StatsCount32        nfsResourceErr;          /* file too big, no space, over quota,
*/
    StatsRatePerS       nfsResourceErrRate; /* or write cache flushed */

    StatsCount32        nfsUserErr;              /* no entry or dir, file exists, no
device, */
    StatsRatePerS       nfsUserErrRate;          /* not dir, is dir, name too long, dir
not empty */

    StatsNfsState       *state_ptr;
    } StatsNfsDialogEntry;


/*
 * Nfs file entry
 *
 * If we start keeping statistics on file systems, StatsNfsFileSystem should probably
 * become a variation of StatsAddrEntry...
 */
typedef struct {
    Byte fileSystem[NFS_MAXPATHLEN];
    } NfsFileSystem;


typedef struct stats_nfs_file_system {
    FBQentry_type                       link;
    Uint32                                   status;          /* NFS_FILE_IN_USE or
NFS_FILE_NEW */
    struct stats_nfs_file_system  *hash_link;
    Uint32                                   serverIpAddr;
    Uint32                                   fileSystemLength;
    Byte                                fileSystem[STATS_NFS_FILE_SYSTEM_LENGTH];
    } StatsNfsFileSystem;


typedef struct file_system_link_type {
    FBQentry_type           link;
```

```
    StatsNfsFileSystem      *file_system_ptr;
    } StatsNfsFileSystemLink;


/*
 * Global Data Structures
 */
Import StatsAddrEntry              *nfs_this_seg_addr_ptr;
Import StatsNfsSegment             *nfs_this_seg_stats_ptr;

Import StatsAddrEntry              *nfs_src_seg_addr_ptr, *nfs_dst_seg_addr_ptr;
Import StatsNfsSegment             *nfs_src_seg_stats_ptr, *nfs_dst_seg_stats_ptr;
Import StatsProtocolEntry          *nfs_src_seg_protocol_ptr, *nfs_dst_seg_protocol_ptr;

Import StatsAddrEntry              *nfs_src_node_addr_ptr, *nfs_dst_node_addr_ptr;
Import StatsNfsAddr                *nfs_src_node_stats_ptr, *nfs_dst_node_stats_ptr;

Import StatsAddrEntry              *nfs_dialog_addr_ptr;
Import StatsNfsDialogEntry         *nfs_dialog_stats_ptr;

Import StatsAddrEntry             *nfs_client_hash_table[NFS_CLIENT_HASH_TABLE_SIZE];
Import StatsAddrEntry              *nfs_server_hash_table[NFS_SERVER_HASH_TABLE_SIZE];
Import StatsAddrEntry              *nfs_dialog_hash_table[NFS_DIALOG_HASH_TABLE_SIZE];
Import StatsNfsFileSystem          *nfs_file_hash_table[NFS_FILE_HASH_TABLE_SIZE];

Import Uint32                     nfs_client_hash;
Import StatsAddrEntry             *nfs_client_hash_link;
Import StatsAddrEntry             *nfs_previous_client_hash_link;

Import Uint32                     nfs_server_hash;
Import StatsAddrEntry             *nfs_server_hash_link;
Import StatsAddrEntry             *nfs_previous_server_hash_link;

Import Uint32                     nfs_dialog_hash;
Import StatsAddrEntry             *nfs_dialog_hash_link;
Import StatsAddrEntry             *nfs_previous_dialog_hash_link;
```

```
Import Uint32                   nfs_file_hash;
Import StatsNfsFileSystem       *nfs_file_hash_link;
Import StatsNfsFileSystem       *nfs_previous_file_hash_link;

Import FBQentry_type               *statsNextNfsServerEntry;
Import FBQentry_type               *statsNextNfsClientEntry;
Import FBQentry_type               *statsNextNfsPairEntry;

Import VarBind                     *(*statsGetNfsSeg[NFS_SEG_END]) ();
Import VarBind                     *(*statsGetNfsSegO[NFS_SEG_END]) ();
Import VarBind                     *(*statsGetNfsAddr[NFS_ADDR_END]) ();
Import VarBind                     *(*statsGetNfsAddrO[NFS_ADDR_END]) ();

Import Uint32                      (*statsSetNfsSeg[NFS_SEG_END]) ();
Import Uint32                      (*statsSetNfsAddr[NFS_ADDR_END]) ();


/*
 * Global Routines
 */
Import Uint32                  stats_nfs_init ();

Import StatsAddrEntry          *stats_nfs_get_segment        (Uint32);
Import StatsAddrEntry          *stats_nfs_get_client         (Uint32 *);
Import StatsAddrEntry          *stats_nfs_get_server         (Uint32 *);
Import StatsAddrEntry          *stats_nfs_get_dialog         (Uint32 *, Uint32 *);
Import Uint32             stats_nfs_get_stats           (StatsAddrEntry *);

Import StatsAddrEntry          *stats_nfs_lookup_segment     (Uint32);
Import StatsAddrEntry          *stats_nfs_lookup_client (Uint32 *);
Import StatsAddrEntry          *stats_nfs_lookup_server (Uint32 *);
Import StatsAddrEntry          *stats_nfs_lookup_dialog (Uint32 *, Uint32 *);

Import StatsNfsFileSystem      *stats_nfs_get_file_system         (StatsAddrEntry
*,StatsAddrEntry *,                                                    Uint32,

NfsFileSystem *);
```

```
Import StatsNfsFileSystem     *stats_nfs_lookup_file_system (StatsAddrEntry *,
StatsAddrEntry *,                                                Uint32,

NfsFileSystem *);

StatsNfsFileSystemLink        *stats_nfs_lookup_file_system_link (StatsNfsAddr *,

StatsNfsFileSystem *);
void                               stats_nfs_deallocate_file_system (StatsAddrEntry *,
StatsAddrEntry *,

StatsNfsFileSystem *);


Import void                        statsNfsAgeClient   (StatsAddrEntry *);
Import void                        statsNfsAgeServer   (StatsAddrEntry *);
Import void                        statsNfsAgeDialog   (StatsAddrEntry *);


Import VarBind                *statsGetNfs            (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                *statsGetNfsO           (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import VarBind                *getNfsSegSum           (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                *getNfsSegSumO          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                *getNfsSegVal           (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                *getNfsSegValO          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                *getNfsSegMostActServer  (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                *getNfsSegMostActServerO (OID *, OID *, Uint32, VarEntry *,
Uint32);
```

```
Import VarBind                  *getNfsSegMostActClient  (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsSegMostActClient0 (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsSegServers         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsSegServers0        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsSegTbl         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsSegTbl0             (OID *, OID *, Uint32, VarEntry
*, Uint32);

Import VarBind                  *getNfsServerSum             (OID *, OID *, Uint32,
VarEntry *, Uint32);
Import VarBind                  *getNfsServerSum0            (OID *, OID *, Uint32,
VarEntry *, Uint32);
Import VarBind                  *getNfsServerVal             (OID *, OID *, Uint32,
VarEntry *, Uint32);
Import VarBind                  *getNfsServerVal0            (OID *, OID *, Uint32,
VarEntry *, Uint32);
Import VarBind                  *getNfsServerMostActClient   (OID *, OID *, Uint32,
VarEntry *, Uint32);
Import VarBind                  *getNfsServerMostActClient0  (OID *, OID *, Uint32,
VarEntry *, Uint32);
Import VarBind                  *getNfsServerFileSystems (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsServerFileSystems0    (OID *, OID *, Uint32,
VarEntry *, Uint32);
Import VarBind                  *getNfsServerTbl         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsServerTbl0        (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import VarBind                  *getNfsClientSum         (OID *, OID *, Uint32, VarEntry *,
Uint32);
```

```
Import VarBind                  *getNfsClientSum0        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsClientVal         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsClientVal0        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsClientMostActServer   (OID *, OID *, Uint32, VarEntry
*, Uint32);
Import VarBind                  *getNfsClientMostActServer0  (OID *, OID *, Uint32, VarEntry
*, Uint32);
Import VarBind                  *getNfsClientFileSystems (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsClientFileSystems0     (OID *, OID *, Uint32,
VarEntry *, Uint32);
Import VarBind                  *getNfsClientThld        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsClientThld0       (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import VarBind                  *getNfsServerPair        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsServerPair0       (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsServerPairThld    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsServerPairThld0   (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import VarBind                  *getNfsClientPair        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsClientPair0       (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsClientPairThld    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getNfsClientPairThld0   (OID *, OID *, Uint32, VarEntry *,
Uint32);
```

```
Import Uint32
*);
Import Uint32
*);
Import Uint32
*);
Import Uint32
*);
Import Uint32
*);
Import Uint32
*);

Import Uint32
ObjectSyntax *);
Import Uint32
ObjectSyntax *);
Import Uint32
ObjectSyntax *);
Import Uint32
ObjectSyntax *);
Import Uint32
ObjectSyntax *);

Import Uint32
ObjectSyntax *);
Import Uint32
ObjectSyntax *);
Import Uint32
ObjectSyntax *);
Import Uint32
ObjectSyntax *);
Import Uint32
ObjectSyntax *);

Import Uint32
ObjectSyntax *);
```

```
setNfsSegSum            (OID *, OID *, Uint32, ObjectSyntax
setNfsSegVal            (OID *, OID *, Uint32, ObjectSyntax
setNfsSegMostActServer  (OID *, OID *, Uint32, ObjectSyntax
setNfsSegMostActClient  (OID *, OID *, Uint32, ObjectSyntax
setNfsSegServers        (OID *, OID *, Uint32, ObjectSyntax
setNfsSegThld           (OID *, OID *, Uint32, ObjectSyntax

setNfsServerSum           (OID *, OID *, Uint32,
setNfsServerVal           (OID *, OID *, Uint32,
setNfsServerMostActClient (OID *, OID *, Uint32,
setNfsServerFileSystem     (OID *, OID *, Uint32,
setNfsServerThld          (OID *, OID *, Uint32,

setNfsClientSum           (OID *, OID *, Uint32,
setNfsClientVal           (OID *, OID *, Uint32,
setNfsClientMostActServer (OID *, OID *, Uint32,
setNfsClientFileSystem     (OID *, OID *, Uint32,
setNfsClientThld          (OID *, OID *, Uint32,

setNfsServerPair          (OID *, OID *, Uint32,
```

```
Import Uint32                   setNfsServerPairThld      (OID *, OID *, Uint32,
ObjectSyntax *);

Import Uint32                   setNfsClientPair          (OID *, OID *, Uint32,
ObjectSyntax *);
Import Uint32                   setNfsClientPairThld      (OID *, OID *, Uint32,
ObjectSyntax *);


Import void                          setNfsServerDflts      (MibNfsAddrDefaults *,
StatsNfsAddr *);
Import void                          setNfsClientDflts      (MibNfsAddrDefaults *,
StatsNfsAddr *);
Import void                          setNfsSegDflts         (MibNfsSegDefaults *,
StatsNfsSegment *);
Import void                          setNfsAddrPairDflts    (MibNfsAddrPairDefaults *,
                                                                 StatsNfsDialogEntry *);


Import void                          statsNfsSegRate        (Uint32);
Import void                          statsNfsAddrRate       (Uint32);
Import void                          statsNfsSocketRate     (Uint32);
Import void                          statsNfsDialogRate     (Uint32);



/*****************************************************************
 *
 * For Nfs, look up the following data structures:
 *
 *        dst_seg_stats_ptr
 *        this_seg_addr_ptr
 *        src_seg_addr_ptr
 *        src_seg_stats_ptr
 *        dst_seg_addr_ptr
 *        this_seg_stats_ptr
 *        src_node_addr_ptr
```

```
 *        src_node_stats_ptr
 *        dst_node_addr_ptr
 *        dst_stats_addr_ptr
 *        dialog_addr_ptr
 *        dialog_stats_ptr
 */
#define stats_nfs_lookup_ptrs(ip_src_addr, src_port, ip_dst_addr, dst_port) { \
\
     if ( (src_port == MOUNT_PORT) || (src_port == NFS_PORT) ) { \
          dialog_addr_ptr = stats_nfs_lookup_dialog (ip_src_addr, ip_dst_addr); \
          src_node_addr_ptr = stats_nfs_lookup_server (ip_src_addr); \
          dst_node_addr_ptr = stats_nfs_lookup_client (ip_src_addr); } \
     else  { \
          dialog_addr_ptr = stats_nfs_lookup_dialog (ip_dst_addr, ip_src_addr); \
          src_node_addr_ptr = stats_nfs_lookup_client (ip_src_addr); \
          dst_node_addr_ptr = stats_nfs_lookup_server (ip_src_addr); } \
\
     if (src_node_addr_ptr != NULL) { \
          src_node_stats_ptr = (StatsNfsAddr *) src_node_addr_ptr->stats_ptr; \
          src_seg_addr_ptr = stats_nfs_lookup_segment
(src_node_addr_ptr->address.segment1); } \
     else { \
          src_node_stats_ptr = NULL; \
          ip_src_addr_ptr = (StatsAddrEntry *) stats_ip_lookup_addr (ip_src_addr); \
          src_seg_addr_ptr = stats_nfs_lookup_segment (ip_src_addr_ptr->address.segment1);
\
} \
\
     if (src_seg_addr_ptr != NULL) \
          src_seg_stats_ptr = (StatsNfsSegment *) src_seg_addr_ptr->stats_ptr; \
     else \
          src_seg_stats_ptr = NULL; \
\
     if (dst_node_addr_ptr != NULL) { \
          dst_node_stats_ptr = (StatsNfsAddr *) dst_node_addr_ptr->stats_ptr; \
          dst_seg_addr_ptr = stats_nfs_lookup_segment
(dst_node_addr_ptr->address.segment1); } \
     else { \
```

```
            dst_node_stats_ptr = NULL; \
            ip_dst_addr_ptr = (StatsAddrEntry *) stats_ip_lookup_addr (ip_dst_addr); \
            dst_seg_addr_ptr = stats_nfs_lookup_segment (ip_dst_addr_ptr->address.segment1);
} \
 \
    if (dst_seg_addr_ptr != NULL) \
        dst_seg_stats_ptr = (StatsNfsSegment *) dst_seg_addr_ptr->stats_ptr; \
    else \
        dst_seg_stats_ptr = NULL; \
 \
    this_seg_addr_ptr = stats_nfs_lookup_segment (mySegmentId); \
    if (this_seg_addr_ptr != NULL) \
        this_seg_stats_ptr = (StatsNfsSegment *) this_seg_addr_ptr->stats_ptr; \
 \
    if (dialog_addr_ptr != NULL) \
        dialog_stats_ptr = (StatsNfsDialogEntry *) dialog_addr_ptr->stats_ptr; }


#endif /* stats_nfs_h */
```

```
/*
 *  stats_pmap.h
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/include/SCCS/s.stats_pmap.h
 *
 *  Date:       8/8/91
 *
 *  Revision:   1.1
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   07-24-91  KR        Created
 */

#ifndef stats_pmap_h
#define stats_pmap_h

/*
 * Format of structure used to keep a port to program mapping
 * These are placed in a queue associated with the ip address.
 * The queue is pointed to from protocol_specific_ptr in
 * the StatsAddrEntry type.
 */

typedef struct pmapping_type {
     FBQentry_type  link;
```

```
    Uint32              port;
    Uint32              protocol;
    Uint32              program;
    Uint32              version;
    Uint32              transport_protocol;
    } StatsPmapping;


Import StatsPmapping *stats_pmap_lookup     (StatsAddrEntry*, Uint32, Uint32);
Import StatsPmapping *stats_pmap_get        (StatsAddrEntry*, Uint32, Uint32);
Import void           stats_pmap_deallocate (StatsAddrEntry*, Uint32, Uint32);

#endif /* stats_pmap_h */
```

101: Utility?
GDF: poor, see p. 10-12 (I), you have to be assertive, and you don't even have to prove it in the application, as long as you can prove it later, and you only have to be right on one use, not all that are asserted in the appln.
(If there is a potential, then go ahead, but the appln has to be corrected by refiling.)
UOG: none
Abs:

102: assuming search has been done, no problem.

103: assuming search has been done and they are unknown protein, no problem at all.

If it is a "known" protein, then there might be some problem. Amgen:

112, 1st: how to make. Too broad (can be corrected, e.g., separate whole seq. from epitope seq.).

how to use? No disclosure at all. A big problem, but can be corrected by refiling, if you want to bet on its potential use.

written description. Fine.

best mode. Fine.

112, 2nd: no definition of epitope
(a related problem, "substantially identical" is defined and not used.)

**** didn't claim Ab

```
/*
 *  stats_rpc.h
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/include/SCCS/s.stats_rpc.h
 *
 *  Date:       6/5/91
 *
 *  Revision:   1.3
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO        Description of change. (latest first)
 *   --------  ---        ---------------------------------------------
 *
 *   05-29-91  KR         Created
 */

#ifndef stats_rpc_h
#define stats_rpc_h

#include "cci_std.h"
#include "address.h"
#include "kuser.h"
#include "util.h"
#include "mib_defs.h"
#include "mib_monitor.h"
#include "snmpd.h"
#include "alarms.h"
#include "bsd43/sys/time.h"
```

```
/*
 * Remote Procedure Call Statistics
 */
typedef struct stats_rpc_type {
     StatsCount32        versionMismatch;
     StatsRatePerS       versionMisnatchRate;
     StatsCount32        authBadCred;
     StatsRatePerS       authBadCredRate;
     StatsCount32        authRejectedCred;
     StatsRatePerS       authRejectedCredRate;
     StatsCount32        authBadVerf;
     StatsRatePerS       authBadVerfRate;
     StatsCount32        authRejectedVerf;
     StatsRatePerS       authRejectedVerfRate;
     StatsCount32        authTooWeak;
     StatsRatePerS       authTooWeakRate;
     StatsCount32        authOther;
     StatsRatePerS       authOtherRate;
     StatsCount32        progUnavail;
     StatsRatePerS       progUnavailRate;
     StatsCount32        progVersionMismatch;
     StatsRatePerS       progVersionMismatchRate;
     StatsCount32        procUnavail;
     StatsRatePerS       procUnavailRate;
     StatsCount32        procGarbageArgs;
     StatsRatePerS       procGarbageArgsRate;
     } StatsRpc;


#endif /* stats_rpc_h */
```

```
/*
 *  stats_tcp.h
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/include/SCCS/s.stats_tcp.h
 *
 *  Date:       6/11/91
 *
 *  Revision:   1.15
 *
 *
 *  Changes:
 *
 *   MM-DD-YY   WHO        Description of change. (latest first)
 *   --------   ---        ---------------------------------------------
 *
 *   06-07-91   KR         added failed connection macros
 *   06-04-91   DPD        added buckets to seg and addr for successful, failed
 *                              and conn retries.
 *
 *   06-03-91   KR         added stats_tcp_lookup_ptrs and active_connections macros
 *   05-06-91   DPD        added common data structure for addresses and sockets
 *                              and added macros needed for sockets
 *
 *   04-18-91   DPD        added missing counters in StatsTcpSegment and StatsTcpAddr
 *   04-16-91   KR         changed sequence number variables in TcpSocketStateType
 *   04-15-91   KR         added tcp_protocol
 *   04-10-91   DPD        added imports for thld gets/set
 *   03-20-91   KR         Added socket pointers and lookup and get routines
 *   03-15-91   DPD        Added macro's and imports for gets and sets
 *   02-14-91   KR         Created
 */
```

```
#ifndef stats_tcp_h
#define stats_tcp_h

#include "cci_std.h"
#include "address.h"
#include "util.h"
#include "kuser.h"
#include "stats.h"


/*
 * Following are to be deleted
 */
#define TCP_MAX_ADDR                    32
#define TCP_MAX_PAIRS                   32

/* end of to be deleted */


#define TCP_CLOSE_WAIT_TIME             120     /* In seconds */
#define TCP_HASH_TABLE_SIZE             48
#define TCP_DIALOG_HASH_TABLE_SIZE 48
#define TCP_SOCKET_HASH_TABLE_SIZE 48
#define TCP_MAX_HISTORY                 32



/*
 * TCP Segment Statistics
 */
typedef struct stats_tcp_segment_type
     {
     StatsCount32       frames;
     StatsRatePerS      frameRate;
     StatsBucketRate         frameBuckets;

     StatsCount32       bytes;
```

```
StatsRatePerS       byteRate;
StatsBucketRate         byteBuckets;

StatsCount32        errors;
StatsRatePerS       errorRate;
StatsBucketRate         errorBuckets;

StatsCount32        rcvOffSegs;
StatsRatePerS       rcvOffSegRate;
StatsBucketRate         rcvOffSegBuckets;

StatsCount32        xmtOffSegs;
StatsRatePerS       xmtOffSegRate;
StatsBucketRate         xmtOffSegBuckets;

StatsCount32        transits;
StatsRatePerS       transitRate;
StatsBucketRate         transitBuckets;

StatsCount32        flowCtrls;
StatsRatePerS       flowCtrlRate;
StatsBucketRate         flowCtrlBuckets;

StatsCount32        frgnts;
StatsRatePerS       frgmtRate;
StatsBucketRate         frgmtBuckets;

StatsCount32        rexmts;
StatsRatePerS       rexmtRate;
StatsBucketRate         rexmtBuckets;

StatsCount32        hdrBytes;
StatsRatePerS       hdrByteRate;

StatsCount32        rexmtBytes;
StatsRatePerS       rexmtByteRate;
```

```
StatsCount32        keepAlives;
StatsRatePerS       keepAliveRate;

StatsCount32        windowProbes;
StatsRatePerS       windowProbeRate;

StatsCount32        outOfOrder;
StatsRatePerS       outOfOrderRate;

StatsCount32        afterWindow;
StatsRatePerS       afterWindowRate;

StatsCount32        afterClose;
StatsRatePerS       afterCloseRate;

StatsCount32        urgs;
StatsRatePerS       urgRate;

StatsCount32        rsts;
StatsRatePerS       rstRate;

StatsCount32        successfulConnections;
StatsRatePerS       successfulConnectionRate;
StatsBucketRate          successfulConnectionBuckets;

StatsCount32        connectionRetries;
StatsRatePerS       connectionRetryRate;
StatsBucketRate          connectionRetryBuckets;

StatsCount32        failedConnections;
StatsRatePerS       failedConnectionRate;
StatsBucketRate          failedConnectionBuckets;

StatsCount32        activeConnections;

Qhead_type               protocolQ;
```

```
    } StatsTcpSegment;


/*
 * TCP Address/Socket Common Statistics
 */
typedef struct stats_tcp_common_type
    {
    StatsCount32                frames;
    StatsRatePerS               frameRate;
    StatsBucketRate                 frameBuckets;
    StatsCount32                rcvFrames;
    StatsRatePerS               rcvFrameRate;
    StatsCount32                xmtFrames;
    StatsRatePerS               xmtFrameRate;

    StatsCount32                bytes;
    StatsRatePerS               byteRate;
    StatsBucketRate                 byteBuckets;
    StatsCount32                rcvBytes;
    StatsRatePerS               rcvByteRate;
    StatsCount32                xmtBytes;
    StatsRatePerS               xmtByteRate;

    StatsCount32                errors;
    StatsRatePerS               errorRate;
    StatsBucketRate                 errorBuckets;
    StatsCount32                rcvErrors;
    StatsRatePerS               rcvErrorRate;
    StatsCount32                xmtErrors;
    StatsRatePerS               xmtErrorRate;

    StatsCount32                rcvOffSegs;
    StatsRatePerS               rcvOffSegRate;
    StatsBucketRate                 rcvOffSegBuckets;
```

```
StatsCount32            xmtOffSegs;
StatsRatePerS           xmtOffSegRate;
StatsBucketRate              xmtOffSegBuckets;

StatsCount32          flowCtrls;
StatsRatePerS         flowCtrlRate;
StatsBucketRate       flowCtrlBuckets;

StatsCount32            rcvHdrBytes;
StatsRatePerS           rcvHdrByteRate;
StatsCount32            xmtHdrBytes;
StatsRatePerS           xmtHdrByteRate;

StatsCount32            frgmts;
StatsRatePerS           frgmtRate;
StatsBucketRate              frgmtBuckets;
StatsCount32            rcvFrgmts;
StatsRatePerS           rcvFrgmtRate;
StatsCount32            xmtFrgmts;
StatsRatePerS           xmtFrgmtRate;

StatsCount32            rexmts;
StatsRatePerS           rexmtRate;
StatsBucketRate              rexmtBuckets;
StatsCount32            rcvRexmts;
StatsRatePerS           rcvRexmtRate;
StatsCount32            xmtRexmts;
StatsRatePerS           xmtRexmtRate;

StatsCount32            rcvRexmtBytes;
StatsRatePerS           rcvRexmtByteRate;
StatsCount32            xmtRexmtBytes;
StatsRatePerS           xmtRexmtByteRate;

StatsCount32            rcvKeepAlives;
StatsRatePerS           rcvKeepAliveRate;
StatsCount32            xmtKeepAlives;
```

```
StatsRatePerS           xmtKeepAliveRate;

StatsCount32            rcvWindowProbes;
StatsRatePerS           rcvWindowProbeRate;
StatsCount32            xmtWindowProbes;
StatsRatePerS           xmtWindowProbeRate;

StatsCount32            rcvOutOfOrder;
StatsRatePerS           rcvOutOfOrderRate;
StatsCount32            xmtOutOfOrder;
StatsRatePerS           xmtOutOfOrderRate;

StatsCount32            rcvAfterWindow;
StatsRatePerS           rcvAfterWindowRate;
StatsCount32            xmtAfterWindow;
StatsRatePerS           xmtAfterWindowRate;

StatsCount32            rcvAfterWindowBytes;
StatsRatePerS           rcvAfterWindowByteRate;
StatsCount32            xmtAfterWindowBytes;
StatsRatePerS           xmtAfterWindowByteRate;

StatsCount32            rcvAfterClose;
StatsRatePerS           rcvAfterCloseRate;
StatsCount32            xmtAfterClose;
StatsRatePerS           xmtAfterCloseRate;

StatsCount32            rcvUrgs;
StatsRatePerS           rcvUrgRate;
StatsCount32            xmtUrgs;
StatsRatePerS           xmtUrgRate;

StatsCount32            rcvRsts;
StatsRatePerS           rcvRstRate;
StatsCount32            xmtRsts;
StatsRatePerS           xmtRstRate;
```

```
        StatsCount32                    rcvSuccessfulConnections;
        StatsRatePerS                   rcvSuccessfulConnectionRate;
        StatsBucketRate                     rcvSuccessfulConnectionBuckets;
        StatsCount32                    xmtSuccessfulConnections;
        StatsRatePerS                   xmtSuccessfulConnectionRate;
        StatsBucketRate                     xmtSuccessfulConnectionBuckets;

        StatsCount32                    rcvConnectionRetries;
        StatsRatePerS                   rcvConnectionRetryRate;
        StatsBucketRate                     rcvConnectionRetryBuckets;
        StatsCount32                    xmtConnectionRetries;
        StatsRatePerS                   xmtConnectionRetryRate;
        StatsBucketRate                     xmtConnectionRetryBuckets;

        StatsCount32                    rcvFailedConnections;
        StatsRatePerS                   rcvFailedConnectionRate;
        StatsBucketRate                     rcvFailedConnectionBuckets;
        StatsCount32                    xmtFailedConnections;
        StatsRatePerS                   xmtFailedConnectionRate;
        StatsBucketRate                     xmtFailedConnectionBuckets;
        } StatsTcpCommon;


/*
 * TCP Address Statistics
 */
typedef struct stats_tcp_addr_type
        {
        StatsTcpCommon                  common;
        StatsCount32                    activeConnections;
        FBQhead_type                    dialogQ;        /* Queue of StatsDialogLink types */
        Qhead_type                          protocolQ;
        } StatsTcpAddr;
```

```
/*
 * TCP Socket Statistics
 */
typedef struct stats_tcp_socket_type
     {
     StatsTcpCommon                  common;
     FBQhead_type                    dialogQ;       /* Queue of StatsDialogLink types */
     } StatsTcpSocket;


/*
 * Definitions for TCP state tracking.  The StatsDialogEntry state_ptr
 * points to a structure of type TcpConnectionStateType.
 */

typedef struct tcp_state_history_entry
     {
     Uint32    state;
     Uint32    event;
     Uint32    data_length;
     Uint32    seq;
     Uint32    ack;
     Uint32    flags;
     Uint32    initiator;                  /* 1 or 2 */
     } TcpStateHistoryEntry;


typedef   struct tcp_socket_state_type
     {
     Uint32              indicator;
     Uint32              current_state;
     Uint32              current_event;

     Uint32              max_ack_sent;       /* highest seq number acked */
     Uint32              max_seq_sent;       /* highest seq num sent - used to recognize
rexmts */
```

```
    Uint32              last_length_sent;   /* length of data in pkt with highest seq num
*/
    Uint32              max_data_sent;      /* most sent into peer's window */
    Uint32              last_window_sent;
    Uint32              max_window_sent;
    Uint32              min_window_sent;
    Uint32              snd_wl1;            /* window update seq seg num*/
    Uint32              snd_wl2;            /* window update seq ack num */

    } TcpSocketStateType;


typedof struct tcp_connection_state_type
    {
    TcpSocketStateType          tcp1;
    TcpSocketStateType          tcp2;
    Uint32                          histx;
    TcpStateHistoryEntry          history[TCP_MAX_HISTORY];

    } TcpConnectionStateType;


/*
 * Global Data Structures
 */
Import StatsAddrEntry               *tcp_this_seg_addr_ptr;
Import StatsTcpSegment              *tcp_this_seg_stats_ptr;
Import StatsProtocolEntry           *tcp_this_seg_protocol_ptr;

Import StatsAddrEntry               *tcp_src_seg_addr_ptr, *tcp_dst_seg_addr_ptr;
Import StatsTcpSegment              *tcp_src_seg_stats_ptr, *tcp_dst_seg_stats_ptr;
Import StatsProtocolEntry           *tcp_src_seg_protocol_ptr, *tcp_dst_seg_protocol_ptr;

Import StatsAddrEntry               *tcp_src_node_addr_ptr, *tcp_dst_node_addr_ptr;
Import StatsTcpAddr                 *tcp_src_node_stats_ptr, *tcp_dst_node_stats_ptr;
```

```
Import StatsAddrEntry               *tcp_src_socket_addr_ptr, *tcp_dst_socket_addr_ptr;
Import StatsTcpSocket               *tcp_src_socket_stats_ptr, *tcp_dst_socket_stats_ptr;

Import StatsAddrEntry               *tcp_dialog_addr_ptr;
Import StatsDialogEntry             *tcp_dialog_stats_ptr;

Import StatsProtocolEntry           *tcp_src_node_protocol_ptr, *tcp_dst_node_protocol_ptr;


Import Uint32                       tcp_protocol;
Import TcpConnectionStateType *tcp_connection_state_ptr;
Import TcpSocketStateType           *tcp_src_state_ptr;
Import TcpSocketStateType           *tcp_dst_state_ptr;

Import StatsAddrEntry               *tcp_hash_table[TCP_HASH_TABLE_SIZE];
Import StatsAddrEntry               *tcp_socket_hash_table[TCP_SOCKET_HASH_TABLE_SIZE];
Import StatsAddrEntry               *tcp_dialog_hash_table[TCP_DIALOG_HASH_TABLE_SIZE];

Import FBQentry_type                *statsNextTcpAddrEntry;
Import FBQentry_type                *statsNextTcpSocketEntry;
Import FBQentry_type                *statsNextTcpPairEntry;

Import VarBind                      *(*statsGetTcpSeg[5]) ();
Import VarBind                      *(*statsGetTcpSeg0[5]) ();
Import VarBind                      *(*statsGetTcpAddr[7]) ();
Import VarBind                      *(*statsGetTcpAddr0[7]) ();

Import Uint32                       (*statsSetTcpSeg[5]) ();
Import Uint32                       (*statsSetTcpAddr[7]) ();


/*
 * Local data structures
 */
```

```
Import Uint32                   tcp_hash;
Import StatsAddrEntry           *tcp_hash_link;
Import StatsAddrEntry           *tcp_previous_hash_link;

Import Uint32                   tcp_socket_hash;
Import StatsAddrEntry           *tcp_socket_hash_link;
Import StatsAddrEntry           *tcp_previous_socket_hash_link;

Import Uint32                   tcp_dialog_hash;
Import StatsAddrEntry           *tcp_dialog_hash_link;
Import StatsAddrEntry           *tcp_previous_dialog_hash_link;


/*
 * Global Routines
 */
Import Uint32                   stats_tcp_init ();

Import StatsAddrEntry           *stats_tcp_get_segment      (Uint32);
Import StatsAddrEntry           *stats_tcp_get_addr         (Uint32 *);
Import StatsAddrEntry           *stats_tcp_get_socket       (Uint32 *, Uint32);
Import StatsAddrEntry           *stats_tcp_get_dialog       (Uint32 *, Uint32, Uint32 *,
Uint32);
Import StatsProtocolEntry       *stats_tcp_get_protocol     (StatsTcpAddr *, Uint32);
Import Uint32             stats_tcp_get_stats          (StatsAddrEntry *);

Import StatsAddrEntry           *stats_tcp_lookup_segment   (Uint32);
Import StatsAddrEntry           *stats_tcp_lookup_addr      (Uint32 *);
Import StatsAddrEntry           *stats_tcp_lookup_socket (Uint32 *, Uint32);
Import StatsAddrEntry           *stats_tcp_lookup_dialog (Uint32 *, Uint32, Uint32 *,
Uint32);
Import StatsProtocolEntry       *stats_tcp_lookup_protocol  (StatsTcpAddr *, Uint32);

Import void                          statsTcpAgeAddr        (StatsAddrEntry *);
Import void                          statsTcpAgeSocket   (StatsAddrEntry *);
```

```
Import void                    statsTcpAgeDialog   (StatsAddrEntry *);

Import VarBind                 *statsGetTcp        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *statsGetTcpO       (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import VarBind                 *getTcpSegSum       (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpSegSumO      (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpSegVal       (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpSegValO      (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpSegProtocol  (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpSegProtocolO (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpSegMostActive (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpSegMostActiveO (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpSegThld      (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpSegThldO          (OID *, OID *, Uint32, VarEntry
*, Uint32);

Import VarBind                 *getTcpAddrSum      (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpAddrSumO     (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *gotTcpAddrVal      (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                 *getTcpAddrValO     (OID *, OID *, Uint32, VarEntry *,
Uint32);
```

```
Import VarBind                  *getTcpAddrProtocol1     (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getTcpAddrProtocol0     (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getTcpAddrMostActive    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getTcpAddrMostActive0   (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getTcpAddrPair          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getTcpAddrPair0         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getTcpAddrThld          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getTcpAddrThld0         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getTcpAddrPairThld      (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getTcpAddrPairThld0     (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import Uint32                   setTcpSegSum             (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setTcpSegVal             (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setTcpSegProtocol        (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setTcpSegMostActive      (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setTcpSegThld            (OID *, OID *, Uint32, ObjectSyntax
*);

Import Uint32                   setTcpAddrSum            (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setTcpAddrVal            (OID *, OID *, Uint32, ObjectSyntax
*);
```

```
Import Uint32                   setTcpAddrProtocol      (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setTcpAddrMostActive    (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setTcpAddrPair          (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setTcpAddrThld          (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setTcpAddrPairThld      (OID *, OID *, Uint32, ObjectSyntax
*);

Import void                         setTcpAddrDflts         (MibTcpAddrDefaults *,
StatsTcpCommon *);
Import void                         setTcpSegDflts          (MibTcpSegDefaults *,
StatsTcpSegment *);
Import void                         setTcpAddrPairDflts     (MibTcpAddrPairDefaults *,
StatsDialogEntry *);

Import void                         statsTcpSegRate         (Uint32);
Import void                         statsTcpAddrRate        (Uint32);
Import void                         statsTcpSocketRate      (Uint32);
Import void                         statsTcpDialogRate      (Uint32);


/*
 * TCP Macro's
 */

/*  GetTcpSocketFromOid - gets a ip address from the end of the OID */
#define GetTcpSocketFromOid(source, ip_addr, tcp_port) \
ip_addr = (source->oid_ptr[source->length - 2]) + \
(source->oid_ptr[source->length - 3] << 8) + \
(source->oid_ptr[source->length - 4] << 16) + \
(source->oid_ptr[source->length - 5] << 24); \
tcp_port = source->oid_ptr[source->length - 1]

/*  GetTcpSocketsFromOid - gets a ip address from the end of the OID */
```

```
#define GetTcpSocketsFromOid(source, ip_addr1, tcp_port1, ip_addr2, tcp_port2) \
ip_addr1 = (source->oid_ptr[source->length - 7]) + \
(source->oid_ptr[source->length - 8] << 8) + \
(source->oid_ptr[source->length - 9] << 16) + \
(source->oid_ptr[source->length - 10] << 24); \
tcp_port1 = source->oid_ptr[source->length - 6]; \
ip_addr2 = (source->oid_ptr[source->length - 2]) + \
(source->oid_ptr[source->length - 3] << 8) + \
(source->oid_ptr[source->length - 4] << 16) + \
(source->oid_ptr[source->length - 5] << 24); \
tcp_port2 = source->oid_ptr[source->length - 1]

/*  GetTcpSocketIndexFromOid - gets a ip address from the end of the OID */
#define GetTcpSocketIndexFromOid(source, ip_addr, tcp_port, indx) \
ip_addr = (source->oid_ptr[source->length - 3]) + \
(source->oid_ptr[source->length - 4] << 8) + \
(source->oid_ptr[source->length - 5] << 16) + \
(source->oid_ptr[source->length - 6] << 24); \
tcp_port = source->oid_ptr[source->length - 2]; \
indx = source->oid_ptr[source->length - 1]


/*****************************************************************
 *
 * For Tcp, look up the following data structures:  The invoker
 *
 *        dst_seg_stats_ptr
 *        this_seg_addr_ptr
 *        src_seg_addr_ptr
 *        src_seg_stats_ptr
 *        dst_seg_addr_ptr
 *        this_seg_stats_ptr
 *        src_node_addr_ptr
 *        src_node_stats_ptr
 *        dst_node_addr_ptr
 *        dst_stats_addr_ptr
```

```
 *        src_socket_addr_ptr
 *        src_socket_stats_ptr
 *        dst_socket_addr_ptr
 *        dst_socket_stats_ptr
 *        dialog_addr_ptr
 *        dialog_stats_ptr
 */
#define stats_tcp_lookup_ptrs(ip_src_addr, src_port, ip_dst_addr, dst_port) { \
     src_node_addr_ptr = stats_tcp_lookup_addr (ip_src_addr); \
     if (src_node_addr_ptr != NULL) \
          { \
          src_node_stats_ptr = (StatsTcpAddr *) src_node_addr_ptr->stats_ptr; \
          src_seg_addr_ptr = stats_tcp_lookup_segment
(src_node_addr_ptr->address.segment1); } \
     else \
          { \
          src_node_stats_ptr = NULL; \
          ip_src_addr_ptr = (StatsAddrEntry *) stats_ip_lookup_addr (ip_src_addr); \
          src_seg_addr_ptr = stats_tcp_lookup_segment (ip_src_addr_ptr->address.segment1);
} \
\
     if (src_seg_addr_ptr != NULL) \
          src_seg_stats_ptr = (StatsTcpSegment *) src_seg_addr_ptr->stats_ptr; \
     else \
          src_seg_stats_ptr = NULL; \
 \
     dst_node_addr_ptr = stats_tcp_lookup_addr (ip_dst_addr); \
     if (dst_node_addr_ptr != NULL) \
          { \
          dst_node_stats_ptr = (StatsTcpAddr *) dst_node_addr_ptr->stats_ptr; \
          dst_seg_addr_ptr = stats_tcp_lookup_segment
(dst_node_addr_ptr->address.segment1); } \
     else \
          { \
          dst_node_stats_ptr = NULL; \
          ip_dst_addr_ptr = (StatsAddrEntry *) stats_ip_lookup_addr (ip_dst_addr); \
```

```
        dst_seg_addr_ptr = stats_tcp_lookup_segment (ip_dst_addr_ptr->address.segment1);
; \
 \
    if (dst_seg_addr_ptr != NULL) \
        dst_seg_stats_ptr = (StatsTcpSegment *) dst_seg_addr_ptr->stats_ptr; \
    else \
        dst_seg_stats_ptr = NULL; \
 \
    src_socket_addr_ptr = stats_tcp_lookup_socket (ip_src_addr, src_port); \
    if (src_socket_addr_ptr != NULL) \
        src_socket_stats_ptr = (StatsTcpSocket *) src_socket_addr_ptr->stats_ptr; \
 \
    dst_socket_addr_ptr = stats_tcp_lookup_socket (ip_dst_addr, dst_port); \
    if (dst_socket_addr_ptr != NULL) \
        dst_socket_stats_ptr = (StatsTcpSocket *) dst_socket_addr_ptr->stats_ptr; \
 \
    this_seg_addr_ptr = stats_tcp_lookup_segment (mySegmentId); \
    if (this_seg_addr_ptr != NULL) \
        this_seg_stats_ptr = (StatsTcpSegment *) this_seg_addr_ptr->stats_ptr; \
 \
    dialog_addr_ptr = \
        stats_tcp_lookup_dialog (ip_src_addr, src_port, ip_dst_addr, dst_port); \
        if (dialog_addr_ptr != NULL) \
            dialog_stats_ptr = (StatsDialogEntry *) dialog_addr_ptr->stats_ptr; }


#define tcp_stats_active_connections { \
    stats_count_for_segs     (activeConnections, TCP_SEGMENT, AL_ACTIVE_CONNECTIONS,
NULL); \
    stats_count_for_nodes    (activeConnections, TCP_NODE, AL_ACTIVE_CONNECTIONS, NULL); }


#define tcp_stats_decr_active_connections { \
    stats_decr_count_for_segs     (activeConnections, TCP_SEGMENT, AL_ACTIVE_CONNECTIONS,
NULL); \
    stats_decr_count_for_nodes  (activeConnections, TCP_NODE, AL_ACTIVE_CONNECTIONS,
NULL); }
```

```
#define tcp_stats_failed_connections { \
     stats_for_segs        (failedConnections, failedConnectionRate, \
                                 TCP_SEGMENT, AL_FAILED_CONNECTIONS, NULL); \
     stats_for_rcv_node    (common.rcvFailedConnections, common.rcvFailedConnectionRate, \

                           TCP_NODE, AL_RCV_FAILED_CONNECTIONS, NULL); \
     stats_for_xmt_node    (common.xmtFailedConnections, common.xmtFailedConnectionRate, \

                                 TCP_NODE, AL_XMT_FAILED_CONNECTIONS, NULL); \
     stats_for_rcv_socket (common.rcvFailedConnections, common.rcvFailedConnectionRate, \

                            TCP_SOCKET, AL_RCV_FAILED_CONNECTIONS, NULL); \
     stats_for_xmt_socket (common.xmtFailedConnections, common.xmtFailedConnectionRate, \

                                 TCP_SOCKET, AL_XMT_FAILED_CONNECTIONS, NULL); }


#endif /* stats_tcp_h */
```

```
/*
 *  stats_udp.h
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/include/SCCS/s.stats_udp.h
 *
 *  Date:       6/19/91
 *
 *  Revision:   1.7
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   06-18091  KR        added stats_udp_lookup_ptrs
 *   05-14-91  DPD       added GetUdpSocketAddrFromOld macro
 *   05-07-91  DPD       added common data structure for addresses and sockets
 *   04-25-91  KR        Created
 *
 */

#ifndef stats_udp_h
#define stats_udp_h

#include "cci_std.h"
#include "address.h"
#include "util.h"
#include "kuser.h"
#include "stats.h"
```

```
/*
 * Following are to be deleted
 */
#define UDP_MAX_ADDR                    32
#define UDP_MAX_PAIRS                   32

/* end of to be deleted */


#define UDP_HASH_TABLE_SIZE             48
#define UDP_DIALOG_HASH_TABLE_SIZE 48
#define UDP_SOCKET_HASH_TABLE_SIZE 48
#define UDP_MAX_HISTORY                 32



/*
 * UDP Segment Statistics
 */
typedef struct stats_udp_segment_type
     {
     StatsCount32        frames;
     StatsRatePerS       frameRate;
     StatsBucketRate          frameBuckets;

     StatsCount32        bytes;
     StatsRatePerS       byteRate;
     StatsBucketRate          byteBuckets;

     StatsCount32        errors;
     StatsRatePerS       errorRate;
     StatsBucketRate          errorBuckets;

     StatsCount32        rcvOffSegs;
     StatsRatePerS       rcvOffSegRate;
```

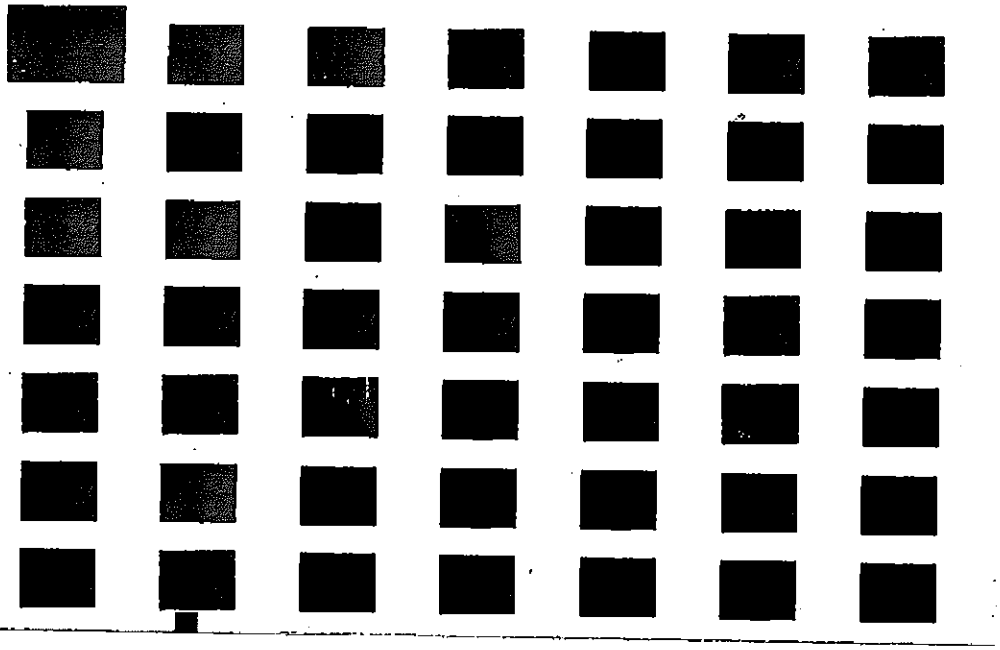
```
        StatsBucketRate         rcvOffSegBuckets;

        StatsCount32        xmtOffSegs;
        StatsRatePerS       xmtOffSegRate;
        StatsBucketRate         xmtOffSegBuckets;

        StatsCount32        transits;
        StatsRatePerS       transitRate;
        StatsBucketRate         transitBuckets;

        StatsCount32        flowCtrls;
        StatsRatePerS       flowCtrlRate;
        StatsBucketRate         flowCtrlBuckets;

        StatsCount32        frgmts;
        StatsRatePerS       frgmtRate;
        StatsBucketRate         frgmtBuckets;

        Qhead_type              protocolQ;

        } StatsUdpSegment;


/*
 * UDP Address/Socket Common Statistics
 */
typedef struct stats_udp_common_type
        {
        StatsCount32                frames;
        StatsRatePerS               frameRate;
        StatsBucketRate                 frameBuckets;
        StatsCount32                rcvFrames;
        StatsRatePerS               rcvFrameRate;
        StatsCount32                xmtFrames;
        StatsRatePerS               xmtFrameRate;
```

```
StatsCount32        bytes;
StatsRatePerS       byteRate;
StatsBucketRate           byteBuckets;
StatsCount32        rcvBytes;
StatsRatePerS       rcvByteRate;
StatsCount32        xmtBytes;
StatsRatePerS       xmtByteRate;

StatsCount32        errors;
StatsRatePerS       errorRate;
StatsBucketRate           errorBuckets;
StatsCount32        rcvErrors;
StatsRatePerS       rcvErrorRate;
StatsCount32        xmtErrors;
StatsRatePerS       xmtErrorRate;

StatsCount32        rcvOffSegs;
StatsRatePerS       rcvOffSegRate;
StatsBucketRate           rcvOffSegBuckets;

StatsCount32        xmtOffSegs;
StatsRatePerS       xmtOffSegRate;
StatsBucketRate           xmtOffSegBuckets;

StatsCount32      flowCtrls;
StatsRatePerS     flowCtrlRate;
StatsBucketRate   flowCtrlBuckets;

StatsCount32      frgmts;
 StatsRatePerS    frgmtRate;
StatsBucketRate   frgmtBuckets;
StatsCount32      rcvFrgmts;
 StatsRatePerS    rcvFrgmtRate;
 StatsCount32     xmtFrgmts;
 StatsRatePerS    xmtFrgmtRate;
 } StatsUdpCommon;
```

```
/*
 * UDP Address Statistics
 */
typedef struct stats_udp_addr_type
    {
    StatsUdpCommon              common;
    Qhead_type                      protocolQ;
    FBQhead_type                dialogQ;        /* Queue of StatsDialogLink types */
    } StatsUdpAddr;


/*
 * UDP Socket Statistics
 */
typedef struct stats_udp_socket_type
    {
    StatsUdpCommon              common;
    FBQhead_type                dialogQ;        /* Queue of StatsDialogLink types */
    } StatsUdpSocket;


/*
 * Global Data Structures
 */
Import StatsAddrEntry           *udp_this_seg_addr_ptr;
Import StatsUdpSegment          *udp_this_seg_stats_ptr;
Import StatsProtocolEntry       *udp_this_seg_protocol_ptr;

Import StatsAddrEntry           *udp_src_seg_addr_ptr, *udp_dst_seg_addr_ptr;
Import StatsUdpSegment          *udp_src_seg_stats_ptr, *udp_dst_seg_stats_ptr;
Import StatsProtocolEntry       *udp_src_seg_protocol_ptr, *udp_dst_seg_protocol_ptr;

Import StatsAddrEntry           *udp_src_node_addr_ptr, *udp_dst_node_addr_ptr;
Import StatsUdpAddr             *udp_src_node_stats_ptr, *udp_dst_node_stats_ptr;
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones, Mary Robertson, David M. Thompson and Gerard White

07



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
Import StatsAddrEntry           *udp_src_socket_addr_ptr, *udp_dst_socket_addr_ptr;
Import StatsUdpSocket           *udp_src_socket_stats_ptr, *udp_dst_socket_stats_ptr;

Import StatsAddrEntry           *udp_dialog_addr_ptr;
Import StatsDialogEntry         *udp_dialog_stats_ptr;

Import StatsProtocolEntry       *udp_src_node_protocol_ptr, *udp_dst_node_protocol_ptr;


Import Uint32                   udp_protocol;


Import StatsAddrEntry           *udp_hash_table[UDP_HASH_TABLE_SIZE];
Import StatsAddrEntry           *udp_socket_hash_table[UDP_SOCKET_HASH_TABLE_SIZE];
Import StatsAddrEntry           *udp_dialog_hash_table[UDP_DIALOG_HASH_TABLE_SIZE];

Import FBQentry_type            *statsNextUdpAddrEntry;
Import FBQentry_type            *statsNextUdpSocketEntry;
Import FBQentry_type            *statsNextUdpPairEntry;

Import VarBind                  *(*statsGetUdpSeg[5]) ();
Import VarBind                  *(*statsGetUdpSegO[5]) ();
Import VarBind                  *(*statsGetUdpAddr[7]) ();
Import VarBind                  *(*statsGetUdpAddrO[7]) ();

Import Uint32                   (*statsSetUdpSeg[5]) ();
Import Uint32                   (*statsSetUdpAddr[7]) ();


/*
 * Local data structures
 */

Import Uint32                   udp_hash;
```

```
Import StatsAddrEntry              *udp_hash_link;
Import StatsAddrEntry              *udp_previous_hash_link;

Import Uint32                      udp_socket_hash;
Import StatsAddrEntry              *udp_socket_hash_link;
Import StatsAddrEntry              *udp_previous_socket_hash_link;

Import Uint32                      udp_dialog_hash;
Import StatsAddrEntry              *udp_dialog_hash_link;
Import StatsAddrEntry              *udp_previous_dialog_hash_link;


/*
 * Global Routines
 */
Import Uint32                  stats_udp_init ();

Import StatsAddrEntry          *stats_udp_get_segment        (Uint32);
Import StatsAddrEntry          *stats_udp_get_addr           (Uint32 *);
Import StatsAddrEntry          *stats_udp_get_socket         (Uint32 *, Uint32);
Import StatsAddrEntry          *stats_udp_get_dialog         (Uint32 *, Uint32, Uint32 *,
Uint32);
Import StatsProtocolEntry      *stats_udp_get_protocol       (StatsUdpAddr *, Uint32);
Import Uint32              stats_udp_get_stats           (StatsAddrEntry *);

Import StatsAddrEntry          *stats_udp_lookup_segment     (Uint32);
Import StatsAddrEntry          *stats_udp_lookup_addr        (Uint32 *);
Import StatsAddrEntry          *stats_udp_lookup_socket (Uint32 *, Uint32);
Import StatsAddrEntry          *stats_udp_lookup_dialog (Uint32 *, Uint32, Uint32 *,
Uint32);
Import StatsProtocolEntry      *stats_udp_lookup_protocol    (StatsUdpAddr *, Uint32);

Import void                        statsUdpAgeAddr           (StatsAddrEntry *);
Import void                        statsUdpAgeSocket    (StatsAddrEntry *);
Import void                        statsUdpAgeDialog    (StatsAddrEntry *);
```

```
Import VarBind                  *statsGetUdp            (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *statsGetUdpO           (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import VarBind                  *getUdpSegSum           (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpSegSumO          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpSegVal           (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpSegValO          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpSegProtocol      (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpSegProtocolO     (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpSegMostActive    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpSegMostActiveO   (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpSegThld          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpSegThldO             (OID *, OID *, Uint32, VarEntry
*, Uint32);

Import VarBind                  *getUdpAddrSum          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrSumO         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrVal          (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrValO         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrProtocol     (OID *, OID *, Uint32, VarEntry *,
Uint32);
```

```
Import VarBind                  *getUdpAddrProtocol0    (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrMostActive   (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUd AddrMostActive0  (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrPair         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrPair0        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrThld         (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrThld0        (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrPairThld     (OID *, OID *, Uint32, VarEntry *,
Uint32);
Import VarBind                  *getUdpAddrPairThld0    (OID *, OID *, Uint32, VarEntry *,
Uint32);

Import Uint32                   setUdpSegSum            (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setUdpSegVal            (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setUdpSegProtocol       (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setUdpSegMostActive     (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setUdpSegThld           (OID *, OID *, Uint32, ObjectSyntax
*);

Import Uint32                   setUdpAddrSum           (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setUdpAddrVal           (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setUdpAddrProtocol      (OID *, OID *, Uint32, ObjectSyntax
*);
```

```
Import Uint32                   setUdpAddrMostActive    (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setUdpAddrPair          (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setUdpAddrThld          (OID *, OID *, Uint32, ObjectSyntax
*);
Import Uint32                   setUdpAddrPairThld      (OID *, OID *, Uint32, ObjectSyntax
*);

Import void                         setUdpAddrDflts         (MibUdpAddrDefaults *,
StatsUdpCommon *);
Import void                         setUdpSegDflts          (MibUdpSegDefaults *,
StatsUdpSegment *);
Import void                         setUdpAddrPairDflts     (MibUdpAddrPairDefaults *,
StatsDialogEntry *);

Import void                         statsUdpSegRate         (Uint32);
Import void                         statsUdpAddrRate        (Uint32);
Import void                         statsUdpSocketRate      (Uint32);
Import void                         statsUdpDialogRate      (Uint32);


/*
 * UDP Macro's
 */

/*  GetUdpSocketFromOid - gets a ip address from the end of the OID */
#define GetUdpSocketFromOid(source, ip_addr, udp_port) \
ip_addr = (source->oid_ptr[source->length - 2]) + \
(source->oid_ptr[source->length - 3] << 8) + \
(source->oid_ptr[source->length - 4] << 16) + \
(source->oid_ptr[source->length - 5] << 24); \
udp_port = source->oid_ptr[source->length - 1]

/*  GetUdpSocketAddrFromOid */
#define GetUdpSocketAddrFromOid(source, ip_addr1, udp_port1, ip_addr2) \
ip_addr1 = (source->oid_ptr[source->length - 6]) + \
```

```
(source->oid_ptr[source->length - 7] << 8) + \
(source->oid_ptr[source->length - 8] << 16) + \
(source->oid_ptr[source->length - 9] << 24); \
udp_port1 = source->oid_ptr[source->length - 5]; \
ip_addr2 = (source->oid_ptr[source->length - 1]) + \
(source->oid_ptr[source->length - 2] << 8) + \
(source->oid_ptr[source->length - 3] << 16) + \
(source->oid_ptr[source->length - 4] << 24)

/*  GetUdpSocketsFromOid - gets a ip address from the end of the OID */
#define GetUdpSocketsFromOid(source, ip_addr1, udp_port1, ip_addr2, udp_port2) \
ip_addr1 = (source->oid_ptr[source->length - 7]) + \
(source->oid_ptr[source->length - 8] << 8) + \
(source->oid_ptr[source->length - 9] << 16) + \
(source->oid_ptr[source->length - 10] << 24); \
udp_port1 = source->oid_ptr[source->length - 6]; \
ip_addr2 = (source->oid_ptr[source->length - 2]) + \
(source->oid_ptr[source->length - 3] << 8) + \
(source->oid_ptr[source->length - 4] << 16) + \
(source->oid_ptr[source->length - 5] << 24); \
udp_port2 = source->oid_ptr[source->length - 1]

/*  GetUdpSocketIndexFromOid - gets a ip address from the end of the OID */
#define GetUdpSocketIndexFromOid(source, ip_addr, udp_port, indx) \
ip_addr = (source->oid_ptr[source->length - 3]) + \
(source->oid_ptr[source->length - 4] << 8) + \
(source->oid_ptr[source->length - 5] << 16) + \
(source->oid_ptr[source->length - 6] << 24); \
udp_port = source->oid_ptr[source->length - 2]; \
indx = source->oid_ptr[source->length - 1]


/*---------------------------------------------------------------*
 *
 * For Udp, look up the following data structures:  The invoker
 *
```

```
 *        dst_seg_stats_ptr
 *        this_seg_addr_ptr
 *        src_seg_addr_ptr
 *        src_seg_stats_ptr
 *        dst_seg_addr_ptr
 *        this_seg_stats_ptr
 *        src_node_addr_ptr
 *        src_node_stats_ptr
 *        dst_node_addr_ptr
 *        dst_stats_addr_ptr
 *        src_socket_addr_ptr
 *        src_socket_stats_ptr
 *        dst_socket_addr_ptr
 *        dst_socket_stats_ptr
 *        dialog_addr_ptr
 *        dialog_stats_ptr
 */
#define stats_udp_lookup_ptrs(ip_src_addr, src_port, ip_dst_addr, dst_port) { \
     src_node_addr_ptr = stats_udp_lookup_addr (ip_src_addr); \
     if (src_node_addr_ptr != NULL) \
          { \
          src_node_stats_ptr = (StatsUdpAddr *) src_node_addr_ptr->stats_ptr; \
          src_seg_addr_ptr = stats_udp_lookup_segment
(src_node_addr_ptr->address.segment1); } \
     else \
          { \
          src_node_stats_ptr = NULL; \
          ip_src_addr_ptr = (StatsAddrEntry *) stats_ip_lookup_addr (ip_src_addr); \
          src_seg_addr_ptr = stats_udp_lookup_segment (ip_src_addr_ptr->address.segment1);
} \
\
     if (src_seg_addr_ptr != NULL) \
          src_seg_stats_ptr = (StatsUdpSegment *) src_seg_addr_ptr->stats_ptr; \
     else \
          src_seg_stats_ptr = NULL; \
 \
     dst_node_addr_ptr = stats_udp_lookup_addr (ip_dst_addr); \
```

```
    if (dst_node_addr_ptr != NULL) \
        { \
        dst_node_stats_ptr = (StatsUdpAddr *) dst_node_addr_ptr->stats_ptr; \
        dst_seg_addr_ptr = stats_udp_lookup_segment
(dst_node_addr_ptr->address.segment1); } \
    else \
        { \
        dst_node_stats_ptr = NULL; \
        ip_dst_addr_ptr = (StatsAddrEntry *) stats_ip_lookup_addr (ip_dst_addr); \
        dst_seg_addr_ptr = stats_udp_lookup_segment (ip_dst_addr_ptr->address.segment1);
} \
 \
    if (dst_seg_addr_ptr != NULL) \
        dst_seg_stats_ptr = (StatsUdpSegment *) dst_seg_addr_ptr->stats_ptr; \
    else \
        dst_seg_stats_ptr = NULL; \
 \
    src_socket_addr_ptr = stats_udp_lookup_socket (ip_src_addr, src_port); \
    if (src_socket_addr_ptr != NULL) \
        src_socket_stats_ptr = (StatsUdpSocket *) src_socket_addr_ptr->stats_ptr; \
 \
    dst_socket_addr_ptr = stats_udp_lookup_socket (ip_dst_addr, dst_port); \
    if (dst_socket_addr_ptr != NULL) \
        dst_socket_stats_ptr = (StatsUdpSocket *) dst_socket_addr_ptr->stats_ptr; \
 \
    this_seg_addr_ptr = stats_udp_lookup_segment (mySegmentId); \
    if (this_seg_addr_ptr != NULL) \
        this_seg_stats_ptr = (StatsUdpSegment *) this_seg_addr_ptr->stats_ptr; \
 \
    dialog_addr_ptr = \
        stats_udp_lookup_dialog (ip_src_addr, src_port, ip_dst_addr, dst_port); \
        if (dialog_addr_ptr != NULL) \
            dialog_stats_ptr = (StatsDialogEntry *) dialog_addr_ptr->stats_ptr; }


#endif /* stats_udp_h */
```

```
/*
 *  stats_dll_a.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_dll_a.c
 *
 *  Date:       8/13/91
 *
 *  Revision:   1.6
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   08-13-91  DPD       Added rate calculations in statsDllSegRate and
 *                            statsDllAddrRate for runt, llc, and enet frames
 *   06-05-91  DPD       Fixed aging bug
 *   06-01-91  DPD       Fixed rate bug
 *
 */

static char stats_dll_a_c [] = "@(#)stats_dll_a.c 1.6";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
#include </usr/include/bsd43/time.h>
#include "util.h"
#include "kuser.h"
#include "nbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "mib_dll.h"
#include "mib_defs.h"
#include "stats.h"
#include "stats_dll.h"
#include "stats_mib.h"
#include "snmpd.h"
#include "mib_alarms.h"
#include "em_ctrl.h"


/*****************************************************************************
 *
 * Given a DLL dialog, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsDllAgeDialog (dialog_addr_ptr)

     StatsAddrEntry      *dialog_addr_ptr;

{
     register Uint32                  xdialog_hash;
     register StatsAddrEntry         *xdialog_hash_link;
```

```
    register StatsAddrEntry       *xprevious_dialog_hash_link;

    register StatsDialogLink *dialog_link_ptr;
    register StatsAddrEntry       *addr_record_ptr;
    register StatsDllAddr         *dll_stats_ptr;
    struct timeval                current_time;
    Uint32                             i;

if (monCtrl.dllDialogAgeTimer == 0)
    return;

if (dialog_addr_ptr == NULL)
    return;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - dialog_addr_ptr->lastTime < monCtrl.dllDialogAgeTimer)
    return;

/*
 * Time to chuck this dialog.  Remove it from the dialog hash table.
 * Look up the dialog in order to set dialog_hash, previous_dialog_hash_link
 * and dialog_hash_link.
 */
addr_record_ptr = stats_dll_lookup_dialog (&dialog_addr_ptr->address.macAddress1,

&dialog_addr_ptr->address.macAddress2);
if (addr_record_ptr != dialog_addr_ptr)
    {
    mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsDllAgeDialog: ptrs not equal");
    return;
    }

xdialog_hash = dialog_hash;
xdialog_hash_link = dialog_hash_link;
xprevious_dialog_hash_link = previous_dialog_hash_link;
```

```
if ( (xprevious_dialog_hash_link == NULL) && (xdialog_hash_link->hash_link == NULL) )
     dialog_hash_table[xdialog_hash] = NULL;
else
     {
     if (xprevious_dialog_hash_link != NULL)
          xprevious_dialog_hash_link->hash_link = xdialog_hash_link->hash_link;
     else
          if (xdialog_hash_link->hash_link != NULL)
               dialog_hash_table[xdialog_hash] = xdialog_hash_link->hash_link;
     }

/*
 * Remove the dialog from the dialogQ for each of the mac addresses
 * and deallocate the structure that was on the dialogQ which pointed
 * to the dialog.
 */
addr_record_ptr = stats_dll_lookup_addr (&dialog_addr_ptr->address.macAddress1);
for (i=0; i<2; i++)
     {
     if (addr_record_ptr != NULL)
          {
          dll_stats_ptr = (StatsDllAddr *) addr_record_ptr->stats_ptr;
          if (dll_stats_ptr != NULL)
               {
               dialog_link_ptr = (StatsDialogLink *) dll_stats_ptr->dialogQ.pFlink;
               while (dialog_link_ptr != NULL)
                    {
                    if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                         break;
                    dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
                    }
               if (dialog_link_ptr != NULL)
                    {
                    FBRemqm (&dll_stats_ptr->dialogQ, (PFBQentry_type) dialog_link_ptr);
                    stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                    }
               }
```

```
        }
    addr_record_ptr = stats_dll_lookup_addr (&dialog_addr_ptr->address.macAddress2);
    }

/*
 * Remove the dialog from the statsDllDialogQ and deallocate the structures
 * associated with the dialog.
 */
PBRemqm (&statsDllDialogQ, (PFBQentry_type) dialog_addr_ptr);

if (dialog_addr_ptr->stats_ptr != NULL)
    stats_deallocate (dialog_addr_ptr->stats_ptr, sizeof(StatsDialogEntry) );

stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );

}


/*****************************************************************************
 *
 * Determine if address statistics should be aged out, and, if so, get rid of it.
 */

void statsDllAgeAddr (addr_ptr)

    StatsAddrEntry      *addr_ptr;

{
    register StatsDialogLink *dialog_link_ptr;
    register StatsDllAddr         *stats_ptr;
    struct timeval                current_time;

if (monCtrl.dllNodeAgeTimer == 0)
    return;

if (addr_ptr == NULL)
```

```
    return;

stats_ptr = (StatsDllAddr *) addr_ptr->stats_ptr;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - addr_ptr->lastTime < monCtrl.dllNodeAgeTimer)
    return;

/*
 * Time to chuck these stats. Not the address - just the stats.
 */

/*
 * Remove the dialog links from the stats and deallocate the stats.
 */
if (stats_ptr != NULL)
    {
    dialog_link_ptr = (StatsDialogLink *) FBRemqh (&stats_ptr->dialogQ);
    while (dialog_link_ptr != NULL)
        {
        stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
        dialog_link_ptr = (StatsDialogLink *) FBRemqh (&stats_ptr->dialogQ);
        }
    stats_deallocate (stats_ptr, sizeof(StatsDllAddr) );

    addr_ptr->stats_ptr = NULL;
    }

}


/*
 ************************************************************************
 *
       *
```

```
 * DLL rate routines
*
 *
       *
 ********************************************************************************
 */


/*
 * statsDllProtocolRate
 */
void statsDllProtocolRate (dll_protocol_ptr, dll_addr_ptr, rate_type, time)
     StatsProtocolEntry              *dll_protocol_ptr;
     StatsAddrEntry                  *dll_addr_ptr;
     Uint32                               rate_type;
     Uint32                               time;
{
     AlarmUserData                   dll_alarm_data;

     while (dll_protocol_ptr != NULL)
     {
          /*
           * Pass the protocol as alarm data in case an alarm occurs
           */
          dll_alarm_data.length = 4;
          bcopy (&dll_protocol_ptr->protocol, dll_alarm_data.data, 4);

          statsCalcRates(&dll_protocol_ptr->frameRate, NULL,
                         rate_type, (StatsAddrEntry *)dll_addr_ptr,
                         DLL_PROTOCOL, AL_FRAMES, time, NULL);
          dll_protocol_ptr = dll_protocol_ptr->link;
     }

}
```

```
/*
 * statsDllSegRate
 */
void statsDllSegRate (rate_type)
     Uint32                    rate_type;
{
     register StatsDllSegment *dll_seg_ptr;
     register StatsAddrEntry       *dll_seg_addr_ptr;
     struct timeval                current_time;
     register StatsProtocolEntry   *dll_protocol_ptr;

     /* get the current time */
     mon_gettimeofday(&current_time, 0);

     dll_seg_addr_ptr = (StatsAddrEntry *) statsDllSegQ.pFlink;

     while (dll_seg_addr_ptr != NULL)
     {
          dll_seg_ptr = (StatsDllSegment *) dll_seg_addr_ptr->stats_ptr;
          if (dll_seg_ptr != NULL)
          {
               if ((current_time.tv_sec - dll_seg_addr_ptr->seconds_start_time) >=
monCtrl.rateTimer)
               {
               statsCalcRates(&dll_seg_ptr->frameRate, &dll_seg_ptr->frameBuckets,
                              rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                              AL_FRAMES, current_time.tv_sec, NULL);
               statsCalcRates(&dll_seg_ptr->byteRate, &dll_seg_ptr->byteBuckets,
                              rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                              AL_BYTES, current_time.tv_sec, NULL);
               statsCalcRates(&dll_seg_ptr->errorRate, &dll_seg_ptr->errorBuckets,
                              rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                              AL_ERRORS, current_time.tv_sec, NULL);
```

```
                statsCalcRates(&dll_seg_ptr->rcvOffSegRate, &dll_seg_ptr->rcvOffSegBuckets,
                               rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                               AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
                statsCalcRates(&dll_seg_ptr->xmtOffSegRate, &dll_seg_ptr->xmtOffSegBuckets,
                               rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                               AL_XMT_OFF_SEG, current_time.tv_sec, NULL);
                statsCalcRates(&dll_seg_ptr->transitRate, &dll_seg_ptr->transitBuckets,
                               rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                               AL_TRANSIT, current_time.tv_sec, NULL);
                statsCalcRates(&dll_seg_ptr->bcastRate, &dll_seg_ptr->bcastBuckets,
                               rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                               AL_BCAST, current_time.tv_sec, NULL);
                statsCalcRates(&dll_seg_ptr->mcastRate, &dll_seg_ptr->mcastBuckets,
                               rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                               AL_MCAST, current_time.tv_sec, NULL);
                statsCalcRates(&dll_seg_ptr->collisionRate, NULL,
                               rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                               AL_COLLISION, current_time.tv_sec, NULL);
                statsCalcRates(&dll_seg_ptr->alignmentErrorRate, NULL,
                               rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                               AL_ALIGNMENT, current_time.tv_sec, NULL);

                statsCalcRates(&dll_seg_ptr->enetFrameRate, NULL,
                               rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                               AL_ENET_FRAMES, current_time.tv_sec, NULL);
                statsCalcRates(&dll_seg_ptr->llcFrameRate, NULL,
```

```
                                        rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                                        AL_LLC_FRAMES, current_time.tv_sec, NULL);
                statsCalcRates(&dll_seg_ptr->runtFrameRate, NULL,
                                        rate_type, (StatsAddrEntry *)dll_seg_addr_ptr,
DLL_SEGMENT,
                                        AL_RUNT_FRAMES, current_time.tv_sec, NULL);

                /* calculate the rates on the protocolQ */
                dll_protocol_ptr = (StatsProtocolEntry *) dll_seg_ptr->protocolQ.pFlink;
                statsDllProtocolRate (dll_protocol_ptr, dll_seg_addr_ptr,
                                                rate_type, current_time.tv_sec);
                dll_seg_addr_ptr->seconds_start_time = current_time.tv_sec;
                }

                statsDllDialogRate (rate_type);
            }
            dll_seg_addr_ptr = (StatsAddrEntry *)dll_seg_addr_ptr->link.pFlink;
        }
}

/*
 * statsDllAddrRate
 */
void statsDllAddrRate (rate_type)
    Uint32                      rate_type;
{
    register StatsDllAddr           *dll_addr_ptr;
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
    register StatsAddrEntry         *dll_entry_ptr;
    register Uint32                      loop_count;
    struct timeval                  current_time;
    register StatsProtocolEntry     *dll_protocol_ptr;

    /* get the current time */
    mon_gettimeofday(&current_time, 0);
```

```
        loop_count = 0;

        /***** AGING *****/
        if (statsNextDllAddrEntry != NULL)
             entry_ptr = statsNextDllAddrEntry;      /* pick up where we left off */
        else
        {
             /* Only age structures if we've processed them all */
             entry_ptr = statsMacAddrQ.pFlink;
             while (entry_ptr != NULL)
             {
                  next_entry_ptr = entry_ptr->pFlink;
                  statsDllAgeAddr ((StatsAddrEntry *) entry_ptr);
                  entry_ptr = next_entry_ptr;
             }

             entry_ptr = statsMacAddrQ.pFlink;
        }
        /***** AGING *****/

        while ((entry_ptr != NULL) && (loop_count < stats_q_count))
        {
             dll_entry_ptr = (StatsAddrEntry *)entry_ptr;
             if (((dll_entry_ptr->em_control & rate_type) != 0)
                                  && (dll_entry_ptr->stats_ptr != NULL))
             {
                  dll_addr_ptr = (StatsDllAddr *)dll_entry_ptr->stats_ptr;

                  if ((current_time.tv_sec - dll_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
                  {
                  statsCalcRates(&dll_addr_ptr->frameRate, &dll_addr_ptr->frameBuckets,
                                  rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                  DLL_NODE, AL_FRAMES, current_time.tv_sec, NULL);
                  statsCalcRates(&dll_addr_ptr->rcvFrameRate, NULL,
                                  rate_type, (StatsAddrEntry *)dll_entry_ptr,
```

```
                                DLL_NODE, AL_RCV_FRAMES, current_time.tv_sec, NULL);
                statsCalcRates(&dll_addr_ptr->xmtFrameRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_XMT_FRAMES, current_time.tv_sec, NULL);

                statsCalcRates(&dll_addr_ptr->byteRate, &dll_addr_ptr->byteBuckets,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_BYTES, current_time.tv_sec, NULL);
                statsCalcRates(&dll_addr_ptr->rcvByteRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_RCV_BYTES, current_time.tv_sec, NULL);
                statsCalcRates(&dll_addr_ptr->xmtByteRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_XMT_BYTES, current_time.tv_sec, NULL);

                statsCalcRates(&dll_addr_ptr->errorRate, &dll_addr_ptr->errorBuckets,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_ERRORS, current_time.tv_sec, NULL);
                statsCalcRates(&dll_addr_ptr->rcvErrorRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_RCV_ERRORS, current_time.tv_sec, NULL);
                statsCalcRates(&dll_addr_ptr->xmtErrorRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_XMT_ERRORS, current_time.tv_sec, NULL);

                statsCalcRates(&dll_addr_ptr->rcvOffSegRate
,&dll_addr_ptr->rcvOffSegBuckets,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
                statsCalcRates(&dll_addr_ptr->xmtOffSegRate,
&dll_addr_ptr->xmtOffSegBuckets,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_XMT_OFF_SEG, current_time.tv_sec, NULL);
                statsCalcRates(&dll_addr_ptr->xmtBcastRate, &dll_addr_ptr->xmtBcastBuckets,

                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_NODE, AL_BCAST, current_time.tv_sec, NULL);
```

```
                statsCalcRates(&dll_addr_ptr->xmtMcastRate, &dll_addr_ptr->xmtMcastBuckets,
                                        rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                        DLL_NODE, AL_MCAST, current_time.tv_sec, NULL);

                statsCalcRates(&dll_addr_ptr->enetFrameRate, NULL,
                                        rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                        DLL_NODE, AL_ENET_FRAMES, current_time.tv_sec, NULL);
                statsCalcRates(&dll_addr_ptr->llcFrameRate, NULL,
                                        rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                        DLL_NODE, AL_LLC_FRAMES, current_time.tv_sec, NULL);
                statsCalcRates(&dll_addr_ptr->runtFrameRate, NULL,
                                        rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                        DLL_NODE, AL_RUNT_FRAMES, current_time.tv_sec, NULL);

                /* calculate the rates on the protocolQ */
                dll_protocol_ptr = (StatsProtocolEntry *) dll_addr_ptr->protocolQ.pFlink;
                statsDllProtocolRate (dll_protocol_ptr, dll_entry_ptr,
                                        rate_type, current_time.tv_sec);

                dll_entry_ptr->seconds_start_time = current_time.tv_sec;
                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }

        statsNextDllAddrEntry = entry_ptr;

        if (statsNextDllAddrEntry != NULL)
            statsRatesNotDone |= STATS_DLL_ADDR_RATE;
}

/*
 * statsDllDialogRate
 */
void statsDllDialogRate (rate_type)
```

```
    Uint32                      rate_type;
{
    register StatsDialogEntry     *dll_dialog_ptr;
    register FBQentry_type        *entry_ptr;
    register FBQentry_type        *next_entry_ptr;
    register StatsAddrEntry       *dll_entry_ptr;
    register Uint32                    loop_count;
    struct timeval                current_time;

    /* get the current time */
    mon_gettimeofday(&current_time, 0);

    if (statsNextDllPairEntry != NULL)
        entry_ptr = statsNextDllPairEntry;      /* pick up where we left off */
    else
    {
        /* Only age structures if we've processed then all */
        entry_ptr = statsDllDialogQ.pFlink;
        while (entry_ptr != NULL)
        {
            next_entry_ptr = entry_ptr->pFlink;
            statsDllAgeDialog ((StatsAddrEntry *) entry_ptr);
            entry_ptr = next_entry_ptr;
        }

        entry_ptr = statsDllDialogQ.pFlink;
    }

    loop_count = 0;
    while ((entry_ptr != NULL) && (loop_count < stats_q_count))
    {
        dll_entry_ptr = (StatsAddrEntry *)entry_ptr;
        if (((dll_entry_ptr->em_control & rate_type) != 0)
                                && (dll_entry_ptr->stats_ptr != NULL))
        {
            dll_dialog_ptr = (StatsDialogEntry *)dll_entry_ptr->stats_ptr;
```

```
                if ((current_time.tv_sec - dll_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
                {
                statsCalcRates(&dll_dialog_ptr->packetRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_PAIR, AL_FRAMES, current_time.tv_sec, NULL);

                statsCalcRates(&dll_dialog_ptr->byteRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_PAIR, AL_BYTES, current_time.tv_sec, NULL);

                statsCalcRates(&dll_dialog_ptr->errorRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_PAIR, AL_ERRORS, current_time.tv_sec, NULL);

                statsCalcRates(&dll_dialog_ptr->fragmentRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_PAIR, AL_FRAGMENTS, current_time.tv_sec, NULL);

                statsCalcRates(&dll_dialog_ptr->rexmtRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_PAIR, AL_REXMTS, current_time.tv_sec, NULL);

                statsCalcRates(&dll_dialog_ptr->flowCtrlRate, NULL,
                                rate_type, (StatsAddrEntry *)dll_entry_ptr,
                                DLL_PAIR, AL_FLOW_CTRLS, current_time.tv_sec, NULL);

                dll_entry_ptr->seconds_start_time = current_time.tv_sec;

                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }
        statsNextDllPairEntry = entry_ptr;

        if (statsNextDllPairEntry != NULL)
```

```
            statsRatesNotDone |= STATS_DLL_PAIR_RATE;
}
```

```
/*
 *  stats_dll_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_dll_p.c
 *
 *  Date:       8/23/91
 *
 *  Revision:   1.6
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   08-15-91  KR        If mac addr is mcast or bcast, set seg id to mySegmentId so
 *                            frames to them won't be counted as xmt off seg
 *   06-12-91  KR        Changed segment id default for mac addresses
 */

static char stats_dll_p_c [] = "@(#)stats_dll_p.c 1.6";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <bsd43/sys/time.h>
#include </usr/include/bsd43/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtn_in.h"
#include "rtp.h"
#include "alarms.h"
#include "mib_dll.h"
#include "mib_defs.h"
#include "stats.h"
#include "stats_dll.h"
#include "stats_mib.h"
#include "snmpd.h"
#include "mib_alarms.h"
#include "em_ctrl.h"


/*
 * Global data structures
 */

StatsAddrEntry      *dll_this_seg_addr_ptr, *dll_src_seg_addr_ptr, *dll_dst_seg_addr_ptr;

StatsDllSegment     *dll_this_seg_stats_ptr, *dll_src_seg_stats_ptr,
*dll_dst_seg_stats_ptr;

StatsAddrEntry      *mac_hash_table[MAC_HASH_TABLE_SIZE];
StatsAddrEntry      *dll_src_node_addr_ptr, *dll_dst_node_addr_ptr;
StatsDllAddr        *dll_src_node_stats_ptr, *dll_dst_node_stats_ptr;
StatsProtocolEntry  *dll_src_node_protocol_ptr, *dll_dst_node_protocol_ptr;
```

```
StatsProtocolEntry  *dll_this_seg_protocol_ptr;
StatsProtocolEntry  *dll_src_seg_protocol_ptr, *dll_dst_seg_protocol_ptr;

StatsAddrEntry      *dialog_hash_table[DLL_DIALOG_HASH_TABLE_SIZE];
StatsAddrEntry      *dll_dialog_addr_ptr;
StatsDialogEntry    *dll_dialog_stats_ptr;


MacAddress          this_src_mac_addr;
MacAddress          this_dst_mac_addr;


/*
 * Local data structures
 */
Uint32              mac_hash;
StatsAddrEntry      *mac_hash_link;
StatsAddrEntry      *previous_mac_hash_link;
Uint32              dialog_hash;
StatsAddrEntry      *dialog_hash_link;
StatsAddrEntry      *previous_dialog_hash_link;



/*************************************************************************
 *
 * Look for the segment address structure.
 * If no match is found, NULL is returned.
 */

StatsAddrEntry *stats_dll_lookup_segment (segment)

     Uint32                  segment;
{
     register StatsAddrEntry       *seg_addr_ptr;
```

```
seg_addr_ptr = (StatsAddrEntry *) statsDllSegQ.pFlink;

while (seg_addr_ptr != NULL)
     {
     if (seg_addr_ptr->address.segment1 == segment)
          break;
     seg_addr_ptr = (StatsAddrEntry *)seg_addr_ptr->link.pFlink;
     }

     return (seg_addr_ptr);
}



/*****************************************************************************
 *
 * Find the structure for keeping Dll segment statistics for the given
 * segment.  If one is not found, attempt to allocate one.
 */

StatsAddrEntry *stats_dll_get_segment (segment)

     Uint32                    segment;
{
     register StatsAddrEntry       *seg_addr_ptr;
     register StatsDllSegment *seg_stats_ptr;

 seg_addr_ptr = stats_dll_lookup_segment (segment);

/*
 * If not found, try to allocate a structure for this segment
 * If one can't be obtained, count this as a drop.
 */
if (seg_addr_ptr == NULL)
     {
     seg_addr_ptr = (StatsAddrEntry *) stats_allocate ( sizeof(StatsAddrEntry) );
     if (seg_addr_ptr != NULL)
```

```
            {
            seg_addr_ptr->address.addressType = MibSegment1;
            seg_addr_ptr->address.segment1 = segment;
            seg_addr_ptr->parse_control = monCtrl.segParseCtrl;
            FBInsqt (&statsDllSegQ, (PFBQentry_type) seg_addr_ptr);
            }
      else
            {
            stats_mon_dll_dropped;
            return (NULL);
            }
      }

/*
 * If there's an address structure, see if there's a statistics structure
 * for Dll attached, if not and parse control allows, allocate one.
 */
if (seg_addr_ptr != NULL)
      {
      seg_stats_ptr = (StatsDllSegment *) seg_addr_ptr->stats_ptr;
      if (seg_stats_ptr == NULL)
            {
            /*
             * If Dll parsing is enabled, allocate Dll statistics structure if
             * one has not already been allocated.  If one can't be obtained, count
             * this as a drop.
             */
            if (seg_addr_ptr->parse_control & MibParseDll)
                  {
                  seg_stats_ptr = (StatsDllSegment *) stats_allocate (sizeof
(StatsDllSegment) );
                  if (seg_stats_ptr == NULL)
                        stats_mon_dll_dropped;
                  else
                        {
                        seg_addr_ptr->stats_ptr = (Uint32 *) seg_stats_ptr;
                        seg_stats_ptr->frameRate.type = STATS_RATE_10S;
```

```
                    seg_stats_ptr->byteRate.type = STATS_RATE_10S;
                    seg_stats_ptr->errorRate.type = STATS_RATE_10S;
                    seg_stats_ptr->rcvOffSegRate.type = STATS_RATE_10S;
                    seg_stats_ptr->xmtOffSegRate.type = STATS_RATE_10S;
                    seg_stats_ptr->transitRate.type = STATS_RATE_10S;
                    seg_stats_ptr->bcastRate.type = STATS_RATE_10S;
                    seg_stats_ptr->mcastRate.type = STATS_RATE_10S;
                    seg_stats_ptr->collisionRate.type = STATS_RATE_10S;
                    seg_stats_ptr->alignmentErrorRate.type = STATS_RATE_10S;
                    seg_stats_ptr->enetFrameRate.type = STATS_RATE_10S;
                    seg_stats_ptr->llcFrameRate.type = STATS_RATE_10S;
                    seg_stats_ptr->runtFrameRate.type = STATS_RATE_10S;
                    Initqh (&seg_stats_ptr->protocolQ);

                    setDllSegDflts (&mibSegDefaults.mibDllSegDefaults, seg_stats_ptr);
                    }
                }
            }
        }

    return (seg_addr_ptr);
    }


/*****************************************************************************
 *
 *  Find the mac address record.
 *  A 3 word mac address hash is done and a pointer to the first
 *  StatsAddrEntry with the same hash is found.  Structures with the
 *  same hash are linked.  The link must be walked and the mac address
 *  compared with the mac address in each of the structures until a match
 *  is found.  If no match is found, NULL is returned.
 */
StatsAddrEntry *stats_dll_lookup_addr (mac_addr)

    MacAddress          *mac_addr;
```

```
{
     Uint32              i;

/*
 * Compute mac address hash to get index into hash table
 */
mac_hash = mac_addr->double_bytes[0] + mac_addr->double_bytes[1] +
mac_addr->double_bytes[2];
mac_hash = ((mac_hash + ((mac_hash & 0xff00) >> 8)) & (MAC_HASH_TABLE_SIZE - 1));


previous_mac_hash_link = NULL;
mac_hash_link = (StatsAddrEntry *) mac_hash_table[mac_hash];

/*
 * Walk linked list for exact entry
 */
while (mac_hash_link != NULL)
     {
     if ( (mac_hash_link->address.macAddress1.double_bytes[2] ==
               mac_addr->double_bytes[2]) &&
          (mac_hash_link->address.macAddress1.double_bytes[1] ==
               mac_addr->double_bytes[1]) &&
          (mac_hash_link->address.macAddress1.double_bytes[0] ==
               mac_addr->double_bytes[0]) )
          return (mac_hash_link);
     else
          {
          previous_mac_hash_link = mac_hash_link;
          mac_hash_link = mac_hash_link->hash_link;
          }
     }


/*
 * No entry found.
 */
```

```
return (NULL);
}


/*****************************************************************
 *
 * Allocate a stats structure if parse control is turned on.
 */

 Uint32 stats_dll_get_stats (mac_addr_record_ptr)

     StatsAddrEntry      *mac_addr_record_ptr;
 {
     register StatsDllAddr          *dll_addr_stats_ptr;

     if ((mac_addr_record_ptr != NULL) && (mac_addr_record_ptr->stats_ptr == NULL))
     {
          /*
           * If parse control is turned on for Dll, allocate a structure for Dll
           * address statistics and initialize it.
           */
          if (mac_addr_record_ptr->parse_control & MibParseDll)
          {
               mac_addr_record_ptr->stats_ptr = (Uint32 *) stats_allocate
(sizeof(StatsDllAddr));
               dll_addr_stats_ptr = (StatsDllAddr *) mac_addr_record_ptr->stats_ptr;
               if (dll_addr_stats_ptr != NULL)
               {
               dll_addr_stats_ptr->frameRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->rcvFrameRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->xmtFrameRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->byteRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->rcvByteRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->xmtByteRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->errorRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->rcvErrorRate.type = STATS_RATE_10S;
```

```
                    dll_addr_stats_ptr->xmtErrorRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->rcvOffSegRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->xmtOffSegRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->xmtBcastRate.type = STATS_RATE_10S;
                    dll_addr_stats_ptr->xmtMcastRate.type = STATS_RATE_10S;
                    Initqh (&dll_addr_stats_ptr->protocolQ);
                    FBInitqh (&dll_addr_stats_ptr->dialogQ);

                    setDllAddrDflts (&mibNodeDefaults.mibDllAddrDefaults,
dll_addr_stats_ptr);
               }
               else
                    return (FALSE);
          }
     }
     return (TRUE);
 }


/*************************************************************************
 *
 * Generate a new node alarm for this mac address
 */
void stats_dll_new_node_alarm (mac_addr)

     MacAddress          *mac_addr;
{
     register MibAddress      *mib_address;

     mib_address = (MibAddress *)mon_cache_malloc(sizeof(MibAddress));
     if (mib_address == NULL)
          return;
     bzero(mib_address, sizeof(MibAddress));

     /* set the address of the node that's alarming */
```

```
    mib_address->addressType = MibMacAddress1;
    mib_address->macAddress1.double_bytes[2] = mac_addr->double_bytes[2];
    mib_address->macAddress1.double_bytes[1] = mac_addr->double_bytes[1];
    mib_address->macAddress1.double_bytes[0] = mac_addr->double_bytes[0];

    stats_new_node_alarm (mib_address);
}


/*************************************************************************
 *
 * Initialize the structure for keeping dll address statistics for the
 * given dll address.
 */

 StatsAddrEntry *stats_dll_init_addr (mac_addr)

    MacAddress              *mac_addr;
{
    register StatsAddrEntry      *mac_addr_record_ptr;

    /*
     * Try to allocate a statistics structure for this address.
     * If one can't be obtained, count this as a drop.
     */
    mac_addr_record_ptr = (StatsAddrEntry *)stats_allocate (sizeof (StatsAddrEntry) );
    if (mac_addr_record_ptr == NULL)
        {
        stats_mon_dll_dropped;
        return (NULL);
        }
    else
        {
        mac_addr_record_ptr->hash_link = NULL;
        mac_addr_record_ptr->address.addressType = MibMacAddress1 + MibSegment1;
```

```
          mac_addr_record_ptr->address.macAddress1.double_bytes[2] =
mac_addr->double_bytes[2];
          mac_addr_record_ptr->address.macAddress1.double_bytes[1] =
mac_addr->double_bytes[1];
          mac_addr_record_ptr->address.macAddress1.double_bytes[0] =
mac_addr->double_bytes[0];
          if (rtp_broadcast || rtp_multicast)
               mac_addr_record_ptr->address.segment1 = mySegmentId;
          else
               mac_addr_record_ptr->address.segment1 = UNKNOWN_SEGMENT_ID;
          mac_addr_record_ptr->parse_control = monCtrl.nodeParseCtrl;
          FBInsqt (&statsMacAddrQ, (PFBQentry_type) mac_addr_record_ptr);
          }

     /*
      * If parse control is turned on, but a structure can't be obtained,
      * stats_icmp_get_stats will return FALSE, so count this as a drop.
      */
     if (stats_dll_get_stats (mac_addr_record_ptr) == FALSE)
          stats_mon_dll_dropped;

     /*
      * Put this mac address record into the hash table.
      * Variable hash was set when stats_dll_lookup_addr was executed.
      */
     if ( mac_hash_table[mac_hash] == NULL)
          mac_hash_table[mac_hash] = mac_addr_record_ptr;
     else
          {
          /*
           * Find the last structure of the hash link
           */
          mac_hash_link = mac_hash_table[mac_hash];

          while (mac_hash_link->hash_link != NULL)
               mac_hash_link = mac_hash_link->hash_link;
```

```
        mac_hash_link->hash_link = mac_addr_record_ptr;
        }

    return (mac_addr_record_ptr);
}


/*****************************************************************************
 *
 * Find the structure for keeping dll address statistics for the given
 * mac address.  If one not found, attempt to allocate one.
 */

StatsAddrEntry *stats_dll_get_parse (mac_addr)

    MacAddress              *mac_addr;
{
    register StatsAddrEntry       *mac_addr_record_ptr;

mac_addr_record_ptr = stats_dll_lookup_addr (mac_addr);

if (mac_addr_record_ptr == NULL)
    mac_addr_record_ptr = stats_dll_init_addr (mac_addr);

return (mac_addr_record_ptr);
}


/*****************************************************************************
 *
 * Find the structure for keeping dll address statistics for the given
 * mac address.  If one not found, attempt to allocate one.
 */

StatsAddrEntry *stats_dll_get_addr (mac_addr)
```

```
    MacAddress                *mac_addr;
{
    register StatsAddrEntry   *mac_addr_record_ptr;

mac_addr_record_ptr = stats_dll_lookup_addr (mac_addr);

if (mac_addr_record_ptr == NULL)
    {
    stats_dll_new_node_alarm (mac_addr);
    mac_addr_record_ptr = stats_dll_init_addr (mac_addr);
    }
else
    if (stats_dll_get_stats (mac_addr_record_ptr) == FALSE)
        stats_mon_dll_dropped;

return (mac_addr_record_ptr);
}


/************************ stats_dll_lookup_dialog ******************************
 *
 * Find a dll dialog given 2 mac addresses.
 * If the dialog is not found, NULL is returned.
 */
StatsAddrEntry *stats_dll_lookup_dialog (mac_addr1, mac_addr2)

    MacAddress                *mac_addr1, *mac_addr2;
{
    register Uint32                xdialog_hash;
    register StatsAddrEntry        *xdialog_hash_link;
    register StatsAddrEntry        *xprevious_dialog_hash_link;

/*
 * Compute hash-function on both MAC addresses
```

```
 */
xdialog_hash = mac_addr1->double_bytes[0] + mac_addr1->double_bytes[1] +
                mac_addr1->double_bytes[2] + mac_addr2->double_bytes[0] +
                mac_addr2->double_bytes[1] + mac_addr2->double_bytes[2];
xdialog_hash = ((xdialog_hash + ((xdialog_hash & 0xff00) >> 8)) &
                  (DLL_DIALOG_HASH_TABLE_SIZE - 1));

xprevious_dialog_hash_link = NULL;
xdialog_hash_link = (StatsAddrEntry *) dialog_hash_table[xdialog_hash];

/*
 * Walk linked list for exact entry
 */
while (xdialog_hash_link != NULL)
     {
     if (((xdialog_hash_link->address.macAddress1.double_bytes[2] ==
mac_addr1->double_bytes[2]) &&
          (xdialog_hash_link->address.macAddress1.double_bytes[1] ==
mac_addr1->double_bytes[1]) &&
          (xdialog_hash_link->address.macAddress1.double_bytes[0] ==
mac_addr1->double_bytes[0]) &&
          (xdialog_hash_link->address.macAddress2.double_bytes[2] ==
mac_addr2->double_bytes[2]) &&
          (xdialog_hash_link->address.macAddress2.double_bytes[1] ==
mac_addr2->double_bytes[1]) &&
          (xdialog_hash_link->address.macAddress2.double_bytes[0] ==
mac_addr2->double_bytes[0]))||

         ((xdialog_hash_link->address.macAddress1.double_bytes[2] ==
mac_addr2->double_bytes[2]) &&
          (xdialog_hash_link->address.macAddress1.double_bytes[1] ==
mac_addr2->double_bytes[1]) &&
          (xdialog_hash_link->address.macAddress1.double_bytes[0] ==
mac_addr2->double_bytes[0]) &&
          (xdialog_hash_link->address.macAddress2.double_bytes[2] ==
mac_addr1->double_bytes[2]) &&
```

```
            (xdialog_hash_link->address.macAddress2.double_bytes[1] ==
mac_addr1->double_bytes[1]) &&
            (xdialog_hash_link->address.macAddress2.double_bytes[0] ==
mac_addr1->double_bytes[0]) ))
                {
                break;
                }
           else
                {
                xprevious_dialog_hash_link = xdialog_hash_link;
                xdialog_hash_link = xdialog_hash_link->hash_link;
                }
      }

dialog_hash = xdialog_hash;
dialog_hash_link = xdialog_hash_link;
previous_dialog_hash_link = xprevious_dialog_hash_link;

return (xdialog_hash_link);
}



/************************ stats_dll_get_dialog ******************************
 *
 * Find or allocate an dll dialog given 2 dll addresses.
 * If the dialog is not found, attempt to allocate a structure.
 */
StatsAddrEntry *stats_dll_get_dialog (mac_addr1, mac_addr2)

     MacAddress                *mac_addr1, *mac_addr2;
{
     register StatsAddrEntry       *dialog_addr_ptr;
     register StatsDialogEntry     *dialog_stats_ptr;
     register StatsDialogLink *dialog_link;
     register StatsAddrEntry       *mac_addr1_record_ptr, *mac_addr2_record_ptr;
```

```
    register StatsDllAddr          *mac_addr1_stats_ptr, *mac_addr2_stats_ptr;
    register Uint32                    parse_control;

dialog_addr_ptr = stats_dll_lookup_dialog (mac_addr1, mac_addr2);
if (dialog_addr_ptr != NULL)
    return (dialog_addr_ptr);

/*
 * Check parse control for the dll addresses.
 */
mac_addr1_record_ptr = stats_dll_lookup_addr (mac_addr1);
mac_addr2_record_ptr = stats_dll_lookup_addr (mac_addr2);

if ( (mac_addr1_record_ptr == NULL) || (mac_addr2_record_ptr == NULL) )
    return (NULL);

if (mac_addr1_record_ptr != NULL)
    parse_control = mac_addr1_record_ptr->parse_control;
if (mac_addr2_record_ptr != NULL)
    parse_control |= mac_addr2_record_ptr->parse_control;


/*
 * If dll is turned on for these mac addresses, allocate structures
 * for the dialog.  If can't get them, count this as a drop.
 */
if ((parse_control & MibParseDll) != MibParseDll)
    return (NULL);

/*
 * Try to allocate structures for this dialog.
 * If they can't be obtained, count this as a drop.
 */
dialog_addr_ptr = (StatsAddrEntry *) stats_allocate (sizeof (StatsAddrEntry) );
if (dialog_addr_ptr == NULL)
    {
    stats_mon_dll_dropped;
```

```
        return (NULL);
        }

    dialog_stats_ptr = (StatsDialogEntry *) stats_allocate (sizeof (StatsDialogEntry) );
    if (dialog_stats_ptr == NULL)
        {
        stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );
        stats_mon_dll_dropped;
        return (NULL);
        }

    /*
     * Initialize the structures
     */
    dialog_addr_ptr->hash_link                                    = NULL;
    dialog_addr_ptr->address.addressType                          = MibMacAddress1 + MibMacAddress2;

    dialog_addr_ptr->address.macAddress1.double_bytes[2]= mac_addr1->double_bytes[2];
    dialog_addr_ptr->address.macAddress1.double_bytes[1]= mac_addr1->double_bytes[1];
    dialog_addr_ptr->address.macAddress1.double_bytes[0]= mac_addr1->double_bytes[0];
    dialog_addr_ptr->address.macAddress2.double_bytes[2]= mac_addr2->double_bytes[2];
    dialog_addr_ptr->address.macAddress2.double_bytes[1]= mac_addr2->double_bytes[1];
    dialog_addr_ptr->address.macAddress2.double_bytes[0]= mac_addr2->double_bytes[0];
    dialog_addr_ptr->parse_control                                = parse_control;
    dialog_addr_ptr->startTime                                    = stats_start_time.tv_sec;
    dialog_addr_ptr->lastTime                                     = stats_start_time.tv_sec;
    dialog_addr_ptr->stats_ptr                                    = (Uint32 *) dialog_stats_ptr;


    FBInsqt (&statsDllDialogQ, (PFBQentry_type) dialog_addr_ptr);

    dialog_stats_ptr->packetRate.type   = STATS_RATE_10S;
    dialog_stats_ptr->byteRate.type     = STATS_RATE_10S;
    dialog_stats_ptr->errorRate.type    = STATS_RATE_10S;
    dialog_stats_ptr->fragmentRate.type = STATS_RATE_10S;
    dialog_stats_ptr->rexmtRate.type    = STATS_RATE_10S;
    dialog_stats_ptr->flowCtrlRate.type = STATS_RATE_10S;
```

```
setDllAddrPairDflts (&mibNodeDefaults.mibDllAddrPairDefaults, dialog_stats_ptr);

/*
 * Link the dialog statistics into the dialog queue for each mac address stats.
 */
if (mac_addr1_record_ptr != NULL)
     {
     if (mac_addr1_record_ptr->stats_ptr != NULL)
          {
          dialog_addr_ptr->address.addressType |= MibSegment1;
          dialog_addr_ptr->address.segment1 = mac_addr1_record_ptr->address.segment1;
          mac_addr1_stats_ptr = (StatsDllAddr *) mac_addr1_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&mac_addr1_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
               }
          }
     }
if (mac_addr2_record_ptr != NULL)
     {
     if (mac_addr2_record_ptr->stats_ptr != NULL)
          {
          dialog_addr_ptr->address.addressType |= MibSegment2;
          dialog_addr_ptr->address.segment2 = mac_addr2_record_ptr->address.segment1;
          mac_addr2_stats_ptr = (StatsDllAddr *) mac_addr2_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&mac_addr2_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
               }
          }
     }
```

```
/*
 * Put the new dialog address structure in the dialog hash table.
 * Variable hash was set up when stats_dll_lookup_dialog was executed.
 */
if (dialog_hash_table[dialog_hash] == NULL)
     dialog_hash_table[dialog_hash] = dialog_addr_ptr;
else
     {
     /*
      * Find the last structure of the hash link.
      */
     dialog_hash_link = dialog_hash_table[dialog_hash];
     while (dialog_hash_link->hash_link != NULL)
          dialog_hash_link = dialog_hash_link->hash_link;

     dialog_hash_link->hash_link = dialog_addr_ptr;
     }

return (dialog_addr_ptr);
}


/************************ setDllAddrDflts ******************************
 *
 *    set dll address thresholds from downloaded defaults
 *
 */
void setDllAddrDflts (mib_dll_addr_defs, dll_stats_addr_ptr)
     MibDllAddrDefaults  *mib_dll_addr_defs;
     StatsDllAddr        *dll_stats_addr_ptr;

{
          dll_stats_addr_ptr->rcvFrames.high_thld =
                                   mib_dll_addr_defs->dllAddrRcvFrameHighThld;
          dll_stats_addr_ptr->rcvFrameRate.high_thld =
                                   mib_dll_addr_defs->dllAddrRcvFrameRateHighThld;
```

```
dll_stats_addr_ptr->rcvBytes.high_thld =
                        nib_dll_addr_defs->dllAddrRcvByteHighThld;
dll_stats_addr_ptr->rcvByteRate.high_thld =
                        nib_dll_addr_defs->dllAddrRcvByteRateHighThld;

dll_stats_addr_ptr->rcvErrors.high_thld =
                        nib_dll_addr_defs->dllAddrRcvErrorHighThld;
dll_stats_addr_ptr->rcvErrorRate.high_thld =
                        nib_dll_addr_defs->dllAddrRcvErrorRateHighThld;

dll_stats_addr_ptr->xmtFrames.high_thld =
                        nib_dll_addr_defs->dllAddrXmtFrameHighThld;
dll_stats_addr_ptr->xmtFrameRate.high_thld =
                        nib_dll_addr_defs->dllAddrXmtFrameRateHighThld;

dll_stats_addr_ptr->xmtBytes.high_thld =
                        nib_dll_addr_defs->dllAddrXmtByteHighThld;
dll_stats_addr_ptr->xmtByteRate.high_thld =
                        nib_dll_addr_defs->dllAddrXmtByteRateHighThld;

dll_stats_addr_ptr->xmtErrors.high_thld =
                        nib_dll_addr_defs->dllAddrXmtErrorHighThld;
dll_stats_addr_ptr->xmtErrorRate.high_thld =
                        nib_dll_addr_defs->dllAddrXmtErrorRateHighThld;

dll_stats_addr_ptr->rcvOffSegs.high_thld =
                        nib_dll_addr_defs->dllAddrRcvOffSegHighThld;
dll_stats_addr_ptr->rcvOffSegRate.high_thld =
                        nib_dll_addr_defs->dllAddrRcvOffSegRateHighThld;

dll_stats_addr_ptr->xmtOffSegs.high_thld =
                        nib_dll_addr_defs->dllAddrXmtOffSegHighThld;
dll_stats_addr_ptr->xmtOffSegRate.high_thld =
                        nib_dll_addr_defs->dllAddrXmtOffSegRateHighThld;

dll_stats_addr_ptr->xmtBcasts.high_thld =
                        nib_dll_addr_defs->dllAddrXmtBcastHighThld;
```

```
        dll_stats_addr_ptr->xmtBcastRate.high_thld =
                                mib_dll_addr_defs->dllAddrXmtBoastRateHighThld;

        dll_stats_addr_ptr->xmtMcasts.high_thld =
                                mib_dll_addr_defs->dllAddrXmtMcastHighThld;
        dll_stats_addr_ptr->xmtMcastRate.high_thld =
                                mib_dll_addr_defs->dllAddrXmtMcastRateHighThld;

        dll_stats_addr_ptr->enetFrames.high_thld =
                                mib_dll_addr_defs->dllAddrEnetFrameHighThld;
        dll_stats_addr_ptr->enetFrameRate.high_thld =
                                mib_dll_addr_defs->dllAddrEnetFrameRateHighThld;

        dll_stats_addr_ptr->llcFrames.high_thld =
                                mib_dll_addr_defs->dllAddrLlcFrameHighThld;
        dll_stats_addr_ptr->llcFrameRate.high_thld =
                                mib_dll_addr_defs->dllAddrLlcFrameRateHighThld;

        dll_stats_addr_ptr->runtFrames.high_thld =
                                mib_dll_addr_defs->dllAddrRuntFrameHighThld;
        dll_stats_addr_ptr->runtFrameRate.high_thld =
                                mib_dll_addr_defs->dllAddrRuntFrameRateHighThld;
}


/************************ setDllAddrPairDflts ***************************
 *
 *    set dll address pair thresholds from downloaded defaults
 *
 */
void setDllAddrPairDflts (mib_dll_addr_pair_defs, dialog_ptr)
     MibDllAddrPairDefaults  *mib_dll_addr_pair_defs;
     StatsDialogEntry        *dialog_ptr;
{
     dialog_ptr->packets.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairFrameHighThld;
```

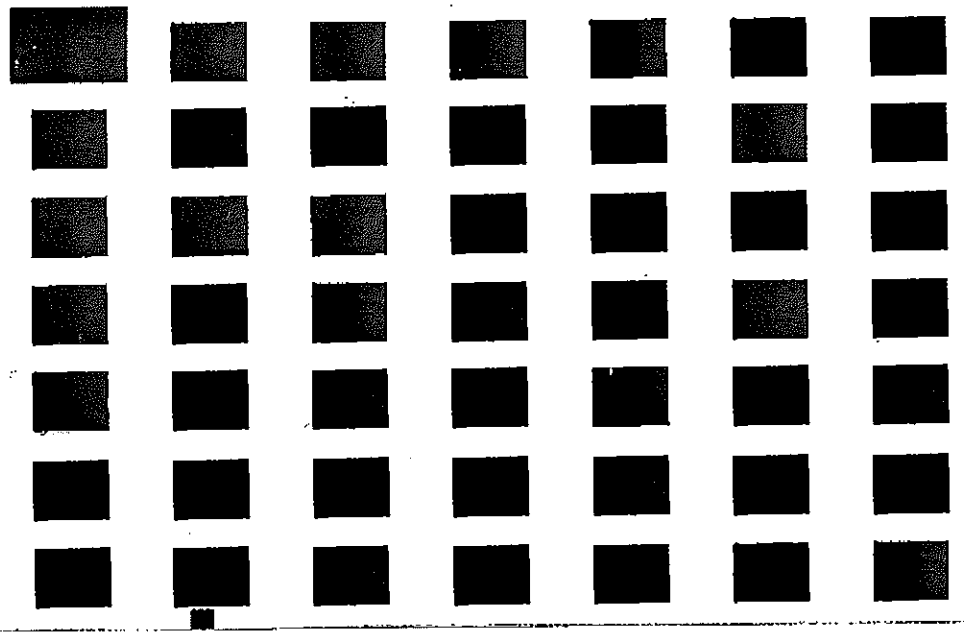
```
    dialog_ptr->packetRate.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairFrameRateHighThld;

    dialog_ptr->bytes.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairByteHighThld;
    dialog_ptr->byteRate.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairByteRateHighThld;

    dialog_ptr->errors.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairErrorHighThld;
    dialog_ptr->errorRate.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairErrorRateHighThld;

    dialog_ptr->fragments.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairFrgmtHighThld;
    dialog_ptr->fragmentRate.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairFrgmtRateHighThld;

    dialog_ptr->rexmts.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairRexmtHighThld;
    dialog_ptr->rexmtRate.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairRexmtRateHighThld;

    dialog_ptr->flowCtrls.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairFlowCtrlHighThld;
    dialog_ptr->flowCtrlRate.high_thld =
                        mib_dll_addr_pair_defs->dllAddrPairFlowCtrlRateHighThld;
}

/************************ setDllSegDflts ***************************
 *
 *    set dll segment thresholds from downloaded defaults
 *
 */
void setDllSegDflts (mib_dll_seg_defs, dll_seg_ptr)
     MibDllSegDefaults   *mib_dll_seg_defs;
```

```
StatsDllSegment     *dll_seg_ptr;
{
dll_seg_ptr->frames.high_thld =
                    mib_dll_seg_defs->dllSegFrameHighThld;
dll_seg_ptr->frameRate.high_thld =
                    mib_dll_seg_defs->dllSegFrameRateHighThld;

dll_seg_ptr->bytes.high_thld =
                    mib_dll_seg_defs->dllSegByteHighThld;
dll_seg_ptr->byteRate.high_thld =
                    mib_dll_seg_defs->dllSegByteRateHighThld;

dll_seg_ptr->errors.high_thld =
                    mib_dll_seg_defs->dllSegErrorHighThld;
dll_seg_ptr->errorRate.high_thld =
                    mib_dll_seg_defs->dllSegErrorRateHighThld;

dll_seg_ptr->rcvOffSegs.high_thld =
                    mib_dll_seg_defs->dllSegRcvOffSegHighThld;
dll_seg_ptr->rcvOffSegRate.high_thld =
                    mib_dll_seg_defs->dllSegRcvOffSegRateHighThld;

dll_seg_ptr->xmtOffSegs.high_thld =
                    mib_dll_seg_defs->dllSegXmtOffSegHighThld;
dll_seg_ptr->xmtOffSegRate.high_thld =
                    mib_dll_seg_defs->dllSegXmtOffSegRateHighThld;

dll_seg_ptr->transits.high_thld =
                    mib_dll_seg_defs->dllSegTransitHighThld;
dll_seg_ptr->transitRate.high_thld =
                    mib_dll_seg_defs->dllSegTransitRateHighThld;

dll_seg_ptr->bcasts.high_thld =
                    mib_dll_seg_defs->dllSegBcastHighThld;
dll_seg_ptr->bcastRate.high_thld =
                    mib_dll_seg_defs->dllSegBcastRateHighThld;
```

```
    dll_seg_ptr->mcasts.high_thld =
                        mib_dll_seg_defs->dllSegMcastHighThld;
    dll_seg_ptr->mcastRate.high_thld =
                        mib_dll_seg_defs->dllSegMcastRateHighThld;

    dll_seg_ptr->collisions.high_thld =
                        mib_dll_seg_defs->dllSegCollisionsHighThld;
    dll_seg_ptr->collisionRate.high_thld =
                        mib_dll_seg_defs->dllSegCollisionsRateHighThld;

    dll_seg_ptr->alignmentErrors.high_thld =
                        mib_dll_seg_defs->dllSegAlignmtErrorsHighThld;
    dll_seg_ptr->alignmentErrorRate.high_thld =
                        mib_dll_seg_defs->dllSegAlignmtErrorsRateHighThld;

    dll_seg_ptr->enetFrames.high_thld =
                        mib_dll_seg_defs->dllSegEnetFrameHighThld;
    dll_seg_ptr->enetFrameRate.high_thld =
                        mib_dll_seg_defs->dllSegEnetFrameRateHighThld;

    dll_seg_ptr->llcFrames.high_thld =
                        mib_dll_seg_defs->dllSegLlcFrameHighThld;
    dll_seg_ptr->llcFrameRate.high_thld =
                        mib_dll_seg_defs->dllSegLlcFrameRateHighThld;

    dll_seg_ptr->runtFrames.high_thld =
                        mib_dll_seg_defs->dllSegRuntFrameHighThld;
    dll_seg_ptr->runtFrameRate.high_thld =
                        mib_dll_seg_defs->dllSegRuntFrameRateHighThld;
}
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones, Kary Robertson, David M. Thompson and Gerard White

08



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David H. Thompson and
Gerald White

```
/*
 *  stats_icmp_a.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_icmp_a.c
 *
 *  Date:       6/17/91
 *
 *  Revision:   1.6
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO        Description of change. (latest first)
 *   --------  ---        ---------------------------------------------
 *
 *   06-17-91  DPD        Fixed bug in statsIcmpAddrRate (frame buckets)
 *   06-01-91  DPD        Fixed aging bug
 *   06-01-91  DPD        Fixed rate bug
 *
 */

static char stats_icmp_a_c [] = "@(#)stats_icmp_a.c    1.6";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
#include "mib_defs.h"
```

```
#include "mib_icmp.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "ntm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_icmp.h"
#include "stats_mib.h"
#include "snmpd.h"
#include "en_ctrl.h"


/*****************************************************************************
 *
 * Given an ICMP dialog, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsIcmpAgeDialog (dialog_addr_ptr)

     StatsAddrEntry      *dialog_addr_ptr;

{
     register Uint32                 xicmp_dialog_hash;
     register StatsAddrEntry        *xicmp_dialog_hash_link;
     register StatsAddrEntry        *xicmp_previous_dialog_hash_link;
```

```
    register StatsDialogLink *dialog_link_ptr;
    register StatsAddrEntry       *addr_record_ptr;
    register StatsIcmpAddr        *icmp_stats_ptr;
    struct timeval                current_time;
    Uint32                             i;

if (monCtrl.icmpDialogAgeTimer == 0)
    return;

if (dialog_addr_ptr == NULL)
    return;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - dialog_addr_ptr->lastTime < monCtrl.icmpDialogAgeTimer)
    return;

/*
 * Time to chuck this dialog.  Remove it from the dialog hash table.
 * Look up the dialog in order to set icmp_dialog_hash, icmp_previous_dialog_hash_link
 * and icmp_dialog_hash_link.
 */
addr_record_ptr = stats_icmp_lookup_dialog (
    &dialog_addr_ptr->address.netAddress1.u.ipAddress,
    &dialog_addr_ptr->address.netAddress2.u.ipAddress);

if (addr_record_ptr != dialog_addr_ptr)
    {
    mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsIcmpAgeDialog: ptrs not equal");
    return;
    }

xicmp_dialog_hash = icmp_dialog_hash;
xicmp_dialog_hash_link = icmp_dialog_hash_link;
xicmp_previous_dialog_hash_link = icmp_previous_dialog_hash_link;
```

```
if ( (xicmp_previous_dialog_hash_link == NULL) && (xicmp_dialog_hash_link->hash_link ==
NULL) )
     icmp_dialog_hash_table[xicmp_dialog_hash] = NULL;
else
     {
     if (xicmp_previous_dialog_hash_link != NULL)
          xicmp_previous_dialog_hash_link->hash_link = xicmp_dialog_hash_link->hash_link;

     else
          if (xicmp_dialog_hash_link->hash_link != NULL)
               icmp_dialog_hash_table[xicmp_dialog_hash] =
xicmp_dialog_hash_link->hash_link;
     }

/*
 * Remove the dialog from the dialogQ for each of the addresses
 * and deallocate the structure that was on the dialogQ which pointed
 * to the dialog.
 */
addr_record_ptr = stats_icmp_lookup_addr (
     &dialog_addr_ptr->address.netAddress1.u.ipAddress);

for (i=0; i<2; i++)
     {
     if (addr_record_ptr != NULL)
          {
          icmp_stats_ptr = (StatsIcmpAddr *) addr_record_ptr->stats_ptr;
          if (icmp_stats_ptr != NULL)
               {
               dialog_link_ptr = (StatsDialogLink *) icmp_stats_ptr->dialogQ.pFlink;
               while (dialog_link_ptr != NULL)
                    {
                    if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                         break;
                    dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
                    }
               if (dialog_link_ptr != NULL)
```

```
                    {
                    FBRemqm (&icmp_stats_ptr->dialogQ, (PFBQentry_type) dialog_link_ptr);

                    stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                    }
               }
          }
          addr_record_ptr = state_icmp_lookup_addr (
               &dialog_addr_ptr->address.netAddress2.u.ipAddress);
     }


/*
 * Remove the dialog from the statsIcmpDialogQ and deallocate the structures
 * associated with the dialog.
 */
FBRemqm (&statsIcmpDialogQ, (PFBQentry_type) dialog_addr_ptr);

if (dialog_addr_ptr->stats_ptr != NULL)
     stats_deallocate (dialog_addr_ptr->stats_ptr, sizeof(StatsDialogEntry) );

stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );

}


/*****************************************************************************
 *
 * Determine if address statistics should be aged out, and, if so, get rid of it.
 */

void statsIcmpAgeAddr (addr_ptr)

     StatsAddrEntry       *addr_ptr;

{
     register StatsDialogLink *dialog_link_ptr;
```

```
    register StatsIcmpAddr          *stats_ptr;
    struct timeval                  current_time;

if (monCtrl.icmpNodeAgeTimer == 0)
    return;

if (addr_ptr == NULL)
    return;

stats_ptr = (StatsIcmpAddr *) addr_ptr->stats_ptr;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - addr_ptr->lastTime < monCtrl.icmpNodeAgeTimer)
    return;

/*
 * Time to chuck these stats. Not the address - just the stats.
 */

/*
 * Remove the dialog links from the stats and deallocate the stats.
 */
if (stats_ptr != NULL)
    {
    dialog_link_ptr = (StatsDialogLink *) FBRemqh (&stats_ptr->dialogQ);
    while (dialog_link_ptr != NULL)
        {
        stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
        dialog_link_ptr = (StatsDialogLink *) FBRemqh (&stats_ptr->dialogQ);
        }
    stats_deallocate (stats_ptr, sizeof(StatsIcmpAddr) );

    addr_ptr->stats_ptr = NULL;
    }

}
```

```
/*****************************************************************************
 *
 * ICMP rate routines
 */

void statsIcmpProtocolRate (icmp_protocol_ptr, icmp_addr_ptr, rate_type, time)
     StatsProtocolEntry  *icmp_protocol_ptr;
     StatsAddrEntry                *icmp_addr_ptr;
     Uint32                             rate_type;
     Uint32                             time;
{
}


void statsIcmpDialogRate (rate_type)
     Uint32                        rate_type;
{
     register StatsDialogEntry     *icmp_dialog_ptr;
    register FBQentry_type         *entry_ptr;
    register FBQentry_type         *next_entry_ptr;
     register StatsAddrEntry      *icmp_entry_ptr;
     register Uint32                    loop_count;
     struct timeval                 current_time;

     /* get the current time */
     mon_gettimeofday(&current_time, 0);

     if (statsNextIcmpPairEntry != NULL)
          entry_ptr = statsNextIcmpPairEntry;          /* pick up where we left off */
     else
     {
          /* only age structures if we've processed them all */
          entry_ptr = statsIcmpDialogQ.pFlink;
```

```
        while (entry_ptr != NULL)
        {
            next_entry_ptr = entry_ptr->pFlink;
            statsIcmpAgeDialog ((StatsAddrEntry *) entry_ptr);
            entry_ptr = next_entry_ptr;
        }

        entry_ptr = statsIcmpDialogQ.pFlink;          /* start at the beginning */
    }

    loop_count = 0;
    while ((entry_ptr != NULL) && (loop_count < stats_q_count))
    {
        icmp_entry_ptr = (StatsAddrEntry *)entry_ptr;
        if (((icmp_entry_ptr->em_control & rate_type) != 0)
                                && (icmp_entry_ptr->stats_ptr != NULL))

        {
            icmp_dialog_ptr = (StatsDialogEntry *)icmp_entry_ptr->stats_ptr;

            if ((current_time.tv_sec - icmp_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
            {
            statsCalcRates(&icmp_dialog_ptr->packetRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_PAIR, AL_FRAMES, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_dialog_ptr->byteRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_PAIR, AL_BYTES, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_dialog_ptr->errorRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_PAIR, AL_ERRORS, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_dialog_ptr->fragmentRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_PAIR, AL_FRAGMENTS, current_time.tv_sec, NULL);
```

```
                statsCalcRates(&icmp_dialog_ptr->rexmtRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_PAIR, AL_REXMTS, current_time.tv_sec, NULL);

                statsCalcRates(&icmp_dialog_ptr->flowCtrlRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_PAIR, AL_FLOW_CTRLS, current_time.tv_sec, NULL);

                icmp_entry_ptr->seconds_start_time = current_time.tv_sec;
                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }
        statsNextIcmpPairEntry = entry_ptr;

        if (statsNextIcmpPairEntry != NULL)
            statsRatesNotDone |= STATS_ICMP_PAIR_RATE;
}


void statsIcmpSegRate (rate_type)
        Uint32                  rate_type;
{
        register StatsIcmpSegment:      *icmp_seg_ptr;
        register StatsAddrEntry         *icmp_seg_addr_ptr;
        struct timeval                  current_time;
        register StatsProtocolEntry     *icmp_protocol_ptr;

        icmp_seg_addr_ptr = (StatsAddrEntry *) statsIcmpSegQ.pFlink;

        while (icmp_seg_addr_ptr != NULL)
        {
            icmp_seg_ptr = (StatsIcmpSegment *) icmp_seg_addr_ptr->stats_ptr;
            if (icmp_seg_ptr != NULL)
```

```
        {
            /* get the current time */
            mon_gettimeofday(&current_time, 0);

            statsCalcRates(&icmp_seg_ptr->frameRate, &icmp_seg_ptr->frameBuckets,
                           rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,              AL_FRAMES, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->byteRate, &icmp_seg_ptr->byteBuckets,
                           rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,              AL_BYTES, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->errorRate, &icmp_seg_ptr->errorBuckets,
                           rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,              AL_ERRORS, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_seg_ptr->rcvOffSegRate,
&icmp_seg_ptr->rcvOffSegBuckets,
                           rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,              AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->xmtOffSegRate,
&icmp_seg_ptr->xmtOffSegBuckets,
                           rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,              AL_XMT_OFF_SEG, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->transitRate, &icmp_seg_ptr->transitBuckets,
                           rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,              AL_TRANSIT, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_seg_ptr->echoReqRate, NULL,
                           rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,              AL_ECHO, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->echoReplyRate, NULL,
```

```
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_ECHO_REPLY, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_seg_ptr->destUnrNetRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_UNREACH_NET, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->destUnrHostRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_UNREACH_HOST, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->destUnrProtocolRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_UNREACH_PROTOCOL, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->destUnrPortRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_UNREACH_PORT, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->destUnrFrgmtRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_UNREACH_FRAG, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->destUnrSrcRouteRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_UNREACH_SRC_FAIL, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->destUnrNetUnknownRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_UNREACH_NET_UNKNOWN, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->destUnrHostUnknownRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_UNREACH_HOST_UNKNOWN, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->destUnrHostIsolatedRate, NULL,
```

```
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_UNREACH_HOST_ISOLATED, current_time.tv_sec, NULL);
                statsCalcRates(&icmp_seg_ptr->destUnrNetProhibitedRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_UNREACH_NET_PROHIBITED, current_time.tv_sec, NULL);

                statsCalcRates(&icmp_seg_ptr->destUnrHostProhibitedRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_UNREACH_HOST_PROHIBITED, current_time.tv_sec, NULL);

                statsCalcRates(&icmp_seg_ptr->destUnrNetTosRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_UNREACH_NET_TOS, current_time.tv_sec, NULL);
                statsCalcRates(&icmp_seg_ptr->destUnrHostTosRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_UNREACH_HOST_TOS, current_time.tv_sec, NULL);

                statsCalcRates(&icmp_seg_ptr->srcQuenchRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_SRC_QUENCH, current_time.tv_sec, NULL);

                statsCalcRates(&icmp_seg_ptr->redirNetRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_REDIR_NET, current_time.tv_sec, NULL);
                statsCalcRates(&icmp_seg_ptr->redirHostRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_REDIR_HOST, current_time.tv_sec, NULL);
                statsCalcRates(&icmp_seg_ptr->redirNetTosRate, NULL,
```

```
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_REDIR_NET_TOS, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->redirHostTosRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_REDIR_HOST_TOS, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_seg_ptr->timeExceededInTransitRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_TIME_EXCEEDED_IN_TRANS, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_seg_ptr->timeExceededInReassRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_TIME_EXCEEDED_IN_REASS, current_time.tv_sec, NULL);


            statsCalcRates(&icmp_seg_ptr->paramProblemRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_PARAM, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->paramProblemOptionRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_PARAM_OPTION, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_seg_ptr->timestampReqRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_TIMESTAMP, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_seg_ptr->timestampReplyRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,                   AL_TIMESTAMP_REPLY, current_time.tv_sec, NULL);
```

```
                statsCalcRates(&icmp_seg_ptr->addrMaskReqRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_MASK, current_time.tv_sec, NULL);
                statsCalcRates(&icmp_seg_ptr->addrMaskReplyRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_seg_addr_ptr,
ICMP_SEGMENT,
                                AL_MASK_REPLY, current_time.tv_sec, NULL);

                statsIcmpDialogRate (rate_type);

                if ((current_time.tv_sec - icmp_seg_addr_ptr->seconds_start_time) >=
monCtrl.rateTimer)
                    icmp_seg_addr_ptr->seconds_start_time = current_time.tv_sec;

            }
            icmp_seg_addr_ptr = (StatsAddrEntry *) icmp_seg_addr_ptr->link.pFlink;
        }
}


void statsIcmpAddrRate (rate_type)
    Uint32                      rate_type;
{
    register StatsIcmpAddr          *icmp_addr_ptr;
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
    register StatsAddrEntry         *icmp_entry_ptr;
    register Uint32                      loop_count;
    struct timeval                  current_time;
    register StatsProtocolEntry     *icmp_protocol_ptr;

    /* get the current time */
    mon_gettimeofday(&current_time, 0);

    loop_count = 0;
```

```
    /***** AGING *****/
    if (statsNextIcmpAddrEntry != NULL)
         entry_ptr = statsNextIcmpAddrEntry;          /* pick up where we left off */
    else
    {
         /* Only age structures if we've processed them all */
         entry_ptr = statsIcmpAddrQ.pFlink;
         while (entry_ptr != NULL)
         {
              next_entry_ptr = entry_ptr->pFlink;
              statsIcmpAgeAddr ((StatsAddrEntry *) entry_ptr);
              entry_ptr = next_entry_ptr;
         }

         entry_ptr = statsIcmpAddrQ.pFlink;      /* start at the beginning */
    }
    /***** AGING *****/

    while ((entry_ptr != NULL) && (loop_count < stats_q_count))
    {
         icmp_entry_ptr = (StatsAddrEntry *)entry_ptr;
         if (((icmp_entry_ptr->em_control & rate_type) != 0)
                              && (icmp_entry_ptr->stats_ptr != NULL))

         {
              icmp_addr_ptr = (StatsIcmpAddr *)icmp_entry_ptr->stats_ptr;

              if ((current_time.tv_sec - icmp_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
              {
              statsCalcRates(&icmp_addr_ptr->frameRate, &icmp_addr_ptr->frameBuckets,
                              rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                              ICMP_NODE, AL_FRAMES, current_time.tv_sec, NULL);
              statsCalcRates(&icmp_addr_ptr->rcvFrameRate, NULL,
                              rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                              ICMP_NODE, AL_RCV_FRAMES, current_time.tv_sec, NULL);
              statsCalcRates(&icmp_addr_ptr->xmtFrameRate, NULL,
                              rate_type, (StatsAddrEntry *)icmp_entry_ptr,
```

```
                                ICMP_NODE, AL_XMT_FRAMES, current_time.tv_sec, NULL);

        statsCalcRates(&icmp_addr_ptr->byteRate, &icmp_addr_ptr->byteBuckets,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_BYTES, current_time.tv_sec, NULL);
        statsCalcRates(&icmp_addr_ptr->rcvByteRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_BYTES, current_time.tv_sec, NULL);
        statsCalcRates(&icmp_addr_ptr->xmtByteRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_BYTES, current_time.tv_sec, NULL);

        statsCalcRates(&icmp_addr_ptr->errorRate, &icmp_addr_ptr->errorBuckets,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_ERRORS, current_time.tv_sec, NULL);
        statsCalcRates(&icmp_addr_ptr->rcvErrorRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_ERRORS, current_time.tv_sec, NULL);
        statsCalcRates(&icmp_addr_ptr->xmtErrorRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_ERRORS, current_time.tv_sec, NULL);

        statsCalcRates(&icmp_addr_ptr->rcvOffSegRate,
&icmp_addr_ptr->rcvOffSegBuckets,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_OFF_SEG, current_time.tv_sec, NULL);

        statsCalcRates(&icmp_addr_ptr->xmtOffSegRate,
&icmp_addr_ptr->xmtOffSegBuckets,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_OFF_SEG, current_time.tv_sec, NULL);


        statsCalcRates(&icmp_addr_ptr->rcvEchoReqRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_ECHO, current_time.tv_sec, NULL);
        statsCalcRates(&icmp_addr_ptr->xmtEchoReqRate, NULL,
```

```
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_ECHO, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_addr_ptr->rcvEchoReplyRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_ECHO_REPLY, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->xmtEchoReplyRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_ECHO_REPLY, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvDestUnrNetRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_NET, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvDestUnrHostRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_HOST, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvDestUnrProtocolRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_PROTOCOL,
current_time.tv_sec, NULL);
            statsCalcRates(&icmp_addr_ptr->rcvDestUnrPortRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_PORT, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvDestUnrFrgmtRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_FRAG, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvDestUnrSrcRouteRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_SRC_FAIL,
current_time.tv_sec, NULL);
            statsCalcRates(&icmp_addr_ptr->rcvDestUnrNetUnknownRate, NULL,
```

```
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_NET_UNKNOWN,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->rcvDestUnrHostUnknownRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_HOST_UNKNOWN,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->rcvDestUnrHostIsolatedRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_HOST_ISOLATED,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->rcvDestUnrNetProhibitedRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_NET_PROHIBITED,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->rcvDestUnrHostProhibitedRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_HOST_PROHIBITED,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->rcvDestUnrNetTosRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_NET_TOS, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->rcvDestUnrHostTosRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_UNREACH_HOST_TOS,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrNetRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_NET, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrHostRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_HOST, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrProtocolRate, NULL,
```

```
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_PROTOCOL,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrPortRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_PORT, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrFrgmtRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_FRAG, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrSrcRouteRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_SRC_FAIL,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrNetUnknownRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_NET_UNKNOWN,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrHostUnknownRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_HOST_UNKNOWN,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrHostIsolatedRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_HOST_ISOLATED,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrNetProhibitedRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_NET_PROHIBITED,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrHostProhibitedRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_UNREACH_HOST_PROHIBITED,
current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->xmtDestUnrNetTosRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
```

```
                                ICMP_NODE, AL_XMT_UNREACH_NET_TOS, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->xmtDestUnrHostTosRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_UNREACH_HOST_TOS,
current_time.tv_sec, NULL);
            statsCalcRates(&icmp_addr_ptr->rcvSrcQuenchRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_SRC_QUENCH, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->xmtSrcQuenchRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_SRC_QUENCH, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvRedirNetRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_REDIR_NET, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvRedirHostRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_REDIR_HOST, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvRedirNetTosRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_REDIR_NET_TOS, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvRedirHostTosRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_REDIR_HOST_TOS, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->xmtRedirNetRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_REDIR_NET, current_time.tv_sec,
NULL);
```

```
                statsCalcRates(&icmp_addr_ptr->xmtRedirHostRate, NULL,
                               rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                               ICMP_NODE, AL_XMT_REDIR_HOST, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->xmtRedirNetTosRate, NULL,
                               rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                               ICMP_NODE, AL_XMT_REDIR_NET_TOS, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->xmtRedirHostTosRate, NULL,
                               rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                               ICMP_NODE, AL_XMT_REDIR_HOST_TOS, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->rcvTimeExceededInTransitRate, NULL,
                               rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                               ICMP_NODE, AL_RCV_TIME_EXCEEDED, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->rcvTimeExceededInReassRate, NULL,
                               rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                               ICMP_NODE, AL_RCV_TIME_EXCEEDED, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->xmtTimeExceededInTransitRate, NULL,
                               rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                               ICMP_NODE, AL_XMT_TIME_EXCEEDED, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->xmtTimeExceededInReassRate, NULL,
                               rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                               ICMP_NODE, AL_XMT_TIME_EXCEEDED, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->rcvParamProblemRate, NULL,
                               rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                               ICMP_NODE, AL_RCV_PARAM, current_time.tv_sec, NULL);
                statsCalcRates(&icmp_addr_ptr->rcvParamProblemOptionRate, NULL,
                               rate_type, (StatsAddrEntry *)icmp_entry_ptr,
```

```
                                ICMP_NODE, AL_RCV_PARAM_OPTION, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->xmtParamProblemRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_PARAM, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_addr_ptr->xmtParamProblemOptionRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_PARAM_OPTION, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvTimestampReqRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_TIMESTAMP, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvTimestampReplyRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_TIMESTAMP_REPLY, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->xmtTimestampReqRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_TIMESTAMP, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->xmtTimestampReplyRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_TIMESTAMP_REPLY, current_time.tv_sec,
NULL);
            statsCalcRates(&icmp_addr_ptr->rcvAddrMaskReqRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_RCV_MASK, current_time.tv_sec, NULL);
            statsCalcRates(&icmp_addr_ptr->xmtAddrMaskReqRate, NULL,
                                rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                ICMP_NODE, AL_XMT_MASK, current_time.tv_sec, NULL);

            statsCalcRates(&icmp_addr_ptr->rcvAddrMaskReplyRate, NULL,
```

```
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_RCV_MASK_REPLY, current_time.tv_sec,
NULL);
                statsCalcRates(&icmp_addr_ptr->xmtAddrMaskReplyRate, NULL,
                                    rate_type, (StatsAddrEntry *)icmp_entry_ptr,
                                    ICMP_NODE, AL_XMT_MASK_REPLY, current_time.tv_sec,
NULL);
                icmp_entry_ptr->seconds_start_time = current_time.tv_sec;
                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }
        statsNextIcmpAddrEntry = entry_ptr;          /* pick up where we left off */

        if (statsNextIcmpAddrEntry != NULL)
            statsRatesNotDone |= STATS_ICMP_ADDR_RATE;
}
```

```
/*
 *  stats_icmp_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                   All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_icmp_p.c
 *
 *  Date:       5/29/91
 *
 *  Revision:   1.4
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 */

static char stats_icmp_p_c [] = "@(#)stats_icmp_p.c    1.4";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
#include "mib_defs.h"
#include "mib_icmp.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
```

```
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_icmp.h"
#include "stats_mib.h"
#include "snmpd.h"
#include "en_ctrl.h"


/*
 * Global Data Structures
 */

StatsAddrEntry      *icmp_this_seg_addr_ptr;
StatsIcmpSegment    *icmp_this_seg_stats_ptr;

StatsAddrEntry      *icmp_src_seg_addr_ptr, *icmp_dst_seg_addr_ptr;
StatsIcmpSegment    *icmp_src_seg_stats_ptr, *icmp_dst_seg_stats_ptr;

StatsAddrEntry      *icmp_src_node_addr_ptr, *icmp_dst_node_addr_ptr;
StatsIcmpAddr       *icmp_src_node_stats_ptr, *icmp_dst_node_stats_ptr;

StatsAddrEntry      *icmp_dialog_addr_ptr;
StatsDialogEntry    *icmp_dialog_stats_ptr;

StatsAddrEntry      *icmp_hash_table[ICMP_HASH_TABLE_SIZE];
StatsAddrEntry      *icmp_dialog_hash_table[ICMP_DIALOG_HASH_TABLE_SIZE];
```

```
/*
 * Local data structures
 */
Uint32          icmp_hash;
StatsAddrEntry  *icmp_hash_link;
StatsAddrEntry  *icmp_previous_hash_link;
Uint32          icmp_dialog_hash;
StatsAddrEntry  *icmp_dialog_hash_link;
StatsAddrEntry  *icmp_previous_dialog_hash_link;


/*************************************************************************
 *
 * Look for the segment address structure.
 * If no match is found, NULL is returned.
 */

StatsAddrEntry *stats_icmp_lookup_segment (segment)

     Uint32                    segment;
{
     register StatsAddrEntry       *seg_addr_ptr;

seg_addr_ptr = (StatsAddrEntry *) statsIcmpSegQ.pFlink;

while (seg_addr_ptr != NULL)
     {
     if (seg_addr_ptr->address.segment1 == segment)
          break;
     seg_addr_ptr = (StatsAddrEntry *) seg_addr_ptr->link.pFlink;
     }

     return (seg_addr_ptr);
}
```

```
/*****************************************************************
 *
 * Find the structure for keeping Icmp segment statistics for the given
 * segment.  If one is not found, attempt to allocate one.
 */

StatsAddrEntry *stats_icmp_get_segment (segment)

     Uint32                         segment;
{
     register StatsAddrEntry        *seg_addr_ptr;
     register StatsIcmpSegment      *seg_stats_ptr;

 seg_addr_ptr = stats_icmp_lookup_segment (segment);

/*
 * If not found, try to allocate a structure for this segment.
 * If one can't be obtained, count this as a drop.
 */
 if (seg_addr_ptr == NULL)
     {
     seg_addr_ptr = (StatsAddrEntry *) stats_allocate ( sizeof(StatsAddrEntry) );
     if (seg_addr_ptr != NULL)
          {
          seg_addr_ptr->address.addressType  = MibSegment1;
          seg_addr_ptr->address.segment1          = segment;
          seg_addr_ptr->parse_control        = monCtrl.segParseCtrl;
          seg_addr_ptr->startTime            = stats_start_time.tv_sec;
          seg_addr_ptr->lestTime                  = stats_start_time.tv_sec;
          FBInsqt (&statsIcmpSegQ, (PFBQentry_type) seg_addr_ptr);
          }
     else
          {
          stats_mon_icmp_dropped;
```

```
            return (NULL);
            }
        }

    /*
     * If there's an address structure, see if there's a statistics structure
     * for Icmp attached, if not and parse control allows, allocate one.
     * If one can't be obtained, count this as a drop.
     */
     if (seg_addr_ptr != NULL)
         {
         seg_stats_ptr = (StatsIcmpSegment *) seg_addr_ptr->stats_ptr;
         if (seg_stats_ptr == NULL)
              {
              /*
               * If Icmp parsing is enabled, allocate Icmp statistics structure if
               * one has not already been allocated.  If one can't be obtained, count
               * this as a drop.
               */
              if (seg_addr_ptr->parse_control & MibParseIcmp)
                   {
                   seg_stats_ptr = (StatsIcmpSegment *) stats_allocate (sizeof
    (StatsIcmpSegment) );
                   if (seg_stats_ptr == NULL)
                        stats_mon_icmp_dropped;
                   else
                        {
                        seg_addr_ptr->stats_ptr = (Uint32 *) seg_stats_ptr;
                        seg_stats_ptr->frameRate.type                    = STATS_RATE_10S;
                        seg_stats_ptr->byteRate.type                     = STATS_RATE_10S;
                        seg_stats_ptr->errorRate.type                    = STATS_RATE_10S;
                        seg_stats_ptr->rcvOffSegRate.type                = STATS_RATE_10S;
                        seg_stats_ptr->xmtOffSegRate.type                = STATS_RATE_10S;
                        seg_stats_ptr->transitRate.type                  =
    STATS_RATE_10S;
                        seg_stats_ptr->echoReqRate.type                  =
    STATS_RATE_10S;
```

```
                seg_stats_ptr->echoReplyRate.type                    = STATS_RATE_10S;
                seg_stats_ptr->destUnrNetRate.type                   = STATS_RATE_10S;
                seg_stats_ptr->destUnrHostRate.type                      =
STATS_RATE_10S;
                seg_stats_ptr->destUnrProtocolRate.type              = STATS_RATE_10S;
                seg_stats_ptr->destUnrPortRate.type                      =
STATS_RATE_10S;
                seg_stats_ptr->destUnrFrgmtRate.type                 = STATS_RATE_10S;
                seg_stats_ptr->destUnrSrcRouteRate.type              = STATS_RATE_10S;
                seg_stats_ptr->destUnrNetUnknownRate.type            = STATS_RATE_10S;
                seg_stats_ptr->destUnrHostUnknownRate.type           = STATS_RATE_10S;
                seg_stats_ptr->destUnrHostIsolatedRate.type          = STATS_RATE_10S;
                seg_stats_ptr->destUnrNetProhibitedRate.type         = STATS_RATE_10S;
                seg_stats_ptr->destUnrHostProhibitedRate.type        = STATS_RATE_10S;
                seg_stats_ptr->destUnrNetTosRate.type                = STATS_RATE_10S;
                seg_stats_ptr->destUnrHostTosRate.type               = STATS_RATE_10S;
                seg_stats_ptr->srcQuenchRate.type                    = STATS_RATE_10S;
                seg_stats_ptr->redirNetRate.type                     = STATS_RATE_10S;
                seg_stats_ptr->redirHostRate.type                        =
                seg_stats_ptr->redirNetTosRate.type
STATS_RATE_10S;
                seg_stats_ptr->redirHostTosRate.type                 = STATS_RATE_10S;
                seg_stats_ptr->timeExceededInTransitRate.type        = STATS_RATE_10S;
                seg_stats_ptr->timeExceededInReassRate.type          = STATS_RATE_10S;
                seg_stats_ptr->paramProblemRate.type                 = STATS_RATE_10S;
                seg_stats_ptr->paramProblemOptionRate.type           = STATS_RATE_10S;
                seg_stats_ptr->timestampReqRate.type                 = STATS_RATE_10S;
                seg_stats_ptr->timestampReplyRate.type                   =
                seg_stats_ptr->addrMaskReqRate.type
STATS_RATE_10S;
                seg_stats_ptr->addrMaskReplyRate.type                = STATS_RATE_10S;

                setIcmpSegDflts (&mibSegDefaults.mibIcmpSegDefaults, seg_stats_ptr);
                }
            }
        }
    }
```

```
return (seg_addr_ptr);
}


/*****************************************************************************
 *
 *  Find the icmp address record.
 *  A hash is done on the ip address and a pointer to the first
 *  StatsAddrEntry with the same hash is found.  Structures with the
 *  same hash are linked.  The link must be walked and the ip address
 *  compared with the ip address in each of the structures until a match
 *  is found.  If no match is found, NULL is returned.
 */
StatsAddrEntry *stats_icmp_lookup_addr (ip_addr)

Uint32          *ip_addr;
{
Uint32          i;

/*
 * Compute ip address hash to get index into hash table
 */
icmp_hash = (*ip_addr & 0xffff0000) + (*ip_addr & 0x0000ffff);
icmp_hash = ((icmp_hash + ((icmp_hash & 0xff00) >> 8)) & (ICMP_HASH_TABLE_SIZE - 1));

icmp_previous_hash_link = NULL;
icmp_hash_link = (StatsAddrEntry *) icmp_hash_table[icmp_hash];

/*
 * Walk linked list for exact entry
 */
while (icmp_hash_link != NULL)
     {
     if (icmp_hash_link->address.netAddress1.u.ipAddress == *ip_addr)
          return (icmp_hash_link);
```

```
    else
        {
        icmp_previous_hash_link = icmp_hash_link;
        icmp_hash_link = icmp_hash_link->hash_link;
        }
    }

/*
 * No entry found.
 */
return (NULL);
}


/*****************************************************************************
 *
 * Allocate a stats structure if parse control is turned on.
 *
 */

Uint32 stats_icmp_get_stats (icmp_addr_record_ptr)

    StatsAddrEntry      *icmp_addr_record_ptr;
{
    register StatsIcmpAddr          *icmp_addr_stats_ptr;

    if ((icmp_addr_record_ptr != NULL) && (icmp_addr_record_ptr->stats_ptr == NULL))
    {
        /*
         * If parse control is turned on for Icmp, allocate a structure for Icmp
         * address statistics and initialize it.
         */
        if (icmp_addr_record_ptr->parse_control & MibParseIcmp)
        {
```

```
                icmp_addr_record_ptr->stats_ptr = (Uint32 *) stats_allocate (sizeof
(StatsIcmpAddr) );
                icmp_addr_stats_ptr = (StatsIcmpAddr *) icmp_addr_record_ptr->stats_ptr;
                if (icmp_addr_stats_ptr != NULL)
                {
                    icmp_addr_stats_ptr->frameRate.type                    =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->rcvFrameRate.type                 =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->xmtFrameRate.type                 =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->byteRate.type                     =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->rcvByteRate.type                  =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->xmtByteRate.type                  =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->errorRate.type                    =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->rcvErrorRate.type                 =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->xmtErrorRate.type                 =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->rcvOffSegRate.type                =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->xmtOffSegRate.type                =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->rcvEchoReqRate.type               =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->xmtEchoReqRate.type               =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->rcvEchoReplyRate.type             =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->xmtEchoReplyRate.type             =
STATS_RATE_10S;
                    icmp_addr_stats_ptr->rcvDestUnrNetRate.type            =
STATS_RATE_10S;
```

```
                    icmp_addr_stats_ptr->xmtDestUnrNetRate.type                =
STATS_RATE_10S;     icmp_addr_stats_ptr->rcvDestUnrHostRate.type               =
STATS_RATE_10S;     icmp_addr_stats_ptr->xmtDestUnrHostRate.type               =
STATS_RATE_10S;     icmp_addr_stats_ptr->rcvDestUnrProtocolRate.type           =
STATS_RATE_10S;     icmp_addr_stats_ptr->xmtDestUnrProtocolRate.type           =
STATS_RATE_10S;     icmp_addr_stats_ptr->rcvDestUnrPortRate.type               =
STATS_RATE_10S;     icmp_addr_stats_ptr->xmtDestUnrPortRate.type               =
STATS_RATE_10S;     icmp_addr_stats_ptr->rcvDestUnrFrgmtRate.type              =
STATS_RATE_10S;     icmp_addr_stats_ptr->xmtDestUnrFrgmtRate.type              =
STATS_RATE_10S;     icmp_addr_stats_ptr->rcvDestUnrSrcRouteRate.type           =
STATS_RATE_10S;     icmp_addr_stats_ptr->xmtDestUnrSrcRouteRate.type           =
STATS_RATE_10S;     icmp_addr_stats_ptr->rcvDestUnrNetUnknownRate.type         =
STATS_RATE_10S;     icmp_addr_stats_ptr->xmtDestUnrNetUnknownRate.type         =
STATS_RATE_10S;     icmp_addr_stats_ptr->rcvDestUnrHostUnknownRate.type        =
STATS_RATE_10S;     icmp_addr_stats_ptr->xmtDestUnrHostUnknownRate.type        =
STATS_RATE_10S;     icmp_addr_stats_ptr->rcvDestUnrHostIsolatedRate.type       =
STATS_RATE_10S;     icmp_addr_stats_ptr->xmtDestUnrHostIsolatedRate.type       =
STATS_RATE_10S;     icmp_addr_stats_ptr->rcvDestUnrNetProhibitedRate.type      =
STATS_RATE_10S;
```

```
                icmp_addr_stats_ptr->xmtDestUnrNetProhibitedRate.type  =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvDestUnrHostProhibitedRate.type =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtDestUnrHostProhibitedRate.type =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvDestUnrNetTosRate.type         =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtDestUnrNetTosRate.type         =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtDestUnrHostTosRate.type        =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvDestUnrHostTosRate.type        =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvSrcQuenchRate.type             =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtSrcQuenchRate.type             =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvRedirNetRate.type              =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtRedirNetRate.type              =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvRedirHostRate.type             =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtRedirHostRate.type             =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvRedirNetTosRate.type           =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtRedirNetTosRate.type           =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvRedirHostTosRate.type          =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtRedirHostTosRate.type          =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvTimeExceededInTransitRate.type =
STATS_RATE_10S;
```

```
                icmp_addr_stats_ptr->xmtTimeExceededInTransitRate.type =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvTimeExceededInReassRate.type   =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtTimeExceededInReassRate.type   =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvParamProblemRate.type                =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtParamProblemRate.type                =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvParamProblemOptionRate.type          =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtParamProblemOptionRate.type          =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvTimestampReqRate.type                =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtTimestampReqRate.type                =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvTimestampReplyRate.type              =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtTimestampReplyRate.type              =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvAddrMaskReqRate.type           =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtAddrMaskReqRate.type           =
STATS_RATE_10S;
                icmp_addr_stats_ptr->rcvAddrMaskReplyRate.type               =
STATS_RATE_10S;
                icmp_addr_stats_ptr->xmtAddrMaskReplyRate.type               =
STATS_RATE_10S;

                FBInitqh (&icmp_addr_stats_ptr->dialogQ);

                setIcmpAddrDflts (&mibNodeDefaults.mibIcmpAddrDefaults,
icmp_addr_stats_ptr);
              }
              else
```

```
                return (FALSE);
            }
        }
        return (TRUE);
}



/************************************************************************
 *
 * Find the structure for keeping icmp address statistics for the given
 * ip address.  If one not found, attempt to allocate one.
 */

StatsAddrEntry *stats_icmp_get_addr (ip_addr)

    Uint32              *ip_addr;
{
    register StatsAddrEntry     *icmp_addr_record_ptr;
    register StatsIcmpAddr      *icmp_addr_stats_ptr;
    register StatsAddrEntry     *ip_addr_record_ptr;


 icmp_addr_record_ptr = stats_icmp_lookup_addr (ip_addr);

 if (icmp_addr_record_ptr == NULL)
     {
     /*
      * Try to allocate a statistics structure for this address.
      * If icmp is turned on for this ip address, but a structure can't
      * be obtained, count this as a drop.  If icmp is not turned on, just
      * return NULL.
      */
     ip_addr_record_ptr = stats_ip_lookup_addr (ip_addr);
     if (ip_addr_record_ptr == NULL)
```

```
        return (NULL);

    if ((ip_addr_record_ptr->parse_control & MibParseIcmp) != MibParseIcmp)
        return (NULL);

    icmp_addr_record_ptr = (StatsAddrEntry *)stats_allocate (sizeof (StatsAddrEntry) );
    if (icmp_addr_record_ptr == NULL)
        {
        stats_mon_icmp_dropped;
        return (NULL);
        }
    else
        {
        icmp_addr_record_ptr->hash_link                                    = NULL;
        icmp_addr_record_ptr->address.addressType                          = MibNetAddress1 |
MibSegment1;
        icmp_addr_record_ptr->address.netAddress1.netAddressType= NetTcpIp;
        icmp_addr_record_ptr->address.netAddress1.length        = 4;
        icmp_addr_record_ptr->address.netAddress1.u.ipAddress   = *ip_addr;
        icmp_addr_record_ptr->startTime                                    =
stats_start_time.tv_sec;
        icmp_addr_record_ptr->lastTime
stats_start_time.tv_sec;

        FBInsqt (&statsIcmpAddrQ, (PFBQentry_type) icmp_addr_record_ptr);

        icmp_addr_record_ptr->parse_control = ip_addr_record_ptr->parse_control;
        icmp_addr_record_ptr->address.segment1 = ip_addr_record_ptr->address.segment1;
        }

    /*
     * If parse control is turned on, but a structure can't be obtained,
     * stats_icmp_get_stats will return FALSE, so count this as a drop.
     */
    if (stats_icmp_get_stats (icmp_addr_record_ptr) == FALSE)
        stats_mon_icmp_dropped;
```

```
    /*
     * Put this icmp address record into the hash table.
     * Variable hash was set when stats_icmp_lookup_addr was executed.
     */
    if ( icmp_hash_table[icmp_hash] == NULL)
         icmp_hash_table[icmp_hash] = icmp_addr_record_ptr;
    else
         {
         /*
          * Find the last structure of the hash link
          */
         icmp_hash_link = icmp_hash_table[icmp_hash];

         while (icmp_hash_link->hash_link != NULL)
              icmp_hash_link = icmp_hash_link->hash_link;

         icmp_hash_link->hash_link = icmp_addr_record_ptr;
         }
    )

return (icmp_addr_record_ptr);
}


/************************ stats_icmp_lookup_dialog *******************************
 *
 * Find a icmp dialog given 2 ip addresses.
 * If the dialog is not found, NULL is returned.
 */

StatsAddrEntry *stats_icmp_lookup_dialog (icmp_addr1, icmp_addr2)

    Uint32                    *icmp_addr1, *icmp_addr2;

{
    register Uint32               xicmp_dialog_hash;
    register StatsAddrEntry       *xicmp_dialog_hash_link;
```

```
    register StatsAddrEntry        *xicmp_previous_dialog_hash_link;

    register StatsAddrEntry        *dialog_addr_ptr;
/*
 * Compute hash function on both IP addresses
 */
xicmp_dialog_hash = (*icmp_addr1 & 0xffff0000) + (*icmp_addr1 & 0x0000ffff) +
                    (*icmp_addr2 & 0xffff0000) + (*icmp_addr2 & 0x0000ffff);
xicmp_dialog_hash = ((xicmp_dialog_hash + ((xicmp_dialog_hash & 0xff00) >> 8))
                     & (ICMP_DIALOG_HASH_TABLE_SIZE - 1));

xicmp_previous_dialog_hash_link = NULL;
xicmp_dialog_hash_link = (StatsAddrEntry *) icmp_dialog_hash_table[xicmp_dialog_hash];

/*
 * Walk linked list for exact entry
 */
while (xicmp_dialog_hash_link != NULL)
     {
     if (((xicmp_dialog_hash_link->address.netAddress1.u.ipAddress == *icmp_addr1) &&
          (xicmp_dialog_hash_link->address.netAddress2.u.ipAddress == *icmp_addr2)) ||

         ((xicmp_dialog_hash_link->address.netAddress1.u.ipAddress == *icmp_addr2) &&
          (xicmp_dialog_hash_link->address.netAddress2.u.ipAddress == *icmp_addr1) ))
              {
              break;
              }
         else
              {
              xicmp_previous_dialog_hash_link = xicmp_dialog_hash_link;
              xicmp_dialog_hash_link = xicmp_dialog_hash_link->hash_link;
              }
     }

icmp_dialog_hash = xicmp_dialog_hash;
icmp_dialog_hash_link = xicmp_dialog_hash_link;
```

```
icmp_previous_dialog_hash_link = xicmp_previous_dialog_hash_link;

return (xicmp_dialog_hash_link);
}


/************************ stats_icmp_get_dialog ******************************
 *
 * Find or allocate an icmp dialog given 2 ip addresses.
 * If the dialog is not found, attempt to allocate a structure.
 */
StatsAddrEntry *stats_icmp_get_dialog (icmp_addr1, icmp_addr2)

     Uint32                     *icmp_addr1, *icmp_addr2;
{
     register StatsAddrEntry        *dialog_addr_ptr;
     register StatsDialogEntry      *dialog_stats_ptr;
     register StatsDialogLink *dialog_link;
     register StatsAddrEntry        *icmp_addr1_record_ptr, *icmp_addr2_record_ptr;
     register StatsIcmpAddr         *icmp_addr1_stats_ptr, *icmp_addr2_stats_ptr;
     register StatsAddrEntry        *ip_addr1_record_ptr, *ip_addr2_record_ptr;
     register Uint32                     parse_control;


dialog_addr_ptr = stats_icmp_lookup_dialog (icmp_addr1, icmp_addr2);
if (dialog_addr_ptr != NULL)
     return (dialog_addr_ptr);

/*
 * Check parse control for the ip addresses.
 */
ip_addr1_record_ptr = stats_ip_lookup_addr (icmp_addr1);
ip_addr2_record_ptr = stats_ip_lookup_addr (icmp_addr2);

if ( (ip_addr1_record_ptr == NULL) && (ip_addr2_record_ptr == NULL) )
     return (NULL);
```

```
if (ip_addr1_record_ptr != NULL)
     parse_control = ip_addr1_record_ptr->parse_control;
if (ip_addr2_record_ptr != NULL)
     parse_control |= ip_addr2_record_ptr->parse_control;


/*
 * If icmp is turned on for these ip addresses, allocate structures
 * for the dialog.  If can't get them, count this as a drop.
 */
if ((parse_control & MibParseIcmp) != MibParseIcmp)
     return (NULL);

/*
 * Try to allocate structures for this dialog.
 * If they can't be obtained, count this as a drop.
 */
dialog_addr_ptr = (StatsAddrEntry *) stats_allocate (sizeof (StatsAddrEntry) );
if (dialog_addr_ptr == NULL)
     {
     stats_mon_icmp_dropped;
     return (NULL);
     }

dialog_stats_ptr = (StatsDialogEntry *) stats_allocate (sizeof (StatsDialogEntry) );
if (dialog_stats_ptr == NULL)
     {
     stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );
     stats_mon_icmp_dropped;
     return (NULL);
     }

/*
 * Initialize the structures
 */
dialog_addr_ptr->hash_link                                    = NULL;
```

```
dialog_addr_ptr->address.addressType                    = MibNetAddress1 | MibNetAddress2;

dialog_addr_ptr->address.netAddress1.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress1.length         = 4;
dialog_addr_ptr->address.netAddress1.u.ipAddress    = *icmp_addr1;
dialog_addr_ptr->address.netAddress2.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress2.length         = 4;
dialog_addr_ptr->address.netAddress2.u.ipAddress    = *icmp_addr2;
dialog_addr_ptr->parse_control                              = parse_control;
dialog_addr_ptr->startTime                                  = stats_start_time.tv_sec;
dialog_addr_ptr->lastTime                                   = stats_start_time.tv_sec;
dialog_addr_ptr->stats_ptr                                  = (Uint32 *) dialog_stats_ptr;


FBInsqt (&statsIcmpDialogQ, (FFBQentry_type) dialog_addr_ptr);

dialog_stats_ptr->packetRate.type   = STATS_RATE_10S;
dialog_stats_ptr->byteRate.type     = STATS_RATE_10S;
dialog_stats_ptr->errorRate.type    = STATS_RATE_10S;
dialog_stats_ptr->fragmentRate.type = STATS_RATE_10S;
dialog_stats_ptr->rexmtRate.type    = STATS_RATE_10S;
dialog_stats_ptr->flowCtrlRate.type = STATS_RATE_10S;

setIcmpAddrPairDflts (&mibNodeDefaults.mibIcmpAddrPairDefaults, dialog_stats_ptr);

/*
 * Link the dialog statistics into the dialog queue for each icmp address stats.
 */
icmp_addr1_record_ptr = stats_icmp_lookup_addr (icmp_addr1);
icmp_addr2_record_ptr = stats_icmp_lookup_addr (icmp_addr2);

if (icmp_addr1_record_ptr != NULL)
     {
     if (icmp_addr1_record_ptr->stats_ptr != NULL)
          {
          dialog_addr_ptr->address.addressType |= MibSegment1;
          dialog_addr_ptr->address.segment1 = icmp_addr1_record_ptr->address.segment1;
```

```
            icmp_addr1_stats_ptr = (StatsIcmpAddr *) icmp_addr1_record_ptr->stats_ptr;
            dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
            if (dialog_link != NULL)
                 {
                 dialog_link->dialog_addr_ptr = dialog_addr_ptr;
                 FBInsqt (&icmp_addr1_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
                 }
            }
       }
if (icmp_addr2_record_ptr != NULL)
       {
       if (icmp_addr2_record_ptr->stats_ptr != NULL)
            {
            dialog_addr_ptr->address.addressType |= MibSegment2;
            dialog_addr_ptr->address.segment2 = icmp_addr2_record_ptr->address.segment1;
            icmp_addr2_stats_ptr = (StatsIcmpAddr *) icmp_addr2_record_ptr->stats_ptr;
            dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
            if (dialog_link != NULL)
                 {
                 dialog_link->dialog_addr_ptr = dialog_addr_ptr;
                 FBInsqt (&icmp_addr2_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
                 }
            }
       }

/*
 * Put the new dialog address structure in the icmp dialog hash table.
 * Variable hash was set up when stats_icmp_lookup_dialog was executed.
 */
if (icmp_dialog_hash_table[icmp_dialog_hash] == NULL)
       icmp_dialog_hash_table[icmp_dialog_hash] = dialog_addr_ptr;
else
       {
       /*
        * Find the last structure of the hash link.
        */
       icmp_dialog_hash_link = icmp_dialog_hash_table[icmp_dialog_hash];
```

```
    while (icmp_dialog_hash_link->hash_link != NULL)
        icmp_dialog_hash_link = icmp_dialog_hash_link->hash_link;

    icmp_dialog_hash_link->hash_link = dialog_addr_ptr;
    }

return (dialog_addr_ptr);
}


void setIcmpAddrDflts (mib_icmp_addr_defs, icmp_stats_addr_ptr)
    MibIcmpAddrDefaults *mib_icmp_addr_defs;
    StatsIcmpAddr       *icmp_stats_addr_ptr;
{
    icmp_stats_addr_ptr->rcvFrames.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvPktHighThld;
    icmp_stats_addr_ptr->rcvFrameRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvPktRateHighThld;

    icmp_stats_addr_ptr->rcvBytes.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvByteHighThld;
    icmp_stats_addr_ptr->rcvByteRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvByteRateHighThld;

    icmp_stats_addr_ptr->rcvErrors.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvErrorHighThld;
    icmp_stats_addr_ptr->rcvErrorRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvErrorRateHighThld;

    icmp_stats_addr_ptr->xmtFrames.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtPktHighThld;
    icmp_stats_addr_ptr->xmtFrameRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtPktRateHighThld;

    icmp_stats_addr_ptr->xmtBytes.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtByteHighThld;
```

```
icmp_stats_addr_ptr->xmtByteRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtByteRateHighThld;

icmp_stats_addr_ptr->xmtErrors.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtErrorHighThld;
icmp_stats_addr_ptr->xmtErrorRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtErrorRateHighThld;

icmp_stats_addr_ptr->rcvOffSegs.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvOffSegHighThld;
icmp_stats_addr_ptr->rcvOffSegRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvOffSegRateHighThld;

icmp_stats_addr_ptr->xmtOffSegs.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtOffSegHighThld;
icmp_stats_addr_ptr->xmtOffSegRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtOffSegRateHighThld;

icmp_stats_addr_ptr->rcvEchoReq.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvEchoReqHighThld;
icmp_stats_addr_ptr->rcvEchoReqRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvEchoReqRateHighThld;

icmp_stats_addr_ptr->xmtEchoReq.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtEchoReqHighThld;
icmp_stats_addr_ptr->xmtEchoReqRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtEchoReqRateHighThld;

icmp_stats_addr_ptr->rcvEchoReply.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvEchoRespHighThld;
icmp_stats_addr_ptr->rcvEchoReplyRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvEchoRespRateHighThld;

icmp_stats_addr_ptr->xmtEchoReply.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtEchoRespHighThld;
icmp_stats_addr_ptr->xmtEchoReplyRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtEchoRespRateHighThld;
```

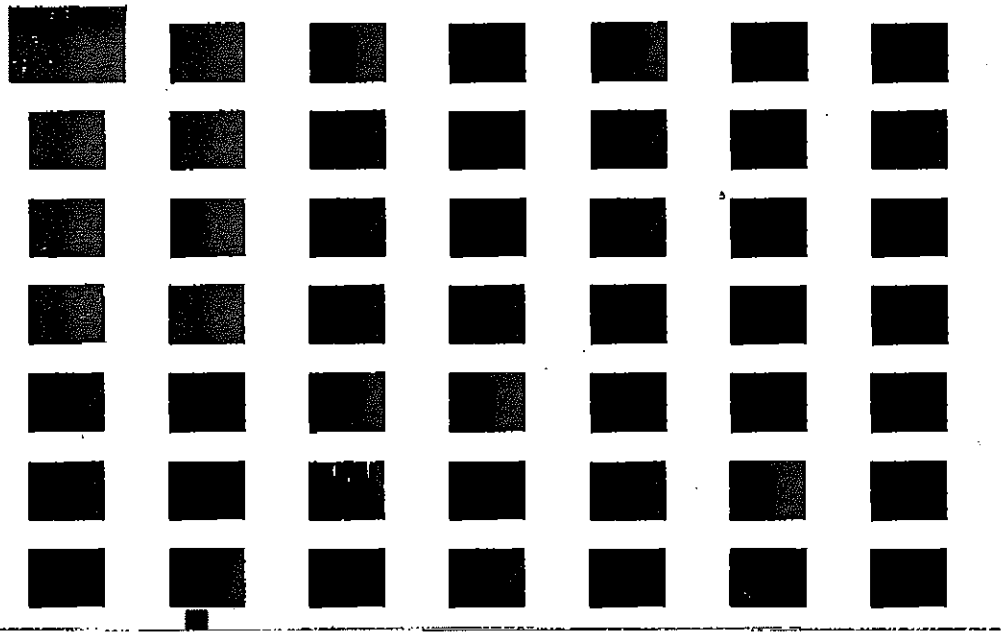
```
icmp_stats_addr_ptr->rcvDestUnrNet.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrNetHighThld;
icmp_stats_addr_ptr->rcvDestUnrNetRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrNetRateHighThld;

icmp_stats_addr_ptr->rcvDestUnrHost.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrHostHighThld;
icmp_stats_addr_ptr->rcvDestUnrHostRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrHostRateHighThld;

icmp_stats_addr_ptr->rcvDestUnrProtocol.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrProtHighThld;
icmp_stats_addr_ptr->rcvDestUnrProtocolRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrProtRateHighThld;

icmp_stats_addr_ptr->rcvDestUnrPort.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrPortHighThld;
icmp_stats_addr_ptr->rcvDestUnrPortRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrPortRateHighThld;

icmp_stats_addr_ptr->rcvDestUnrFrgmt.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrFragHighThld;
icmp_stats_addr_ptr->rcvDestUnrFrgmtRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrFragRateHighThld;

icmp_stats_addr_ptr->rcvDestUnrSrcRoute.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrSrcRouteHighThld;
icmp_stats_addr_ptr->rcvDestUnrSrcRouteRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrSrcRouteRateHighThld;


icmp_stats_addr_ptr->rcvDestUnrNetUnknown.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrNetUnkHighThld;
icmp_stats_addr_ptr->rcvDestUnrNetUnknownRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvDestUnrNetUnkRateHighThld;
```

```
    icmp_stats_addr_ptr->rcvDestUnrHostUnknown.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrHostUnkHighThld;
    icmp_stats_addr_ptr->rcvDestUnrHostUnknownRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrHostUnkRateHighThld;

    icmp_stats_addr_ptr->rcvDestUnrHostIsolated.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrSrcHostIsolHighThld;
    icmp_stats_addr_ptr->rcvDestUnrHostIsolatedRate.high_thld =
mib_icmp_addr_defs->icmpAddrRcvDestUnrSrcHostIsolRateHighThld;

    icmp_stats_addr_ptr->rcvDestUnrNetProhibited.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrNetProhibHighThld;
    icmp_stats_addr_ptr->rcvDestUnrNetProhibitedRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrNetProhibRateHighThld;

    icmp_stats_addr_ptr->rcvDestUnrHostProhibited.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrHostProhibHighThld;
    icmp_stats_addr_ptr->rcvDestUnrHostProhibitedRate.high_thld =
mib_icmp_addr_defs->icmpAddrRcvDestUnrHostProhibRateHighThld;

    icmp_stats_addr_ptr->rcvDestUnrNetTos.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrNetTosHighThld;
    icmp_stats_addr_ptr->rcvDestUnrNetTosRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrNetTosRateHighThld;

    icmp_stats_addr_ptr->rcvDestUnrHostTos.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrHostTosHighThld;
    icmp_stats_addr_ptr->rcvDestUnrHostTosRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvDestUnrHostTosRateHighThld;

    icmp_stats_addr_ptr->xmtDestUnrNet.high_thld =
                                mib_icmp_addr_defs->icmpAddrXmtDestUnrNetHighThld;
    icmp_stats_addr_ptr->xmtDestUnrNetRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrXmtDestUnrNetRateHighThld;
```

```
    icmp_stats_addr_ptr->xmtDestUnrHost.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrHostHighThld;
    icmp_stats_addr_ptr->xmtDestUnrHostRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrHostRateHighThld;

    icmp_stats_addr_ptr->xmtDestUnrProtocol.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrProtHighThld;
    icmp_stats_addr_ptr->xmtDestUnrProtocolRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrProtRateHighThld;

    icmp_stats_addr_ptr->xmtDestUnrPort.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrPortHighThld;
    icmp_stats_addr_ptr->xmtDestUnrPortRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrPortRateHighThld;

    icmp_stats_addr_ptr->xmtDestUnrFrgmt.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrFragHighThld;
    icmp_stats_addr_ptr->xmtDestUnrFrgmtRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrFragRateHighThld;

    icmp_stats_addr_ptr->xmtDestUnrSrcRoute.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrSrcRouteHighThld;
    icmp_stats_addr_ptr->xmtDestUnrSrcRouteRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrSrcRouteRateHighThld;


    icmp_stats_addr_ptr->xmtDestUnrNetUnknown.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrNetUnkHighThld;
    icmp_stats_addr_ptr->xmtDestUnrNetUnknownRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrNetUnkRateHighThld;

    icmp_stats_addr_ptr->xmtDestUnrHostUnknown.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrHostUnkHighThld;
    icmp_stats_addr_ptr->xmtDestUnrHostUnknownRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrHostUnkRateHighThld;
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones,
Mary Robertson, David M. Thompson and
Gerard White

09



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
    icmp_stats_addr_ptr->xmtDestUnrHostIsolated.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrSrcHostIsolHighThld;
    icmp_stats_addr_ptr->xmtDestUnrHostIsolatedRate.high_thld =

mib_icmp_addr_defs->icmpAddrXmtDestUnrSrcHostIsolRateHighThld;

    icmp_stats_addr_ptr->xmtDestUnrNetProhibited.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrNetProhibHighThld;
    icmp_stats_addr_ptr->xmtDestUnrNetProhibitedRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrNetProhibRateHighThld;


    icmp_stats_addr_ptr->xmtDestUnrHostProhibited.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrHostProhibHighThld;
    icmp_stats_addr_ptr->xmtDestUnrHostProhibitedRate.high_thld =

mib_icmp_addr_defs->icmpAddrXmtDestUnrHostProhibRateHighThld;

    icmp_stats_addr_ptr->xmtDestUnrNetTos.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrNetTosHighThld;
    icmp_stats_addr_ptr->xmtDestUnrNetTosRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrNetTosRateHighThld;

    icmp_stats_addr_ptr->xmtDestUnrHostTos.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrHostTosHighThld;
    icmp_stats_addr_ptr->xmtDestUnrHostTosRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtDestUnrHostTosRateHighThld;

    icmp_stats_addr_ptr->rcvSrcQuench.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvSrcQuenchHighThld;
    icmp_stats_addr_ptr->rcvSrcQuenchRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvSrcQuenchRateHighThld;

    icmp_stats_addr_ptr->xmtSrcQuench.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtSrcQuenchHighThld;
    icmp_stats_addr_ptr->xmtSrcQuenchRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtSrcQuenchRateHighThld;
```

```
icmp_stats_addr_ptr->rcvRedirNet.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvRedirNetHighThld;
icmp_stats_addr_ptr->rcvRedirNetRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvRedirNetRateHighThld;

icmp_stats_addr_ptr->rcvRedirHost.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvRedirHostHighThld;
icmp_stats_addr_ptr->rcvRedirHostRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvRedirHostRateHighThld;

icmp_stats_addr_ptr->rcvRedirNetTos.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvRedirNetTosHighThld;
icmp_stats_addr_ptr->rcvRedirNetTosRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvRedirNetTosRateHighThld;

icmp_stats_addr_ptr->rcvRedirHostTos.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvRedirHostTosHighThld;
icmp_stats_addr_ptr->rcvRedirHostTosRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvRedirHostTosRateHighThld;

icmp_stats_addr_ptr->xmtRedirNet.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtRedirNetHighThld;
icmp_stats_addr_ptr->xmtRedirNetRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtRedirNetRateHighThld;

icmp_stats_addr_ptr->xmtRedirHost.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtRedirHostHighThld;
icmp_stats_addr_ptr->xmtRedirHostRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtRedirHostRateHighThld;

icmp_stats_addr_ptr->xmtRedirNetTos.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtRedirNetTosHighThld;
icmp_stats_addr_ptr->xmtRedirNetTosRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtRedirNetTosRateHighThld;

icmp_stats_addr_ptr->xmtRedirHostTos.high_thld =
```

```
                                mib_icmp_addr_defs->icmpAddrXmtRedirHostTosHighThld;
icmp_stats_addr_ptr->xmtRedirHostTosRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrXmtRedirHostTosRateHighThld;

icmp_stats_addr_ptr->rcvTimeExceededInTransit.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvTimeExcTransHighThld;
icmp_stats_addr_ptr->rcvTimeExceededInTransitRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvTimeExcTransRateHighThld;

icmp_stats_addr_ptr->rcvTimeExceededInReass.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvTimeExcReassHighThld;
icmp_stats_addr_ptr->rcvTimeExceededInReassRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvTimeExcReassRateHighThld;

icmp_stats_addr_ptr->xmtTimeExceededInTransit.high_thld =
                                mib_icmp_addr_defs->icmpAddrXmtTimeExcTransHighThld;
icmp_stats_addr_ptr->xmtTimeExceededInTransitRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrXmtTimeExcTransRateHighThld;

icmp_stats_addr_ptr->xmtTimeExceededInReass.high_thld =
                                mib_icmp_addr_defs->icmpAddrXmtTimeExcReassHighThld;
icmp_stats_addr_ptr->xmtTimeExceededInReassRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrXmtTimeExcReassRateHighThld;

icmp_stats_addr_ptr->rcvParamProblem.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvParamProbHighThld;
icmp_stats_addr_ptr->rcvParamProblemRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvParamProbRateHighThld;

icmp_stats_addr_ptr->rcvParamProblemOption.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvParamProbOptHighThld;
icmp_stats_addr_ptr->rcvParamProblemOptionRate.high_thld =
                                mib_icmp_addr_defs->icmpAddrRcvParamProbOptRateHighThld;

icmp_stats_addr_ptr->xmtParamProblem.high_thld =
                                mib_icmp_addr_defs->icmpAddrXmtParamProbHighThld;
icmp_stats_addr_ptr->xmtParamProblemRate.high_thld =
```

```
                    mi.. icmp_addr_defs->icmpAddrXmtParamProbRateHighThld;

icmp_stats_addr_ptr->xmtParamProblemOption.high_thld =
                    mib_icmp_addr_defs->icmpAddrXmtParamProbOptHighThld;
icmp_stats_addr_ptr->xmtParamProblemOptionRate.high_thld =
                    mib_icmp_addr_defs->icmpAddrXmtParamProbOptRateHighThld;

icmp_stats_addr_ptr->rcvTimestampReq.high_thld =
                    mib_icmp_addr_defs->icmpAddrRcvTimeStReqHighThld;
icmp_stats_addr_ptr->rcvTimestampReqRate.high_thld =
                    mib_icmp_addr_defs->icmpAddrRcvTimeStReqRateHighThld;

icmp_stats_addr_ptr->xmtTimestampReq.high_thld =
                    mib_icmp_addr_defs->icmpAddrXmtTimeStReqHighThld;
icmp_stats_addr_ptr->xmtTimestampReqRate.high_thld =
                    mib_icmp_addr_defs->icmpAddrXmtTimeStReqRateHighThld;

icmp_stats_addr_ptr->rcvTimestampReply.high_thld =
                    mib_icmp_addr_defs->icmpAddrRcvTimeStRespHighThld;
icmp_stats_addr_ptr->rcvTimestampReplyRate.high_thld =
                    mib_icmp_addr_defs->icmpAddrRcvTimeStRespRateHighThld;

icmp_stats_addr_ptr->xmtTimestampReply.high_thld =
                    mib_icmp_addr_defs->icmpAddrXmtTimeStRespHighThld;
icmp_stats_addr_ptr->xmtTimestampReplyRate.high_thld =
                    mib_icmp_addr_defs->icmpAddrXmtTimeStRespRateHighThld;

icmp_stats_addr_ptr->rcvAddrMaskReq.high_thld =
                    mib_icmp_addr_defs->icmpAddrRcvAddrMaskReqHighThld;
icmp_stats_addr_ptr->rcvAddrMaskReqRate.high_thld =
                    mib_icmp_addr_defs->icmpAddrRcvAddrMaskReqRateHighThld;

icmp_stats_addr_ptr->xmtAddrMaskReq.high_thld =
                    mib_icmp_addr_defs->icmpAddrXmtAddrMaskReqHighThld;
icmp_stats_addr_ptr->xmtAddrMaskReqRate.high_thld =
                    mib_icmp_addr_defs->icmpAddrXmtAddrMaskReqRateHighThld;
```

```
    icmp_stats_addr_ptr->rcvAddrMaskReply.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvAddrMaskRespHighThld;
    icmp_stats_addr_ptr->rcvAddrMaskReplyRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrRcvAddrMaskRespRateHighThld;

    icmp_stats_addr_ptr->xmtAddrMaskReply.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtAddrMaskRespHighThld;
    icmp_stats_addr_ptr->xmtAddrMaskReplyRate.high_thld =
                        mib_icmp_addr_defs->icmpAddrXmtAddrMaskRespRateHighThld;
}


void setIcmpAddrPairDflts (mib_icmp_addr_pair_defs, dialog_ptr)
    MibIcmpAddrPairDefaults  *mib_icmp_addr_pair_defs;
    StatsDialogEntry         *dialog_ptr;
{
    dialog_ptr->packets.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairPktHighThld;
    dialog_ptr->packetRate.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairPktRateHighThld;

    dialog_ptr->bytes.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairByteHighThld;
    dialog_ptr->byteRate.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairByteRateHighThld;

    dialog_ptr->errors.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairErrorHighThld;
    dialog_ptr->errorRate.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairErrorRateHighThld;

    dialog_ptr->fragments.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairFrgmtHighThld;
    dialog_ptr->fragmentRate.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairFrgmtRateHighThld;
```

```
    dialog_ptr->rexmts.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairRexmtHighThld;
    dialog_ptr->rexmtRate.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairRexmtRateHighThld;

    dialog_ptr->flowCtrls.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairFlowCtrlHighThld;
    dialog_ptr->flowCtrlRate.high_thld =
                        mib_icmp_addr_pair_defs->icmpAddrPairFlowCtrlRateHighThld;
}


void setIcmpSegDflts (mib_icmp_seg_defs, icmp_seg_ptr)
    MibIcmpSegDefaults      *mib_icmp_seg_defs;
    StatsIcmpSegment        *icmp_seg_ptr;
{
    icmp_seg_ptr->frames.high_thld =
                        mib_icmp_seg_defs->icmpSegPktHighThld;
    icmp_seg_ptr->frameRate.high_thld =
                        mib_icmp_seg_defs->icmpSegPktRateHighThld;

    icmp_seg_ptr->bytes.high_thld =
                        mib_icmp_seg_defs->icmpSegByteHighThld;
    icmp_seg_ptr->byteRate.high_thld =
                        mib_icmp_seg_defs->icmpSegByteRateHighThld;

    icmp_seg_ptr->errors.high_thld =
                        mib_icmp_seg_defs->icmpSegErrorHighThld;
    icmp_seg_ptr->errorRate.high_thld =
                        mib_icmp_seg_defs->icmpSegErrorRateHighThld;

    icmp_seg_ptr->rcvOffSegs.high_thld =
                        mib_icmp_seg_defs->icmpSegRcvOffSegHighThld;
    icmp_seg_ptr->rcvOffSegRate.high_thld =
                        mib_icmp_seg_defs->icmpSegRcvOffSegRateHighThld;
```

```
icmp_seg_ptr->xmtOffSegs.high_thld =
                    mib_icmp_seg_defs->icmpSegXmtOffSegHighThld;
icmp_seg_ptr->xmtOffSegRate.high_thld =
                    mib_icmp_seg_defs->icmpSegXmtOffSegRateHighThld;

icmp_seg_ptr->transits.high_thld =
                    mib_icmp_seg_defs->icmpSegTransitHighThld;
icmp_seg_ptr->transitRate.high_thld =
                    mib_icmp_seg_defs->icmpSegTransitRateHighThld;

icmp_seg_ptr->echoReq.high_thld =
                    mib_icmp_seg_defs->icmpSegEchoReqHighThld;
icmp_seg_ptr->echoReqRate.high_thld =
                    mib_icmp_seg_defs->icmpSegEchoReqRateHighThld;

icmp_seg_ptr->echoReply.high_thld =
                    mib_icmp_seg_defs->icmpSegEchoRespHighThld;
icmp_seg_ptr->echoReplyRate.high_thld =
                    mib_icmp_seg_defs->icmpSegEchoRespRateHighThld;

icmp_seg_ptr->destUnrNet.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrNetHighThld;
icmp_seg_ptr->destUnrNetRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrNetRateHighThld;

icmp_seg_ptr->destUnrHost.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrHostHighThld;
icmp_seg_ptr->destUnrHostRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrHostRateHighThld;

icmp_seg_ptr->destUnrProtocol.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrProtHighThld;
icmp_seg_ptr->destUnrProtocolRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrProtRateHighThld;

icmp_seg_ptr->destUnrPort.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrPortHighThld;
```

```
icmp_seg_ptr->destUnrPortRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrPortRateHighThld;

icmp_seg_ptr->destUnrFrgmt.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrFragHighThld;
icmp_seg_ptr->destUnrFrgmtRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrFragRateHighThld;

icmp_seg_ptr->destUnrSrcRoute.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrSrcRouteHighThld;
icmp_seg_ptr->destUnrSrcRouteRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrSrcRouteRateHighThld;

icmp_seg_ptr->destUnrNetUnknown.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrNetUnkHighThld;
icmp_seg_ptr->destUnrNetUnknownRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrNetUnkRateHighThld;

icmp_seg_ptr->destUnrHostUnknown.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrHostUnkHighThld;
icmp_seg_ptr->destUnrHostUnknownRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrHostUnkRateHighThld;

icmp_seg_ptr->destUnrHostIsolated.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrSrcHostIsolHighThld;
icmp_seg_ptr->destUnrHostIsolatedRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrSrcHostIsolRateHighThld;

icmp_seg_ptr->destUnrNetProhibited.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrNetProhibHighThld;
icmp_seg_ptr->destUnrNetProhibitedRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrNetProhibRateHighThld;

icmp_seg_ptr->destUnrHostProhibited.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrHostProhibHighThld;
icmp_seg_ptr->destUnrHostProhibitedRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrHostProhibRateHighThld;
```

```
icmp_seg_ptr->destUnrNetTos.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrNetTosHighThld;
icmp_seg_ptr->destUnrNetTosRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrNetTosRateHighThld;

icmp_seg_ptr->destUnrHostTos.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrHostTosHighThld;
icmp_seg_ptr->destUnrHostTosRate.high_thld =
                    mib_icmp_seg_defs->icmpSegDestUnrHostTosRateHighThld;

icmp_seg_ptr->srcQuench.high_thld =
                    mib_icmp_seg_defs->icmpSegSrcQuenchHighThld;
icmp_seg_ptr->srcQuenchRate.high_thld =
                    mib_icmp_seg_defs->icmpSegSrcQuenchRateHighThld;

icmp_seg_ptr->redirNet.high_thld =
                    mib_icmp_seg_defs->icmpSegRedirNetHighThld;
icmp_seg_ptr->redirNetRate.high_thld =
                    mib_icmp_seg_defs->icmpSegRedirNetRateHighThld;

icmp_seg_ptr->redirHost.high_thld =
                    mib_icmp_seg_defs->icmpSegRedirHostHighThld;
icmp_seg_ptr->redirHostRate.high_thld =
                    mib_icmp_seg_defs->icmpSegRedirHostRateHighThld;

icmp_seg_ptr->redirNetTos.high_thld =
                    mib_icmp_seg_defs->icmpSegRedirNetTosHighThld;
icmp_seg_ptr->redirNetTosRate.high_thld =
                    mib_icmp_seg_defs->icmpSegRedirNetTosRateHighThld;

icmp_seg_ptr->redirHostTos.high_thld =
                    mib_icmp_seg_defs->icmpSegRedirHostTosHighThld;
icmp_seg_ptr->redirHostTosRate.high_thld =
                    mib_icmp_seg_defs->icmpSegRedirHostTosRateHighThld;

icmp_seg_ptr->timeExceededInTransit.high_thld =
```

```
                    mib_icmp_seg_defs->icmpSegTimeExcTransHighThld;
icmp_seg_ptr->timeExceededInTransitRate.high_thld =
                    mib_icmp_seg_defs->icmpSegTimeExcTransRateHighThld;

icmp_seg_ptr->timeExceededInReass.high_thld =
                    mib_icmp_seg_defs->icmpSegTimeExcReassHighThld;
icmp_seg_ptr->timeExceededInReassRate.high_thld =
                    mib_icmp_seg_defs->icmpSegTimeExcReassRateHighThld;

icmp_seg_ptr->paramProblem.high_thld =
                    mib_icmp_seg_defs->icmpSegParamProbHighThld;
icmp_seg_ptr->paramProblemRate.high_thld =
                    mib_icmp_seg_defs->icmpSegParamProbRateHighThld;

icmp_seg_ptr->paramProblemOption.high_thld =
                    mib_icmp_seg_defs->icmpSegParamProbOptHighThld;
icmp_seg_ptr->paramProblemOptionRate.high_thld =
                    mib_icmp_seg_defs->icmpSegParamProbOptRateHighThld;

icmp_seg_ptr->timestampReq.high_thld =
                    mib_icmp_seg_defs->icmpSegTimeStReqHighThld;
icmp_seg_ptr->timestampReqRate.high_thld =
                    mib_icmp_seg_defs->icmpSegTimeStReqRateHighThld;

icmp_seg_ptr->timestampReply.high_thld =
                    mib_icmp_seg_defs->icmpSegTimeStRespHighThld;
icmp_seg_ptr->timestampReplyRate.high_thld =
                    mib_icmp_seg_defs->icmpSegTimeStRespRateHighThld;

icmp_seg_ptr->addrMaskReq.high_thld =
                    mib_icmp_seg_defs->icmpSegAddrMaskReqHighThld;
icmp_seg_ptr->addrMaskReqRate.high_thld =
                    mib_icmp_seg_defs->icmpSegTimeStReqRateHighThld;

icmp_seg_ptr->addrMaskReply.high_thld =
                    mib_icmp_seg_defs->icmpSegAddrMaskRespHighThld;
icmp_seg_ptr->addrMaskReplyRate.high_thld =
```

```
        mib_icmp_seg_defs->icmpSegTimeStRespRateHighThld;
}
```

```
/*
 *  stats_ip_a.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_ip_a.c
 *
 *  Date:       8/12/91
 *
 *  Revision:   1.6
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   08-12-91  DPD       Added missing rate calculations in statsIpAddrRate
 *                            and statsIpSegRate
 *
 *   06-05-91  DPD       Fixed aging bug
 *   06-01-91  DPD       Fixed rate bug
 *
 */

static char stats_ip_a_c [] = "@(#)stats_ip_a.c   1.6";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
```

```
#include "mib_defs.h"
#include "mib_ip.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
#include </usr/include/bsd43/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_mib.h"
#include "snmpd.h"
#include "mib_alarms.h"
#include "em_ctrl.h"


/*****************************************************************************
 *
 * Given a IP dialog, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsIpAgeDialog (dialog_addr_ptr)

     StatsAddrEntry      *dialog_addr_ptr;

{
     register Uint32                  xip_dialog_hash;
```

```
    register StatsAddrEntry       *xip_dialog_hash_link;
    register StatsAddrEntry       *xip_previous_dialog_hash_link;

    register StatsDialogLink *dialog_link_ptr;
    register StatsAddrEntry       *addr_record_ptr;
    register StatsIpAddr          *ip_stats_ptr;
    struct timeval      current_time;
    Uint32                    i;

if (monCtrl.ipDialogAgeTimer == 0)
    return;

if (dialog_addr_ptr == NULL)
    return;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - dialog_addr_ptr->lastTime < monCtrl.ipDialogAgeTimer)
    return;

/*
 * Time to chuck this dialog.  Remove it from the dialog hash table.
 * Look up the dialog in order to set ip_dialog_hash, ip_previous_dialog_hash_link
 * and ip_dialog_hash_link.
 */
addr_record_ptr = stats_ip_lookup_dialog (
    &dialog_addr_ptr->address.netAddress1.u.ipAddress,
    &dialog_addr_ptr->address.netAddress2.u.ipAddress);

if (addr_record_ptr != dialog_addr_ptr)
    {
    mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsIpAgeDialog: ptrs not equal");
    return;
    }

xip_dialog_hash = ip_dialog_hash;
xip_dialog_hash_link = ip_dialog_hash_link;
```

```
xip_previous_dialog_hash_link = ip_previous_dialog_hash_link;

if ( (xip_previous_dialog_hash_link == NULL) && (xip_dialog_hash_link->hash_link == NULL)
)
     ip_dialog_hash_table[xip_dialog_hash] = NULL;
else
     {
     if (xip_previous_dialog_hash_link != NULL)
          xip_previous_dialog_hash_link->hash_link = xip_dialog_hash_link->hash_link;
     else
          if (xip_dialog_hash_link->hash_link != NULL)
               ip_dialog_hash_table[xip_dialog_hash] = xip_dialog_hash_link->hash_link;
     }

/*
 * Remove the dialog from the dialogQ for each of the ip addressos
 * and deallocate the structure that was on the dialogQ which pointed
 * to the dialog.
 */
addr_record_ptr = stats_ip_lookup_addr (
     &dialog_addr_ptr->address.netAddress1.u.ipAddress);

for (i=0; i<2; i++)
     {
     if (addr_record_ptr != NULL)
          {
          ip_stats_ptr = (StatsIpAddr *) addr_record_ptr->stats_ptr;
          if (ip_stats_ptr != NULL)
               {
               dialog_link_ptr = (StatsDialogLink *) ip_stats_ptr->dialogQ.pFlink;
               while (dialog_link_ptr != NULL)
                    {
                    if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                         break;
                    dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
                    }
               if (dialog_link_ptr != NULL)
```

```
                    {
                    FBRemqm (&ip_stats_ptr->dialogQ, (PFBQentry_type) dialog_link_ptr);
                    stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                    }
               }
          }
          addr_record_ptr = stats_ip_lookup_addr (
               &dialog_addr_ptr->address.netAddress2.u.ipAddress);
     }



/*
 * Remove the dialog from the statsIpDialogQ and deallocate the structures
 * associated with the dialog.
 */
FBRemqm (&statsIpDialogQ, (PFBQentry_type) dialog_addr_ptr);

if (dialog_addr_ptr->stats_ptr != NULL)
     stats_deallocate (dialog_addr_ptr->stats_ptr, sizeof(StatsDialogEntry) );

stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );

}


/*****************************************************************************
 *
 * Determine if address statistics should be aged out, and, if so, get rid of it.
 */

void statsIpAgeAddr (addr_ptr)

     StatsAddrEntry      *addr_ptr;
```

```
{
     register StatsDialogLink *dialog_link_ptr;
     register StatsIpAddr          *stats_ptr;
     struct timeval                current_time;

if (monCtrl.ipNodeAgeTimer == 0)
     return;

if (addr_ptr == NULL)
     return;

stats_ptr = (StatsIpAddr *) addr_ptr->stats_ptr;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - addr_ptr->lastTime < monCtrl.ipNodeAgeTimer)
     return;

/*
 * Time to chuck these stats. Not the address - just the stats.
 */

/*
 * Remove the dialog links from the stats and deallocate the stats.
 */
if (stats_ptr != NULL)
     {
     dialog_link_ptr = (StatsDialogLink *) FBRemqh (&stats_ptr->dialogQ);
     while (dialog_link_ptr != NULL)
          {
          stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
          dialog_link_ptr = (StatsDialogLink *) FBRemqh (&stats_ptr->dialogQ);
          }
     stats_deallocate (stats_ptr, sizeof(StatsIpAddr) );

     addr_ptr->stats_ptr = NULL;
     }
```

```
}


/*****************************************************************************
 *
 * IP rate routines
 */

void statsIpProtocolRate (ip_protocol_ptr, ip_addr_ptr, rate_type, time)
     StatsProtocolEntry  *ip_protocol_ptr;
     StatsAddrEntry                *ip_addr_ptr;
     Uint32                             rate_type;
     Uint32                             time;
{
     AlarmUserData       ip_alarm_data;

     while (ip_protocol_ptr != NULL)
     {
          /*
           * Pass the protocol as alarm data in case an alarm occurs
           */
          ip_alarm_data.length = 4;
          bcopy (&ip_protocol_ptr->protocol, ip_alarm_data.data, 4);

          statsCalcRates(&ip_protocol_ptr->frameRate, NULL,
                         rate_type, (StatsAddrEntry *)ip_addr_ptr,
                         IP_PROTOCOL, AL_FRAMES, time, NULL);
          ip_protocol_ptr = ip_protocol_ptr->link;
     }

}


void statsIpSegRate (rate_type)
```

```
    Uint32                      rate_type;
{
    register StatsIpSegment         *ip_seg_ptr;
    register StatsAddrEntry         *ip_seg_addr_ptr;
    struct timeval                  current_time;
    register StatsProtocolEntry     *ip_protocol_ptr;

    ip_seg_addr_ptr = (StatsAddrEntry *) statsIpSegC.pFlink;

    while (ip_seg_addr_ptr != NULL)
    {
        ip_seg_ptr = (StatsIpSegment *) ip_seg_addr_ptr->stats_ptr;
        if (ip_seg_ptr != NULL)
        {
            /* get the current time */
            mon_gettimeofday(&current_time, 0);

            statsCalcRates(&ip_seg_ptr->frameRate, &ip_seg_ptr->frameBuckets,
                           rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                           AL_FRAMES, current_time.tv_sec, NULL);
            statsCalcRates(&ip_seg_ptr->byteRate, &ip_seg_ptr->byteBuckets,
                           rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                           AL_BYTES, current_time.tv_sec, NULL);
            statsCalcRates(&ip_seg_ptr->errorRate, &ip_seg_ptr->errorBuckets,
                           rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                           AL_ERRORS, current_time.tv_sec, NULL);
            statsCalcRates(&ip_seg_ptr->rcvOffSegRate, &ip_seg_ptr->rcvOffSegBuckets,
                           rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                           AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
            statsCalcRates(&ip_seg_ptr->xmtOffSegRate, &ip_seg_ptr->xmtOffSegBuckets,
                           rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                           AL_XMT_OFF_SEG, current_time.tv_sec, NULL);
```

```
                statsCalcRates(&ip_seg_ptr->transitRate, &ip_seg_ptr->transitBuckets,
                                rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                                AL_TRANSIT, current_time.tv_sec, NULL);
                statsCalcRates(&ip_seg_ptr->bcastRate, &ip_seg_ptr->bcastBuckets,
                                rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                                AL_BCAST, current_time.tv_sec, NULL);
                statsCalcRates(&ip_seg_ptr->mcastRate, &ip_seg_ptr->mcastBuckets,
                                rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                                AL_MCAST, current_time.tv_sec, NULL);

                statsCalcRates(&ip_seg_ptr->flowCtrlRate, &ip_seg_ptr->flowCtrlBuckets,
                                rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                                AL_FLOW_CTRLS, current_time.tv_sec, NULL);

                statsCalcRates(&ip_seg_ptr->frgmtRate, &ip_seg_ptr->frgmtBuckets,
                                rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                                AL_FRAGMENTS, current_time.tv_sec, NULL);

                statsCalcRates(&ip_seg_ptr->hdrByteRate, NULL,
                                rate_type, (StatsAddrEntry *)ip_seg_addr_ptr,
IP_SEGMENT,
                                AL_HDR_BYTES, current_time.tv_sec, NULL);

                /* calculate the rates on the protocolQ */
                ip_protocol_ptr = (StatsProtocolEntry *) ip_seg_ptr->protocolQ.pFlink;
                statsIpProtocolRate (ip_protocol_ptr, ip_seg_addr_ptr,
                                        rate_type, current_time.tv_sec);

                statsIpDialogRate (rate_type);

                if ((current_time.tv_sec - ip_seg_addr_ptr->seconds_start_time) >=
monCtrl.rateTimer)
```

```
                ip_seg_addr_ptr->seconds_start_time = current_time.tv_sec;
            }
            ip_seg_addr_ptr = (StatsAddrEntry *)ip_seg_addr_ptr->link.pFlink;
        }
    }

void statsIpAddrRate (rate_type)
    Uint32                  rate_type;
{
    register StatsIpAddr            *ip_addr_ptr;
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
    register StatsAddrEntry         *ip_entry_ptr;
    register Uint32                  loop_count;
    struct timeval                  current_time;
    register StatsProtocolEntry     *ip_protocol_ptr;

    /* get the current time */
    mon_gettimeofday(&current_time, 0);

    loop_count = 0;

    /***** AGING *****/
    if (statsNextIpAddrEntry != NULL)
        entry_ptr = statsNextIpAddrEntry;       /* pick up where we left off */
    else
    {
        /* Only age structures if we've processed them all */
        entry_ptr = statsIpAddrQ.pFlink;
        while (entry_ptr != NULL)
        {
            next_entry_ptr = entry_ptr->pFlink;
            statsIpAgeAddr ((StatsAddrEntry *) entry_ptr);
            entry_ptr = next_entry_ptr;
        }
        entry_ptr = statsIpAddrQ.pFlink;        /* start at the beginning */
```

```
    }
    /***** AGING *****/

    while ((entry_ptr != NULL) && (loop_count < stats_q_count))
    {
        ip_entry_ptr = (StatsAddrEntry *)entry_ptr;
        if (((ip_entry_ptr->em_control & rate_type) != 0)
                        && (ip_entry_ptr->stats_ptr != NULL))

        {
            ip_addr_ptr = (StatsIpAddr *)ip_entry_ptr->stats_ptr;

            if ((current_time.tv_sec - ip_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
            {
            statsCalcRates(&ip_addr_ptr->frameRate, &ip_addr_ptr->frameBuckets,
                        rate_type, (StatsAddrEntry *)ip_entry_ptr,
                        IP_NODE, AL_FRAMES, current_time.tv_sec, NULL);
            statsCalcRates(&ip_addr_ptr->rcvFrameRate, NULL,
                        rate_type, (StatsAddrEntry *)ip_entry_ptr,
                        IP_NODE, AL_RCV_FRAMES, current_time.tv_sec, NULL);
            statsCalcRates(&ip_addr_ptr->xmtFrameRate, NULL,
                        rate_type, (StatsAddrEntry *)ip_entry_ptr,
                        IP_NODE, AL_XMT_FRAMES, current_time.tv_sec, NULL);

            statsCalcRates(&ip_addr_ptr->byteRate, &ip_addr_ptr->byteBuckets,
                        rate_type, (StatsAddrEntry *)ip_entry_ptr,
                        IP_NODE, AL_BYTES, current_time.tv_sec, NULL);
            statsCalcRates(&ip_addr_ptr->rcvByteRate, NULL,
                        rate_type, (StatsAddrEntry *)ip_entry_ptr,
                        IP_NODE, AL_RCV_BYTES, current_time.tv_sec, NULL);
            statsCalcRates(&ip_addr_ptr->xmtByteRate, NULL,
                        rate_type, (StatsAddrEntry *)ip_entry_ptr,
                        IP_NODE, AL_XMT_BYTES, current_time.tv_sec, NULL);

            statsCalcRates(&ip_addr_ptr->errorRate, &ip_addr_ptr->errorBuckets,
                        rate_type, (StatsAddrEntry *)ip_entry_ptr,
                        IP_NODE, AL_ERRORS, current_time.tv_sec, NULL);
```

```
statsCalcRates(&ip_addr_ptr->rcvErrorRate, NULL,
                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
                    IP_NODE, AL_RCV_ERRORS, current_time.tv_sec, NULL);
statsCalcRates(&ip_addr_ptr->xmtErrorRate, NULL,
                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
                    IP_NODE, AL_XMT_ERRORS, current_time.tv_sec, NULL);

statsCalcRates(&ip_addr_ptr->rcvOffSegRate, &ip_addr_ptr->rcvOffSegBuckets,

                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
                    IP_NODE, AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
statsCalcRates(&ip_addr_ptr->xmtOffSegRate, &ip_addr_ptr->xmtOffSegBuckets,

                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
                    IP_NODE, AL_XMT_OFF_SEG, current_time.tv_sec, NULL);
statsCalcRates(&ip_addr_ptr->xmtBcastRate, &ip_addr_ptr->xmtBcastBuckets,
                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
                    IP_NODE, AL_BCAST, current_time.tv_sec, NULL);
statsCalcRates(&ip_addr_ptr->xmtMcastRate, &ip_addr_ptr->xmtMcastBuckets,
                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
                    IP_NODE, AL_MCAST, current_time.tv_sec, NULL);

statsCalcRates(&ip_addr_ptr->flowCtrlRate, &ip_addr_ptr->flowCtrlBuckets,
                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
                    IP_NODE, AL_FLOW_CTRLS, current_time.tv_sec, NULL);

statsCalcRates(&ip_addr_ptr->rcvHdrByteRate, NULL,
                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
                    IP_NODE, AL_RCV_HDR_BYTES, current_time.tv_sec, NULL);

statsCalcRates(&ip_addr_ptr->xmtHdrByteRate, NULL,
                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
                    IP_NODE, AL_XMT_HDR_BYTES, current_time.tv_sec, NULL);


statsCalcRates(&ip_addr_ptr->frgmtRate, &ip_addr_ptr->frgmtBuckets,
                    rate_type, (StatsAddrEntry *)ip_entry_ptr,
```

```
                                IP_NODE, AL_FRAGMENTS, current_time.tv_sec, NULL);
            statsCalcRates(&ip_addr_ptr->rcvFrgmtRate, NULL,
                                rate_type, (StatsAddrEntry *)ip_entry_ptr,
                                IP_NODE, AL_RCV_FRAGMENTS, current_time.tv_sec, NULL);

            statsCalcRates(&ip_addr_ptr->xmtFrgmtRate, NULL,
                                rate_type, (StatsAddrEntry *)ip_entry_ptr,
                                IP_NODE, AL_XMT_FRAGMENTS, current_time.tv_sec, NULL);


            /* calculate the rates on the protocolQ */
            ip_protocol_ptr = (StatsProtocolEntry *) ip_addr_ptr->protocolQ.pFlink;
            statsIpProtocolRate (ip_protocol_ptr, ip_entry_ptr,
                                        rate_type, current_time.tv_sec);

            ip_entry_ptr->seconds_start_time = current_time.tv_sec;
            loop_count++;
            }
        }
        entry_ptr = entry_ptr->pFlink;
    }
    statsNextIpAddrEntry = entry_ptr;       /* pick up where we left off */

    if (statsNextIpAddrEntry != NULL)
        statsRatesNotDone |= STATS_IP_ADDR_RATE;
}

void statsIpDialogRate (rate_type)
    Uint32                      rate_type;
{
    register StatsDialogEntry     *ip_dialog_ptr;
    register FBQentry_type        *entry_ptr;
    register FBQentry_type        *next_entry_ptr;
    register StatsAddrEntry      *ip_entry_ptr;
    register Uint32                loop_count;
    struct timeval                 current_time;
```

```
    /* get the current time */
    mon_gettimeofday(&current_time, 0);

    if (statsNextIpPairEntry != NULL)
        entry_ptr = statsNextIpPairEntry;       /* pick up where we left off */
    else
    {
        /* Only age structures if we've processed them all */
        entry_ptr = statsIpDialogQ.pFlink;
        while (entry_ptr != NULL)
        {
            next_entry_ptr = entry_ptr->pFlink;
            statsIpAgeDialog ((StatsAddrEntry *) entry_ptr);
            entry_ptr = next_entry_ptr;
        }

        entry_ptr = statsIpDialogQ.pFlink;      /* start at the beginning */
    }

    loop_count = 0;
    while ((entry_ptr != NULL) && (loop_count < stats_q_count))
    {
        ip_entry_ptr = (StatsAddrEntry *)entry_ptr;
        if (((ip_entry_ptr->em_control & rate_type) != 0)
                            && (ip_entry_ptr->stats_ptr != NULL))
        {
            ip_dialog_ptr = (StatsDialogEntry *)ip_entry_ptr->stats_ptr;

            if ((current_time.tv_sec - ip_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
            {
            statsCalcRates(&ip_dialog_ptr->packetRate, NULL,
                           rate_type, (StatsAddrEntry *)ip_entry_ptr,
                           IP_PAIR, AL_FRAMES, current_time.tv_sec, NULL);

            statsCalcRates(&ip_dialog_ptr->byteRate, NULL,
                           rate_type, (StatsAddrEntry *)ip_entry_ptr,
```

```
                                IP_PAIR, AL_BYTES, current_time.tv_sec, NULL);

                statsCalcRates(&ip_dialog_ptr->errorRate, NULL,
                                rate_type, (StatsAddrEntry *)ip_entry_ptr,
                                IP_PAIR, AL_ERRORS, current_time.tv_sec, NULL);

                statsCalcRates(&ip_dialog_ptr->fragmentRate, NULL,
                                rate_type, (StatsAddrEntry *)ip_entry_ptr,
                                IP_PAIR, AL_FRAGMENTS, current_time.tv_sec, NULL);

                statsCalcRates(&ip_dialog_ptr->rexmtRate, NULL,
                                rate_type, (StatsAddrEntry *)ip_entry_ptr,
                                IP_PAIR, AL_REXMTS, current_time.tv_sec, NULL);

                statsCalcRates(&ip_dialog_ptr->flowCtrlRate, NULL,
                                rate_type, (StatsAddrEntry *)ip_entry_ptr,
                                IP_PAIR, AL_FLOW_CTRLS, current_time.tv_sec, NULL);

                ip_entry_ptr->seconds_start_time = current_time.tv_sec;
                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }
        statsNextIpPairEntry = entry_ptr;

        if (statsNextIpPairEntry != NULL)
            statsRatesNotDone |= STATS_IP_PAIR_RATE;
    }
```

```
/*
 *  stats_ip_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                   All rights reserved.
 *
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_ip_p.c
 *
 *  Date:       8/23/91
 *
 *  Revision:   1.7
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   08-15-91  KR        if mcast or bcast addr, set seg to mySegmentId
 *   06-12-91  KR        Changed default segment id for ip addresses
 */

static char stats_ip_p_c [] = "@(#)stats_ip_p.c   1.7";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
#include "mib_defs.h"
#include "mib_ip.h"
#include <sys/types.h>
```

```
#include <sys/socket.h>
#include <bsd43/sys/time.h>
#include </usr/include/bsd43/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_mib.h"
#include "snmpd.h"
#include "mib_alarms.h"
#include "em_ctrl.h"



/*
 * Global Data Structures
 */

StatsAddrEntry      *ip_this_seg_addr_ptr;
StatsIpSegment      *ip_this_seg_stats_ptr;
StatsProtocolEntry  *ip_this_seg_protocol_ptr;

StatsAddrEntry      *ip_src_seg_addr_ptr, *ip_dst_seg_addr_ptr;
StatsIpSegment      *ip_src_seg_stats_ptr, *ip_dst_seg_stats_ptr;
StatsProtocolEntry  *ip_src_seg_protocol_ptr, *ip_dst_seg_protocol_ptr;

StatsAddrEntry      *ip_hash_table[IP_HASH_TABLE_SIZE];
StatsAddrEntry      *ip_src_node_addr_ptr, *ip_dst_node_addr_ptr;
```

```
StatsIpAddr          *ip_src_node_stats_ptr, *ip_dst_node_stats_ptr;
StatsProtocolEntry   *ip_src_node_protocol_ptr, *ip_dst_node_protocol_ptr;

StatsProtocolEntry   *ip_this_seg_protocol_ptr;
StatsProtocolEntry   *ip_src_seg_protocol_ptr, *ip_dst_seg_protocol_ptr;

StatsAddrEntry       *ip_dialog_hash_table[IP_DIALOG_HASH_TABLE_SIZE];
StatsAddrEntry       *ip_dialog_addr_ptr;
StatsDialogEntry     *ip_dialog_stats_ptr;



/*
 * Local data structures
 */
Uint32                    ip_hash;
StatsAddrEntry       *ip_hash_link;
StatsAddrEntry       *ip_previous_hash_link;
Uint32                    ip_dialog_hash;
StatsAddrEntry       *ip_dialog_hash_link;
StatsAddrEntry       *ip_previous_dialog_hash_link;

Import MacAddress    mac_src_addr;
Import MacAddress    mac_dst_addr;
Import Uint32        ip_src_addr;
Import Uint32        ip_dst_addr;


/*****************************************************************************
 *
 * Look for the segment address structure.
 * If no match is found, NULL is returned.
 */

StatsAddrEntry *stats_ip_lookup_segment (segment)
```

```
    Uint32                  segment;
{
    register StatsAddrEntry       *seg_addr_ptr;

seg_addr_ptr = (StatsAddrEntry *) statsIpSegQ.pFlink;

while (seg_addr_ptr != NULL)
    {
    if (seg_addr_ptr->address.segment1 == segment)
        break;
    seg_addr_ptr = (StatsAddrEntry *)seg_addr_ptr->link.pFlink;
    }

    return (seg_addr_ptr);
}


/*************************************************************************
 *
 * Find the structure for keeping Ip segment statistics for the given
 * segment.  If one is not found, attempt to allocate one.
 */

StatsAddrEntry *stats_ip_get_segment (segment)

    Uint32              segment;
{
    register StatsAddrEntry       *seg_addr_ptr;
    register StatsIpSegment       *seg_stats_ptr;

 seg_addr_ptr = stats_ip_lookup_segment (segment);

/*
 * If not found, try to allocate a structure for this segment
 * If one can't be obtained, count this as a drop.
 */
```

```
if (seg_addr_ptr == NULL)
    {
    seg_addr_ptr = (StatsAddrEntry *) stats_allocate ( sizeof(StatsAddrEntry) );
    if (seg_addr_ptr != NULL)
        {
        seg_addr_ptr->address.addressType = MibSegment1;
        seg_addr_ptr->address.segment1 = segment;
        seg_addr_ptr->parse_control = monCtrl.segParseCtrl;
        seg_addr_ptr->startTime = stats_start_time.tv_sec;
        seg_addr_ptr->lastTime = stats_start_time.tv_sec;
        FBInsqt (&statsIpSegQ, (PFBQentry_type) seg_addr_ptr);
        }
    else
        {
        stats_mon_ip_dropped;
        return (NULL);
        }
    }

/*
 * If there's an address structure, see if there's a statistics structure
 * for Ip attached, if not and parse control allows, allocate one.
 */
if (seg_addr_ptr != NULL)
    {
    seg_stats_ptr = (StatsIpSegment *) seg_addr_ptr->stats_ptr;
    if (seg_stats_ptr == NULL)
        {
        /*
         * If Ip parsing is enabled, allocate Ip statistics structure if
         * one has not already been allocated. If one can't be obtained, count
         * this as a drop.
         */
        if (seg_addr_ptr->parse_control & MibParseIp)
            {
            seg_stats_ptr = (StatsIpSegment *) stats_allocate (sizeof (StatsIpSegment)
);
```

```
                if (seg_stats_ptr == NULL)
                     stats_mon_ip_dropped;
                else
                     {
                     seg_addr_ptr->stats_ptr = (Uint32 *) seg_stats_ptr;
                     seg_stats_ptr->frameRate.type = STATS_RATE_10S;
                     seg_stats_ptr->byteRate.type = STATS_RATE_10S;
                     seg_stats_ptr->errorRate.type = STATS_RATE_10S;
                     seg_stats_ptr->rcvOffSegRate.type = STATS_RATE_10S;
                     seg_stats_ptr->xmtOffSegRate.type = STATS_RATE_10S;
                     seg_stats_ptr->transitRate.type = STATS_RATE_10S;
                     seg_stats_ptr->bcastRate.type = STATS_RATE_10S;
                     seg_stats_ptr->mcastRate.type = STATS_RATE_10S;
                     seg_stats_ptr->frgmtRate.type = STATS_RATE_10S;
                     seg_stats_ptr->flowCtrlRate.type = STATS_RATE_10S;
                     seg_stats_ptr->hdrByteRate.type = STATS_RATE_10S;
                     InitQh (&seg_stats_ptr->protocolQ);

                     setIpSegDflts (&mibSegDefaults.mibIpSegDefaults, seg_stats_ptr);
                     }
                }
           }
      }

return (seg_addr_ptr);
}


/*****************************************************************************
 *
 *  Find the ip address record.
 *  A hash is done on the ip address and a pointer to the first
 *  StatsAddrEntry with the same hash is found.  Structures with the
 *  same hash are linked.  The link must be walked and the ip address
 *  compared with the ip address in each of the structures until a match
 *  is found.  If no match is found, NULL is returned.
```

```
 */
StatsAddrEntry *stats_ip_lookup_addr (ip_addr)

     Uint32          *ip_addr;
{
     Uint32          i;

/*
 * Compute ip address hash to get index into hash table
 */
ip_hash = (*ip_addr & 0xffff0000) + (*ip_addr & 0x0000ffff);
ip_hash = ((ip_hash + ((ip_hash & 0xff00) >> 8)) & (IP_HASH_TABLE_SIZE - 1));


ip_previous_hash_link = NULL;
ip_hash_link = (StatsAddrEntry *) ip_hash_table[ip_hash];

/*
 * Walk linked list for exact entry
 */
while (ip_hash_link != NULL)
     {
     if (ip_hash_link->address.netAddress1.u.ipAddress == *ip_addr)
          return (ip_hash_link);
     else
          {
          ip_previous_hash_link = ip_hash_link;
          ip_hash_link = ip_hash_link->hash_link;
          }
     }


/*
 * No entry found.
 */
return (NULL);
}
```

```
/*****************************************************************************
 *
 * Allocate a stats structure if parse control is turned on.
 *
 */

Uint32 stats_ip_get_stats (ip_addr_record_ptr)

    StatsAddrEntry     *ip_addr_record_ptr;
{
    register StatsIpAddr     *ip_addr_stats_ptr;

    if ((ip_addr_record_ptr != NULL) && (ip_addr_record_ptr->stats_ptr == NULL))
    {
         /*
          * If parse control is turned on for Ip, allocate a structure for Ip
          * address statistics and initialize it.
          */
         if (ip_addr_record_ptr->parse_control & MibParseIp)
         {
              ip_addr_record_ptr->stats_ptr = (Uint32 *) stats_allocate (sizeof
(StatsIpAddr) );
              ip_addr_stats_ptr = (StatsIpAddr *) ip_addr_record_ptr->stats_ptr;
              if (ip_addr_stats_ptr != NULL)
              {
                   ip_addr_stats_ptr = (StatsIpAddr *) ip_addr_record_ptr->stats_ptr;
              ip_addr_stats_ptr->frameRate.type = STATS_RATE_10S;
                   ip_addr_stats_ptr->rcvFrameRate.type = STATS_RATE_10S;
                   ip_addr_stats_ptr->xmtFrameRate.type = STATS_RATE_10S;
                   ip_addr_stats_ptr->byteRate.type = STATS_RATE_10S;
                   ip_addr_stats_ptr->rcvByteRate.type = STATS_RATE_10S;
                   ip_addr_stats_ptr->xmtByteRate.type = STATS_RATE_10S;
                   ip_addr_stats_ptr->errorRate.type = STATS_RATE_10S;
                   ip_addr_stats_ptr->rcvErrorRate.type = STATS_RATE_10S;
```

```
                    ip_addr_stats_ptr->xmtErrorRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->frgmtRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->rcvFrgmtRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->xmtFrgmtRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->flowCtrlRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->rcvOffSegRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->xmtOffSegRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->xmtBcastRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->xmtMcastRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->rcvHdrByteRate.type = STATS_RATE_10S;
                    ip_addr_stats_ptr->xmtHdrByteRate.type = STATS_RATE_10S;
                    InItqh (&ip_addr_stats_ptr->protocolQ);
                    FBInitqh (&ip_addr_stats_ptr->dialogQ);

                    setIpAddrDflts (&mibNodeDefaults.mibIpAddrDefaults,
ip_addr_stats_ptr);
                }
                else
                    return (FALSE);
            }
        }
        return (TRUE);
    }


/*****************************************************************************
 *
 * Generate a new node alarm for this ip address
 */
void stats_ip_new_node_alarm (ip_addr)

    Uint32              *ip_addr;
{
    register MibAddress     *mib_address;
```

```
    mib_address = (MibAddress *)mon_cache_malloc(sizeof(MibAddress));
    if (mib_address == NULL)
        return;
    bzero(mib_address, sizeof(MibAddress));

    /* set the address of the node that's alarming */
    mib_address->addressType = MibNetAddress1 + MibMacAddress1;

    mib_address->netAddress1.netAddressType = NetTcpIp;
    mib_address->netAddress1.length = 4;
    mib_address->netAddress1.u.ipAddress = *ip_addr;

    /* now get the mac address for this node */
    if (*ip_addr == ip_src_addr)
    {
        mib_address->macAddress1.double_bytes[2] = mac_src_addr.double_bytes[2];
        mib_address->macAddress1.double_bytes[1] = mac_src_addr.double_bytes[1];
        mib_address->macAddress1.double_bytes[0] = mac_src_addr.double_bytes[0];
    }
    else
    {
        mib_address->macAddress1.double_bytes[2] = mac_dst_addr.double_bytes[2];
        mib_address->macAddress1.double_bytes[1] = mac_dst_addr.double_bytes[1];
        mib_address->macAddress1.double_bytes[0] = mac_dst_addr.double_bytes[0];
    }

    stats_new_node_alarm (mib_address);
}


/*****************************************************************************
 *
 * Initialize the structure for keeping ip address statistics for the
 * given ip address.
 */
```

```
StatsAddrEntry *stats_ip_init_addr (ip_addr)

    Uint32                  *ip_addr;
{
    register StatsAddrEntry       *ip_addr_record_ptr;

    /*
     * Try to allocate a statistics structure for this address.
     * If one can't be obtained, count this as a drop.
     */
    ip_addr_record_ptr = (StatsAddrEntry *)stats_allocate (sizeof (StatsAddrEntry) );
    if (ip_addr_record_ptr == NULL)
        {
        stats_mon_ip_dropped;
        return (NULL);
        }
    else
        {
        ip_addr_record_ptr->hash_link                                    = NULL;
        ip_addr_record_ptr->address.addressType = MibNetAddress1 + MibSegment1;
        ip_addr_record_ptr->address.netAddress1.netAddressType      = NetTcpIp;
        ip_addr_record_ptr->address.netAddress1.length              = 4;
        ip_addr_record_ptr->address.netAddress1.u.ipAddress    = *ip_addr;
        ip_addr_record_ptr->parse_control                                =
monCtrl.nodeParseCtrl;
        ip_addr_record_ptr->startTime                                    =
stats_start_time.tv_sec;
        ip_addr_record_ptr->lastTime                                     =
stats_start_time.tv_sec;
        if (rtp_net_broadcast || rtp_net_multicast)
            ip_addr_record_ptr->address.segment1                         = mySegmentId;
        else
            ip_addr_record_ptr->address.segment1                         =
UNKNOWN_SEGMENT_ID;

        FBInsqt (&statsIpAddrQ, (PFBQentry_type) ip_addr_record_ptr);
        }
```

```
    /*
     * If parse control is turned on, but a structure can't be obtained,
     * stats_ip_get_stats will return FALSE, so count this as a drop.
     */
    if (stats_ip_get_stats (ip_addr_record_ptr) == FALSE)
        stats_mon_ip_dropped;

    /*
     * Put this ip address record into the hash table.
     * Variable hash was set when stats_ip_lookup_addr was executed.
     */
    if ( ip_hash_table[ip_hash] == NULL)
        ip_hash_table[ip_hash] = ip_addr_record_ptr;
    else
        {
        /*
         * Find the last structure of the hash link
         */
        ip_hash_link = ip_hash_table[ip_hash];

        while (ip_hash_link->hash_link != NULL)
             ip_hash_link = ip_hash_link->hash_link;

        ip_hash_link->hash_link = ip_addr_record_ptr;
        }

     return (ip_addr_record_ptr);
}


/*****************************************************************************
 *
 * Find the structure for keeping ip address statistics for the given
 * ip address.  If one not found, attempt to allocate one.
 */
```

```
StatsAddrEntry *stats_ip_get_addr (ip_addr)

     Uint32              *ip_addr;
{
     register StatsAddrEntry       *ip_addr_record_ptr;

 ip_addr_record_ptr = stats_ip_lookup_addr (ip_addr);

 if (ip_addr_record_ptr == NULL)
     {
     stats_ip_new_node_alarm (ip_addr);
     ip_addr_record_ptr = stats_ip_init_addr (ip_addr);
     }
else
     if (stats_ip_get_stats (ip_addr_record_ptr) == FALSE)
          stats_mon_ip_dropped;

 return (ip_addr_record_ptr);
 }


/*****************************************************************
 *
 * Find the structure for keeping ip address statistics for the given
 * ip address.  If one not found, attempt to allocate one.
 */
StatsAddrEntry *stats_ip_get_parse (ip_addr)

     Uint32              *ip_addr;
{
     register StatsAddrEntry       *ip_addr_record_ptr;

     ip_addr_record_ptr = stats_ip_lookup_addr (ip_addr);
```

```
    if (ip_addr_record_ptr == NULL)
        ip_addr_record_ptr = stats_ip_init_addr (ip_addr);

    return (ip_addr_record_ptr);
}


/************************ stats_ip_lookup_dialog ******************************
 *
 * Find a ip dialog given 2 ip addresses.
 * If the dialog is not found, NULL is returned.
 */

StatsAddrEntry *stats_ip_lookup_dialog (ip_addr1, ip_addr2)

    Uint32                  *ip_addr1, *ip_addr2;
{
    register Uint32             xip_dialog_hash;
    register StatsAddrEntry     *xip_dialog_hash_link;
    register StatsAddrEntry     *xip_previous_dialog_hash_link;

    register StatsAddrEntry     *dialog_addr_ptr;

/*
 * Compute hash function on both IP addresses
 */
xip_dialog_hash = (*ip_addr1 & 0xffff0000) + (*ip_addr1 & 0x0000ffff) +
                  (*ip_addr2 & 0xffff0000) + (*ip_addr2 & 0x0000ffff);
xip_dialog_hash = ((xip_dialog_hash + ((xip_dialog_hash & 0xff00) >> 8))
                   & (IP_DIALOG_HASH_TABLE_SIZE - 1));

xip_previous_dialog_hash_link = NULL;
xip_dialog_hash_link = (StatsAddrEntry *) ip_dialog_hash_table[xip_dialog_hash];

/*
 * Walk linked list for exact entry
```

```
 */
while (xip_dialog_hash_link != NULL)
     {
     if (((xip_dialog_hash_link->address.netAddress1.u.ipAddress == *ip_addr1) &&
          (xip_dialog_hash_link->address.netAddress2.u.ipAddress == *ip_addr2)) ||

          ((xip_dialog_hash_link->address.netAddress1.u.ipAddress == *ip_addr2) &&
           (xip_dialog_hash_link->address.netAddress2.u.ipAddress == *ip_addr1) ))
               {
               break;
               }
          else
               {
               xip_previous_dialog_hash_link = xip_dialog_hash_link;
               xip_dialog_hash_link = xip_dialog_hash_link->hash_link;
               }
     }

ip_dialog_hash = xip_dialog_hash;
ip_dialog_hash_link = xip_dialog_hash_link;
ip_previous_dialog_hash_link = xip_previous_dialog_hash_link;

return (xip_dialog_hash_link);
}



/************************ stats_ip_get_dialog ******************************
 *
 * Find or allocate an ip dialog given 2 ip addresses.
 * If the dialog is not found, attempt to allocate a structure.
 */
StatsAddrEntry *stats_ip_get_dialog (ip_addr1, ip_addr2)

     Uint32                    *ip_addr1, *ip_addr2;
{
```

```
    register StatsAddrEntry       *dialog_addr_ptr;
    register StatsDialogEntry     *dialog_stats_ptr;
    register StatsDialogLink *dialog_link;
    register StatsAddrEntry       *ip_addr1_record_ptr, *ip_addr2_record_ptr;
    register StatsIpAddr          *ip_addr1_stats_ptr, *ip_addr2_stats_ptr;
    register Uint32                    parse_control;


dialog_addr_ptr = stats_ip_lookup_dialog (ip_addr1, ip_addr2);
if (dialog_addr_ptr != NULL)
    return (dialog_addr_ptr);

/*
 * Check parse control for the ip addresses.
 */
ip_addr1_record_ptr = stats_ip_lookup_addr (ip_addr1);
ip_addr2_record_ptr = stats_ip_lookup_addr (ip_addr2);

if ( (ip_addr1_record_ptr == NULL) && (ip_addr2_record_ptr == NULL) )
    return (NULL);

if (ip_addr1_record_ptr != NULL)
    parse_control = ip_addr1_record_ptr->parse_control;
if (ip_addr2_record_ptr != NULL)
    parse_control |= ip_addr2_record_ptr->parse_control;


/*
 * If ip is turned on for these ip addresses, allocate structures
 * for the dialog.  If can't get them, count this as a drop.
 */
if ((parse_control & MibParseIp) != MibParseIp)
    return (NULL);

/*
 * Try to allocate structures for this dialog.
 * If they can't be obtained, count this as a drop.
```

```
 */
dialog_addr_ptr = (StatsAddrEntry *) stats_allocate (sizeof (StatsAddrEntry) );
if (dialog_addr_ptr == NULL)
     {
     stats_mon_ip_dropped;
     return (NULL);
     }

dialog_stats_ptr = (StatsDialogEntry *) stats_allocate (sizeof (StatsDialogEntry) );
if (dialog_stats_ptr == NULL)
     {
     stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );
     stats_mon_ip_dropped;
     return (NULL);
     }

/*
 * Initialize the structures
 */
dialog_addr_ptr->hash_link                                    = NULL;
dialog_addr_ptr->address.addressType                  = MibNetAddress1 | MibNetAddress2;

dialog_addr_ptr->address.netAddress1.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress1.length         = 4;
dialog_addr_ptr->address.netAddress1.u.ipAddress    = *ip_addr1;
dialog_addr_ptr->address.netAddress2.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress2.length         = 4;
dialog_addr_ptr->address.netAddress2.u.ipAddress    = *ip_addr2;
dialog_addr_ptr->parse_control                                = parse_control;
dialog_addr_ptr->startTime                                    = stats_start_time.tv_sec;
dialog_addr_ptr->lastTime                                     = stats_start_time.tv_sec;
dialog_addr_ptr->stats_ptr                                    = (Uint32 *) dialog_stats_ptr;


FBInsqt (&statsIpDialogQ, (PFBQentry_type) dialog_addr_ptr);

dialog_stats_ptr->packetRate.type  = STATS_RATE_10S;
```

```
dialog_stats_ptr->byteRate.type    = STATS_RATE_10S;
dialog_stats_ptr->errorRate.type   = STATS_RATE_10S;
dialog_stats_ptr->fragmentRate.type = STATS_RATE_10S;
dialog_stats_ptr->rexmtRate.type   = STATS_RATE_10S;
dialog_stats_ptr->flowCtrlRate.type = STATS_RATE_10S;

setIpAddrPairDflts (&mibNodeDefaults.mibIpAddrPairDefaults, dialog_stats_ptr);

/*
 * Link the dialog statistics into the dialog queue for each ip address stats.
 */
if (ip_addr1_record_ptr != NULL)
    {
    if (ip_addr1_record_ptr->stats_ptr != NULL)
        {
        dialog_addr_ptr->address.addressType |= MibSegment1;
        dialog_addr_ptr->address.segment1 = ip_addr1_record_ptr->address.segment1;
        ip_addr1_stats_ptr = (StatsIpAddr *) ip_addr1_record_ptr->stats_ptr;
        dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
        if (dialog_link != NULL)
            {
            dialog_link->dialog_addr_ptr = dialog_addr_ptr;
            FBInsqt (&ip_addr1_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
            }
        }
    }
if (ip_addr2_record_ptr != NULL)
    {
    if (ip_addr2_record_ptr->stats_ptr != NULL)
        {
        dialog_addr_ptr->address.addressType |= MibSegment2;
        dialog_addr_ptr->address.segment2 = ip_addr2_record_ptr->address.segment1;
        ip_addr2_stats_ptr = (StatsIpAddr *) ip_addr2_record_ptr->stats_ptr;
        dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
        if (dialog_link != NULL)
            {
            dialog_link->dialog_addr_ptr = dialog_addr_ptr;
```

```
                FBInsqt (&ip_addr2_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
                }
            }
        }

/*
 * Put the new dialog address structure in the ip dialog hash table.
 * Variable hash was set up when stats_ip_lookup_dialog was executed.
 */
if (ip_dialog_hash_table[ip_dialog_hash] == NULL)
    ip_dialog_hash_table[ip_dialog_hash] = dialog_addr_ptr;
else
    {
    /*
     * Find the last structure of the hash link.
     */
    ip_dialog_hash_link = ip_dialog_hash_table[ip_dialog_hash];
    while (ip_dialog_hash_link->hash_link != NULL)
        ip_dialog_hash_link = ip_dialog_hash_link->hash_link;

    ip_dialog_hash_link->hash_link = dialog_addr_ptr;
    }

return (dialog_addr_ptr);
}

void setIpAddrDflts (mib_ip_addr_defs, ip_stats_addr_ptr)
    MibIpAddrDefaults   *mib_ip_addr_defs;
    StatsIpAddr         *ip_stats_addr_ptr;
{
    ip_stats_addr_ptr->rcvFrames.high_thld =
                        mib_ip_addr_defs->ipAddrRcvPktHighThld;
    ip_stats_addr_ptr->rcvFramesRate.high_thld =
                        mib_ip_addr_defs->ipAddrRcvPktRateHighThld;

    ip_stats_addr_ptr->rcvBytes.high_thld =
```

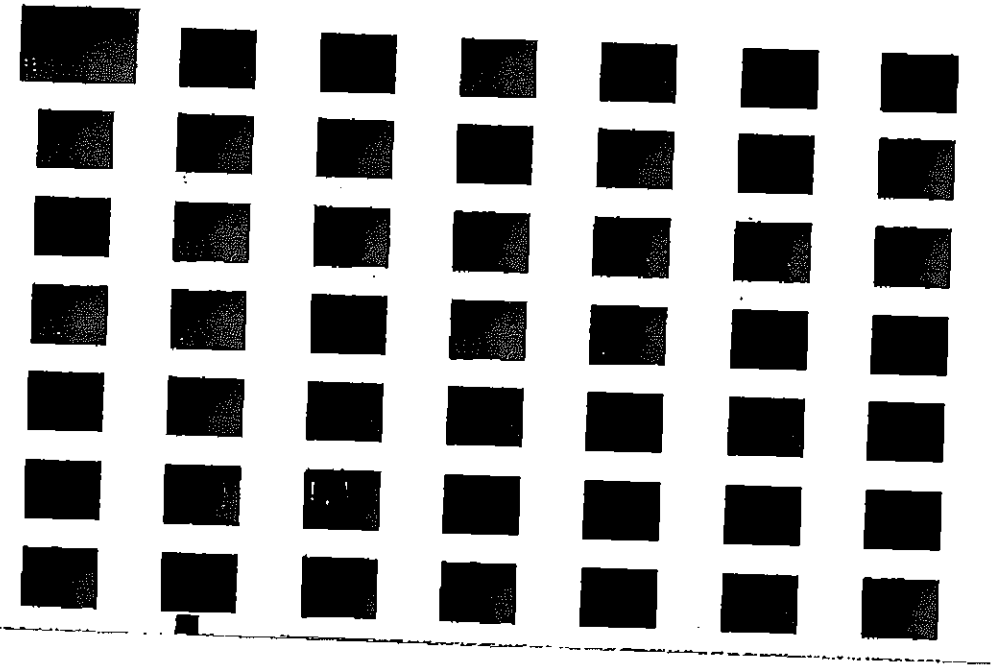
```
                        mib_ip_addr_defs->ipAddrRcvByteHighThld;
ip_stats_addr_ptr->rcvByteRate.high_thld =
                        mib_ip_addr_defs->ipAddrRcvByteRateHighThld;

ip_stats_addr_ptr->rcvErrors.high_thld =
                        mib_ip_addr_defs->ipAddrRcvErrorHighThld;
ip_stats_addr_ptr->rcvErrorRate.high_thld =
                        mib_ip_addr_defs->ipAddrRcvErrorRateHighThld;

ip_stats_addr_ptr->xmtFrames.high_thld =
                        mib_ip_addr_defs->ipAddrXmtPktHighThld;
ip_stats_addr_ptr->xmtFrameRate.high_thld =
                        mib_ip_addr_defs->ipAddrXmtPktRateHighThld;

ip_stats_addr_ptr->xmtBytes.high_thld =
                        mib_ip_addr_defs->ipAddrXmtByteHighThld;
ip_stats_addr_ptr->xmtByteRate.high_thld =
                        mib_ip_addr_defs->ipAddrXmtByteRateHighThld;

ip_stats_addr_ptr->xmtErrors.high_thld =
                        mib_ip_addr_defs->ipAddrXmtErrorHighThld;
ip_stats_addr_ptr->xmtErrorRate.high_thld =
                        mib_ip_addr_defs->ipAddrXmtErrorRateHighThld;

ip_stats_addr_ptr->rcvOffSegs.high_thld =
                        mib_ip_addr_defs->ipAddrRcvOffSegHighThld;
ip_stats_addr_ptr->rcvOffSegRate.high_thld =
                        mib_ip_addr_defs->ipAddrRcvOffSegRateHighThld;

ip_stats_addr_ptr->xmtOffSegs.high_thld =
                        mib_ip_addr_defs->ipAddrXmtOffSegHighThld;
ip_stats_addr_ptr->xmtOffSegRate.high_thld =
                        mib_ip_addr_defs->ipAddrXmtOffSegRateHighThld;

ip_stats_addr_ptr->xmtBcasts.high_thld =
                        mib_ip_addr_defs->ipAddrXmtBcastHighThld;
ip_stats_addr_ptr->xmtBcastRate.high_thld =
```

```
                                mib_ip_addr_defs->ipAddrXmtBcastRateHighThld;

    ip_stats_addr_ptr->xmtMcasts.high_thld =
                                mib_ip_addr_defs->ipAddrXmtMcastHighThld;
    ip_stats_addr_ptr->xmtMcastRate.high_thld =
                                mib_ip_addr_defs->ipAddrXmtMcastRateHighThld;

    ip_stats_addr_ptr->rcvHdrBytes.high_thld =
                                mib_ip_addr_defs->ipAddrRcvHdrByteHighThld;
    ip_stats_addr_ptr->rcvHdrByteRate.high_thld =
                                mib_ip_addr_defs->ipAddrRcvHdrByteRateHighThld;

    ip_stats_addr_ptr->xmtHdrBytes.high_thld =
                                mib_ip_addr_defs->ipAddrXmtHdrByteHighThld;
    ip_stats_addr_ptr->xmtHdrByteRate.high_thld =
                                mib_ip_addr_defs->ipAddrXmtHdrByteRateHighThld;

    ip_stats_addr_ptr->rcvFrgmts.high_thld =
                                mib_ip_addr_defs->ipAddrRcvFrgmtHighThld;
    ip_stats_addr_ptr->rcvFrgmtRate.high_thld =
                                mib_ip_addr_defs->ipAddrRcvFrgmtRateHighThld;

    ip_stats_addr_ptr->xmtFrgmts.high_thld =
                                mib_ip_addr_defs->ipAddrXmtFrgmtHighThld;
    ip_stats_addr_ptr->xmtFrgmtRate.high_thld =
                                mib_ip_addr_defs->ipAddrXmtFrgmtRateHighThld;
}

void setIpAddrPairDflts (mib_ip_addr_pair_defs, dialog_ptr)
    MibIpAddrPairDefaults    *mib_ip_addr_pair_defs;
    StatsDialogEntry         *dialog_ptr;
{
    dialog_ptr->packets.high_thld =
                            mib_ip_addr_pair_defs->ipAddrPairPktHighThld;
    dialog_ptr->packetRate.high_thld =
                            mib_ip_addr_pair_defs->ipAddrPairPktRateHighThld;
```

```
    dialog_ptr->bytes.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairByteHighThld;
    dialog_ptr->byteRate.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairByteRateHighThld;

    dialog_ptr->errors.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairErrorHighThld;
    dialog_ptr->errorRate.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairErrorRateHighThld;

    dialog_ptr->fragments.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairFrgmtHighThld;
    dialog_ptr->fragmentRate.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairFrgmtRateHighThld;

    dialog_ptr->rexmts.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairRexmtHighThld;
    dialog_ptr->rexmtRate.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairRexmtRateHighThld;

    dialog_ptr->flowCtrls.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairFlowCtrlHighThld;
    dialog_ptr->flowCtrlRate.high_thld =
                        mib_ip_addr_pair_defs->ipAddrPairFlowCtrlRateHighThld;
}

void setIpSegDflts (mib_ip_seg_defs, ip_seg_ptr)
    MibIpSegDefaults    *mib_ip_seg_defs;
    StatsIpSegment      *ip_seg_ptr;
{
    ip_seg_ptr->frames.high_thld =
                        mib_ip_seg_defs->ipSegPktHighThld;
    ip_seg_ptr->frameRate.high_thld =
                        mib_ip_seg_defs->ipSegPktRateHighThld;
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones, Mary Robertson, David M. Thompson and Gerard White

10



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
ip_seg_ptr->bytes.high_thld =
                    mib_ip_seg_defs->ipSegByteHighThld;
ip_seg_ptr->byteRate.high_thld =
                    mib_ip_seg_defs->ipSegByteRateHighThld;

ip_seg_ptr->errors.high_thld =
                    mib_ip_seg_defs->ipSegErrorHighThld;
ip_seg_ptr->errorRate.high_thld =
                    mib_ip_seg_defs->ipSegErrorRateHighThld;

ip_seg_ptr->rcvOffSegs.high_thld =
                    mib_ip_seg_defs->ipSegRcvOffSegHighThld;
ip_seg_ptr->rcvOffSegRate.high_thld =
                    mib_ip_seg_defs->ipSegRcvOffSegRateHighThld;

ip_seg_ptr->xmtOffSegs.high_thld =
                    mib_ip_seg_defs->ipSegXmtOffSegHighThld;
ip_seg_ptr->xmtOffSegRate.high_thld =
                    mib_ip_seg_defs->ipSegXmtOffSegRateHighThld;

ip_seg_ptr->transits.high_thld =
                    mib_ip_seg_defs->ipSegTransitHighThld;
ip_seg_ptr->transitRate.high_thld =
                    mib_ip_seg_defs->ipSegTransitRateHighThld;

ip_seg_ptr->flowCtrls.high_thld =
                    mib_ip_seg_defs->ipSegFlowCtrlHighThld;
ip_seg_ptr->flowCtrlRate.high_thld =
                    mib_ip_seg_defs->ipSegFlowCtrlRateHighThld;

ip_seg_ptr->bcasts.high_thld =
                    mib_ip_seg_defs->ipSegBcastHighThld;
ip_seg_ptr->bcastRate.high_thld =
                    mib_ip_seg_defs->ipSegBcastRateHighThld;

ip_seg_ptr->mcasts.high_thld =
                    mib_ip_seg_defs->ipSegMcastHighThld;
```

```
        ip_seg_ptr->mcastRate.high_thld =
                            mib_ip_seg_defs->ipSegMcastRateHighThld;

        ip_seg_ptr->frgmts.high_thld =
                            mib_ip_seg_defs->ipSegFrgmtHighThld;
        ip_seg_ptr->frgmtRate.high_thld =
                            mib_ip_seg_defs->ipSegFrgmtRateHighThld;

        ip_seg_ptr->hdrBytes.high_thld =
                            mib_ip_seg_defs->ipSegHdrByteHighThld;
        ip_seg_ptr->hdrByteRate.high_thld =
                            mib_ip_seg_defs->ipSegHdrByteRateHighThld;
}
```

```
/*
 *  stats_nfs_a.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_nfs_a.c
 *
 *  Date:       6/6/91
 *
 *  Revision:   1.11
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   06-06-91  KR        Deallocated file systen links in client and server aging
 *   06-05-91  KR        Changed StatsDialogEntry to StatsNfsDialogEntry
 *   06-05-91  DPD       Fixed aging bug
 *   06-02-91  KR        Added deallocate of state info in dialog age routine
 *   06-01-91  DPD       Fixed rate bug
 *   05-31-91  DPD       Added rate routines
 *
 */

static char stats_nfs_a_c [] = "@(#)stats_nfs_a.c 1.11";

#include <stdio.h>

#include <cci_std.h>
```

```
#include "system.h"
#include "address.h"
#include "mib_defs.h"
#include "mib_nfs.h"

#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "nbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "protocols.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_rpc.h"
#include "stats_nfs.h"
#include "stats_mib.h"
#include "snmpd.h"
#include "em_ctrl.h"



/*****************************************************************************
 *
 * Given an NFS dialog, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsNfsAgeDialog (dialog_addr_ptr)
```

```
    StatsAddrEntry      *dialog_addr_ptr;

{
    register Uint32                         xnfs_dialog_hash;
    register StatsAddrEntry                *xnfs_dialog_hash_link;
    register StatsAddrEntry                *xnfs_previous_dialog_hash_link;

    register StatsNfsDialogEntry  *dialog_stats_ptr;
    register StatsDialogLink      *dialog_link_ptr;
    register StatsAddrEntry                *addr_record_ptr;
    register StatsNfsAddr                  *nfs_stats_ptr;
    struct timeval                          current_time;
    Uint32                                  i;

if (monCtrl.nfsDialogAgeTimer == 0)
    return;

if (dialog_addr_ptr == NULL)
    return;

dialog_stats_ptr = (StatsNfsDialogEntry *) dialog_addr_ptr->stats_ptr;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - dialog_addr_ptr->lastTime < monCtrl.nfsDialogAgeTimer)
    return;

/*
 * Time to chuck this dialog.  Remove it from the dialog hash table.
 * Look up the dialog in order to set nfs_dialog_hash, nfs_previous_dialog_hash_link
 * and nfs_dialog_hash_link.
 */
addr_record_ptr = stats_nfs_lookup_dialog (
    &dialog_addr_ptr->address.netAddress1.u.ipAddress,
    &dialog_addr_ptr->address.netAddress2.u.ipAddress);
```

```
if (addr_record_ptr != dialog_addr_ptr)
     {
     mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsNfsAgeDialog: ptrs not equal");
     return;
     }

xnfs_dialog_hash = nfs_dialog_hash;
xnfs_dialog_hash_link = nfs_dialog_hash_link;
xnfs_previous_dialog_hash_link = nfs_previous_dialog_hash_link;

if ( (xnfs_previous_dialog_hash_link == NULL) && (xnfs_dialog_hash_link->hash_link ==
NULL) )
     nfs_dialog_hash_table[xnfs_dialog_hash] = NULL;
else
     {
     if (xnfs_previous_dialog_hash_link != NULL)
          xnfs_previous_dialog_hash_link->hash_link = xnfs_dialog_hash_link->hash_link;
     else
          if (xnfs_dialog_hash_link->hash_link != NULL)
               nfs_dialog_hash_table[xnfs_dialog_hash] = xnfs_dialog_hash_link->hash_link;

     }

/*
 * Remove the dialog from the dialogQ for each of the nfs stats structures for the
 * ip addresses and deallocate the structure that was on the dialogQ which pointed
 * to the dialog.
 */
addr_record_ptr = stats_nfs_lookup_server
(&dialog_addr_ptr->address.netAddress1.u.ipAddress);

for (i=0; i<2; i++)
     {
     if (addr_record_ptr != NULL)
          {
          nfs_stats_ptr = (StatsNfsAddr *) addr_record_ptr->stats_ptr;
          if (nfs_stats_ptr != NULL)
```

```
                {
                dialog_link_ptr = (StatsDialogLink *) nfs_stats_ptr->dialogQ.pFlink;
                while (dialog_link_ptr != NULL)
                     {
                     if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                          break;
                     dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
                     }
                if (dialog_link_ptr != NULL)
                     {
                     FBRemqm (&nfs_stats_ptr->dialogQ, (PFBQentry_type) dialog_link_ptr);
                     stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                     }
                }
           }
           addr_record_ptr = stats_nfs_lookup_client (
                &dialog_addr_ptr->address.netAddress2.u.ipAddress);
      }


/*
 * Remove the dialog from the statsNfsDialogQ and deallocate the structures
 * associated with the dialog.
 */
FBRemqm (&statsNfsDialogQ, (PFBQentry_type) dialog_addr_ptr);

stats_deallocate (dialog_stats_ptr->state_ptr, sizeof(StatsNfsState) );
stats_deallocate (dialog_stats_ptr, sizeof(StatsNfsDialogEntry) );
stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );

}



/*****************************************************************************
 *
```

```
 * Given a client, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsNfsAgeClient (nfs_addr_ptr)

     StatsAddrEntry      *nfs_addr_ptr;

{
     register StatsDialogLink      *dialog_link_ptr;
     register StatsAddrEntry            *addr_record_ptr;
     register StatsNfsFileSystemLink    *file_system_link_ptr;
     register StatsNfsAddr              *nfs_stats_ptr;
     struct timeval                     current_time;

if (monCtrl.nfsClientAgeTimer == 0)
     return;

if (nfs_addr_ptr == NULL)
     return;

nfs_stats_ptr = (StatsNfsAddr *) nfs_addr_ptr->stats_ptr;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - nfs_addr_ptr->lastTime < monCtrl.nfsClientAgeTimer)
     return;

/*
 * Time to chuck this client.  Remove it from the hash table.
 * Look up the ip address in order to set nfs_client_hash, nfs_previous_client_hash_link

 * and nfs_client_hash_link.
 */
addr_record_ptr = stats_nfs_lookup_client (
     &nfs_addr_ptr->address.netAddress1.u.ipAddress);
```

```
if (addr_record_ptr != nfs_addr_ptr)
     {
     mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsNfsAgeClient: ptrs not equal");
     return;
     }

if ( (nfs_previous_client_hash_link == NULL) && (nfs_client_hash_link->hash_link == NULL)
)
     nfs_client_hash_table[nfs_client_hash] = NULL;
else
     {
     if (nfs_previous_client_hash_link != NULL)
          nfs_previous_client_hash_link->hash_link = nfs_client_hash_link->hash_link;
     else
          if (nfs_client_hash_link->hash_link != NULL)
               nfs_client_hash_table[nfs_client_hash] = nfs_client_hash_link->hash_link;
     }

/*
 * Remove the dialog links and the file system link from the client stats then
 * deallocate the client stats structure.
 */
if (nfs_stats_ptr != NULL)
     {
     dialog_link_ptr = (StatsDialogLink *) FBRemqh (&nfs_stats_ptr->dialogQ);
     while (dialog_link_ptr != NULL)
          {
          stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
          dialog_link_ptr = (StatsDialogLink *) FBRemqh (&nfs_stats_ptr->dialogQ);
          }
     file_system_link_ptr = (StatsNfsFileSystemLink *) FBRemqh
(&nfs_stats_ptr->fileSystemQ);
     while (file_system_link_ptr != NULL)
          {
          stats_deallocate (file_system_link_ptr, sizeof(StatsNfsFileSystemLink) );
          file_system_link_ptr = (StatsNfsFileSystemLink *) FBRemqh
(&nfs_stats_ptr->fileSystemQ);
```

```
        }
    stats_deallocate (nfs_stats_ptr, sizeof(StatsNfsAddr) );
    }


/*
 * Remove the address from the statsNfsClientQ and deallocate the address structure.
 */
FBRemqm (&statsNfsClientQ, (PFBQentry_type) nfs_addr_ptr);
stats_deallocate (nfs_addr_ptr, sizeof(StatsAddrEntry) );

}




/*********************************************************************
 *
 * Given a server, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsNfsAgeServer (server_addr_ptr)

    StatsAddrEntry      *server_addr_ptr;

{
    register StatsDialogLink      *dialog_link_ptr;
    register StatsAddrEntry            *addr_record_ptr;
    register StatsNfsFileSystemLink    *file_system_link_ptr;
    register StatsNfsAddr              *server_stats_ptr;
    struct timeval                     current_time;

if (monCtrl.nfsServerAgeTimer == 0)
```

```
        return;

    if (server_addr_ptr == NULL)
        return;

    server_stats_ptr = (StatsNfsAddr *) server_addr_ptr->stats_ptr;

    mon_gettimeofday (&current_time, 0);

    if (current_time.tv_sec - server_addr_ptr->lastTime < monCtrl.nfsServerAgeTimer)
        return;

    /*
     * Time to chuck this server.  Remove it from the server hash table.
     * Look up the server in order to set nfs_server_hash, nfs_previous_server_hash_link
     * and nfs_server_hash_link.
     */
    addr_record_ptr = stats_nfs_lookup_server (
        &server_addr_ptr->address.netAddress1.u.ipAddress);

    if (addr_record_ptr != server_addr_ptr)
        {
        mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsNfsAgeServer: ptrs not equal");
        return;
        }

    if ( (nfs_previous_server_hash_link == NULL) && (nfs_server_hash_link->hash_link == NULL)
    )
        nfs_server_hash_table[nfs_server_hash] = NULL;
    else
        {
        if (nfs_previous_server_hash_link != NULL)
            nfs_previous_server_hash_link->hash_link = nfs_server_hash_link->hash_link;
        else
            if (nfs_server_hash_link->hash_link != NULL)
                nfs_server_hash_table[nfs_server_hash] = nfs_server_hash_link->hash_link;
        }
```

```
/*
 * Remove the dialog links and the file system link from the server stats then
 * deallocate the server stats structure.
 */
if (server_stats_ptr != NULL)
     {
     dialog_link_ptr = (StatsDialogLink *) FBRemqh (&server_stats_ptr->dialogQ);
     while (dialog_link_ptr != NULL)
          {
          stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
          dialog_link_ptr = (StatsDialogLink *) FBRemqh (&server_stats_ptr->dialogQ);
          }
     file_system_link_ptr = (StatsNfsFileSystemLink *) FBRemqh
(&server_stats_ptr->fileSystemQ);
     while (file_system_link_ptr != NULL)
          {
          stats_deallocate (file_system_link_ptr, sizeof(StatsNfsFileSystemLink) );
          file_system_link_ptr = (StatsNfsFileSystemLink *)
FBRemqh(&server_stats_ptr->fileSystemQ);
          }
     stats_deallocate (server_stats_ptr, sizeof(StatsNfsAddr) );
     }


/*
 * Remove the server from the statsNfsServerQ and deallocate the address structure.
 */
FBRemqm (&statsNfsServerQ, (PFBQentry_type) server_addr_ptr);
stats_deallocate (server_addr_ptr, sizeof(StatsAddrEntry) );

}


/*****************************************************************************
 *
```

```
 * NFS rate routines
 */


void statsNfsSegRate (rate_type)
     Uint32                    rate_type;
{
     register StatsNfsSegment *nfs_seg_ptr;
     register StatsAddrEntry       *nfs_seg_addr_ptr;
     struct timeval                current_time;
     register StatsProtocolEntry   *nfs_protocol_ptr;

     nfs_seg_addr_ptr = (StatsAddrEntry *) statsNfsSegQ.pFlink;

     while (nfs_seg_addr_ptr != NULL)
     {
          nfs_seg_ptr = (StatsNfsSegment *) nfs_seg_addr_ptr->stats_ptr;
          if (nfs_seg_ptr != NULL)
          {
               /* get the current time */
               mon_gettimeofday(&current_time, 0);

               statsCalcRates(&nfs_seg_ptr->opRate, &nfs_seg_ptr->opBuckets,
                              rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                              AL_OPS, current_time.tv_sec, NULL);
               statsCalcRates(&nfs_seg_ptr->readOpRate, NULL,
                              rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                              AL_READ_OPS, current_time.tv_sec, NULL);
               statsCalcRates(&nfs_seg_ptr->writeOpRate, NULL,
                              rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                              AL_WRITE_OPS, current_time.tv_sec, NULL);
               statsCalcRates(&nfs_seg_ptr->writeCacheOpRate, NULL,
                              rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
```

```
                                AL_WRITE_CACHE_OPS, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->byteRate, &nfs_seg_ptr->byteBuckets,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_BYTES, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->readByteRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_READ_BYTES, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->writeByteRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_WRITE_BYTES, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rcvOffSegRate, &nfs_seg_ptr->rcvOffSegBuckets,

                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->xmtOffSegRate, &nfs_seg_ptr->xmtOffSegBuckets,

                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_XMT_OFF_SEG, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->transitRate, &nfs_seg_ptr->transitBuckets,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_TRANSIT, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->flowCtrlRate, &nfs_seg_ptr->flowCtrlBuckets,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_FLOW_CTRLS, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rexmtRate, &nfs_seg_ptr->rexmtBuckets,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_REXMTS, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->frgmtRate, NULL,
```

```
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_FRAGMENTS, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->errorRate, &nfs_seg_ptr->errorBuckets,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_ERRORS, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->mountFailureRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_MOUNT_FAILURES, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrPermRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_PERM, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrNoEntRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_NO_ENT, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrIORate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_IO, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrNXIORate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_NXIO, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrAccessRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_ACCESS, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrExistRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_EXIST, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrNoDevRate, NULL,
```

```
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_NO_DEV, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrNotDirRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_NOT_DIR, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrIsDirRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_IS_DIR, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrFBigRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_FBIG, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrNoSpaceRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_NO_SPACE, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrROFSRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_ROFS, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrNameTooLongRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_NAME_TOO_LONG, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrNotEmptyRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_NOT_EMPTY, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrDQuotaRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                AL_NFS_ERR_DQUOTA, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrStaleRate, NULL,
```

```
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
        NFS_SEGMENT,
                                AL_NFS_ERR_STALE, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->nfsErrWFlushRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
        NFS_SEGMENT,
                                AL_NFS_ERR_WFLUSH, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.versionMismatchRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
        NFS_SEGMENT,
                                AL_RPC_VERSION, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.authBadCredRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
        NFS_SEGMENT,
                                AL_RPC_BAD_CRED, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.authRejectedCredRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
        NFS_SEGMENT,
                                AL_RPC_REJECT_CRED, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.authBadVerfRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
        NFS_SEGMENT,
                                AL_RPC_BAD_VERF, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.authRejectedVerfRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
        NFS_SEGMENT,
                                AL_RPC_REJECT_VERF, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.authTooWeakRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
        NFS_SEGMENT,
                                AL_RPC_AUTH_WEAK, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.authOtherRate, NULL,
                                rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
        NFS_SEGMENT,
                                AL_RPC_AUTH_OTHER, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.progUnavailRate, NULL,
```

```
                                        rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                        AL_RPC_PROG_UNAVAIL, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.progVersionMismatchRate, NULL,
                                        rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                        AL_RPC_PROG_VERSION, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.procUnavailRate, NULL,
                                        rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                        AL_RPC_PROC_UNAVAIL, current_time.tv_sec, NULL);
                statsCalcRates(&nfs_seg_ptr->rpcStats.procGarbageArgsRate, NULL,
                                        rate_type, (StatsAddrEntry *)nfs_seg_addr_ptr,
NFS_SEGMENT,
                                        AL_RPC_PROC_GARBAGE, current_time.tv_sec, NULL);

                statsNfsDialogRate (rate_type);

                if ((current_time.tv_sec - nfs_seg_addr_ptr->seconds_start_time) >=
monCtrl.rateTimer)
                        nfs_seg_addr_ptr->seconds_start_time = current_time.tv_sec;
            }
            nfs_seg_addr_ptr = (StatsAddrEntry *) nfs_seg_addr_ptr->link.pFlink;
        }
}


void statsNfsRate (nfs_addr_stats_ptr, nfs_entry_ptr, rate_type, current_time)
        StatsNfsAddr        *nfs_addr_stats_ptr;
        StatsAddrEntry      *nfs_entry_ptr;
        Uint32                  rate_type;
        struct timeval      current_time;
{
        statsCalcRates(&nfs_addr_stats_ptr->opRate, &nfs_addr_stats_ptr->opBuckets,
                        rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                        NFS_NODE, AL_OPS, current_time.tv_sec, NULL);
```

```
    statsCalcRates(&nfs_addr_stats_ptr->readOpRate, NULL,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_READ_OPS, current_time.tv_sec, NULL);
    statsCalcRates(&nfs_addr_stats_ptr->writeOpRate, NULL,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_WRITE_OPS, current_time.tv_sec, NULL);
    statsCalcRates(&nfs_addr_stats_ptr->writeCacheOpRate, NULL,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_WRITE_CACHE_OPS, current_time.tv_sec, NULL);

    statsCalcRates(&nfs_addr_stats_ptr->byteRate, &nfs_addr_stats_ptr->byteBuckets,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_BYTES, current_time.tv_sec, NULL);
    statsCalcRates(&nfs_addr_stats_ptr->readByteRate, NULL,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_READ_BYTES, current_time.tv_sec, NULL);
    statsCalcRates(&nfs_addr_stats_ptr->writeByteRate, NULL,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_WRITE_BYTES, current_time.tv_sec, NULL);

    statsCalcRates(&nfs_addr_stats_ptr->errorRate, &nfs_addr_stats_ptr->errorBuckets,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_ERRORS, current_time.tv_sec, NULL);

    statsCalcRates(&nfs_addr_stats_ptr->rcvOffSegRate,
&nfs_addr_stats_ptr->rcvOffSegBuckets,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
    statsCalcRates(&nfs_addr_stats_ptr->xmtOffSegRate,
&nfs_addr_stats_ptr->xmtOffSegBuckets,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_XMT_OFF_SEG, current_time.tv_sec, NULL);

    statsCalcRates(&nfs_addr_stats_ptr->flowCtrlRate,
&nfs_addr_stats_ptr->flowCtrlBuckets,
                   rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                   NFS_NODE, AL_FLOW_CTRLS, current_time.tv_sec, NULL);
```

```
statsCalcRates(&nfs_addr_stats_ptr->rcvFrgmtRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_RCV_FRAGMENTS, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->xmtFrgmtRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_XMT_FRAGMENTS, current_time.tv_sec, NULL);

statsCalcRates(&nfs_addr_stats_ptr->rexmtRate, &nfs_addr_stats_ptr->rexmtBuckets,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_REXMTS, current_time.tv_sec, NULL);

statsCalcRates(&nfs_addr_stats_ptr->mountFailureRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_MOUNT_FAILURES, current_time.tv_sec, NULL);

statsCalcRates(&nfs_addr_stats_ptr->nfsErrPermRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_PERM, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrNoEntRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_NO_ENT, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrIORate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_IO, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrNXIORate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_NXIO, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrAccessRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_ACCESS, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrExistRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_EXIST, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrNoDevRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_NO_DEV, current_time.tv_sec, NULL);
```

```
statsCalcRates(&nfs_addr_stats_ptr->nfsErrNotDirRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_NOT_DIR, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrIsDirRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_IS_DIR, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrFBigRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_FBIG, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrNoSpaceRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_NO_SPACE, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrROFSRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_ROFS, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrNameTooLongRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_NAME_TOO_LONG, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrNotEmptyRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_NOT_EMPTY, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrDQuotaRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_DQUOTA, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrStaleRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_STALE, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->nfsErrWFlushRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_NFS_ERR_WFLUSH, current_time.tv_sec, NULL);

statsCalcRates(&nfs_addr_stats_ptr->rpcStats.versionMismatchRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_RPC_VERSION, current_time.tv_sec, NULL);
statsCalcRates(&nfs_addr_stats_ptr->rpcStats.authBadCredRate, NULL,
                    rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                    NFS_NODE, AL_RPC_BAD_CRED, current_time.tv_sec, NULL);
```

```
        statsCalcRates(&nfs_addr_stats_ptr->rpcStats.authRejectedCredRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_NODE, AL_RPC_REJECT_CRED, current_time.tv_sec, NULL);
        statsCalcRates(&nfs_addr_stats_ptr->rpcStats.authBadVerfRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_NODE, AL_RPC_BAD_VERF, current_time.tv_sec, NULL);
        statsCalcRates(&nfs_addr_stats_ptr->rpcStats.authRejectedVerfRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_NODE, AL_RPC_REJECT_VERF, current_time.tv_sec, NULL);
        statsCalcRates(&nfs_addr_stats_ptr->rpcStats.authTooWeakRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_NODE, AL_RPC_AUTH_WEAK, current_time.tv_sec, NULL);
        statsCalcRates(&nfs_addr_stats_ptr->rpcStats.authOtherRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_NODE, AL_RPC_AUTH_OTHER, current_time.tv_sec, NULL);
        statsCalcRates(&nfs_addr_stats_ptr->rpcStats.progUnavailRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_NODE, AL_RPC_PROG_UNAVAIL, current_time.tv_sec, NULL);
        statsCalcRates(&nfs_addr_stats_ptr->rpcStats.progVersionMismatchRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_NODE, AL_RPC_PROG_VERSION, current_time.tv_sec, NULL);
        statsCalcRates(&nfs_addr_stats_ptr->rpcStats.procUnavailRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_NODE, AL_RPC_PROC_UNAVAIL, current_time.tv_sec, NULL);
        statsCalcRates(&nfs_addr_stats_ptr->rpcStats.procGarbageArgsRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_NODE, AL_RPC_PROC_GARBAGE, current_time.tv_sec, NULL);
}

void statsNfsServerRate (rate_type)
        Uint32                  rate_type;
{
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
     register StatsAddrEntry        *nfs_entry_ptr;
     register StatsNfsAddr          *nfs_addr_stats_ptr;
     register Uint32                     loop_count;
```

```
        struct timeval                  current_time;

        /* get the current time */
        mon_gettimeofday(&current_time, 0);

        loop_count = 0;

        /***** AGING *****/
        if (statsNextNfsServerEntry != NULL)
             entry_ptr = statsNextNfsServerEntry;          /* pick up where we left off */
        else
        {
             /* Only age structures if we've processed them all */
             entry_ptr = statsNfsServerQ.pFlink;
             while (entry_ptr != NULL)
             {
                  next_entry_ptr = entry_ptr->pFlink;
                  statsNfsAgeServer ((StatsAddrEntry *) entry_ptr);
                  entry_ptr = next_entry_ptr;
             }

             entry_ptr = statsNfsServerQ.pFlink;           /* start at the beginning */
        }
        /***** AGING *****/

        while ((entry_ptr != NULL) && (loop_count < stats_q_count))
        {
             nfs_entry_ptr = (StatsAddrEntry *)entry_ptr;
             if (((nfs_entry_ptr->em_control & rate_type) != 0)
                                  && (nfs_entry_ptr->stats_ptr != NULL))

             {
                  if ((current_time.tv_sec - nfs_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
                  {
                  nfs_addr_stats_ptr = (StatsNfsAddr *)nfs_entry_ptr->stats_ptr;
                  statsNfsRate (nfs_addr_stats_ptr, nfs_entry_ptr, rate_type, current_time);
```

```
                nfs_entry_ptr->seconds_start_time = current_time.tv_sec;
                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }
        statsNextNfsServerEntry = entry_ptr;          /* pick up where we left off */

        if (statsNextNfsServerEntry != NULL)
            statsRatesNotDone |= STATS_NFS_SERVER_RATE;
}


void statsNfsClientRate (rate_type)
        Uint32                  rate_type;
{
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
     register StatsAddrEntry        *nfs_entry_ptr;
     register StatsNfsAddr          *nfs_addr_stats_ptr;
     register Uint32                    loop_count;
     struct timeval                 current_time;

     /* get the current time */
     mon_gettimeofday(&current_time, 0);

     loop_count = 0;

     /***** AGING *****/
     if (statsNextNfsClientEntry != NULL)
          entry_ptr = statsNextNfsClientEntry;          /* pick up where we left off */
     else
     {
          /* Only age structures if we've processed them all */
          entry_ptr = statsNfsClientQ.pFlink;
          while (entry_ptr != NULL)
```

```
        {
            next_entry_ptr = entry_ptr->pFlink;
            statsNfsAgeClient ((StatsAddrEntry *) entry_ptr);
            entry_ptr = next_entry_ptr;
        }

        entry_ptr = statsNfsClientQ.pFlink;          /* start at the beginning */
    }
    /***** AGING *****/

    while ((entry_ptr != NULL) && (loop_count < stats_q_count))
    {
        nfs_entry_ptr = (StatsAddrEntry *)entry_ptr;
        if (((nfs_entry_ptr->em_control & rate_type) != 0)
                                && (nfs_entry_ptr->stats_ptr != NULL))
        {
            if ((current_time.tv_sec - nfs_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
            {
            nfs_addr_stats_ptr = (StatsNfsAddr *)nfs_entry_ptr->stats_ptr;
            statsNfsRate (nfs_addr_stats_ptr, nfs_entry_ptr, rate_type, current_time);


            nfs_entry_ptr->seconds_start_time = current_time.tv_sec;
            loop_count++;
            }
        }
        entry_ptr = entry_ptr->pFlink;
    }
    statsNextNfsClientEntry = entry_ptr;         /* pick up where we left off */

    if (statsNextNfsClientEntry != NULL)
        statsRatesNotDone |= STATS_NFS_CLIENT_RATE;
}

void statsNfsDialogRate (rate_type)
    Uint32                  rate_type;
```

```
{
     register StatsNfsDialogEntry  *nfs_dialog_ptr;
    register FBQentry_type              *entry_ptr;
    register FBQentry_type              *next_entry_ptr;
     register StatsAddrEntry            *nfs_entry_ptr;
     register Uint32                         loop_count;
     struct timeval                     current_time;

     /* get the current time */
     mon_gettimeofday(&current_time, 0);

     if (statsNextNfsPairEntry != NULL)
          entry_ptr = statsNextNfsPairEntry;      /* pick up where we left off */
     else
     {
          /* Only age structures if we've processed them all */
          entry_ptr = statsNfsDialogQ.pFlink;
          while (entry_ptr != NULL)
          {
               next_entry_ptr = entry_ptr->pFlink;
               statsNfsAgeDialog ((StatsAddrEntry *) entry_ptr);
               entry_ptr = next_entry_ptr;
          }

          entry_ptr = statsNfsDialogQ.pFlink;          /* start at the beginning */
     }

     loop_count = 0;
     while ((entry_ptr != NULL) && (loop_count < stats_q_count))
     {
          nfs_entry_ptr = (StatsAddrEntry *)entry_ptr;
          if (((nfs_entry_ptr->em_control & rate_type) != 0)
                                   && (nfs_entry_ptr->stats_ptr != NULL))
          {
               if ((current_time.tv_sec - nfs_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
               {
```

```
        nfs_dialog_ptr = (StatsNfsDialogEntry *)nfs_entry_ptr->stats_ptr;

        statsCalcRates(&nfs_dialog_ptr->opRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_PAIR, AL_OPS, current_time.tv_sec, NULL);

        statsCalcRates(&nfs_dialog_ptr->readOpRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_PAIR, AL_READ_OPS, current_time.tv_sec, NULL);

        statsCalcRates(&nfs_dialog_ptr->writeOpRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_PAIR, AL_WRITE_OPS, current_time.tv_sec, NULL);

        statsCalcRates(&nfs_dialog_ptr->writeCacheOpRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_PAIR, AL_WRITE_CACHE_OPS, current_time.tv_sec,
NULL);
        statsCalcRates(&nfs_dialog_ptr->readByteRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_PAIR, AL_READ_BYTES, current_time.tv_sec, NULL);

        statsCalcRates(&nfs_dialog_ptr->writeByteRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_PAIR, AL_WRITE_BYTES, current_time.tv_sec, NULL);

        statsCalcRates(&nfs_dialog_ptr->reXmtRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_PAIR, AL_REXMTS, current_time.tv_sec, NULL);

        statsCalcRates(&nfs_dialog_ptr->rpcProgramFailRate, NULL,
                       rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                       NFS_PAIR, AL_RPC_PROG_FAIL, current_time.tv_sec, NULL);


        statsCalcRates(&nfs_dialog_ptr->rpcMismatchRate, NULL,
```

```
                                        rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                                        NFS_PAIR, AL_RPC_MISMATCH, current_time.tv_sec, NULL);

                statsCalcRates(&nfs_dialog_ptr->rpcAuthFailRate, NULL,
                                        rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                                        NFS_PAIR, AL_RPC_AUTH_FAIL, current_time.tv_sec, NULL);

                statsCalcRates(&nfs_dialog_ptr->nfsAccessErrRate, NULL,
                                        rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                                        NFS_PAIR, AL_RPC_ACCESS_ERR, current_time.tv_sec,
NULL);

                statsCalcRates(&nfs_dialog_ptr->nfsHardErrRate, NULL,
                                        rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                                        NFS_PAIR, AL_RPC_HARD_ERR, current_time.tv_sec, NULL);

                statsCalcRates(&nfs_dialog_ptr->nfsResourceErrRate, NULL,
                                        rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                                        NFS_PAIR, AL_RPC_RESOURCE_ERR, current_time.tv_sec,
NULL);

                statsCalcRates(&nfs_dialog_ptr->nfsUserErrRate, NULL,
                                        rate_type, (StatsAddrEntry *)nfs_entry_ptr,
                                        NFS_PAIR, AL_RPC_USER_ERR, current_time.tv_sec, NULL);

                nfs_entry_ptr->seconds_start_time = current_time.tv_sec;
                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }
        statsNextNfsPairEntry = entry_ptr;
```

```
    if (statsNextNfsPairEntry != NULL)
        statsRatesNotDone |= STATS_NFS_PAIR_RATE;
}
```

```
/*
 *  stats_nfs_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_nfs_p.c
 *
 *  Date:       7/24/91
 *
 *  Revision:   1.13
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO        Description of change. (latest first)
 *   --------  ---        ---------------------------------------------
 *
 *   06-28-91  KR         Fixed comparision in stats_nfs_lookup_file_system
 *   06-14-91  KR         Added stats_nfs_lookup_file_system_link
 *                             and stats_nfs_deallocate_file_system
 *   06-11-91  KR         Added setting of file status to NFS_FILE_NEW
 *   06-06-91  KR         Fixed stats_nfs_get_file_system to put link on client file system
 *                             queue if one wasn't there but lookup of file system was
successful.
 */

static char stats_nfs_p_c [] = "@(#)stats_nfs_p.c 1.13";

#include <stdio.h>

#include <cci_std.h>
```

```
#include "system.h"
#include "address.h"

#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "rtp_rpc.h"
#include "alarms.h"
#include "protocols.h"
#include "mib_defs.h"
#include "mib_nfs.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_rpc.h"
#include "stats_nfs.h"


/*
 * Global Data Structures
 */

StatsAddrEntry          *nfs_this_seg_addr_ptr;
StatsNfsSegment              *nfs_this_seg_stats_ptr;

StatsAddrEntry          *nfs_src_seg_addr_ptr, *nfs_dst_seg_addr_ptr;
StatsNfsSegment              *nfs_src_seg_stats_ptr, *nfs_dst_seg_stats_ptr;
```

```
StatsAddrEntry          *nfs_src_node_addr_ptr, *nfs_dst_node_addr_ptr;
StatsNfsAddr            *nfs_src_node_stats_ptr, *nfs_dst_node_stats_ptr;

StatsAddrEntry          *nfs_dialog_addr_ptr;
StatsNfsDialogEntry     *nfs_dialog_stats_ptr;


/*
 * Local data structures
 */

 StatsAddrEntry  *nfs_client_hash_table[NFS_CLIENT_HASH_TABLE_SIZE];
 Uint32          nfs_client_hash;
 StatsAddrEntry  *nfs_client_hash_link;
 StatsAddrEntry  *nfs_previous_client_hash_link;

 StatsAddrEntry  *nfs_server_hash_table[NFS_SERVER_HASH_TABLE_SIZE];
 Uint32          nfs_server_hash;
 StatsAddrEntry  *nfs_server_hash_link;
 StatsAddrEntry  *nfs_previous_server_hash_link;

 StatsAddrEntry  *nfs_dialog_hash_table[NFS_DIALOG_HASH_TABLE_SIZE];
 Uint32          nfs_dialog_hash;
 StatsAddrEntry  *nfs_dialog_hash_link;
 StatsAddrEntry  *nfs_previous_dialog_hash_link;

 StatsNfsFileSystem *nfs_file_hash_table[NFS_FILE_HASH_TABLE_SIZE];
 Uint32             nfs_file_hash;
 StatsNfsFileSystem *nfs_file_hash_link;
 StatsNfsFileSystem *nfs_previous_file_hash_link;



/*****************************************************************************
 *
 * Look for the segment address structure.
```

```
 * If no match is found, NULL is returned.
 */

 StatsAddrEntry *stats_nfs_lookup_segment (segment)

     Uint32                    segment;
{
     register StatsAddrEntry       *seg_addr_ptr;

seg_addr_ptr = (StatsAddrEntry *) statsNfsSegQ.pFlink;

while (seg_addr_ptr != NULL)
     {
     if (seg_addr_ptr->address.segment1 == segment)
          break;
     seg_addr_ptr = (StatsAddrEntry *) seg_addr_ptr->link.pFlink;
     }

     return (seg_addr_ptr);
}


/*****************************************************************************
 *
 * Find the structure for keeping Nfs segment statistics for the given
 * segment.  If one is not found, attempt to allocate one.
 */

     StatsAddrEntry *stats_nfs_get_segment (segment)

     Uint32         segment;
{
     register StatsAddrEntry       *seg_addr_ptr;
     register StatsNfsSegment *seg_stats_ptr;

 seg_addr_ptr = stats_nfs_lookup_segment (segment);
```

```
/*
 * If not found, try to allocate a structure for this segment.
 * If a structure can't be obtained, count this as a drop.
 */
 if (seg_addr_ptr == NULL)
      {
      seg_addr_ptr = (StatsAddrEntry *) stats_allocate ( sizeof(StatsAddrEntry) );
      if (seg_addr_ptr != NULL)
           {
           seg_addr_ptr->address.addressType = MibSegment1;
           seg_addr_ptr->address.segment1 = segment;
           seg_addr_ptr->parse_control = PARSE_CONTROL_DEFAULT;
           seg_addr_ptr->startTime = stats_start_time.tv_sec;
           seg_addr_ptr->lastTime = stats_start_time.tv_sec;

           FBInsqt (&statsNfsSegQ, (PFBQentry_type) seg_addr_ptr);
           }
      else
           {
           stats_mon_nfs_dropped;
           return (NULL);
           }
      }

/*
 * There's an address structure. See if there's a statistics structure
 * for Nfs attached, if not and parse control allows, allocate one.
 */
seg_stats_ptr = (StatsNfsSegment *) seg_addr_ptr->stats_ptr;
if (seg_stats_ptr == NULL)
      {
      /*
       * If Nfs parsing is enabled, allocate Nfs statistics structure if
       * one has not already been allocated.  If a structure can't be obtained,
       * count this as a drop.
       */
```

```
    if (seg_addr_ptr->parse_control & NibParseNfs)
        {
        seg_stats_ptr = (StatsNfsSegment *) stats_allocate (sizeof (StatsNfsSegment) );

        if (seg_stats_ptr != NULL)
            {
            seg_addr_ptr->stats_ptr = (Uint32 *) seg_stats_ptr;
            seg_stats_ptr->opRate.type                                  = STATS_RATE_10S;
            seg_stats_ptr->readOpRate.type                              = STATS_RATE_10S;
            seg_stats_ptr->writeOpRate.type                             = STATS_RATE_10S;
            seg_stats_ptr->writeCacheOpRate.type                        = STATS_RATE_10S;
            seg_stats_ptr->byteRate.type                                = STATS_RATE_10S;
            seg_stats_ptr->readByteRate.type                            = STATS_RATE_10S;
            seg_stats_ptr->writeByteRate.type                           = STATS_RATE_10S;

            seg_stats_ptr->rcvOffSegRate.type                           = STATS_RATE_10S;
            seg_stats_ptr->xmtOffSegRate.type                           = STATS_RATE_10S;
            seg_stats_ptr->transitRate.type                             = STATS_RATE_10S;
            seg_stats_ptr->flowCtrlRate.type                            = STATS_RATE_10S;
            seg_stats_ptr->rexmtRate.type                               = STATS_RATE_10S;
            seg_stats_ptr->frgmtRate.type                               = STATS_RATE_10S;

            seg_stats_ptr->errorRate.type                               = STATS_RATE_10S;

            seg_stats_ptr->rpcStats.versionMismatchRate.type            = STATS_RATE_10S;
            seg_stats_ptr->rpcStats.authBadCredRate.type                = STATS_RATE_10S;
            seg_stats_ptr->rpcStats.authRejectedCredRate.type           = STATS_RATE_10S;
            seg_stats_ptr->rpcStats.authBadVerfRate.type                = STATS_RATE_10S;
            seg_stats_ptr->rpcStats.authRejectedVerfRate.type           = STATS_RATE_10S;
            seg_stats_ptr->rpcStats.authTooWeakRate.type                = STATS_RATE_10S;
            seg_stats_ptr->rpcStats.authOtherRate.type
                                                                        =
STATS_RATE_10S;
            seg_stats_ptr->rpcStats.progUnavailRate.type                = STATS_RATE_10S;
            seg_stats_ptr->rpcStats.progVersionMismatchRate.type        = STATS_RATE_10S;
            seg_stats_ptr->rpcStats.procUnavailRate.type                = STATS_RATE_10S;
            seg_stats_ptr->rpcStats.procGarbageArgsRate.type            = STATS_RATE_10S;
```

```
                seg_stats_ptr->mountFailureRate.type              = STATS_RATE_10S;

                seg_stats_ptr->nfsErrPernRate.type                = STATS_RATE_10S;
                seg_stats_ptr->nfsErrNoEntRate.type                   = STATS_RATE_10S;
                seg_stats_ptr->nfsErrIORate.type                  = STATS_RATE_10S;
                seg_stats_ptr->nfsErrNXIORate.type                = STATS_RATE_10S;
                seg_stats_ptr->nfsErrAccessRate.type              = STATS_RATE_10S;
                seg_stats_ptr->nfsErrExistRate.type                   = STATS_RATE_10S;
                seg_stats_ptr->nfsErrNoDevRate.type                   = STATS_RATE_10S;
                seg_stats_ptr->nfsErrNotDirRate.type              = STATS_RATE_10S;
                seg_stats_ptr->nfsErrFBigRate.type                = STATS_RATE_10S;
                seg_stats_ptr->nfsErrNoSpaceRate.type             = STATS_RATE_10S;
                seg_stats_ptr->nfsErrROFSRate.type                = STATS_RATE_10S;
                seg_stats_ptr->nfsErrNameTooLongRate.type         = STATS_RATE_10S;
                seg_stats_ptr->nfsErrNotEmptyRate.type            = STATS_RATE_10S;
                seg_stats_ptr->nfsErrDQuotaRate.type              = STATS_RATE_10S;
                seg_stats_ptr->nfsErrStaleRate.type                   = STATS_RATE_10S;
                seg_stats_ptr->nfsErrWFlushRate.type              = STATS_RATE_10S;
                }
            else
                stats_mon_nfs_dropped;
            }
        }

    return (seg_addr_ptr);
    }


    /*************************************************************************
     *
     *  Find the nfs client record.
     *  A hash is done on the ip address.  A pointer to the first
     *  StatsAddrEntry with the same hash is found.  Structures with the
     *  same hash are linked.  The link must be walked and the ip address
     *  compared with the those in each of the structures until a match
```

```
 *  is fouud.  If no match is found, NULL is returned.
 */
StatsAddrEntry *stats_nfs_lookup_client (ip_addr)

     Uint32          *ip_addr;
{
     Uint32          i;

/*
 * Compute ip address hash to get index into hash table
 */
nfs_client_hash = (*ip_addr & 0x0000ffff);
nfs_client_hash = ((nfs_client_hash + ((nfs_client_hash & 0xff00) >> 8)) &
                        (NFS_CLIENT_HASH_TABLE_SIZE - 1));


nfs_previous_client_hash_link = NULL;
nfs_client_hash_link = (StatsAddrEntry *) nfs_client_hash_table[nfs_client_hash];

/*
 * Walk linked list for exact entry
 */
while (nfs_client_hash_link != NULL)
     {
     if (nfs_client_hash_link->address.netAddress1.u.ipAddress == *ip_addr)
          return (nfs_client_hash_link);
     else
          {
          nfs_previous_client_hash_link = nfs_client_hash_link;
          nfs_client_hash_link = nfs_client_hash_link->hash_link;
          }
     }

/*
 * No entry found.
 */
return (NULL);
```

```
}


/*****************************************************************************
 *
 * Allocate a stats structure if parse control is turned on.
 *
 */

 Uint32 stats_nfs_get_stats (nfs_addr_record_ptr)

     StatsAddrEntry      *nfs_addr_record_ptr;
 {
     register StatsNfsAddr          *nfs_addr_stats_ptr;

     if ((nfs_addr_record_ptr != NULL) && (nfs_addr_record_ptr->stats_ptr == NULL))
          {
          /*
          * If parse control is turned on for Nfs, allocate a structure for Nfs
          * address statistics and initialize it.  If parse control is turned on,
          * but a structure can't be obtained, return FALSE
          */
          if (nfs_addr_record_ptr->parse_control & MibParseNfs)
               {
               nfs_addr_record_ptr->stats_ptr = (Uint32 *) stats_allocate (sizeof
(StatsNfsAddr) );
               nfs_addr_stats_ptr = (StatsNfsAddr *) nfs_addr_record_ptr->stats_ptr;
               if (nfs_addr_stats_ptr != NULL)
                    {
                    nfs_addr_stats_ptr->opRate.type                              =
STATS_RATE_10S;
                    nfs_addr_stats_ptr->readOpRate.type                          =
STATS_RATE_10S;
```

```
                nfs_addr_stats_ptr->writeOpRate.type                    =
STATS_RATE_10S;
                nfs_addr_stats_ptr->writeCacheOpRate.type               =
STATS_RATE_10S;
                nfs_addr_stats_ptr->byteRate.type                       =
STATS_RATE_10S;
                nfs_addr_stats_ptr->readByteRate.type                   =
STATS_RATE_10S;
                nfs_addr_stats_ptr->writeByteRate.type                  =
STATS_RATE_10S;

                nfs_addr_stats_ptr->rcvOffSegRate.type                  =
STATS_RATE_10S;
                nfs_addr_stats_ptr->xmtOffSegRate.type                  =
STATS_RATE_10S;
                nfs_addr_stats_ptr->flowCtrlRate.type                   =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rexmtRate.type                      =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rcvFrgmtRate.type                   =
STATS_RATE_10S;
                nfs_addr_stats_ptr->xmtFrgmtRate.type                   =
STATS_RATE_10S;

                nfs_addr_stats_ptr->errorRate.type                      =
STATS_RATE_10S;

                nfs_addr_stats_ptr->rpcStats.versionMismatchRate.type   =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rpcStats.authBadCredRate.type       =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rpcStats.authRejectedCredRate.type  =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rpcStats.authBadVerfRate.type       =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rpcStats.authRejectedVerfRate.type  =
STATS_RATE_10S;
```

```
                nfs_addr_stats_ptr->rpcStats.authTooWeakRate.type          =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rpcStats.authOtherRate.type                =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rpcStats.progUnavailRate.type          =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rpcStats.progVersionMismatchRate.type  =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rpcStats.procUnavailRate.type          =
STATS_RATE_10S;
                nfs_addr_stats_ptr->rpcStats.procGarbageArgsRate.type      =
STATS_RATE_10S;

                nfs_addr_stats_ptr->mountFailureRate.type              =
STATS_RATE_10S;

                nfs_addr_stats_ptr->nfsErrPermRate.type                =
STATS_RATE_10S;
                nfs_addr_stats_ptr->nfsErrNoEntRate.type               =
STATS_RATE_10S;
                nfs_addr_stats_ptr->nfsErrIORate.type                  =
STATS_RATE_10S;
                nfs_addr_stats_ptr->nfsErrNXIORate.type                =
STATS_RATE_10S;
                nfs_addr_stats_ptr->nfsErrAccessRate.type              =
STATS_RATE_10S;
                nfs_addr_stats_ptr->nfsErrExistRate.type               =
STATS_RATE_10S;
                nfs_addr_stats_ptr->nfsErrNoDevRate.type               =
STATS_RATE_10S;
                nfs_addr_stats_ptr->nfsErrNotDirRate.type              =
STATS_RATE_10S;
                nfs_addr_stats_ptr->nfsErrFBigRate.type                =
STATS_RATE_10S;
                nfs_addr_stats_ptr->nfsErrNoSpaceRate.type             =
STATS_RATE_10S;
```

```
                    nfs_addr_stats_ptr->nfsErrROFSRate.type                =
STATS_RATE_10S;
                    nfs_addr_stats_ptr->nfsErrNameTooLongRate.type         =
STATS_RATE_10S;
                    nfs_addr_stats_ptr->nfsErrNotEmptyRate.type            =
STATS_RATE_10S;
                    nfs_addr_stats_ptr->nfsErrDQuotaRate.type              =
STATS_RATE_10S;
                    nfs_addr_stats_ptr->nfsErrStaleRate.type               =
STATS_RATE_10S;
                    nfs_addr_stats_ptr->nfsErrWFlushRate.type              =
STATS_RATE_10S;

                    FBInitqh (&nfs_addr_stats_ptr->dialogQ);
                    }
                else
                    return (FALSE);
            }
        }
        return (TRUE);
    }


/*****************************************************************************
 *
 * Find the structure for keeping nfs client statistics for the given
 * ip address.  If one not found, attempt to allocate one.
 */
 StatsAddrEntry *stats_nfs_get_client (ip_addr)

     Uint32              *ip_addr;
{
     register StatsAddrEntry       *nfs_addr_record_ptr;
```

```
    register StatsNfsAddr          *nfs_addr_stats_ptr;
    register StatsAddrEntry        *ip_addr_record_ptr;


nfs_addr_record_ptr = stats_nfs_lookup_client (ip_addr);


if (nfs_addr_record_ptr == NULL)
    {
    /*
     * Try to allocate a statistics structure for this address.
     * If nfs is turned on for this ip address, but a structure can't
     * be obtained, count this as a drop.  If nfs is not turned on, just
     * return NULL.
     */
    ip_addr_record_ptr = stats_ip_lookup_addr (ip_addr);
    if (ip_addr_record_ptr == NULL)
        return (NULL);

    if ((ip_addr_record_ptr->parse_control & MibParseNfs) != MibParseNfs)
        return (NULL);

    nfs_addr_record_ptr = (StatsAddrEntry *)stats_allocate (sizeof (StatsAddrEntry) );
    if (nfs_addr_record_ptr == NULL)
        {
        stats_mon_nfs_dropped;
        return (NULL);
        }
    else
        {
        nfs_addr_record_ptr->hash_link                          = NULL;
        nfs_addr_record_ptr->startTime                          =
stats_start_time.tv_sec;
        nfs_addr_record_ptr->lastTime                           =
stats_start_time.tv_sec;
        nfs_addr_record_ptr->address.addressType                = MibNotAddress1 |
MibSegment1;
```

```
        nfs_addr_record_ptr->address.netAddress1.netAddressType = NetTopIp;
        nfs_addr_record_ptr->address.netAddress1.length         = 4;
        nfs_addr_record_ptr->address.netAddress1.u.ipAddress    = *ip_addr;

        nfs_addr_record_ptr->parse_control = ip_addr_record_ptr->parse_control;
        nfs_addr_record_ptr->address.segment1 = ip_addr_record_ptr->address.segment1;

        PBInsqt (&statsNfsClientQ, (PFBQentry_type) nfs_addr_record_ptr);

        /*
         * Allocate a stats structure if parse control is turned on.
         * stats_nfs_get_stats will return FALSE, so count this as a drop.
         */
        if (stats_nfs_get_stats (nfs_addr_record_ptr) == FALSE)
            stats_mon_nfs_dropped;
        }

    /*
     * Put this nfs address record into the hash table.
     * Variable hash was set when stats_nfs_lookup_client was executed.
     */
    if ( nfs_client_hash_table[nfs_client_hash] == NULL)
        nfs_client_hash_table[nfs_client_hash] = nfs_addr_record_ptr;
    else
        {
        /*
         * Find the last structure of the hash link
         */
        nfs_client_hash_link = nfs_client_hash_table[nfs_client_hash];

        while (nfs_client_hash_link->hash_link != NULL)
            nfs_client_hash_link = nfs_client_hash_link->hash_link;

        nfs_client_hash_link->hash_link = nfs_addr_record_ptr;
        }
    }
```

```
return (nfs_addr_record_ptr);
}



/*****************************************************************************
 *
 *  Find the nfs server.
 *  A hash is done on the ip address.  A pointer to the first
 *  StatsAddrEntry with the same hash is found.  Structures with the
 *  same hash are linked.  The link must be walked and the ip address
 *  compared with the those in each of the structures until a match
 *  is found.  If no match is found, NULL is returned.
 */
StatsAddrEntry *stats_nfs_lookup_server (ip_addr)

    Uint32          *ip_addr;
{
    Uint32          i;

/*
 * Compute ip address hash to get index into hash table
 */
nfs_server_hash = (*ip_addr & 0x0000ffff);
nfs_server_hash = ((nfs_server_hash + ((nfs_server_hash & 0xff00) >> 8)) &
                   (NFS_SERVER_HASH_TABLE_SIZE - 1));

nfs_previous_server_hash_link = NULL;
nfs_server_hash_link = (StatsAddrEntry *) nfs_server_hash_table[nfs_server_hash];

/*
 * Walk linked list for exact entry
 */
while (nfs_server_hash_link != NULL)
    {
    if (nfs_server_hash_link->address.netAddress1.u.ipAddress == *ip_addr)
```

```
            return (nfs_server_hash_link);
        else
            {
            nfs_previous_server_hash_link = nfs_server_hash_link;
            nfs_server_hash_link = nfs_server_hash_link->hash_link;
            }
        }


/*
 * No entry found.
 */
return (NULL);
}


/************************************************************************
 *
 * Find the structure for keeping nfs statistics for the given
 * ip address.  If one not found, attempt to allocate one.
 */

StatsAddrEntry *stats_nfs_get_server (ip_addr)

    Uint32              *ip_addr;
{
    register StatsAddrEntry     *nfs_addr_ptr;
    register StatsNfsAddr       *nfs_addr_stats_ptr;
    register StatsAddrEntry     *ip_addr_record_ptr;

nfs_addr_ptr = stats_nfs_lookup_server (ip_addr);

if (nfs_addr_ptr == NULL)
    {
    /*
     * Try to allocate a statistics structure for this address.
```

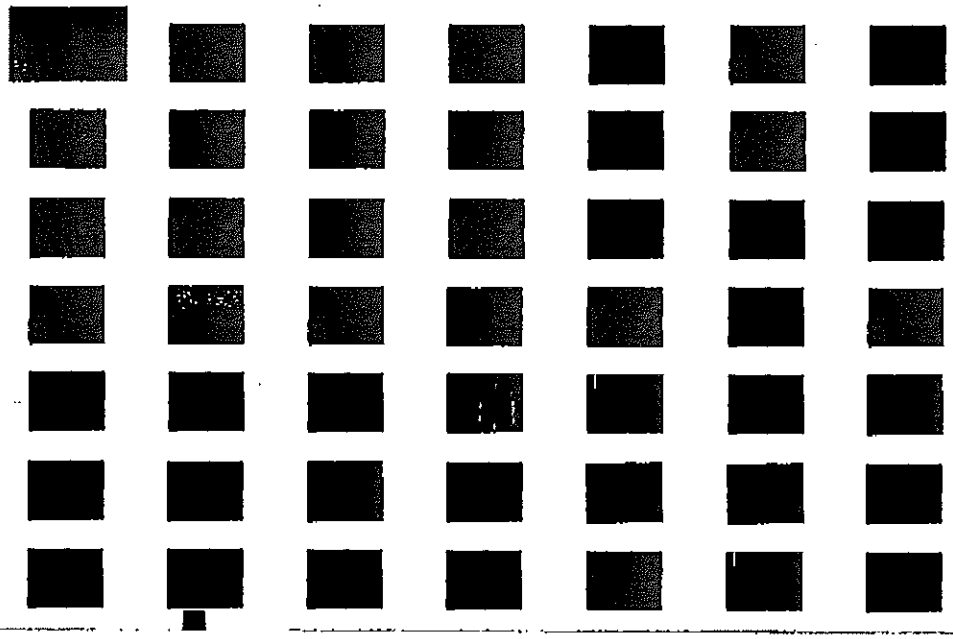
```
 * If nfs is turned on for this ip address, but a structure can't
 * be obtained, count this as a drop.  If nfs is not turned on, just
 * return NULL.
 */
ip_addr_record_ptr = stats_ip_lookup_addr (ip_addr);
if (ip_addr_record_ptr == NULL)
     return (NULL);
if ((ip_addr_record_ptr->parse_control & MibParseNfs) == NULL)
     return (NULL);

nfs_addr_ptr = (StatsAddrEntry *)stats_allocate (sizeof (StatsAddrEntry) );
if (nfs_addr_ptr == NULL)
     {
     stats_mon_nfs_dropped;
     return (NULL);
     }
else
     {
     nfs_addr_ptr->hash_link = NULL;
     nfs_addr_ptr->startTime = stats_start_time.tv_sec;
     nfs_addr_ptr->lastTime = stats_start_time.tv_sec;

     nfs_addr_ptr->address.addressType = MibNetAddress1 | MibSegment1;
     nfs_addr_ptr->address.netAddress1.netAddressType = NetTcpIp;
     nfs_addr_ptr->address.netAddress1.length = 4;
     nfs_addr_ptr->address.netAddress1.u.ipAddress = *ip_addr;

     nfs_addr_ptr->parse_control = ip_addr_record_ptr->parse_control;
     nfs_addr_ptr->address.segment1 = ip_addr_record_ptr->address.segment1;

     FBInsqt (&statsNfsServerQ, (PFBQentry_type) nfs_addr_ptr);

   /*
     * Allocate a stats structure if parse control is turned on.
     * stats_nfs_get_stats will return FALSE, so count this as a drop.
     */
     if (stats_nfs_get_stats (nfs_addr_ptr) == FALSE)
```

```
            stats_mon_nfs_dropped;
        }

    /*
     * Put this nfs server into the hash table.
     * Variable nfs_server_hash was set when stats_nfs_lookup_server was executed.
     */
    if ( nfs_server_hash_table[nfs_server_hash] == NULL)
         nfs_server_hash_table[nfs_server_hash] = nfs_addr_ptr;
    else
         {
         /*
          * Find the last structure of the hash link
          */
         nfs_server_hash_link = nfs_server_hash_table[nfs_server_hash];

         while (nfs_server_hash_link->hash_link != NULL)
              nfs_server_hash_link = nfs_server_hash_link->hash_link;

         nfs_server_hash_link->hash_link = nfs_addr_ptr;
         }
    }

return (nfs_addr_ptr);
}


/************************ stats_nfs_lookup_dialog ******************************
 *
 * Find a nfs dialog given 2 ip addresses.
 * If the dialog is not found, NULL is returned.
 */

StatsAddrEntry *stats_nfs_lookup_dialog (server, client)
```

```
    Uint32                  *server, *client;
{
    register Uint32             xnfs_dialog_hash;
    register StatsAddrEntry     *xnfs_dialog_hash_link;
    register StatsAddrEntry     *xnfs_previous_dialog_hash_link;

    register StatsAddrEntry     *dialog_addr_ptr;

/*
 * Compute hash function on both ip addresses
 */
xnfs_dialog_hash = (*server & 0xffff0000) + ((*server & 0x0000ffff) << 16) +
                   (*client & 0xffff0000) + ((*client & 0x0000ffff) << 16);
xnfs_dialog_hash = ((xnfs_dialog_hash + ((xnfs_dialog_hash & 0xff00) >> 8))
                    & (NFS_DIALOG_HASH_TABLE_SIZE - 1));

xnfs_previous_dialog_hash_link = NULL;
xnfs_dialog_hash_link = (StatsAddrEntry *) nfs_dialog_hash_table[xnfs_dialog_hash];

/*
 * Walk linked list for exact entry
 */
while (xnfs_dialog_hash_link != NULL)
    {
    if ( (xnfs_dialog_hash_link->address.netAddress1.u.ipAddress == *server) &&
         (xnfs_dialog_hash_link->address.netAddress2.u.ipAddress == *client) )
            {
            break;
            }
        else
            {
            xnfs_previous_dialog_hash_link = xnfs_dialog_hash_link;
            xnfs_dialog_hash_link = xnfs_dialog_hash_link->hash_link;
            }
    }

nfs_dialog_hash = xnfs_dialog_hash;
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones, Kary Robertson, David M. Thompson and Gerard White

11



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
nfs_dialog_hash_link = xnfs_dialog_hash_link;
nfs_previous_dialog_hash_link = xnfs_previous_dialog_hash_link;

return (xnfs_dialog_hash_link);
}


/************************* stats_nfs_get_dialog ******************************
 *
 * Find or allocate an nfs dialog given 2 ip addresses.
 * If the dialog is not found, attempt to allocate a structure.
 */
StatsAddrEntry *stats_nfs_get_dialog (server, client)

     Uint32                    *server, *client;
{
     register StatsAddrEntry            *dialog_addr_ptr;
     register StatsNfsDialogEntry  *dialog_stats_ptr;
     register StatsDialogLink      *dialog_link;
     register StatsNfsState             *dialog_state_ptr;
     register StatsAddrEntry            *nfs_server_record_ptr, *nfs_client_record_ptr;
     register StatsNfsAddr              *nfs_server_state_ptr, *nfs_client_stats_ptr;
     register StatsAddrEntry            *ip_server_record_ptr, *ip_client_record_ptr;
     register Uint32                         parse_control;


dialog_addr_ptr = stats_nfs_lookup_dialog (server, client);
if (dialog_addr_ptr != NULL)
     return (dialog_addr_ptr);

/*
 * Check parse control for the ip addresses.
 */
ip_server_record_ptr = stats_ip_lookup_addr (server);
ip_client_record_ptr = stats_ip_lookup_addr (client);
```

```
if ( (ip_server_record_ptr == NULL) || (ip_client_record_ptr == NULL) )
     return (NULL);

if (ip_server_record_ptr != NULL)
     parse_control = ip_server_record_ptr->parse_control;
if (ip_client_record_ptr != NULL)
     parse_control |= ip_client_record_ptr->parse_control;


/*
 * If nfs is turned on for these ip addresses, allocate structures
 * for the dialog.  If can't get them, count this as a drop.
 */
if ((parse_control & MibParseNfs) != MibParseNfs)
     return (NULL);

/*
 * Try to allocate structures for this dialog.
 * If they can't be obtained, count this as a drop.
 */
dialog_addr_ptr = (StatsAddrEntry *) stats_allocate (sizeof (StatsAddrEntry) );
if (dialog_addr_ptr == NULL)
     {
     stats_mon_nfs_dropped;
     return (NULL);
     }

dialog_stats_ptr = (StatsNfsDialogEntry *) stats_allocate (sizeof (StatsNfsDialogEntry) );

if (dialog_stats_ptr == NULL)
     {
     stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );
     stats_mon_nfs_dropped;
     return (NULL);
     }

dialog_state_ptr = (StatsNfsState *) stats_allocate (sizeof (StatsNfsState) );
```

```
if (dialog_state_ptr == NULL)
     {
     stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );
     stats_deallocate (dialog_stats_ptr, sizeof(StatsNfsDialogEntry) );
     stats_mon_nfs_dropped;
     return (NULL);
     }


/*
 * Initialize the structures
 */
dialog_addr_ptr->hash_link                                    = NULL;
dialog_addr_ptr->address.addressType                        = MibNetAddress1 | MibNetAddress2;

dialog_addr_ptr->address.netAddress1.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress1.length         = 4;
dialog_addr_ptr->address.netAddress1.u.ipAddress    = *server;
dialog_addr_ptr->address.netAddress2.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress2.length         = 4;
dialog_addr_ptr->address.netAddress2.u.ipAddress    = *client;
dialog_addr_ptr->parse_control                                = parse_control;
dialog_addr_ptr->startTime                                    = stats_start_time.tv_sec;
dialog_addr_ptr->lastTime                                     = stats_start_time.tv_sec;
dialog_addr_ptr->stats_ptr                                    = (Uint32 *) dialog_stats_ptr;


FBInsqt (&statsNfsDialogQ, (PFBQentry_type) dialog_addr_ptr);

dialog_stats_ptr->opRate.type                       = STATS_RATE_10S;
dialog_stats_ptr->readOpRate.type                 = STATS_RATE_10S;
dialog_stats_ptr->writeOpRate.type                  = STATS_RATE_10S;
dialog_stats_ptr->writeCacheOpRate.type           = STATS_RATE_10S;
dialog_stats_ptr->readByteRate.type               = STATS_RATE_10S;
dialog_stats_ptr->writeByteRate.type              = STATS_RATE_10S;

dialog_stats_ptr->reXmtRate.type                  = STATS_RATE_10S;
```

```
dialog_stats_ptr->rpcProgramFailRate.type     = STATS_RATE_10S;
dialog_stats_ptr->rpcMismatchRate.type        = STATS_RATE_10S;
dialog_stats_ptr->rpcAuthFailRate.type        = STATS_RATE_10S;
dialog_stats_ptr->nfsAccessErrRate.type       = STATS_RATE_10S;
dialog_stats_ptr->nfsHardErrRate.type         = STATS_RATE_10S;
dialog_stats_ptr->nfsResourceErrRate.type     = STATS_RATE_10S;
dialog_stats_ptr->nfsUserErrRate.type         = STATS_RATE_10S;

dialog_stats_ptr->state_ptr                   = (StatsNfsState *) dialog_state_ptr;

bzero (dialog_state_ptr, sizeof(StatsNfsState) );

/*
 * Link the dialog statistics into the dialog queue for each nfs address stats.
 */
nfs_server_record_ptr = stats_nfs_lookup_server (server);
nfs_client_record_ptr = stats_nfs_lookup_client (client);

if (nfs_server_record_ptr != NULL)
     {
     if (nfs_server_record_ptr->stats_ptr != NULL)
          {
          dialog_addr_ptr->address.addressType= NibSegment1;
          dialog_addr_ptr->address.segment1  = nfs_server_record_ptr->address.segment1;
          nfs_server_stats_ptr                    =  (StatsNfsAddr *)
nfs_server_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&nfs_server_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
               }
          }
     }
if (nfs_client_record_ptr != NULL)
     {
     if (nfs_client_record_ptr->stats_ptr != NULL)
```

```
          {
          dialog_addr_ptr->address.addressType|= NibSegment2;
          dialog_addr_ptr->address.segment2  = nfs_client_record_ptr->address.segment1;
          nfs_client_stats_ptr                       = (StatsNfsAddr *)
nfs_client_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&nfs_client_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
               }
          }
     }


/*
 * Put the new dialog address structure in the nfs dialog hash table.
 * Variable hash was set up when stats_nfs_lookup_dialog was executed.
 */
if (nfs_dialog_hash_table[nfs_dialog_hash] == NULL)
     nfs_dialog_hash_table[nfs_dialog_hash] = dialog_addr_ptr;
else
     {
     /*
      * Find the last structure of the hash link.
      */
     nfs_dialog_hash_link = nfs_dialog_hash_table[nfs_dialog_hash];
     while (nfs_dialog_hash_link->hash_link != NULL)
          nfs_dialog_hash_link = nfs_dialog_hash_link->hash_link;

     nfs_dialog_hash_link->hash_link = dialog_addr_ptr;
     }

return (dialog_addr_ptr);
}
```

```
/*****************************************************************************
 *
 * Lookup a file system link given the file system
 *
 */
StatsNfsFileSystemLink *stats_nfs_lookup_file_system_link (nfs_stats_ptr,
nfs_file_system_ptr)

     StatsNfsAddr                        *nfs_stats_ptr;
     StatsNfsFileSystem                  *nfs_file_system_ptr;
{
     register StatsNfsFileSystemLink     *file_system_link_ptr;
     register Uint32                      I;

if (nfs_stats_ptr == NULL)
     return (NULL);

file_system_link_ptr = (StatsNfsFileSystemLink *) nfs_stats_ptr->fileSystemQ.pFlink;
while (file_system_link_ptr != NULL)
     {
     if (file_system_link_ptr->file_system_ptr == nfs_file_system_ptr)
          break;
     else
          file_system_link_ptr = (StatsNfsFileSystemLink *)
file_system_link_ptr->link.pFlink;
     }
return (file_system_link_ptr);
}


/*****************************************************************************
 *
 * Lookup an Nfs file system record
 *  A hash is done on the ip address.  A pointer to the first
 *  StatsAddrEntry with the same hash is found.  Structures with the
```

```
 *  same hash are linked.  The link must be walked and the ip address
 *  compared with the those in each of the structures until a match
 *  is found.  If no match is found, NULL is returned.
 */
StatsNfsFileSystem *stats_nfs_lookup_file_system (nfs_server_addr_ptr,
nfs_client_addr_ptr,
                                                  file_system_length, file_system_ptr)

    StatsAddrEntry          *nfs_server_addr_ptr, *nfs_client_addr_ptr;
    Uint32                       file_system_length;
    NfsFileSystem           *file_system_ptr;
{
    register Uint32              i, ip_addr, length;
    register Bool           match;

/*
 * Compute ip address hash to get index into hash table
 */
if (nfs_server_addr_ptr == NULL)
    return (NULL);

ip_addr = nfs_server_addr_ptr->address.netAddress1.u.ipAddress;

nfs_file_hash = (ip_addr & 0x0000ffff);
nfs_file_hash = ((nfs_file_hash + ((nfs_file_hash & 0xff00) >> 8)) &
                 (NFS_FILE_HASH_TABLE_SIZE - 1));

nfs_previous_file_hash_link = NULL;
nfs_file_hash_link = (StatsNfsFileSystem *) nfs_file_hash_table[nfs_file_hash];

/*
 * Walk linked list for exact entry
 */
while (nfs_file_hash_link != NULL)
    {
    if (nfs_file_hash_link->serverIpAddr == ip_addr)
        {
```

```
            if (file_system_length > STATS_NFS_FILE_SYSTEM_LENGTH)
                 length  = STATS_NFS_FILE_SYSTEM_LENGTH;
            else
                 length = file_system_length;

            if (nfs_file_hash_link->fileSystemLength == file_system_length)
                 {
                 match = TRUE;
                 i = 0;
                 while (i < length)
                      {
                      if (nfs_file_hash_link->fileSystem[i] !=
file_system_ptr->fileSystem[i])
                           {
                           match = FALSE;
                           break;
                           }
                      i++;
                      }
                 }
            if (match)
                 return (nfs_file_hash_link);
            }

       nfs_previous_file_hash_link = nfs_file_hash_link;
       nfs_file_hash_link = nfs_file_hash_link->hash_link;
       }

  /*
   * No entry found.
   */
  return (NULL);
  }


  /*****************************************************************************
   *
```

```
 * Deallocate an Nfs file system record
 */
 void stats_nfs_deallocate_file_system (server_addr_ptr, client_addr_ptr,
nfs_file_system_ptr)

     StatsAddrEntry                    *server_addr_ptr, *client_addr_ptr;
     StatsNfsFileSystem                *nfs_file_system_ptr;
{
     register StatsNfsFileSystem       *file_system_ptr;
     register Uint32                 xnfs_file_hash;
     register StatsNfsFileSystem       *xnfs_file_hash_link;
     register StatsNfsFileSystem       *xnfs_previous_file_hash_link;

file_system_ptr = stats_nfs_lookup_file_system (server_addr_ptr, client_addr_ptr,
     nfs_file_system_ptr->fileSystemLength, (NfsFileSystem
*)nfs_file_system_ptr->fileSystem);

if (file_system_ptr != nfs_file_system_ptr)
     {
     non_panic (RTP_task, STATS_PTRS_NOT_EQUAL, "stats_nfs_deallocate_file_system");
     return;
     }

/*
 * Take the file system out of the hash table.
 */
xnfs_file_hash = nfs_file_hash;
xnfs_file_hash_link = nfs_file_hash_link;
xnfs_previous_file_hash_link = nfs_previous_file_hash_link;

if ( (xnfs_previous_file_hash_link == NULL) &&
      (xnfs_file_hash_link->hash_link == NULL) )
     nfs_file_hash_table[xnfs_file_hash] = NULL;
else
     {
     if (xnfs_previous_file_hash_link != NULL)
          xnfs_previous_file_hash_link->hash_link = xnfs_file_hash_link->hash_link;
```

```
        else
             if (xnfs_file_hash_link->hash_link != NULL)
                  nfs_file_hash_table[xnfs_file_hash] =
                       xnfs_file_hash_link->hash_link;
        }

/*
 * Take the file system off the queue.
 */
FBRemqm (&statsNfsFileSystemQ, (PFBQentry_type) nfs_file_system_ptr);
stats_deallocate (nfs_file_system_ptr, sizeof(StatsNfsFileSystem));
}



/*****************************************************************************
 *
 * Get an Nfs file system record
 */
 StatsNfsFileSystem *stats_nfs_get_file_system (nfs_server_addr_ptr, nfs_client_addr_ptr,
                                                file_system_length,
file_system_ptr)

     StatsAddrEntry                  *nfs_server_addr_ptr, *nfs_client_addr_ptr;
     Uint32                           file_system_length;
     NfsFileSystem                   *file_system_ptr;

{
     register StatsNfsFileSystem     *stats_nfs_file_system_ptr;
     register StatsNfsAddr           *nfs_server_stats_ptr, *nfs_client_stats_ptr;
     register Uint32                  copy_length;
     register StatsNfsFileSystemLink *file_system_link;

if (nfs_server_addr_ptr == NULL)
     return (NULL);
```

```
stats_nfs_file_system_ptr =
     stats_nfs_lookup_file_system (nfs_server_addr_ptr, nfs_client_addr_ptr,
                                        file_system_length, file_system_ptr);

/*
 * If the lookup was successful, see if the client has a link to the file.
 * If not, link it up.
 */
if (stats_nfs_file_system_ptr != NULL)
     {
     if (nfs_client_addr_ptr != NULL)
          {
          if (nfs_client_addr_ptr->stats_ptr != NULL)
               {
               nfs_client_stats_ptr = (StatsNfsAddr *) nfs_client_addr_ptr->stats_ptr;
               file_system_link = (StatsNfsFileSystemLink*)
nfs_client_stats_ptr->fileSystemQ.pFlink;
               while (file_system_link != NULL)
                    {
                    if (file_system_link->file_system_ptr == stats_nfs_file_system_ptr)
                         return (stats_nfs_file_system_ptr);
                    else
                         file_system_link = (StatsNfsFileSystemLink *)
file_system_link->link.pFlink;
                    }

               file_system_link =
                    (StatsNfsFileSystemLink
*)stats_allocate(sizeof(StatsNfsFileSystemLink));
               if (file_system_link != NULL)
                    {
                    file_system_link->file_system_ptr = stats_nfs_file_system_ptr;
                    FDInsqt (&nfs_client_stats_ptr->fileSystemQ,
(PFBQentry_type)file_system_link);
                    }
               }
          }
```

```
        return (stats_nfs_file_system_ptr);
        }

    /*
     * Since there was a StatsAddrEntry for the server, we know that
     * parse control is turned on for the server.  Try to allocate a
     * structure for this file system.  If one can't be obtained, count
     * this as a drop.
     */
    stats_nfs_file_system_ptr = (StatsNfsFileSystem *)stats_allocate
    (sizeof(StatsNfsFileSystem));
    if (stats_nfs_file_system_ptr == NULL)
        {
        stats_mon_nfs_dropped;
        return (NULL);
        }

    stats_nfs_file_system_ptr->hash_link          = NULL;
    stats_nfs_file_system_ptr->status             = NFS_FILE_NEW;
    stats_nfs_file_system_ptr->fileSystemLength = file_system_length;
    stats_nfs_file_system_ptr->serverIpAddr       =
              nfs_server_addr_ptr->address.netAddress1.u.ipAddress;

    if (file_system_length < STATS_NFS_FILE_SYSTEM_LENGTH)
        copy_length = file_system_length;
    else
        copy_length = STATS_NFS_FILE_SYSTEM_LENGTH;
    bcopy (file_system_ptr, stats_nfs_file_system_ptr->fileSystem, copy_length);

    FBInsqt (&statsNfsFileSystemQ, (PFBQentry_type) stats_nfs_file_system_ptr);


    /*
     * Link the file system record into the file system queues for the client and the server.

     */
    if (nfs_server_addr_ptr->stats_ptr != NULL),
```

```
        {
        nfs_server_stats_ptr = (StatsNfsAddr *) nfs_server_addr_ptr->stats_ptr;
        file_system_link = (StatsNfsFileSystemLink
*)stats_allocate(sizeof(StatsNfsFileSystemLink));
        if (file_system_link != NULL)
             {
             file_system_link->file_system_ptr = stats_nfs_file_system_ptr;
             FBInsqt (&nfs_server_stats_ptr->fileSystemQ, (PFBQentry_type)file_system_link);

             }
        }

if (nfs_client_addr_ptr != NULL)
        {
        if (nfs_client_addr_ptr->stats_ptr != NULL)
             {
             nfs_client_stats_ptr = (StatsNfsAddr *) nfs_client_addr_ptr->stats_ptr;
             file_system_link =
                  (StatsNfsFileSystemLink *)stats_allocate(sizeof(StatsNfsFileSystemLink));
             if (file_system_link != NULL)
                  {
                  file_system_link->file_system_ptr = stats_nfs_file_system_ptr;
                  FBInsqt(&nfs_client_stats_ptr->fileSystemQ,
(PFBQentry_type)file_system_link);
                  }
             }
        }

/*
 * Put this file system into the hash table.
 * Variable nfs_file_hash was set when the stats_nfs_lookup_file_system routine
 * was executed.
 */

 if (nfs_file_hash_table[nfs_file_hash] == NULL)
     nfs_file_hash_table[nfs_file_hash] = stats_nfs_file_system_ptr;
else
```

```
        {
        /*
         * Find the last structure of the hash link.
         */
        nfs_file_hash_link = nfs_file_hash_table[nfs_file_hash];

        while (nfs_file_hash_link->hash_link != NULL)
             nfs_file_hash_link = nfs_file_hash_link->hash_link;

        nfs_file_hash_link->hash_link = stats_nfs    .e_system_ptr;
        }

    return (stats_nfs_file_system_ptr);
    }



    /*****************************************************************************
     *
     * Set Nfs node defaults
     */
    void setNfsAddrDflts (mib_nfs_addr_defs, nfs_addr_stats_ptr)
        MibNfsAddrDefaults  *mib_nfs_addr_defs;
        StatsNfsAddr        *nfs_addr_stats_ptr;
    {
        nfs_addr_stats_ptr->ops.high_thld =
                                    mib_nfs_addr_defs->nfsAddrOpHighThld;
        nfs_addr_stats_ptr->opRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrOpRateHighThld;
        nfs_addr_stats_ptr->readOps.high_thld =
                                    mib_nfs_addr_defs->nfsAddrReadOpHighThld;
        nfs_addr_stats_ptr->readOpRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrReadOpRateHighThld;
        nfs_addr_stats_ptr->writeOps.high_thld =
                                    mib_nfs_addr_defs->nfsAddrWriteOpHighThld;
        nfs_addr_stats_ptr->writeOpRate.high_thld =
```

```
                                    mib_nfs_addr_defs->nfsAddrWriteOpRateHighThld;
        nfs_addr_stats_ptr->writeCacheOps.high_thld =
                                    mib_nfs_addr_defs->nfsAddrWriteCacheOpHighThld;
        nfs_addr_stats_ptr->writeCacheOpRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrWriteCacheOpRateHighThld;

        nfs_addr_stats_ptr->readBytes.high_thld =
                                    mib_nfs_addr_defs->nfsAddrReadByteHighThld;
        nfs_addr_stats_ptr->readByteRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrReadByteRateHighThld;
        nfs_addr_stats_ptr->writeBytes.high_thld =
                                    mib_nfs_addr_defs->nfsAddrWriteByteHighThld;
        nfs_addr_stats_ptr->writeByteRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrWriteByteRateHighThld;

    /*
        nfs_addr_stats_ptr->rcvOffSegs.high_thld =
                                    mib_nfs_addr_defs->nfsAddrRcvOffSegHighThld;
        nfs_addr_stats_ptr->rcvOffSegRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrRcvOffSegRateHighThld;
        nfs_addr_stats_ptr->xmtOffSegs.high_thld =
                                    mib_nfs_addr_defs->nfsAddrXmtOffSegHighThld;
        nfs_addr_stats_ptr->xmtOffSegRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrXmtOffSegRateHighThld;
    */

        nfs_addr_stats_ptr->mountFailures.high_thld =
                                    mib_nfs_addr_defs->nfsAddrMountFailureHighThld;
        nfs_addr_stats_ptr->mountFailureRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrMountFailureRateHighThld;

        nfs_addr_stats_ptr->nfsErrPerm.high_thld =
                                    mib_nfs_addr_defs->nfsAddrErrPermHighThld;
        nfs_addr_stats_ptr->nfsErrPermRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrErrPermRateHighThld;

        nfs_addr_stats_ptr->nfsErrNoEnt.high_thld =
```

```
                                mib_nfs_addr_defs->nfsAddrErrNoEntHighThld;
nfs_addr_stats_ptr->nfsErrNoEntRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNoEntRateHighThld;

nfs_addr_stats_ptr->nfsErrIO.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrIOHighThld;
nfs_addr_stats_ptr->nfsErrIORate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrIORateHighThld;

nfs_addr_stats_ptr->nfsErrNXIO.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNXIOHighThld;
nfs_addr_stats_ptr->nfsErrNXIORate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNXIORateHighThld;

nfs_addr_stats_ptr->nfsErrAccess.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrAccessHighThld;
nfs_addr_stats_ptr->nfsErrAccessRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrAccessRateHighThld;

nfs_addr_stats_ptr->nfsErrExist.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrExistHighThld;
nfs_addr_stats_ptr->nfsErrExistRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrExistRateHighThld;

nfs_addr_stats_ptr->nfsErrNoDev.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNoDevHighThld;
nfs_addr_stats_ptr->nfsErrNoDevRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNoDevRateHighThld;

nfs_addr_stats_ptr->nfsErrNotDir.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNotDirHighThld;
nfs_addr_stats_ptr->nfsErrNotDirRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNotDirRateHighThld;

nfs_addr_stats_ptr->nfsErrIsDir.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrIsDirHighThld;
nfs_addr_stats_ptr->nfsErrIsDirRate.high_thld =
```

```
                                mib_nfs_addr_defs->nfsAddrErrIsDirRateHighThld;

    nfs_addr_stats_ptr->nfsErrFBig.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrFBigHighThld;
    nfs_addr_stats_ptr->nfsErrFBigRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrFBigRateHighThld;

    nfs_addr_stats_ptr->nfsErrNoSpace.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNoSpaceHighThld;
    nfs_addr_stats_ptr->nfsErrNoSpaceRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNoSpaceRateHighThld;

    nfs_addr_stats_ptr->nfsErrROFS.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrROFSHighThld;
    nfs_addr_stats_ptr->nfsErrROFSRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrROFSRateHighThld;

    nfs_addr_stats_ptr->nfsErrNameTooLong.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNameTooLongHighThld;
    nfs_addr_stats_ptr->nfsErrNameTooLongRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNameTooLongRateHighThld;

    nfs_addr_stats_ptr->nfsErrNotEmpty.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNotEmptyHighThld;
    nfs_addr_stats_ptr->nfsErrNotEmptyRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrNotEmptyRateHighThld;

    nfs_addr_stats_ptr->nfsErrDQuota.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrDQuotaHighThld;
    nfs_addr_stats_ptr->nfsErrDQuotaRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrDQuotaRateHighThld;

    nfs_addr_stats_ptr->nfsErrStale.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrStaleHighThld;
    nfs_addr_stats_ptr->nfsErrStaleRate.high_thld =
                                mib_nfs_addr_defs->nfsAddrErrStaleRateHighThld;
```

```
        nfs_addr_stats_ptr->nfsErrWFlush.high_thld =
                                    mib_nfs_addr_defs->nfsAddrErrWFlushHighThld;
        nfs_addr_stats_ptr->nfsErrWFlushRate.high_thld =
                                    mib_nfs_addr_defs->nfsAddrErrWFlushRateHighThld;

        nfs_addr_stats_ptr->rpcStats.versionMismatch.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.versionMismatchHighThld;
        nfs_addr_stats_ptr->rpcStats.versionMismatchRate.high_thld =

mib_nfs_addr_defs->rpcThlds.versionMismatchRateHighThld;
        nfs_addr_stats_ptr->rpcStats.authBadCred.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authBadCredHighThld;
        nfs_addr_stats_ptr->rpcStats.authBadCredRate.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authBadCredRateHighThld;
        nfs_addr_stats_ptr->rpcStats.authRejectedCred.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authRejectedCredHighThld;
        nfs_addr_stats_ptr->rpcStats.authRejectedCredRate.high_thld =

mib_nfs_addr_defs->rpcThlds.authRejectedCredRateHighThld;
        nfs_addr_stats_ptr->rpcStats.authBadVerf.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authBadVerfHighThld;
        nfs_addr_stats_ptr->rpcStats.authBadVerfRate.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authBadVerfRateHighThld;
        nfs_addr_stats_ptr->rpcStats.authRejectedVerf.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authRejectedVerfHighThld;
        nfs_addr_stats_ptr->rpcStats.authRejectedVerfRate.high_thld =

mib_nfs_addr_defs->rpcThlds.authRejectedVerfRateHighThld;
        nfs_addr_stats_ptr->rpcStats.authTooWeak.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authTooWeakHighThld;
        nfs_addr_stats_ptr->rpcStats.authTooWeakRate.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authTooWeakRateHighThld;
        nfs_addr_stats_ptr->rpcStats.authOther.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authOtherHighThld;
        nfs_addr_stats_ptr->rpcStats.authOtherRate.high_thld =
                                    mib_nfs_addr_defs->rpcThlds.authOtherRateHighThld;
        nfs_addr_stats_ptr->rpcStats.progUnavail.high_thld =
```

```
                                        mib_nfs_addr_defs->rpcThlds.progUnavailHighThld;
        nfs_addr_stats_ptr->rpcStats.progUnavailRate.high_thld =
                                        mib_nfs_addr_defs->rpcThlds.progUnavailRateHighThld;
        nfs_addr_stats_ptr->rpcStats.progVersionMismatch.high_thld =

mib_nfs_addr_defs->rpcThlds.progVersionMismatchHighThld;
        nfs_addr_stats_ptr->rpcStats.progVersionMismatchRate.high_thld =

mib_nfs_addr_defs->rpcThlds.progVersionMismatchRateHighThld;
        nfs_addr_stats_ptr->rpcStats.procUnavail.high_thld =
                                        mib_nfs_addr_defs->rpcThlds.procUnavailHighThld;
        nfs_addr_stats_ptr->rpcStats.procUnavailRate.high_thld =
                                        mib_nfs_addr_defs->rpcThlds.procUnavailRateHighThld;
        nfs_addr_stats_ptr->rpcStats.procGarbageArgs.high_thld =
                                        mib_nfs_addr_defs->rpcThlds.procGarbageArgsHighThld;
        nfs_addr_stats_ptr->rpcStats.procGarbageArgsRate.high_thld =

mib_nfs_addr_defs->rpcThlds.procGarbageArgsRateHighThld;
}


void setNfsAddrPairDflts (mib_nfs_addr_pair_defs, dialog_ptr)
        MibNfsAddrPairDefaults    *mib_nfs_addr_pair_defs;
        StatsNfsDialogEntry       *dialog_ptr;
{
        dialog_ptr->ops.high_thld =
                                mib_nfs_addr_pair_defs->nfsAddrPairOpsHighThld;
        dialog_ptr->opRate.high_thld =
                                mib_nfs_addr_pair_defs->nfsAddrPairOpRateHighThld;
        dialog_ptr->readOps.high_thld =
                                mib_nfs_addr_pair_defs->nfsAddrPairReadOpsHighThld;
        dialog_ptr->readOpRate.high_thld =
                                mib_nfs_addr_pair_defs->nfsAddrPairReadOpRateHighThld;
        dialog_ptr->writeOps.high_thld =
                                mib_nfs_addr_pair_defs->nfsAddrPairWriteOpsHighThld;
        dialog_ptr->writeOpRate.high_thld =
                                mib_nfs_addr_pair_defs->nfsAddrPairWriteOpRateHighThld;
```

```
dialog_ptr->writeCacheOps.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairWriteCacheOpsHighThld;
dialog_ptr->writeCacheOpRate.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairWriteCacheOpRateHighThld;


dialog_ptr->readBytes.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairReadBytesHighThld;
dialog_ptr->readByteRate.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairReadByteRateHighThld;
dialog_ptr->writeBytes.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairWriteBytesHighThld;
dialog_ptr->writeByteRate.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairWriteByteRateHighThld;

dialog_ptr->rexmts.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairRexmtsHighThld;
dialog_ptr->rexmtRate.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairRexmtRateHighThld;

dialog_ptr->rpcProgramFail.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairRpcProgramFailHighThld;
dialog_ptr->rpcProgramFailRate.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairRpcProgramFailRateHighThld;

dialog_ptr->rpcMismatch.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairRpcMismatchHighThld;
dialog_ptr->rpcMismatchRate.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairRpcMismatchRateHighThld;

dialog_ptr->rpcAuthFail.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairRpcAuthFailHighThld;
dialog_ptr->rpcAuthFailRate.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairRpcAuthFailRateHighThld;

dialog_ptr->nfsAccessErr.high_thld =
                    mib_nfs_addr_pair_defs->nfsAddrPairNfsAccessErrHighThld;
```

```
        dialog_ptr->nfsAccessErrRate.high_thld =
                            mib_nfs_addr_pair_defs->nfsAddrPairNfsAccessErrRateHighThld;

        dialog_ptr->nfsHardErr.high_thld =
                            mib_nfs_addr_pair_defs->nfsAddrPairNfsHardErrHighThld;
        dialog_ptr->nfsHardErrRate.high_thld =
                            mib_nfs_addr_pair_defs->nfsAddrPairNfsHardErrRateHighThld;

        dialog_ptr->nfsResourceErr.high_thld =
                            mib_nfs_addr_pair_defs->nfsAddrPairNfsResourceErrHighThld;
        dialog_ptr->nfsResourceErrRate.high_thld =
                            mib_nfs_addr_pair_defs->nfsAddrPairNfsResourceErrRateHighThld;

        dialog_ptr->nfsUserErr.high_thld =
                            mib_nfs_addr_pair_defs->nfsAddrPairNfsUserErrHighThld;
        dialog_ptr->nfsUserErrRate.high_thld =
                            mib_nfs_addr_pair_defs->nfsAddrPairNfsUserErrRateHighThld;
}


void setNfsSegDflts (mib_nfs_seg_defs, nfs_seg_stats_ptr)
        MibNfsSegDefaults   *mib_nfs_seg_defs;
        StatsNfsSegment          *nfs_seg_stats_ptr;
{
        nfs_seg_stats_ptr->ops.high_thld =
                                    mib_nfs_seg_defs->nfsSegOpHighThld;
        nfs_seg_stats_ptr->opRate.high_thld =
                                    mib_nfs_seg_defs->nfsSegOpRateHighThld;
        nfs_seg_stats_ptr->readOps.high_thld =
                                    mib_nfs_seg_defs->nfsSegReadOpHighThld;
        nfs_seg_stats_ptr->readOpRate.high_thld =
                                    mib_nfs_seg_defs->nfsSegReadOpRateHighThld;
        nfs_seg_stats_ptr->writeOps.high_thld =
                                    mib_nfs_seg_defs->nfsSegWriteOpHighThld;
        nfs_seg_stats_ptr->writeOpRate.high_thld =
                                    mib_nfs_seg_defs->nfsSegWriteOpRateHighThld;
        nfs_seg_stats_ptr->writeCacheOps.high_thld =
```

```
                                mib_nfs_seg_defs->nfsSegWriteCacheOpHighThld;
nfs_seg_stats_ptr->writeCacheOpRate.high_thld =
                                mib_nfs_seg_defs->nfsSegWriteCacheOpRateHighThld;

nfs_seg_stats_ptr->readBytes.high_thld =
                                mib_nfs_seg_defs->nfsSegReadByteHighThld;
nfs_seg_stats_ptr->readByteRate.high_thld =
                                mib_nfs_seg_defs->nfsSegReadByteRateHighThld;
nfs_seg_stats_ptr->writeBytes.high_thld =
                                mib_nfs_seg_defs->nfsSegWriteByteHighThld;
nfs_seg_stats_ptr->writeByteRate.high_thld =
                                mib_nfs_seg_defs->nfsSegWriteByteRateHighThld;

nfs_seg_stats_ptr->rcvOffSegs.high_thld =
                                mib_nfs_seg_defs->nfsSegRcvOffSegHighThld;
nfs_seg_stats_ptr->rcvOffSegRate.high_thld =
                                mib_nfs_seg_defs->nfsSegRcvOffSegRateHighThld;
nfs_seg_stats_ptr->xmtOffSegs.high_thld =
                                mib_nfs_seg_defs->nfsSegXmtOffSegHighThld;
nfs_seg_stats_ptr->xmtOffSegRate.high_thld =
                                mib_nfs_seg_defs->nfsSegXmtOffSegRateHighThld;
nfs_seg_stats_ptr->transits.high_thld =
                                mib_nfs_seg_defs->nfsSegTransitHighThld;
nfs_seg_stats_ptr->transitRate.high_thld =
                                mib_nfs_seg_defs->nfsSegTransitRateHighThld;

nfs_seg_stats_ptr->flowCtrls.high_thld =
                                mib_nfs_seg_defs->nfsSegFlowCtrlHighThld;
nfs_seg_stats_ptr->flowCtrlRate.high_thld =
                                mib_nfs_seg_defs->nfsSegFlowCtrlRateHighThld;

nfs_seg_stats_ptr->mountFailures.high_thld =
                               mib_nfs_seg_defs->nfsSegMountFailureHighThld;
nfs_seg_stats_ptr->mountFailureRate.high_thld =
                               mib_nfs_seg_defs->nfsSegMountFailureRateHighThld;

nfs_seg_stats_ptr->nfsErrPerm.high_thld =
```

```
                                mib_nfs_seg_defs->nfsSegErrPermHighThld;
nfs_seg_stats_ptr->nfsErrPermRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrPermRateHighThld;
nfs_seg_stats_ptr->nfsErrNoEnt.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNoEntHighThld;
nfs_seg_stats_ptr->nfsErrNoEntRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNoEntRateHighThld;
nfs_seg_stats_ptr->nfsErrIO.high_thld =
                                mib_nfs_seg_defs->nfsSegErrIOHighThld;
nfs_seg_stats_ptr->nfsErrIORate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrIORateHighThld;
nfs_seg_stats_ptr->nfsErrNXIO.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNXIOHighThld;
nfs_seg_stats_ptr->nfsErrNXIORate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNXIORateHighThld;
nfs_seg_stats_ptr->nfsErrAccess.high_thld =
                                mib_nfs_seg_defs->nfsSegErrAccessHighThld;
nfs_seg_stats_ptr->nfsErrAccessRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrAccessRateHighThld;
nfs_seg_stats_ptr->nfsErrExist.high_thld =
                                mib_nfs_seg_defs->nfsSegErrExistHighThld;
nfs_seg_stats_ptr->nfsErrExistRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrExistRateHighThld;
nfs_seg_stats_ptr->nfsErrNoDev.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNoDevHighThld;
nfs_seg_stats_ptr->nfsErrNoDevRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNoDevRateHighThld;
nfs_seg_stats_ptr->nfsErrNotDir.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNotDirHighThld;
nfs_seg_stats_ptr->nfsErrNotDirRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNotDirRateHighThld;
nfs_seg_stats_ptr->nfsErrIsDir.high_thld =
                                mib_nfs_seg_defs->nfsSegErrIsDirHighThld;
nfs_seg_stats_ptr->nfsErrIsDirRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrIsDirRateHighThld;
nfs_seg_stats_ptr->nfsErrFBig.high_thld =
                                mib_nfs_seg_defs->nfsSegErrFBigHighThld;
```

```
nfs_seg_stats_ptr->nfsErrFBigRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrFBigRateHighThld;
nfs_seg_stats_ptr->nfsErrNoSpace.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNoSpaceHighThld;
nfs_seg_stats_ptr->nfsErrNoSpaceRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNoSpaceRateHighThld;
nfs_seg_stats_ptr->nfsErrROFS.high_thld =
                                mib_nfs_seg_defs->nfsSegErrROFSHighThld;
nfs_seg_stats_ptr->nfsErrROFSRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrROFSRateHighThld;
nfs_seg_stats_ptr->nfsErrNameTooLong.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNameTooLongHighThld;
nfs_seg_stats_ptr->nfsErrNameTooLongRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNameTooLongRateHighThld;
nfs_seg_stats_ptr->nfsErrNotEmpty.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNotEmptyHighThld;
nfs_seg_stats_ptr->nfsErrNotEmptyRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrNotEmptyRateHighThld;
nfs_seg_stats_ptr->nfsErrDQuota.high_thld =
                                mib_nfs_seg_defs->nfsSegErrDQuotaHighThld;
nfs_seg_stats_ptr->nfsErrDQuotaRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrDQuotaRateHighThld;
nfs_seg_stats_ptr->nfsErrStale.high_thld =
                                mib_nfs_seg_defs->nfsSegErrStaleHighThld;
nfs_seg_stats_ptr->nfsErrStaleRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrStaleRateHighThld;
nfs_seg_stats_ptr->nfsErrWFlush.high_thld =
                                mib_nfs_seg_defs->nfsSegErrWFlushHighThld;
nfs_seg_stats_ptr->nfsErrWFlushRate.high_thld =
                                mib_nfs_seg_defs->nfsSegErrWFlushRateHighThld;

nfs_seg_stats_ptr->rpcStats.versionMismatch.high_thld =
                                mib_nfs_seg_defs->rpcThlds.versionMismatchHighThld;
nfs_seg_stats_ptr->rpcStats.versionMismatchRate.high_thld =
                                mib_nfs_seg_defs->rpcThlds.versionMismatchRateHighThld;

nfs_seg_stats_ptr->rpcStats.authBadCred.high_thld =
```

```
                                    mib_nfs_seg_defs->rpcThlds.authBadCredHighThld;
        nfs_seg_stats_ptr->rpcStats.authBadCredRate.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.authBadCredRateHighThld;
        nfs_seg_stats_ptr->rpcStats.authRejectedCred.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.authRejectedCredHighThld;
        nfs_seg_stats_ptr->rpcStats.authRejectedCredRate.high_thld =

mib_nfs_seg_defs->rpcThlds.authRejectedCredRateHighThld;
        nfs_seg_stats_ptr->rpcStats.authBadVerf.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.authBadVerfHighThld;
        nfs_seg_stats_ptr->rpcStats.authBadVerfRate.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.authBadVerfRateHighThld;
        nfs_seg_stats_ptr->rpcStats.authRejectedVerf.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.authRejectedVerfHighThld;
        nfs_seg_stats_ptr->rpcStats.authRejectedVerfRate.high_thld =

mib_nfs_seg_defs->rpcThlds.authRejectedVerfRateHighThld;
        nfs_seg_stats_ptr->rpcStats.authTooWeak.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.authTooWeakHighThld;
        nfs_seg_stats_ptr->rpcStats.authTooWeakRate.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.authTooWeakRateHighThld;
        nfs_seg_stats_ptr->rpcStats.authOther.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.authOtherHighThld;
        nfs_seg_stats_ptr->rpcStats.authOtherRate.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.authOtherRateHighThld;
        nfs_seg_stats_ptr->rpcStats.progUnavail.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.progUnavailHighThld;
        nfs_seg_stats_ptr->rpcStats.progUnavailRate.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.progUnavailRateHighThld;
        nfs_seg_stats_ptr->rpcStats.progVersionMismatch.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.progVersionMismatchHighThld;

        nfs_seg_stats_ptr->rpcStats.progVersionMismatchRate.high_thld =

mib_nfs_seg_defs->rpcThlds.progVersionMismatchRateHighThld;
        nfs_seg_stats_ptr->rpcStats.procUnavail.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.procUnavailHighThld;
```

```
        nfs_seg_stats_ptr->rpcStats.procUnavailRate.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.procUnavailRateHighThld;
        nfs_seg_stats_ptr->rpcStats.procGarbageArgs.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.procGarbageArgsHighThld;
        nfs_seg_stats_ptr->rpcStats.procGarbageArgsRate.high_thld =
                                    mib_nfs_seg_defs->rpcThlds.procGarbageArgsRateHighThld;

}
```

```
/*
 *  stats_pmap_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                   All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_pmap_p.c
 *
 *  Date:       8/8/91
 *
 *  Revision:   1.1
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   07-24-91  KR        Created
 */

static char stats_pmap_p_c [] = "@(#)stats_pmap_p.c    1.1";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
#include "mib_defs.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
```

```
#include </usr/include/bsd43/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_mib.h"
#include "stats_pmap.h"
#include "rtp.h"
#include "rtp_pmap.h"


/*
 * Local shared variables
 */
static FBQhead_type      *pmapQ_ptr;
static StatsPmapping     *pmap_ptr;


/*****************************************************************************
 *
 * Given an ip address structure, walk the port map queue.  If the port
 * is found, return the pointer to the structure, otherwise return NULL.
 */
StatsPmapping *stats_pmap_lookup (ip_addr_ptr, port, transport_protocol)

    StatsAddrEntry                  *ip_addr_ptr;
    Uint32                          port, transport_protocol;

{
if (ip_addr_ptr != NULL)
    {
    if (ip_addr_ptr->protocol_specific_ptr != NULL)
        {
        pmapQ_ptr = (FBQhead_type*) ip_addr_ptr->protocol_specific_ptr;
        pmap_ptr = (StatsPmapping *) pmapQ_ptr->plink;
```

```
            while (pmap_ptr != NULL)
                  {
                  if ( (pmap_ptr->port == port) &&
                        (pmap_ptr->transport_protocol == transport_protocol) )
                       return (pmap_ptr);
                  pmap_ptr = (StatsPmapping *) pmap_ptr->link.pFlink;
                  }
            }
      }
return (NULL);
}



/*****************************************************************************
 *
 * Given an ip address structure, attempt to allocate a structure and attach it
 * to the ip address record.  Return the pointer to the structure or NULL.
 * The caller should have invoked pmap_look first.  User must initialize
 * the structure after doing this get.
 */
StatsPmapping *stats_pmap_get (ip_addr_ptr, port, transport_protocol)

     StatsAddrEntry                    *ip_addr_ptr;
     Uint32                            port, transport_protocol;

{
if (ip_addr_ptr == NULL)
     return (NULL);

/*
 * If necessary, get a queue structure
 */
if (ip_addr_ptr->protocol_specific_ptr == NULL)
     {
     pmapQ_ptr = (FBQhead_type *) stats_allocate (sizeof(FBQhead_type) );
```

```
     if (pmapQ_ptr == NULL)
          return (NULL);
     FBInitqh (pmapQ_ptr);
     ip_addr_ptr->protocol_specific_ptr = (Uint32 *) pmapQ_ptr;
     }

/*
 * Get the port mapping structure
 */
pmap_ptr = (StatsPmapping *) stats_allocate (sizeof (StatsPmapping));
if (pmap_ptr != NULL)
     FBInsqt (pmapQ_ptr, (PFBQentry_type) pmap_ptr);

return (pmap_ptr);
}



/*************************************************************************
 *
 * Remove a mapping for an ip address.  Look it up.  This will set
 * pmap_ptr and pmapQ_ptr.
 */
 void stats_pmap_deallocate (ip_addr_ptr, port, transport_protocol)

     StatsAddrEntry                    *ip_addr_ptr;
     Uint32                            port, transport_protocol;

{
if ( stats_pmap_lookup (ip_addr_ptr, port, transport_protocol) == NULL )
     return;

FBRemqm (pmapQ_ptr, (PFBQentry_type) pmap_ptr);
stats_deallocate (pmapQ_ptr, sizeof(pmapQ_ptr));
}
```

```
/*
 *  stats_tcp_a.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                   All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_tcp_a.c
 *
 *  Date:       6/11/91
 *
 *  Revision:   1.9
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change, (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   06-10-91  KR        Removed decr of active connection count
 *   06-07-91  KR        Added failed connection count to age of dialogs
 *   06-05-91  DPD       Fixed aging bug.
 *   06-04-91  DPD       Modified rate calcs for hourly rates:
 *                            successful, failed, and conn retries
 *   06-01-91  DPD       Fixed rate bug.
 *
 */

static char stats_tcp_a_c [] = "@(#)stats_tcp_a.c 1.9";

#include <stdio.h>

#include <cci_std.h>
```

```
#include "system.h"
#include "address.h"
#include "mib_defs.h"
#include "mib_ip.h"
#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "protocols.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_tcp.h"
#include "stats_mib.h"
#include "mib_tcp.h"
#include "snmpd.h"
#include "em_ctrl.h"

/*
 * Local TCP data structures
 */
static StatsAddrEntry       *this_seg_addr_ptr;
static StatsTcpSegment      *this_seg_stats_ptr;

static StatsAddrEntry       *src_seg_addr_ptr;
static StatsTcpSegment      *src_seg_stats_ptr;

static StatsAddrEntry       *dst_seg_addr_ptr;
static StatsTcpSegment      *dst_seg_stats_ptr;
```

```
static StatsAddrEntry       *src_node_addr_ptr;
static StatsTcpAddr         *src_node_stats_ptr;

static StatsAddrEntry       *dst_node_addr_ptr;
static StatsTcpAddr         *dst_node_stats_ptr;

static StatsAddrEntry       *src_socket_addr_ptr;
static StatsTcpSocket       *src_socket_stats_ptr;

static StatsAddrEntry       *dst_socket_addr_ptr;
static StatsTcpSocket       *dst_socket_stats_ptr;

static StatsAddrEntry       *dialog_addr_ptr;
static StatsDialogEntry     *dialog_stats_ptr;



/*****************************************************************
 *
 * Given a TCP dialog, determine if it the state should be set to closed,
 * or if it should be aged out.
 */

void statsTcpAgeDialog (dialog_to_age)

     StatsAddrEntry      *dialog_to_age;

{
     register Uint32                 xtcp_dialog_hash;
     register StatsAddrEntry        *xtcp_dialog_hash_link;
     register StatsAddrEntry        *xtcp_previous_dialog_hash_link;

     register StatsDialogLink *dialog_link_ptr;
     register StatsAddrEntry        *addr_record_ptr;
```

```
        register StatsTcpAddr           *tcp_stats_ptr;
        register StatsTcpSocket         *tcp_socket_stats_ptr;
        register StatsAddrEntry         *ip_src_addr_ptr, *ip_dst_addr_ptr;
        struct timeval                  current_time;
        Uint32                              i;

    if (monCtrl.tcpDialogAgeTimer == 0)
        return;

    if (dialog_to_age == NULL)
        return;

    mon_gettimeofday (&current_time, 0);


    if (current_time.tv_sec - dialog_to_age->lastTime < monCtrl.tcpDialogAgeTimer)
        {
        /*
         * Not time to age out, but check the close wait timer.  If it has expired, set
         * the state to closed if the state is closing.
         */
        if (current_time.tv_sec - dialog_to_age->lastTime < TCP_CLOSE_WAIT_TIME)
            return;
        dialog_stats_ptr = (StatsDialogEntry *) dialog_to_age->stats_ptr;
        if (dialog_stats_ptr != NULL)
            {
            if (dialog_stats_ptr->transport.state == ConnectionStateClosing)
                {
                dialog_stats_ptr->transport.state = ConnectionStateClosed;
                dialog_stats_ptr->transport.state = ConnectionStateClosed;
                }
            if (dialog_stats_ptr->transport.state == ConnectionStateClosing)
                {
                dialog_stats_ptr->transport.state = ConnectionStateClosed;
                stats_tcp_lookup_ptrs (
                    &dialog_to_age->address.netAddress1.u.ipAddress,
                    dialog_to_age->address.port1,
                    &dialog_to_age->address.netAddress2.u.ipAddress,
                    dialog_to_age->address.port2);
                if (dialog_to_age != dialog_addr_ptr)
                    {
```

```
                    mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsTcpAgeDialog: ptrs not
equal");
                    return;
                    }
                }
            }
        return;
        }

/*
 * Time to chuck this dialog.  Remove it from the dialog hash table.
 * Look up the dialog in order to set tcp_dialog_hash, tcp_previous_dialog_hash_link
 * and tcp_dialog_hash_link.  Lookup other pointers in order to update statistics.
 * Although src and dst pointers are set up, there is no real concept of direction
 * at this point.
 */
stats_tcp_lookup_ptrs (
     &dialog_to_age->address.netAddress1.u.ipAddress,
     dialog_to_age->address.port1,
     &dialog_to_age->address.netAddress2.u.ipAddress,
     dialog_to_age->address.port2);

if (dialog_to_age != dialog_addr_ptr)
     {
     mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsTcpAgeDialog: ptrs not equal");
     return;
     }

xtcp_dialog_hash = tcp_dialog_hash;
xtcp_dialog_hash_link = tcp_dialog_hash_link;
xtcp_previous_dialog_hash_link = tcp_previous_dialog_hash_link;

if ( (xtcp_previous_dialog_hash_link == NULL) && (xtcp_dialog_hash_link->hash_link ==
NULL) )
     tcp_dialog_hash_table[xtcp_dialog_hash] = NULL;
else
     {
```

```
        if (xtcp_previous_dialog_hash_link != NULL)
             xtcp_previous_dialog_hash_link->hash_link = xtcp_dialog_hash_link->hash_link;
        else
             if (xtcp_dialog_hash_link->hash_link != NULL)
                  tcp_dialog_hash_table[xtcp_dialog_hash] = xtcp_dialog_hash_link->hash_link;

        }

   /*
    * Remove the dialog from the dialogQ for each of the tcp stats structures for the
    * ip addresses and deallocate the structure that was on the dialogQ which pointed
    * to the dialog.
    */

   if (src_node_addr_ptr != NULL)
        {
        if (src_node_stats_ptr != NULL)
             {
             dialog_link_ptr = (StatsDialogLink *) src_node_stats_ptr->dialogQ.pFlink;
             while (dialog_link_ptr != NULL)
                  {
                  if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                       break;
                  dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
                  }
             if (dialog_link_ptr != NULL)
                  {
                  FBRemqm (&src_node_stats_ptr->dialogQ, (PFBQentry_type) dialog_link_ptr);
                  stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                  }
             }
        }

   if (dst_node_addr_ptr != NULL)
        {
        if (dst_node_stats_ptr != NULL)
             {
```

```
            dialog_link_ptr = (StatsDialogLink *) dst_node_stats_ptr->dialogQ.pFlink;
            while (dialog_link_ptr != NULL)
                 {
                 if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                      break;
                 dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
                 }
            if (dialog_link_ptr != NULL)
                 {
                 FBRemqn (&dst_node_stats_ptr->dialogQ, (PFBQentry_type) dialog_link_ptr);
                 stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                 }
            }
       }

  /*
   * If the connection is in data state, the active connection count needs to be
   * decremented.
   */
  if (dialog_stats_ptr->transport.state == ConnectionStateData)
       {
       /*
        * Maybe call rtp_tcp_state here...
        */
        tcp_stats_decr_active_connections;
       }

  /*
   * If the connection is in connecting state, the failed connection count needs to be
   * incremented.
   */
  if (dialog_stats_ptr->transport.state == ConnectionStateConnecting)
       {
       /*
        * Maybe call rtp_tcp_state here...
        */
        tcp_stats_failed_connections;
```

```
     }


/*
 * Remove the dialog from the dialogQ for each of the sockets
 * and deallocate the structure that was on the dialogQ which pointed
 * to the dialog.
 */
if (src_socket_addr_ptr != NULL)
     {
     if (src_socket_stats_ptr != NULL)
          {
          dialog_link_ptr = (StatsDialogLink *) src_socket_stats_ptr->dialogQ.pFlink;
          while (dialog_link_ptr != NULL)
               {
               if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                    break;
               dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
               }
          if (dialog_link_ptr != NULL)
               {
               FBRemqn (&src_socket_stats_ptr->dialogQ, (PFBQentry_type) dialog_link_ptr);

               stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
               }
          }
     }

if (dst_socket_addr_ptr != NULL)
     {
     if (dst_socket_stats_ptr != NULL)
          {
          dialog_link_ptr = (StatsDialogLink *) dst_socket_stats_ptr->dialogQ.pFlink;
          while (dialog_link_ptr != NULL)
               {
               if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                    break;
```

```
                dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
                }
            if (dialog_link_ptr != NULL)
                {
                FBRemqm (&dst_socket_stats_ptr->dialogQ, (PFBQentry_type) dialog_link_ptr);

                stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                }
            }
        }


/*
 * Remove the dialog from the statsTcpDialogQ and deallocate the structures
 * associated with the dialog.
 */
FBRemqm (&statsTcpDialogQ, (PFBQentry_type) dialog_addr_ptr);

if (dialog_stats_ptr->transport.state_ptr != NULL)
     stats_deallocate (dialog_stats_ptr->transport.state_ptr,
sizeof(TcpConnectionStateType));

stats_deallocate (dialog_stats_ptr, sizeof(StatsDialogEntry) );
stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );

}


/*****************************************************************
 *
 * Given a socket, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsTcpAgeAddr (tcp_addr_ptr)
```

```
    StatsAddrEntry      *tcp_addr_ptr;

{
    register StatsDialogLink *dialog_link_ptr;
    register StatsAddrEntry      *addr_record_ptr;
    register StatsTcpAddr        *tcp_stats_ptr;
    struct timeval               current_time;

if (monCtrl.tcpNodeAgeTimer == 0)
    return;

if (tcp_addr_ptr == NULL)
    return;

tcp_stats_ptr = (StatsTcpAddr *) tcp_addr_ptr->stats_ptr;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - tcp_addr_ptr->lastTime < monCtrl.tcpNodeAgeTimer)
    return;

/*
 * Time to chuck this socket.  Remove it from the hash table.
 * Look up the ip address in order to set tcp_hash, tcp_previous_hash_link
 * and tcp_hash_link.
 */
addr_record_ptr = stats_tcp_lookup_addr (
    &tcp_addr_ptr->address.netAddress1.u.ipAddress);

if (addr_record_ptr != tcp_addr_ptr)
    {
    mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsTcpAgeAddr: ptrs not equal");
    return;
    }

if ( (tcp_previous_hash_link == NULL) && (tcp_hash_link->hash_link == NULL) )
    tcp_hash_table[tcp_hash] = NULL;
```

```
else
     {
     if (tcp_previous_hash_link != NULL)
          tcp_previous_hash_link->hash_link = tcp_hash_link->hash_link;
     else
          if (tcp_hash_link->hash_link != NULL)
               tcp_hash_table[tcp_hash] = tcp_hash_link->hash_link;
     }

/*
 * Remove the dialog links from the socket stats and deallocate the stats structure.
 */
if (tcp_stats_ptr != NULL)
     {
     dialog_link_ptr = (StatsDialogLink *) FBRemqh (&tcp_stats_ptr->dialogQ);
     while (dialog_link_ptr != NULL)
          {
          stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
          dialog_link_ptr = (StatsDialogLink *) FBRemqh (&tcp_stats_ptr->dialogQ);
          }
     stats_deallocate (tcp_stats_ptr, sizeof(StatsTcpAddr) );
     }


/*
 * Remove the address from the statsTcpAddrQ and deallocate the address structure.
 */
FBRemqm (&statsTcpAddrQ, (PFBQentry_type) tcp_addr_ptr);
stats_deallocate (tcp_addr_ptr, sizeof(StatsAddrEntry) );
tcp_addr_ptr->stats_ptr = NULL;

}


/*****************************************************************************
 *
```

```
 * Given a socket, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsTcpAgeSocket (socket_addr_ptr)

     StatsAddrEntry      *socket_addr_ptr;

{
     register StatsDialogLink *dialog_link_ptr;
     register StatsAddrEntry       *addr_record_ptr;
     register StatsTcpSocket       *socket_stats_ptr;
     struct timeval                current_time;

if (monCtrl.tcpWellKnownAgeTimer == 0)
     return;

if (socket_addr_ptr == NULL)
     return;

socket_stats_ptr = (StatsTcpSocket *) socket_addr_ptr->stats_ptr;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - socket_addr_ptr->lastTime < monCtrl.tcpWellKnownAgeTimer)
     return;

/*
 * Time to chuck this socket.  Remove it from the socket hash table.
 * Look up the socket in order to set tcp_socket_hash, tcp_previous_socket_hash_link
 * and tcp_socket_hash_link.
 */
addr_record_ptr = stats_tcp_lookup_socket (
     &socket_addr_ptr->address.netAddress1.u.ipAddress,
     socket_addr_ptr->address.port1);

if (addr_record_ptr != socket_addr_ptr)
```

```
        {
        mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsTcpAgeSocket: ptrs not equal");
        return;
        }

if ( (tcp_previous_socket_hash_link == NULL) && (tcp_socket_hash_link->hash_link == NULL)
)
        tcp_socket_hash_table[tcp_socket_hash] = NULL;
else
        {
        if (tcp_previous_socket_hash_link != NULL)
                tcp_previous_socket_hash_link->hash_link = tcp_socket_hash_link->hash_link;
        else
                if (tcp_socket_hash_link->hash_link != NULL)
                        tcp_socket_hash_table[tcp_socket_hash] = tcp_socket_hash_link->hash_link;
        }

/*
 * Remove the dialog links from the socket stats and deallocate the stats structure.
 */
if (socket_stats_ptr != NULL)
        {
        dialog_link_ptr = (StatsDialogLink *) FBRemqh (&socket_stats_ptr->dialogQ);
        while (dialog_link_ptr != NULL)
                {
                stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                dialog_link_ptr = (StatsDialogLink *) FBRemqh (&socket_stats_ptr->dialogQ);
                }
        stats_deallocate (socket_stats_ptr, sizeof(StatsTcpAddr) );
        }


/*
 * Remove the socket from the statsTcpSocketQ and deallocate the address structure.
 */
FBRemqm (&statsTcpSocketQ, (PFBQentry_type) socket_addr_ptr);
stats_deallocate (socket_addr_ptr, sizeof(StatsAddrEntry) );
```

```
socket_addr_ptr->stats_ptr = NULL;
}


/*****************************************************************************
 *
 * TCP rate routines
 */

void statsTcpProtocolRate (tcp_protocol_ptr, tcp_addr_ptr, rate_type, time)
     StatsProtocolEntry  *tcp_protocol_ptr;
     StatsAddrEntry                *tcp_addr_ptr;
     Uint32                             rate_type;
     Uint32                             time;
{
     AlarmUserData       tcp_alarm_data;

     while (tcp_protocol_ptr != NULL)
     {
          /*
           * Pass the protocol as alarm data in case an alarm occurs
           */
          tcp_alarm_data.length = 4;
          bcopy (&tcp_protocol_ptr->protocol, tcp_alarm_data.data, 4);

          statsCalcRates(&tcp_protocol_ptr->frameRate, NULL,
                         rate_type, (StatsAddrEntry *)tcp_addr_ptr,
                         TCP_PROTOCOL, AL_FRAMES, time, NULL);
          tcp_protocol_ptr = tcp_protocol_ptr->link;
     }

}
```

```
void statsTcpSegRate (rate_type)
     Uint32                    rate_type;
{
     register StatsTcpSegment *tcp_seg_ptr;
     register StatsAddrEntry       *tcp_seg_addr_ptr;
     struct timeval                current_time;
     register StatsProtocolEntry   *tcp_protocol_ptr;

     tcp_seg_addr_ptr = (StatsAddrEntry *) statsTcpSegQ.pFlink;

     while (tcp_seg_addr_ptr != NULL)
     {
          tcp_seg_ptr = (StatsTcpSegment *) tcp_seg_addr_ptr->stats_ptr;
          if (tcp_seg_ptr != NULL)
          {
               /* get the current time */
               non_gettimeofday(&current_time, 0);

               statsCalcRates(&tcp_seg_ptr->frameRate, &tcp_seg_ptr->frameBuckets,
                              rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                              AL_FRAMES, current_time.tv_sec, NULL);
               statsCalcRates(&tcp_seg_ptr->byteRate, &tcp_seg_ptr->byteBuckets,
                              rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                              AL_BYTES, current_time.tv_sec, NULL);
               statsCalcRates(&tcp_seg_ptr->errorRate, &tcp_seg_ptr->errorBuckets,
                              rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                              AL_ERRORS, current_time.tv_sec, NULL);
               statsCalcRates(&tcp_seg_ptr->rcvOffSegRate, &tcp_seg_ptr->rcvOffSegBuckets,

                              rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                              AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
               statsCalcRates(&tcp_seg_ptr->xmtOffSegRate, &tcp_seg_ptr->xmtOffSegBuckets,
```

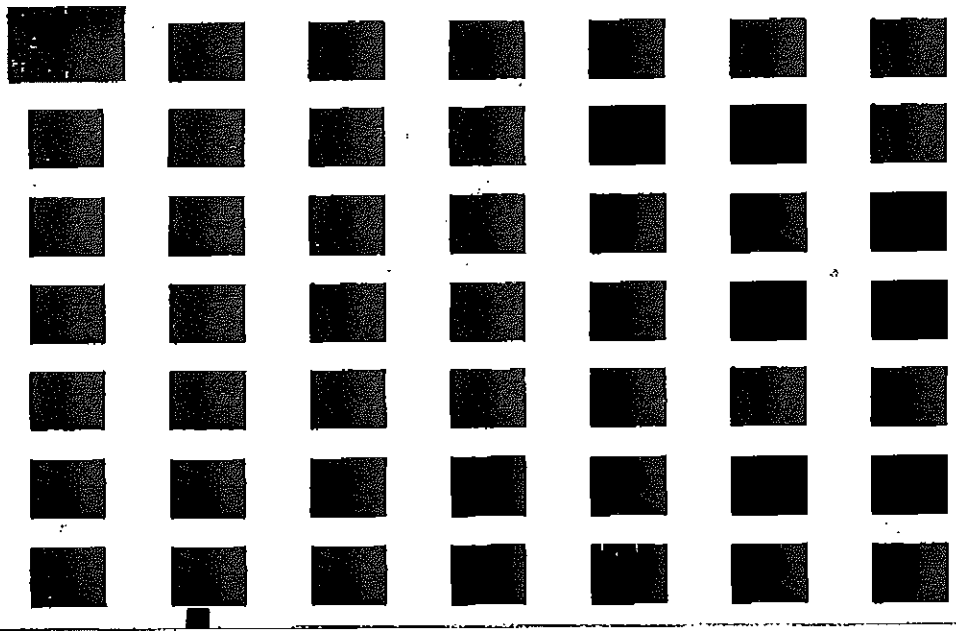
```
                                rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                                AL_XMT_OFF_SEG, current_time.tv_sec, NULL);
                statsCalcRates(&tcp_seg_ptr->transitRate, &tcp_seg_ptr->transitBuckets,
                                rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                                AL_TRANSIT, current_time.tv_sec, NULL);
                statsCalcRates(&tcp_seg_ptr->flowCtrlRate, &tcp_seg_ptr->flowCtrlBuckets,
                                rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                                AL_FLOW_CTRLS, current_time.tv_sec, NULL);
                statsCalcRates(&tcp_seg_ptr->frgmtRate, &tcp_seg_ptr->frgmtBuckets,
                                rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                                AL_FRAGMENTS, current_time.tv_sec, NULL);
                statsCalcRates(&tcp_seg_ptr->rexmtRate, &tcp_seg_ptr->rexmtBuckets,
                                rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                                AL_REXMTS, current_time.tv_sec, NULL);
                statsCalcRates(&tcp_seg_ptr->hdrByteRate, NULL,
                                rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                                AL_HDR_BYTES, current_time.tv_sec, NULL);
                statsCalcRates(&tcp_seg_ptr->rstRate, NULL,
                                rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                                AL_RST, current_time.tv_sec, NULL);
                statsCalcRates(&tcp_seg_ptr->successfulConnectionRate,
                                &tcp_seg_ptr->successfulConnectionBuckets,
                                rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                                AL_SUCCESSFUL_CONNECTIONS, current_time.tv_sec, NULL);

                statsCalcRates(&tcp_seg_ptr->connectionRetryRate,
                                &tcp_seg_ptr->connectionRetryBuckets,
                                rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
```

```
                                        AL_CONNECTION_RETRIES, current_time.tv_sec, NULL);
                statsCalcRates(&tcp_seg_ptr->failedConnectionRate,
                                        &tcp_seg_ptr->failedConnectionBuckets,
                                        rate_type, (StatsAddrEntry *)tcp_seg_addr_ptr,
TCP_SEGMENT,
                                        AL_FAILED_CONNECTIONS, current_time.tv_sec, NULL);

                /* calculate the rates on the protocolQ */
                tcp_protocol_ptr = (StatsProtocolEntry *) tcp_seg_ptr->protocolQ.pFlink;
                statsTcpProtocolRate (tcp_protocol_ptr, tcp_seg_addr_ptr,
                                                rate_type, current_time.tv_sec);

                if ((current_time.tv_sec - tcp_seg_addr_ptr->seconds_start_time) >=
monCtrl.rateTimer)
                     tcp_seg_addr_ptr->seconds_start_time = current_time.tv_sec;

                statsTcpDialogRate (rate_type);
           }
           tcp_seg_addr_ptr = (StatsAddrEntry *) tcp_seg_addr_ptr->link.pFlink;
      }
}


void statsTcpRate (tcp_common_ptr, tcp_entry_ptr, rate_type, current_time)
     StatsTcpCommon      *tcp_common_ptr;
     StatsAddrEntry      *tcp_entry_ptr;
     Uint32                   rate_type;
     struct timeval      current_time;
{
     statsCalcRates(&tcp_common_ptr->frameRate, &tcp_common_ptr->frameBuckets,
                         rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                         TCP_NODE, AL_FRAMES, current_time.tv_sec, NULL);
     statsCalcRates(&tcp_common_ptr->rcvFrameRate, NULL,
                         rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                         TCP_NODE, AL_RCV_FRAMES, current_time.tv_sec, NULL);
     statsCalcRates(&tcp_common_ptr->xmtFrameRate, NULL,
```

```
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_FRAMES, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->byteRate, &tcp_common_ptr->byteBuckets,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_BYTES, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->rcvByteRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_BYTES, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtByteRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_BYTES, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->errorRate, &tcp_common_ptr->errorBuckets,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_ERRORS, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->rcvErrorRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_ERRORS, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtErrorRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_ERRORS, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->rcvOffSegRate, &tcp_common_ptr->rcvOffSegBuckets,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtOffSegRate, &tcp_common_ptr->xmtOffSegBuckets,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_OFF_SEG, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->flowCtrlRate, &tcp_common_ptr->flowCtrlBuckets,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_FLOW_CTRLS, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->rcvHdrByteRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_HDR_BYTES, current_time.tv_sec, NULL);
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones, Mary Robertson, David M. Thompson and Gerard White

12



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
statsCalcRates(&tcp_common_ptr->xmtHdrByteRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_HDR_BYTES, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->frgmtRate, &tcp_common_ptr->frgmtBuckets,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_FRAGMENTS, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->rcvFrgmtRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_FRAGMENTS, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtFrgmtRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_FRAGMENTS, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->rexmtRate, &tcp_common_ptr->rexmtBuckets,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_REXMTS, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->rcvRexmtRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_REXMTS, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtRexmtRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_REXMTS, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->rcvRexmtByteRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_REXMTS, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtRexmtByteRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_REXMTS, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->rcvKeepAliveRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_KEEP_ALIVES, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtKeepAliveRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_KEEP_ALIVES, current_time.tv_sec, NULL);
```

```
statsCalcRates(&tcp_common_ptr->rcvWindowProbeRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_WINDOW_PROBES, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtWindowProbeRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_WINDOW_PROBES, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->rcvOutOfOrderRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_OUT_OF_ORDER, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtOutOfOrderRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_OUT_OF_ORDER, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->rcvAfterWindowRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_AFTER_WINDOW, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtAfterWindowRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_AFTER_WINDOW, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->rcvAfterWindowByteRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_AFTER_WINDOW_BYTES, current_time.tv_sec, NULL);

statsCalcRates(&tcp_common_ptr->xmtAfterWindowByteRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_AFTER_WINDOW_BYTES, current_time.tv_sec, NULL);


statsCalcRates(&tcp_common_ptr->rcvAfterCloseRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_RCV_AFTER_CLOSE, current_time.tv_sec, NULL);
statsCalcRates(&tcp_common_ptr->xmtAfterCloseRate, NULL,
                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                    TCP_NODE, AL_XMT_AFTER_CLOSE, current_time.tv_sec, NULL);
```

```
    statsCalcRates(&tcp_common_ptr->rcvUrgRate, NULL,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_RCV_URG, current_time.tv_sec, NULL);
    statsCalcRates(&tcp_common_ptr->xmtUrgRate, NULL,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_XMT_URG, current_time.tv_sec, NULL);

    statsCalcRates(&tcp_common_ptr->rcvRstRate, NULL,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_RCV_RST, current_time.tv_sec, NULL);
    statsCalcRates(&tcp_common_ptr->xmtRstRate, NULL,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_XMT_RST, current_time.tv_sec, NULL);

    statsCalcRates(&tcp_common_ptr->rcvSuccessfulConnectionRate,
                   &tcp_common_ptr->rcvSuccessfulConnectionBuckets,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_RCV_SUCCESSFUL_CONNECTIONS, current_time.tv_sec,
NULL);
    statsCalcRates(&tcp_common_ptr->xmtSuccessfulConnectionRate,
                   &tcp_common_ptr->xmtSuccessfulConnectionBuckets,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_XMT_SUCCESSFUL_CONNECTIONS, current_time.tv_sec,
NULL);
    statsCalcRates(&tcp_common_ptr->rcvConnectionRetryRate,
                   &tcp_common_ptr->rcvConnectionRetryBuckets,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_RCV_CONNECTION_RETRIES, current_time.tv_sec, NULL);

    statsCalcRates(&tcp_common_ptr->xmtConnectionRetryRate,
                   &tcp_common_ptr->xmtConnectionRetryBuckets,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_XMT_CONNECTION_RETRIES, current_time.tv_sec, NULL);
```

```
    statsCalcRates(&tcp_common_ptr->rcvFailedConnectionRate,
                   &tcp_common_ptr->rcvFailedConnectionBuckets,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_RCV_FAILED_CONNECTIONS, current_time.tv_sec, NULL);

    statsCalcRates(&tcp_common_ptr->xmtFailedConnectionRate,
                   &tcp_common_ptr->xmtFailedConnectionBuckets,
                   rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                   TCP_NODE, AL_XMT_FAILED_CONNECTIONS, current_time.tv_sec, NULL);

}

void statsTcpAddrRate (rate_type)
    Uint32                  rate_type;
{
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
     register StatsAddrEntry        *tcp_entry_ptr;
     register StatsTcpAddr          *tcp_addr_ptr;
     register StatsTcpCommon        *tcp_common_ptr;
     register Uint32                 loop_count;
     struct timeval                 current_time;
     register StatsProtocolEntry    *tcp_protocol_ptr;

     /* get the current time */
     mon_gettimeofday(&current_time, 0);

     loop_count = 0;

     /***** AGING *****/
     if ( tatsNextTcpAddrEntry != NULL)
          entry_ptr = statsNextTcpAddrEntry;      /* pick up where we left off */
     else
     {
          /* Only age structures if we've processed them all */
          entry_ptr = statsTcpAddrQ.pFlink;
          while (entry_ptr != NULL)
```

```
        {
            next_entry_ptr = entry_ptr->pFlink;
            stateTcpAgeAddr ((StatsAddrEntry *) entry_ptr);
            entry_ptr = next_entry_ptr;
        }

        entry_ptr = statsTcpAddrQ.pFlink;       /* start at the beginning */
    }
    /***** AGING *****/

    while ((entry_ptr != NULL) && (loop_count < stats_q_count))
    {
        tcp_entry_ptr = (StatsAddrEntry *)entry_ptr;
        if (((tcp_entry_ptr->em_control & rate_type) != 0)
                                && (tcp_entry_ptr->stats_ptr != NULL))

        {
            if ((current_time.tv_sec - tcp_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
            {
            tcp_addr_ptr = (StatsTcpAddr *)tcp_entry_ptr->stats_ptr;
            tcp_common_ptr = (StatsTcpCommon *) &tcp_addr_ptr->common;
            statsTcpRate (tcp_common_ptr, tcp_entry_ptr, rate_type, current_time);

            /* calculate the rates on the protocolQ */
            tcp_protocol_ptr = (StatsProtocolEntry *) tcp_addr_ptr->protocolQ.pFlink;
            statsTcpProtocolRate (tcp_protocol_ptr, tcp_entry_ptr,
                                  rate_type, current_time.tv_sec);

            tcp_entry_ptr->seconds_start_time = current_time.tv_sec;
            loop_count++;
            }
        }
        entry_ptr = entry_ptr->pFlink;
    }
    statsNextTcpAddrEntry = entry_ptr;      /* pick up where we left off */

    if (statsNextTcpAddrEntry != NULL)
```

```
        statsRatesNotDone |= STATS_TCP_ADDR_RATE;
}

void statsTcpSocketRate (rate_type)
    Uint32                      rate_type;
{
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
     register StatsAddrEntry        *tcp_entry_ptr;
     register StatsTcpSocket        *tcp_socket_ptr;
     register StatsTcpCommon        *tcp_common_ptr;
     register Uint32                     loop_count;
     struct timeval                 current_time;
     register StatsProtocolEntry    *tcp_protocol_ptr;

     /* get the current time */
     mon_gettimeofday(&current_time, 0);

     loop_count = 0;

     /***** AGING *****/
     if (statsNextTcpSocketEntry != NULL)
          entry_ptr = statsNextTcpSocketEntry;          /* pick up where we left off */
     else
     {
          /* Only age structures if we've processed them all */
          entry_ptr = statsTcpSocketQ.pFlink;
          while (entry_ptr != NULL)
          {
               next_entry_ptr = entry_ptr->pFlink;
               statsTcpAgeSocket ((StatsAddrEntry *) entry_ptr);
               entry_ptr = next_entry_ptr;
          }

          entry_ptr = statsTcpSocketQ.pFlink;          /* start at the beginning */
     }
```

```
    /***** AGING *****/

    while ((entry_ptr != NULL) && (loop_count < stats_q_count))
    {
         tcp_entry_ptr = (StatsAddrEntry *)entry_ptr;
         if (((tcp_entry_ptr->em_control & rate_type) != 0)
                              && (tcp_entry_ptr->stats_ptr != NULL))

         {
              if ((current_time.tv_sec - tcp_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
              {
              tcp_socket_ptr = (StatsTcpSocket *)tcp_entry_ptr->stats_ptr;
              tcp_common_ptr = (StatsTcpCommon *) &tcp_socket_ptr->common;
              statsTcpRate (tcp_common_ptr, tcp_entry_ptr, rate_type, current_time);

              tcp_entry_ptr->seconds_start_time = current_time.tv_sec;
              loop_count++;
              }
         }
         entry_ptr = entry_ptr->pFlink;
    }
    statsNextTcpSocketEntry = entry_ptr;          /* pick up where we left off */

    if (statsNextTcpSocketEntry != NULL)
         statsRatesNotDone |= STATS_TCP_SOCKET_RATE;
}

void statsTcpDialogRate (rate_type)
     Uint32                    rate_type;
{
     register StatsDialogEntry      *tcp_dialog_ptr;
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
     register StatsAddrEntry        *tcp_entry_ptr;
     register Uint32                      loop_count;
     struct timeval                 current_time;
```

```
    /* get the current time */
    mon_gettimeofday(&current_time, 0);

    if (statsNextTcpPairEntry != NULL)
        entry_ptr = statsNextTcpPairEntry;      /* pick up where we left off */
    else
    {
        /* Only age structures if we've processed them all */
        entry_ptr = statsTcpDialogQ.pFlink;
        while (entry_ptr != NULL)
        {
            next_entry_ptr = entry_ptr->pFlink;
            statsTcpAgeDialog ((StatsAddrEntry *) entry_ptr);
            entry_ptr = next_entry_ptr;
        }

        entry_ptr = statsTcpDialogQ.pFlink;          /* start at the beginning */
    }

    loop_count = 0;
    while ((entry_ptr != NULL) && (loop_count < stats_q_count))
    {
        tcp_entry_ptr = (StatsAddrEntry *)entry_ptr;
        if (((tcp_entry_ptr->em_control & rate_type) != 0)
                                && (tcp_entry_ptr->stats_ptr != NULL))
        {
            if ((current_time.tv_sec - tcp_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
            {
            tcp_dialog_ptr = (StatsDialogEntry *)tcp_entry_ptr->stats_ptr;

            statsCalcRates(&tcp_dialog_ptr->packetRate, NULL,
                                rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                                TCP_PAIR, AL_FRAMES, current_time.tv_sec, NULL);

            statsCalcRates(&tcp_dialog_ptr->byteRate, NULL,
                                rate_type, (StatsAddrEntry *)tcp_entry_ptr,
```

```
                                    TCP_PAIR, AL_BYTES, current_time.tv_sec, NULL);

                statsCalcRates(&tcp_dialog_ptr->errorRate, NULL,
                                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                                    TCP_PAIR, AL_ERRORS, current_time.tv_sec, NULL);

                statsCalcRates(&tcp_dialog_ptr->fragmentRate, NULL,
                                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                                    TCP_PAIR, AL_FRAGMENTS, current_time.tv_sec, NULL);

                statsCalcRates(&tcp_dialog_ptr->rexmtRate, NULL,
                                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                                    TCP_PAIR, AL_REXMTS, current_time.tv_sec, NULL);

                statsCalcRates(&tcp_dialog_ptr->flowCtrlRate, NULL,
                                    rate_type, (StatsAddrEntry *)tcp_entry_ptr,
                                    TCP_PAIR, AL_FLOW_CTRLS, current_time.tv_sec, NULL);

                tcp_entry_ptr->seconds_start_time = current_time.tv_sec;
                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }
        statsNextTcpPairEntry = entry_ptr;

        if (statsNextTcpPairEntry != NULL)
            statsRatesNotDone |= STATS_TCP_PAIR_RATE;
    }
```

```
/*
 *  stats_tcp_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                   All rights reserved.
 *
 *
 *  Path:       /home/hawk1/malibu/trakker_db/monitor/stats/SCCS/s.stats_tcp_p.c
 *
 *  Date:       6/19/91
 *
 *  Revision:   1.7
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO        Description of change. (latest first)
 *   --------  ---        ---------------------------------------------
 *
 *   06-14-91  KR         Fixed set of application protocol in dialog
 *   06-01-91  DPD        Set rate types that were missing in seg and addr structures
 *
 */

static char stats_tcp_p_c [] = "@(#)stats_tcp_p.c 1.7";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
#include "mib_defs.h"
#include "mib_ip.h"
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "protocols.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_tcp.h"
#include "stats_mib.h"
#include "mib_tcp.h"
#include "snmpd.h"
#include "em_ctrl.h"


/*
 * Global Data Structures
 */

StatsAddrEntry          *tcp_this_seg_addr_ptr;
StatsTcpSegment              *tcp_this_seg_stats_ptr;
StatsProtocolEntry      *tcp_this_seg_protocol_ptr;

StatsAddrEntry          *tcp_src_seg_addr_ptr, *tcp_dst_seg_addr_ptr;
StatsTcpSegment              *tcp_src_seg_stats_ptr, *tcp_dst_seg_stats_ptr;
StatsProtocolEntry      *tcp_src_seg_protocol_ptr, *tcp_dst_seg_protocol_ptr;

StatsAddrEntry          *tcp_src_node_addr_ptr, *tcp_dst_node_addr_ptr;
StatsTcpAddr            *tcp_src_node_stats_ptr, *tcp_dst_node_stats_ptr;
```

```
StatsAddrEntry          *tcp_src_socket_addr_ptr, *tcp_dst_socket_addr_ptr;
StatsTcpSocket          *tcp_src_socket_stats_ptr, *tcp_dst_socket_stats_ptr;

StatsProtocolEntry       *tcp_src_node_protocol_ptr, *tcp_dst_node_protocol_ptr;

StatsAddrEntry           *tcp_dialog_addr_ptr;
StatsDialogEntry         *tcp_dialog_stats_ptr;

TcpConnectionStateType   *tcp_connection_state_ptr;
TcpSocketStateType       *tcp_src_state_ptr;
TcpSocketStateType       *tcp_dst_state_ptr;


/*
 * Local data structures
 */

 StatsAddrEntry  *tcp_hash_table[TCP_HASH_TABLE_SIZE];
 Uint32          tcp_hash;
 StatsAddrEntry  *tcp_hash_link;
 StatsAddrEntry  *tcp_previous_hash_link;

 StatsAddrEntry  *tcp_socket_hash_table[TCP_SOCKET_HASH_TABLE_SIZE];
 Uint32          tcp_socket_hash;
 StatsAddrEntry  *tcp_socket_hash_link;
 StatsAddrEntry  *tcp_previous_socket_hash_link;

 StatsAddrEntry  *tcp_dialog_hash_table[TCP_DIALOG_HASH_TABLE_SIZE];
 Uint32          tcp_dialog_hash;
 StatsAddrEntry  *tcp_dialog_hash_link;
 StatsAddrEntry  *tcp_previous_dialog_hash_link;


/*****************************************************************************
```

```
 *
 * Look for the segment address structure.
 * If no match is found, NULL is returned.
 */

StatsAddrEntry *stats_tcp_lookup_segment (segment)

    Uint32                      segment;
{
    register StatsAddrEntry       *seg_addr_ptr;

seg_addr_ptr = (StatsAddrEntry *) statsTcpSegQ.pFlink;

while (seg_addr_ptr != NULL)
    {
    if (seg_addr_ptr->address.segment1 == segment)
        break;
    seg_addr_ptr = (StatsAddrEntry *) seg_addr_ptr->link.pFlink;
    }

    return (seg_addr_ptr);
}


/*****************************************************************************
 *
 * Find the structure for keeping Tcp segment statistics for the given
 * segment.  If one is not found, attempt to allocate one.
 */

StatsAddrEntry *stats_tcp_get_segment (segment)

    Uint32          segment;

{
    register StatsAddrEntry       *seg_addr_ptr;
```

```
    register StatsTcpSegment *seg_stats_ptr;

 seg_addr_ptr = stats_tcp_lookup_segment (segment);

'*
 * If not found, try to allocate a structure for this segment.
 * If a structure can't be obtained, count this as a drop.
 */
 if (seg_addr_ptr == NULL)
     {
     seg_addr_ptr = (StatsAddrEntry *) stats_allocate ( sizeof(StatsAddrEntry) );
     if (seg_addr_ptr != NULL)
          {
          seg_addr_ptr->address.addressType = MibSegment1;
          seg_addr_ptr->address.segment1 = segment;
          seg_addr_ptr->parse_control = monCtrl.segParseCtrl;
          seg_addr_ptr->startTime = stats_start_time.tv_sec;
          seg_addr_ptr->lastTime = stats_start_time.tv_sec;

          FBInsqt (&statsTcpSegQ, (PFBQentry_type) seg_addr_ptr);
          }
     else
          {
          stats_mon_tcp_dropped;
          return (NULL);
          }
     }

/*
 * There's an address structure. See if there's a statistics structure
 * for Tcp attached, if not and parse control allows, allocate one.
 */
seg_stats_ptr = (StatsTcpSegment *) seg_addr_ptr->stats_ptr;
if (seg_stats_ptr == NULL)
     {
     /*
      * If Tcp parsing is enabled, allocate Tcp statistics structure if
```

```
 * one has not already been allocated.  If a structure can't be obtained,
 * count this as a drop.
 */
if (seg_addr_ptr->parse_control & MibParseTcp)
     {
     seg_stats_ptr = (StatsTcpSegment *) stats_allocate (sizeof (StatsTcpSegment) );

     if (seg_stats_ptr != NULL)
          {
          seg_addr_ptr->stats_ptr = (Uint32 *) seg_stats_ptr;
          seg_stats_ptr->frameRate.type                    = STATS_RATE_10S;
          seg_stats_ptr->byteRate.type                     = STATS_RATE_10S;
          seg_stats_ptr->errorRate.type                    = STATS_RATE_10S;
          seg_stats_ptr->rcvOffSegRate.type                = STATS_RATE_10S;
          seg_stats_ptr->xmtOffSegRate.type                = STATS_RATE_10S;
          seg_stats_ptr->transitRate.type                      = STATS_RATE_10S;
          seg_stats_ptr->flowCtrlRate.type                 = STATS_RATE_10S;
          seg_stats_ptr->frgmtRate.type                    = STATS_RATE_10S;
          seg_stats_ptr->hdrByteRate.type                      = STATS_RATE_10S;
          seg_stats_ptr->rexmtRate.type                    = STATS_RATE_10S;
          seg_stats_ptr->keepAliveRate.type                = STATS_RATE_10S;
          seg_stats_ptr->windowProbeRate.type                  = STATS_RATE_10S;
          seg_stats_ptr->outOfOrderRate.type               = STATS_RATE_10S;
          seg_stats_ptr->afterWindowRate.type                  = STATS_RATE_10S;
          seg_stats_ptr->afterCloseRate.type               = STATS_RATE_10S;
          seg_stats_ptr->urgRate.type                          = STATS_RATE_10S;
          seg_stats_ptr->rstRate.type                          = STATS_RATE_10S;
          seg_stats_ptr->successfulConnectionRate.type = STATS_RATE_HOUR;
          seg_stats_ptr->connectionRetryRate.type          = STATS_RATE_HOUR;
          seg_stats_ptr->failedConnectionRate.type         = STATS_RATE_HOUR;

          setTcpSegDflts (&mibSegDefaults.mibTcpSegDefaults, seg_stats_ptr);
          }
     else
          stats_mon_tcp_dropped;
     }
}
```

```
return (seg_addr_ptr);
}



/*****************************************************************************
 *
 *  Find the tcp ip address record.
 *  A hash is done on the ip address.  A pointer to the first
 *  StatsAddrEntry with the same hash is found.  Structures with the
 *  same hash are linked.  The link must be walked and the ip address
 *  compared with the those in each of the structures until a match
 *  is found.  If no match is found, NULL is returned.
 */
StatsAddrEntry *stats_tcp_lookup_addr (ip_addr)

     Uint32          *ip_addr;
{
     Uint32          i;

/*
 * Compute ip address hash to get index into hash table
 */
tcp_hash = (*ip_addr & 0x0000ffff);
tcp_hash = ((tcp_hash + ((tcp_hash & 0xff00) >> 8)) & (TCP_HASH_TABLE_SIZE - 1));

tcp_previous_hash_link = NULL;
tcp_hash_link = (StatsAddrEntry *) tcp_hash_table[tcp_hash];

/*
 * Walk linked list for exact entry
 */
while (tcp_hash_link != NULL)
     {
     if (tcp_hash_link->address.netAddress1.u.ipAddress == *ip_addr)
```

```
            return (tcp_hash_link);
        else
            {
            tcp_previous_hash_link = tcp_hash_link;
            tcp_hash_link = tcp_hash_link->hash_link;
            }
        }

    /*
     * No entry found.
     */
    return (NULL);
    }



    /*****************************************************************************
     *
     * Allocate a stats structure if parse control is turned on.
     *
     */

    Uint32 stats_tcp_get_stats (tcp_addr_record_ptr)
        StatsAddrEntry      *tcp_addr_record_ptr;
    {
        register StatsTcpAddr    *tcp_addr_stats_ptr;

        if ((tcp_addr_record_ptr != NULL) && (tcp_addr_record_ptr->stats_ptr == NULL))
        {
            /*
             * If parse control is turned on for Tcp, allocate a structure for Tcp
             * address statistics and initialize it.  If parse control is turned on,
             * but a structure can't be obtained, return FALSE
             */
            if (tcp_addr_record_ptr->parse_control & MibParseTcp)
```

```
        {
                tcp_addr_record_ptr->stats_ptr = (Uint32 *) stats_allocate (sizeof
(StatsTcpAddr) );
                tcp_addr_stats_ptr = (StatsTcpAddr *) tcp_addr_record_ptr->stats_ptr;
                if (tcp_addr_stats_ptr != NULL)
                {
                tcp_addr_stats_ptr->common.frameRate.type                    =
STATS_RATE_10S;
                tcp_addr_stats_ptr->common.rcvFrameRate.type                 =
STATS_RATE_10S;
                tcp_addr_stats_ptr->common.xmtFrameRate.type                 =
STATS_RATE_10S;
                tcp_addr_stats_ptr->common.byteRate.type
= STATS_RATE_10S;
                tcp_addr_stats_ptr->common.rcvByteRate.type
= STATS_RATE_10S;
                tcp_addr_stats_ptr->common.xmtByteRate.type
= STATS_RATE_10S;
                tcp_addr_stats_ptr->common.errorRate.type
= STATS_RATE_10S;
                tcp_addr_stats_ptr->common.rcvErrorRate.type                 =
STATS_RATE_10S;
                tcp_addr_stats_ptr->common.xmtErrorRate.type                 =
STATS_RATE_10S;
                tcp_addr_stats_ptr->common.rcvOffSegRate.type
= STATS_RATE_10S;
                tcp_addr_stats_ptr->common.xmtOffSegRate.type
= STATS_RATE_10S;
                tcp_addr_stats_ptr->common.rcvHdrByteRate.type
= STATS_RATE_10S;
                tcp_addr_stats_ptr->common.xmtHdrByteRate.type
= STATS_RATE_10S;
                tcp_addr_stats_ptr->common.flowCtrlRate.type                 =
STATS_RATE_10S;
                tcp_addr_stats_ptr->common.frgmtRate.type
= STATS_RATE_10S;
```

```
                    tcp_addr_stats_ptr->common.rcvFrgmtRate.type                =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.xmtFrgmtRate.type                =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.rexmtRate.type
= STATS_RATE_10S;   tcp_addr_stats_ptr->common.rcvRexmtRate.type                =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.xmtRexmtRate.type                =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.rcvRexmtByteRate.type            =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.xmtRexmtByteRate.type            =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.rcvKeepAliveRate.type            =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.xmtKeepAliveRate.type            =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.rcvWindowProbeRate.type
= STATS_RATE_10S;   tcp_addr_stats_ptr->common.xmtWindowProbeRate.type
= STATS_RATE_10S;   tcp_addr_stats_ptr->common.rcvOutOfOrderRate.type           =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.xmtOutOfOrderRate.type           =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.rcvAfterWindowRate.type
= STATS_RATE_10S;   tcp_addr_stats_ptr->common.xmtAfterWindowRate.type
= STATS_RATE_10S;   tcp_addr_stats_ptr->common.rcvAfterWindowByteRate.type      =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.xmtAfterWindowByteRate.type      =
STATS_RATE_10S;     tcp_addr_stats_ptr->common.rcvAfterCloseRate.type           =
STATS_RATE_10S;
```

```
                    tcp_addr_stats_ptr->common.xmtAfterCloseRate.type                =
STATS_RATE_10S;
                    tcp_addr_stats_ptr->common.rcvUrgRate.type
= STATS_RATE_10S;
                    tcp_addr_stats_ptr->common.xmtUrgRate.type
= STATS_RATE_10S;
                    tcp_addr_stats_ptr->common.rcvRstRate.type
= STATS_RATE_10S;
                    tcp_addr_stats_ptr->common.xmtRstRate.type
= STATS_RATE_10S;
                    tcp_addr_stats_ptr->common.rcvSuccessfulConnectionRate.type      =
STATS_RATE_HOUR;
                    tcp_addr_stats_ptr->common.xmtSuccessfulConnectionRate.type      =
STATS_RATE_HOUR;
                    tcp_addr_stats_ptr->common.rcvConnectionRetryRate.type           =
STATS_RATE_HOUR;
                    tcp_addr_stats_ptr->common.xmtConnectionRetryRate.type           =
STATS_RATE_HOUR;
                    tcp_addr_stats_ptr->common.rcvFailedConnectionRate.type
= STATS_RATE_HOUR;
                    tcp_addr_stats_ptr->common.xmtFailedConnectionRate.type
= STATS_RATE_HOUR;

                    Initqh   (&tcp_addr_stats_ptr->protocolQ);
                    FBInitqh (&tcp_addr_stats_ptr->dialogQ);

                    setTcpAddrDflts (&mibNodeDefaults.mibTcpAddrDefaults,
&tcp_addr_stats_ptr->common);
               }
               else
                    return (FALSE);
          }
     }
     return (TRUE);
}
```

```
/*****************************************************************************
 *
 * Allocate a socket stats structure if parse control is turned on.
 *
 */

Uint32 stats_tcp_get_socket_stats (tcp_addr_record_ptr)
    StatsAddrEntry     *tcp_addr_record_ptr;
{
    register StatsTcpSocket       *tcp_socket_stats_ptr;

    if ((tcp_addr_record_ptr != NULL) && (tcp_addr_record_ptr->stats_ptr == NULL))
    {
         /*
          * If parse control is turned on for Tcp, allocate a structure for Tcp
          * socket statistics and initialize it.  If parse control is turned on,
          * but a structure can't be obtained, return FALSE
          */
         if (tcp_addr_record_ptr->parse_control & MibParseTcp)
         {
              tcp_addr_record_ptr->stats_ptr = (Uint32 *) stats_allocate (sizeof
(StatsTcpSocket) );
              tcp_socket_stats_ptr = (StatsTcpSocket *) tcp_addr_record_ptr->stats_ptr;
              if (tcp_socket_stats_ptr != NULL)
              {
              tcp_socket_stats_ptr->common.frameRate.type                    =
STATS_RATE_10S;
                   tcp_socket_stats_ptr->common.rcvFrameRate.type
= STATS_RATE_10S;
                   tcp_socket_stats_ptr->common.xmtFrameRate.type
= STATS_RATE_10S;
                   tcp_socket_stats_ptr->common.byteRate.type
= STATS_RATE_10S;
                   tcp_socket_stats_ptr->common.rcvByteRate.type
= STATS_RATE_10S;
                   tcp_socket_stats_ptr->common.xmtByteRate.type
= STATS_RATE_10S;
```

```
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.errorRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.rcvErrorRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.xmtErrorRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.rcvOffSegRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.xmtOffSegRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.rcvHdrByteRate.type          =
STATS_RATE_10S;          tcp_socket_stats_ptr->common.xmtHdrByteRate.type          =
STATS_RATE_10S;          tcp_socket_stats_ptr->common.rcvFrgmtRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.xmtFrgmtRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.rcvRexmtRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.xmtRexmtRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.rcvRexmtByteRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.xmtRexmtByteRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.rcvKeepAliveRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.xmtKeepAliveRate.type
= STATS_RATE_10S;        tcp_socket_stats_ptr->common.rcvWindowProbeRate.type      =
STATS_RATE_10S;          tcp_socket_stats_ptr->common.xmtWindowProbeRate.type      =
STATS_RATE_10S;          tcp_socket_stats_ptr->common.rcvOutOfOrderRate.type
= STATS_RATE_10S;
```

```
                    tcp_socket_stats_ptr->common.xmtOutOfOrderRate.type
= STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.rcvAfterWindowRate.type          =
STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.xmtAfterWindowRate.type          =
STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.rcvAfterWindowByteRate.type      =
STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.xmtAfterWindowByteRate.type      =
STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.rcvAfterCloseRate.type
= STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.xmtAfterCloseRate.type
= STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.rcvUrgRate.type                  =
STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.xmtUrgRate.type                  =
STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.rcvRstRate.type                  =
STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.xmtRstRate.type                  =
STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.rcvSuccessfulConnectionRate.type =
STATS_RATE_10S;
                    tcp_socket_stats_ptr->common.xmtSuccessfulConnectionRate.type =
STATS_RATE_HOUR;
                    tcp_socket_stats_ptr->common.rcvConneotionRetryRate.type      =
STATS_RATE_HOUR;
                    tcp_socket_stats_ptr->common.xmtConnectionRetryRate.type      =
STATS_RATE_HOUR;
                    tcp_socket_stats_ptr->common.rcvFailedConnectionRate.type     =
STATS_RATE_HOUR;
                    tcp_socket_stats_ptr->common.xmtFailedConnectionRate.type     =
STATS_RATE_HOUR;

                    FBInitqh (&tcp_socket_stats_ptr->dialogQ);
```

```
                    setTcpAddrDflts (&nibNodeDefaults.mibTcpAddrDefaults,
&tcp_socket_stats_ptr->common);
               }
               else
                    return (FALSE);
          }
     }
     return (TRUE);
}



/************************************************************************
 *
 * Find the structure for keeping tcp address statistics for the given
 * ip address.  If one not found, attempt to allocate one.
 */

StatsAddrEntry *stats_tcp_get_addr (ip_addr)

     Uint32              *ip_addr;
{
     register StatsAddrEntry       *tcp_addr_record_ptr;
     register StatsTcpAddr         *tcp_addr_stats_ptr;
     register StatsAddrEntry       *ip_addr_record_ptr;


tcp_addr_record_ptr = stats_tcp_lookup_addr (ip_addr);


if (tcp_addr_record_ptr == NULL)
     {
     /*
      * Try to allocate a statistics structure for this address.
      * If tcp is turned on, but a structure can't be obtained,
      * count this as a drop.  If tcp is turned off, just return NULL.
```

```
     */
    ip_addr_record_ptr = stats_ip_lookup_addr (ip_addr);
    if (ip_addr_record_ptr == NULL)
        return (NULL);

    if ((ip_addr_record_ptr->parse_control & MibParseTcp) != MibParseTcp)
        return (NULL);

    tcp_addr_record_ptr = (StatsAddrEntry *)stats_allocate (sizeof (StatsAddrEntry) );
    if (tcp_addr_record_ptr == NULL)
        {
        stats_mon_tcp_dropped;
        return (NULL);
        }
    else
        {
        tcp_addr_record_ptr->hash_link                                  = NULL;
        tcp_addr_record_ptr->startTime                                  =
stats_start_time.tv_sec;
        tcp_addr_record_ptr->lastTime                                   =
stats_start_time.tv_sec;
        tcp_addr_record_ptr->address.addressType                        = MibNetAddress1 |
MibSegment1;
        tcp_addr_record_ptr->address.netAddress1.netAddressType = NetTcpIp;
        tcp_addr_record_ptr->address.netAddress1.length         = 4;
        tcp_addr_record_ptr->address.netAddress1.u.ipAddress    = *ip_addr;

        tcp_addr_record_ptr->parse_control = ip_addr_record_ptr->parse_control;
        tcp_addr_record_ptr->address.segment1 = ip_addr_record_ptr->address.segment1;

        FBInsqt (&statsTcpAddrQ, (PFBQentry_type) tcp_addr_record_ptr);

        /*
         * Allocate a stats structure if parse control is turned on.
         * stats_tcp_get_stats will return FALSE, so count this as a drop.
         */
        if (stats_tcp_get_stats (tcp_addr_record_ptr) == FALSE)
```

```
                stats_mon_tcp_dropped;
            }

        /*
         * Put this tcp address record into the hash table.
         * Variable hash was set when stats_tcp_lookup_addr was executed.
         */
        if ( tcp_hash_table[tcp_hash] == NULL)
            tcp_hash_table[tcp_hash] = tcp_addr_record_ptr;
        else
            {
            /*
             * Find the last structure of the hash link
             */
            tcp_hash_link = tcp_hash_table[tcp_hash];

            while (tcp_hash_link->hash_link != NULL)
                tcp_hash_link = tcp_hash_link->hash_link;

            tcp_hash_link->hash_link = tcp_addr_record_ptr;
            }
        }

    return (tcp_addr_record_ptr);
    }



/*****************************************************************************
 *
 *  Find the tcp socket record.
 *  A hash is done on the ip address and port number.  A pointer to the first
 *  StatsAddrEntry with the same hash is found.  Structures with the
 *  same hash are linked.  The link must be walked and the ip address and port
 *  compared with the those in each of the structures until a match
 *  is found.  If no match is found, NULL is returned.
```

```
 */
StatsAddrEntry *stats_tcp_lookup_socket (ip_addr, port)

     Uint32          *ip_addr;
     Uint32          port;
{
     Uint32          i;

/*
 * Compute ip address hash to get index into hash table
 */
tcp_socket_hash = (*ip_addr & 0x0000ffff) + ((port & 0x0000ffff) << 16);
tcp_socket_hash = ((tcp_socket_hash + ((tcp_socket_hash & 0xff00) >> 8)) &
                   (TCP_SOCKET_HASH_TABLE_SIZE - 1));

tcp_previous_socket_hash_link = NULL;
tcp_socket_hash_link = (StatsAddrEntry *) tcp_socket_hash_table[tcp_socket_hash];

/*
 * Walk linked list for exact entry
 */
while (tcp_socket_hash_link != NULL)
     {
     if ((tcp_socket_hash_link->address.port1 == port) &&
          (tcp_socket_hash_link->address.netAddress1.u.ipAddress == *ip_addr) )
          return (tcp_socket_hash_link);
     else
          {
          tcp_previous_socket_hash_link = tcp_socket_hash_link;
          tcp_socket_hash_link = tcp_socket_hash_link->hash_link;
          }
     }


/*
 * No entry found.
 */
```

```
return (NULL);
}


/*****************************************************************************
 *
 * Find the structure for keeping tcp statistics for the given
 * ip address and port.  If one not found, attempt to allocate one.
 */

StatsAddrEntry *stats_tcp_get_socket (ip_addr, port)

    Uint32              *ip_addr;
    Uint32              port;
{
    register StatsAddrEntry       *tcp_socket_addr_ptr;
    register StatsTcpAddr         *tcp_addr_stats_ptr;
    register StatsAddrEntry       *ip_addr_record_ptr;

/*
 * Don't bother keeping stats for ports that aren't well-known yet.
 */
 if (!stats_well_known_port (port) )
    return (NULL);

 tcp_socket_addr_ptr = stats_tcp_lookup_socket (ip_addr, port);


 if (tcp_socket_addr_ptr == NULL)
    {
    /*
     * Try to allocate a statistics structure for this address.
     * If tcp is turned on for this ip address, but a structure can't
     * be obtained, count this as a drop.  If tcp is not turned on, just
     * return NULL.
     */
```

```
        ip_addr_record_ptr = stats_ip_lookup_addr (ip_addr);
        if (ip_addr_record_ptr == NULL)
             return (NULL);
        if ((ip_addr_record_ptr->parse_control & MibParseTcp) != MibParseTcp)
             return (NULL);

        tcp_socket_addr_ptr = (StatsAddrEntry *)stats_allocate (sizeof (StatsAddrEntry) );
        if (tcp_socket_addr_ptr == NULL)
             {
             stats_mon_tcp_dropped;
             return (NULL);
             }
        else
             {
             tcp_socket_addr_ptr->hash_link = NULL;
             tcp_socket_addr_ptr->startTime = stats_start_time.tv_sec;
             tcp_socket_addr_ptr->lastTime = stats_start_time.tv_sec;

             tcp_socket_addr_ptr->address.addressType = MibNetAddress1 | MibSegment1 |
MibPort1;
             tcp_socket_addr_ptr->address.netAddress1.netAddressType = NetTcpIp;
             tcp_socket_addr_ptr->address.netAddress1.length = 4;
             tcp_socket_addr_ptr->address.netAddress1.u.ipAddress = *ip_addr;
             tcp_socket_addr_ptr->address.port1 = port;

             tcp_socket_addr_ptr->parse_control = ip_addr_record_ptr->parse_control;
             tcp_socket_addr_ptr->address.segment1 = ip_addr_record_ptr->address.segment1;

             FBInsqt (&statsTcpSocketQ, (PFBQentry_type) tcp_socket_addr_ptr);

           /*
              * Allocate a stats structure if parse control is turned on.
              * stats_tcp_get_socket_stats will return FALSE, so count this as a drop.
              */
              if (stats_tcp_get_stats (tcp_socket_addr_ptr) == FALSE)
                stats_mon_tcp_dropped;
           }
```

```
{
     register Uint32              xtcp_dialog_hash;
     register StatsAddrEntry      *xtcp_dialog_hash_link;
     register StatsAddrEntry      *xtcp_previous_dialog_hash_link;

     register StatsAddrEntry      *dialog_addr_ptr;
/*
 * Compute hash function on both sockets
 */
xtcp_dialog_hash = (*ip_addr1 & 0x0000ffff) + ((port1 & 0x0000ffff) << 16) +
                   (*ip_addr2 & 0x0000ffff) + ((port2 & 0x0000ffff) << 16);
xtcp_dialog_hash = ((xtcp_dialog_hash + ((xtcp_dialog_hash & 0xff00) >> 8))
                    & (TCP_DIALOG_HASH_TABLE_SIZE - 1));

xtcp_previous_dialog_hash_link = NULL;
xtcp_dialog_hash_link = (StatsAddrEntry *) tcp_dialog_hash_table[xtcp_dialog_hash];

/*
 * Walk linked list for exact entry
 */
while (xtcp_dialog_hash_link != NULL)
     {
     if (((xtcp_dialog_hash_link->address.port1 == port1) &&
          (xtcp_dialog_hash_link->address.port2 == port2) &&
          (xtcp_dialog_hash_link->address.netAddress1.u.ipAddress == *ip_addr1) &&
          (xtcp_dialog_hash_link->address.netAddress2.u.ipAddress == *ip_addr2) ) ||

         ((xtcp_dialog_hash_link->address.port1 == port2) &&
          (xtcp_dialog_hash_link->address.port2 == port1) &&
          (xtcp_dialog_hash_link->address.netAddress1.u.ipAddress == *ip_addr2) &&
          (xtcp_dialog_hash_link->address.netAddress2.u.ipAddress == *ip_addr1) ))
              {
              break;
              }
         else
              {
              xtcp_previous_dialog_hash_link = xtcp_dialog_hash_link;
```

```
                xtcp_dialog_hash_link = xtcp_dialog_hash_link->hash_link;
                }
        }

tcp_dialog_hash = xtcp_dialog_hash;
tcp_dialog_hash_link = xtcp_dialog_hash_link;
tcp_previous_dialog_hash_link = xtcp_previous_dialog_hash_link;

return (xtcp_dialog_hash_link);
}


/************************ stats_tcp_get_dialog ******************************
 *
 * Find or allocate an tcp dialog given 2 sockets.
 * If the dialog is not found, attempt to allocate a structure.
 */
StatsAddrEntry *stats_tcp_get_dialog (ip_addr1, port1, ip_addr2, port2)

        Uint32                  *ip_addr1, *ip_addr2;
        Uint32                  port1, port2;
{
        register StatsAddrEntry         *dialog_addr_ptr;
        register StatsDialogEntry       *dialog_stats_ptr;
        register StatsDialogLink *dialog_link;
        register StatsAddrEntry         *tcp_addr1_record_ptr, *tcp_addr2_record_ptr;
        register StatsTcpAddr           *tcp_addr1_stats_ptr, *tcp_addr2_stats_ptr;
        register StatsAddrEntry         *tcp_socket1_record_ptr, *tcp_socket2_record_ptr;
        register StatsTcpSocket  *tcp_socket1_stats_ptr, *tcp_socket2_stats_ptr;
        register StatsAddrEntry         *ip_addr1_record_ptr, *ip_addr2_record_ptr;
        register Uint32                 parse_control;


dialog_addr_ptr = stats_tcp_lookup_dialog (ip_addr1, port1, ip_addr2, port2);
if (dialog_addr_ptr != NULL)
        return (dialog_addr_ptr);
```

```
/*
 * Check parse control for the ip addresses.
 */
ip_addr1_record_ptr = stats_ip_lookup_addr (ip_addr1);
ip_addr2_record_ptr = stats_ip_lookup_addr (ip_addr2);

if ( (ip_addr1_record_ptr == NULL) && (ip_addr2_record_ptr == NULL) )
     return (NULL);

if (ip_addr1_record_ptr != NULL)
     parse_control = ip_addr1_record_ptr->parse_control;
if (ip_addr2_record_ptr != NULL)
     parse_control |= ip_addr2_record_ptr->parse_control;


/*
 * If tcp is turned on for these ip addresses, allocate structures
 * for the dialog.  If can't get them, count this as a drop.
 */
if ((parse_control & MibParseTcp) != MibParseTcp)
     return (NULL);

/*
 * Try to allocate structures for this dialog.
 * If they can't be obtained, count this as a drop.
 */
dialog_addr_ptr = (StatsAddrEntry *) stats_allocate (sizeof (StatsAddrEntry) );
if (dialog_addr_ptr == NULL)
     {
     stats_mon_tcp_dropped;
     return (NULL);
     }

dialog_stats_ptr = (StatsDialogEntry *) stats_allocate (sizeof (StatsDialogEntry) );
if (dialog_stats_ptr == NULL)
     {
```

```
      stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );
      stats_mon_tcp_dropped;
      return (NULL);
      }

/*
 * Try to get a structure to keep state information in
 * and attach it to the statistics.
 */
dialog_stats_ptr->transport.state_ptr =
          (UInt32 *) stats_allocate (sizeof(TcpConnectionStateType));
if (dialog_stats_ptr->transport.state_ptr == NULL)
      {
      stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );
      stats_deallocate (dialog_stats_ptr, sizeof(StatsDialogEntry) );
      stats_mon_tcp_dropped;
      return (NULL);
      }


/*
 * Initialize the structures
 */
dialog_addr_ptr->hash_link                                    = NULL;
dialog_addr_ptr->address.addressType                  = MibNetAddress1 | MibPort1 |
                                                                MibNetAddress2 |
MibPort2;
dialog_addr_ptr->address.netAddress1.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress1.length         = 4;
dialog_addr_ptr->address.netAddress1.u.ipAddress    = *ip_addr1;
dialog_addr_ptr->address.port1                                = port1;
dialog_addr_ptr->address.netAddress2.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress2.length         = 4;
dialog_addr_ptr->address.netAddress2.u.ipAddress    = *ip_addr2;
dialog_addr_ptr->address.port2                                = port2;
dialog_addr_ptr->parse_control                                = parse_control;
dialog_addr_ptr->startTime                                    = stats_start_time.tv_sec;
```

```
dialog_addr_ptr->lastTime                                   = stats_start_time.tv_sec;
dialog_addr_ptr->stats_ptr                                  = (Uint32 *) dialog_stats_ptr;


FBInsqt (&statsTcpDialogQ, (PFBQentry_type) dialog_addr_ptr);

dialog_stats_ptr->packetRate.type  = STATS_RATE_10S;
dialog_stats_ptr->byteRate.type    = STATS_RATE_10S;
dialog_stats_ptr->errorRate.type   = STATS_RATE_10S;
dialog_stats_ptr->fragmentRate.type = STATS_RATE_10S;
dialog_stats_ptr->rexmtRate.type   = STATS_RATE_10S;
dialog_stats_ptr->flowCtrlRate.type = STATS_RATE_10S;

dialog_stats_ptr->transport.transportProtocol = TCP_PROTOCOL;
dialog_stats_ptr->transport.applicationProtocol = tcp_protocol;
dialog_stats_ptr->transport.initiator = ConnectionInitiatorUnknown;
dialog_stats_ptr->transport.state = ConnectionStateUnknown;
dialog_stats_ptr->transport.closeReason = ConnectionCloseUnknown;

((TcpConnectionStateType *) dialog_stats_ptr->transport.state_ptr)->tcp1.indicator = 1;
((TcpConnectionStateType *) dialog_stats_ptr->transport.state_ptr)->tcp2.indicator = 2;

setTcpAddrPairDflts (&mibNodeDefaults.mibTcpAddrPairDefaults, dialog_stats_ptr);

/*
 * Link the dialog statistics into the dialog queue for each tcp address stats.
 */
tcp_addr1_record_ptr = stats_tcp_lookup_addr (ip_addr1);
tcp_addr2_record_ptr = stats_tcp_lookup_addr (ip_addr2);

if (tcp_addr1_record_ptr != NULL)
     {
     if (tcp_addr1_record_ptr->stats_ptr != NULL)
          {
          dialog_addr_ptr->address.addressType|= MibSegment1;
          dialog_addr_ptr->address.segment1  = tcp_addr1_record_ptr->address.segment1;
```

```
          tcp_addr1_stats_ptr                         = (StatsTcpAddr *)
tcp_addr1_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&tcp_addr1_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
               }
          }
     }
if (tcp_addr2_record_ptr != NULL)
     {
     if (tcp_addr2_record_ptr->stats_ptr != NULL)
          {
          dialog_addr_ptr->address.addressType = MibSegment2;
          dialog_addr_ptr->address.segment2  = tcp_addr2_record_ptr->address.segment1;
          tcp_addr2_stats_ptr                         = (StatsTcpAddr *)
tcp_addr2_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&tcp_addr2_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
               }
          }
     }


/*
 * Link the dialog statistics into the dialog queue for each of
 * the sockets.
 */
tcp_socket1_record_ptr = stats_tcp_lookup_socket (ip_addr1, port1);
tcp_socket2_record_ptr = stats_tcp_lookup_socket (ip_addr2, port2);

if (tcp_socket1_record_ptr != NULL)
     {
```

```
     if (tcp_socket1_record_ptr->stats_ptr != NULL)
          {
          tcp_socket1_stats_ptr =(StatsTcpSocket*)tcp_socket1_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&tcp_socket1_stats_ptr->dialogQ,(PFBQentry_type)dialog_link);
               }
          }
     }
if (tcp_socket2_record_ptr != NULL)
     {
     if (tcp_socket2_record_ptr->stats_ptr != NULL)
          {
          tcp_socket2_stats_ptr =(StatsTcpSocket*)tcp_socket2_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&tcp_socket2_stats_ptr->dialogQ,(PFBQentry_type)dialog_link);
               }
          }
     }

/*
 * Put the new dialog address structure in the tcp dialog hash table.
 * Variable hash was set up when stats_tcp_lookup_dialog was executed.
 */
if (tcp_dialog_hash_table[tcp_dialog_hash] == NULL)
     tcp_dialog_hash_table[tcp_dialog_hash] = dialog_addr_ptr;
else
     {
     /*
      * Find the last structure of the hash link.
      */
     tcp_dialog_hash_link = tcp_dialog_hash_table[tcp_dialog_hash];
```

```
        while (tcp_dialog_hash_link->hash_link != NULL)
             tcp_dialog_hash_link = tcp_dialog_hash_link->hash_link;

        tcp_dialog_hash_link->hash_link = dialog_addr_ptr;
        }

    return (dialog_addr_ptr);
    }


    void setTcpAddrDflts (mib_tcp_addr_defs, tcp_stats_common_ptr)
        MibTcpAddrDefaults  *mib_tcp_addr_defs;
        StatsTcpCommon      *tcp_stats_common_ptr;
    {
        tcp_stats_common_ptr->rcvFrames.high_thld =
                                mib_tcp_addr_defs->tcpAddrRcvPktHighThld;
        tcp_stats_common_ptr->rcvFrameRate.high_thld =
                                mib_tcp_addr_defs->tcpAddrRcvPktRateHighThld;

        tcp_stats_common_ptr->rcvBytes.high_thld =
                                mib_tcp_addr_defs->tcpAddrRcvByteHighThld;
        tcp_stats_common_ptr->rcvByteRate.high_thld =
                                mib_tcp_addr_defs->tcpAddrRcvByteRateHighThld;

        tcp_stats_common_ptr->rcvErrors.high_thld =
                                mib_tcp_addr_defs->tcpAddrRcvErrorHighThld;
        tcp_stats_common_ptr->rcvErrorRate.high_thld =
                                mib_tcp_addr_defs->tcpAddrRcvErrorRateHighThld;

        tcp_stats_common_ptr->xmtFrames.high_thld =
                                mib_tcp_addr_defs->tcpAddrXmtPktHighThld;
        tcp_stats_common_ptr->xmtFrameRate.high_thld =
                                mib_tcp_addr_defs->tcpAddrXmtPktRateHighThld;

        tcp_stats_common_ptr->xmtBytes.high_thld =
                                mib_tcp_addr_defs->tcpAddrXmtByteHighThld;
        tcp_stats_common_ptr->xmtByteRate.high_thld =
```

```
                        mib_tcp_addr_defs->tcpAddrXmtByteRateHighThld;

tcp_stats_common_ptr->xmtErrors.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtErrorHighThld;
tcp_stats_common_ptr->xmtErrorRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtErrorRateHighThld;

tcp_stats_common_ptr->rcvOffSegs.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvOffSegHighThld;
tcp_stats_common_ptr->rcvOffSegRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvOffSegRateHighThld;

tcp_stats_common_ptr->xmtOffSegs.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtOffSegHighThld;
tcp_stats_common_ptr->xmtOffSegRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtOffSegRateHighThld;

tcp_stats_common_ptr->rcvHdrBytes.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvHdrByteHighThld;
tcp_stats_common_ptr->rcvHdrByteRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvHdrByteRateHighThld;

tcp_stats_common_ptr->xmtHdrBytes.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtHdrByteHighThld;
tcp_stats_common_ptr->xmtHdrByteRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtHdrByteRateHighThld;

tcp_stats_common_ptr->rcvFrgmts.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvFrgmtHighThld;
tcp_stats_common_ptr->rcvFrgmtRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvFrgmtRateHighThld;

tcp_stats_common_ptr->xmtFrgmts.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtFrgmtHighThld;
tcp_stats_common_ptr->xmtFrgmtRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtFrgmtRateHighThld;
```

```
tcp_stats_common_ptr->rcvRexmts.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvRexmtHighThld;
tcp_stats_common_ptr->rcvRexmtRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvRexmtRateHighThld;

tcp_stats_common_ptr->xmtRexmts.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtRexmtHighThld;
tcp_stats_common_ptr->xmtRexmtRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtRexmtRateHighThld;

tcp_stats_common_ptr->rcvRexmtBytes.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvRexmtByteHighThld;
tcp_stats_common_ptr->rcvRexmtByteRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvRexmtByteRateHighThld;

tcp_stats_common_ptr->xmtRexmtBytes.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtRexmtByteHighThld;
tcp_stats_common_ptr->xmtRexmtByteRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtRexmtByteRateHighThld;

tcp_stats_common_ptr->rcvKeepAlives.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvKeepAliveHighThld;
tcp_stats_common_ptr->rcvKeepAliveRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvKeepAliveRateHighThld;

tcp_stats_common_ptr->xmtKeepAlives.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtKeepAliveHighThld;
tcp_stats_common_ptr->xmtKeepAliveRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtKeepAliveRateHighThld;

tcp_stats_common_ptr->rcvWindowProbes.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvWindowProbeHighThld;
tcp_stats_common_ptr->rcvWindowProbeRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvWindowProbeRateHighThld;

tcp_stats_common_ptr->xmtWindowProbes.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtWindowProbeHighThld;
```

```
tcp_stats_common_ptr->xmtWindowProbeRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtWindowProbeRateHighThld;

tcp_stats_common_ptr->rcvOutOfOrder.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvOutOfOrderHighThld;
tcp_stats_common_ptr->rcvOutOfOrderRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvOutOfOrderRateHighThld;

tcp_stats_common_ptr->xmtOutOfOrder.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtOutOfOrderHighThld;
tcp_stats_common_ptr->xmtOutOfOrderRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtOutOfOrderRateHighThld;

tcp_stats_common_ptr->rcvAfterWindow.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvAfterWindowHighThld;
tcp_stats_common_ptr->rcvAfterWindowRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvAfterWindowRateHighThld;

tcp_stats_common_ptr->xmtAfterWindow.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtAfterWindowHighThld;
tcp_stats_common_ptr->xmtAfterWindowRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtAfterWindowRateHighThld;

tcp_stats_common_ptr->rcvAfterWindowBytes.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvAfterWindowByteHighThld;
tcp_stats_common_ptr->rcvAfterWindowByteRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvAfterWindowByteRateHighThld;

tcp_stats_common_ptr->xmtAfterWindowBytes.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtAfterWindowByteHighThld;
tcp_stats_common_ptr->xmtAfterWindowByteRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrXmtAfterWindowByteRateHighThld;

tcp_stats_common_ptr->rcvAfterClose.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvAfterCloseHighThld;
tcp_stats_common_ptr->rcvAfterCloseRate.high_thld =
                        mib_tcp_addr_defs->tcpAddrRcvAfterCloseRateHighThld;
```

```
    tcp_stats_common_ptr->xmtAfterClose.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtAfterCloseHighThld;
    tcp_stats_common_ptr->xmtAfterCloseRate.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtAfterCloseRateHighThld;

    tcp_stats_common_ptr->rcvUrgs.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvUrgHighThld;
    tcp_stats_common_ptr->rcvUrgRate.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvUrgRateHighThld;

    tcp_stats_common_ptr->xmtUrgs.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtUrgHighThld;
    tcp_stats_common_ptr->xmtUrgRate.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtUrgRateHighThld;

    tcp_stats_common_ptr->rcvRsts.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvRstHighThld;
    tcp_stats_common_ptr->rcvRstRate.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvRstRateHighThld;

    tcp_stats_common_ptr->xmtRsts.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtRstHighThld;
    tcp_stats_common_ptr->xmtRstRate.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtRstRateHighThld;

    tcp_stats_common_ptr->rcvSuccessfulConnections.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvSuccessfulConnectionHighThld;
    tcp_stats_common_ptr->rcvSuccessfulConnectionRate.high_thld =
mib_tcp_addr_defs->tcpAddrRcvSuccessfulConnectionRateHighThld;

    tcp_stats_common_ptr->xmtSuccessfulConnections.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtSuccessfulConnectionHighThld;
    tcp_stats_common_ptr->xmtSuccessfulConnectionRate.high_thld =
mib_tcp_addr_defs->tcpAddrXmtSuccessfulConnectionRateHighThld;
```

```
    tcp_stats_common_ptr->rcvConnectionRetries.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvConnectionRetryHighThld;
    tcp_stats_common_ptr->rcvConnectionRetryRate.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvConnectionRetryRateHighThld;

    tcp_stats_common_ptr->xmtConnectionRetries.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtConnectionRetryHighThld;
    tcp_stats_common_ptr->xmtConnectionRetryRate.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtConnectionRetryRateHighThld;

    tcp_stats_common_ptr->rcvFailedConnections.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvFailedConnectionHighThld;
    tcp_stats_common_ptr->rcvFailedConnectionRate.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvFailedConnectionRateHighThld;

    tcp_stats_common_ptr->xmtFailedConnections.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtFailedConnectionHighThld;
    tcp_stats_common_ptr->xmtFailedConnectionRate.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtFailedConnectionRateHighThld;

    tcp_stats_common_ptr->rcvSuccessfulConnections.high_thld =
                            mib_tcp_addr_defs->tcpAddrRcvSuccessfulConnectionHighThld;
    tcp_stats_common_ptr->rcvSuccessfulConnectionRate.high_thld =

nib_tcp_addr_defs->tcpAddrRcvSuccessfulConnectionRateHighThld;

    tcp_stats_common_ptr->xmtSuccessfulConnections.high_thld =
                            mib_tcp_addr_defs->tcpAddrXmtSuccessfulConnectionHighThld;
    tcp_stats_common_ptr->xmtSuccessfulConnectionRate.high_thld =

nib_tcp_addr_defs->tcpAddrXmtSuccessfulConnectionRateHighThld;
}

void setTcpAddrPairDflts (mib_tcp_addr_pair_defs, dialog_ptr)
    MibTcpAddrPairDefaults   *mib_tcp_addr_pair_defs;
```

```
    StatsDialogEntry          *dialog_ptr;
{
    dialog_ptr->packets.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairPktHighThld;
    dialog_ptr->packetRate.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairPktRateHighThld;

    dialog_ptr->bytes.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairByteHighThld;
    dialog_ptr->byteRate.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairByteRateHighThld;

    dialog_ptr->errors.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairErrorHighThld;
    dialog_ptr->errorRate.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairErrorRateHighThld;

    dialog_ptr->fragments.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairFrgmtHighThld;
    dialog_ptr->fragmentRate.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairFrgmtRateHighThld;

    dialog_ptr->rexmts.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairRexmtHighThld;
    dialog_ptr->rexmtRate.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairRexmtRateHighThld;

    dialog_ptr->flowCtrls.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairFlowCtrlHighThld;
    dialog_ptr->flowCtrlRate.high_thld =
                        mib_tcp_addr_pair_defs->tcpAddrPairFlowCtrlRateHighThld;
}

void setTcpSegDflts (mib_tcp_seg_defs, tcp_seg_ptr)
    MibTcpSegDefaults   *mib_tcp_seg_defs;
    StatsTcpSegment          *tcp_seg_ptr;
```

```
{
    tcp_seg_ptr->frames.high_thld =
                        mib_tcp_seg_defs->tcpSegPktHighThld;
    tcp_seg_ptr->frameRate.high_thld =
                        mib_tcp_seg_defs->tcpSegPktRateHighThld;

    tcp_seg_ptr->bytes.high_thld =
                        mib_tcp_seg_defs->tcpSegByteHighThld;
    tcp_seg_ptr->byteRate.high_thld =
                        mib_tcp_seg_defs->tcpSegByteRateHighThld;

    tcp_seg_ptr->errors.high_thld =
                        mib_tcp_seg_defs->tcpSegErrorHighThld;
    tcp_seg_ptr->errorRate.high_thld =
                        mib_tcp_seg_defs->tcpSegErrorRateHighThld;

    tcp_seg_ptr->rcvOffSegs.high_thld =
                        mib_tcp_seg_defs->tcpSegRcvOffSegHighThld;
    tcp_seg_ptr->rcvOffSegRate.high_thld =
                        mib_tcp_seg_defs->tcpSegRcvOffSegRateHighThld;

    tcp_seg_ptr->xmtOffSegs.high_thld =
                        mib_tcp_seg_defs->tcpSegXmtOffSegHighThld;
    tcp_seg_ptr->xmtOffSegRate.high_thld =
                        mib_tcp_seg_defs->tcpSegXmtOffSegRateHighThld;

    tcp_seg_ptr->transits.high_thld =
                        mib_tcp_seg_defs->tcpSegTransitHighThld;
    tcp_seg_ptr->transitRate.high_thld =
                        mib_tcp_seg_defs->tcpSegTransitRateHighThld;

    tcp_seg_ptr->flowCtrls.high_thld =
                        mib_tcp_seg_defs->tcpSegFlowCtrlHighThld;
    tcp_seg_ptr->flowCtrlRate.high_thld =
                        mib_tcp_seg_defs->tcpSegFlowCtrlRateHighThld;

    tcp_seg_ptr->frgmts.high_thld =
```

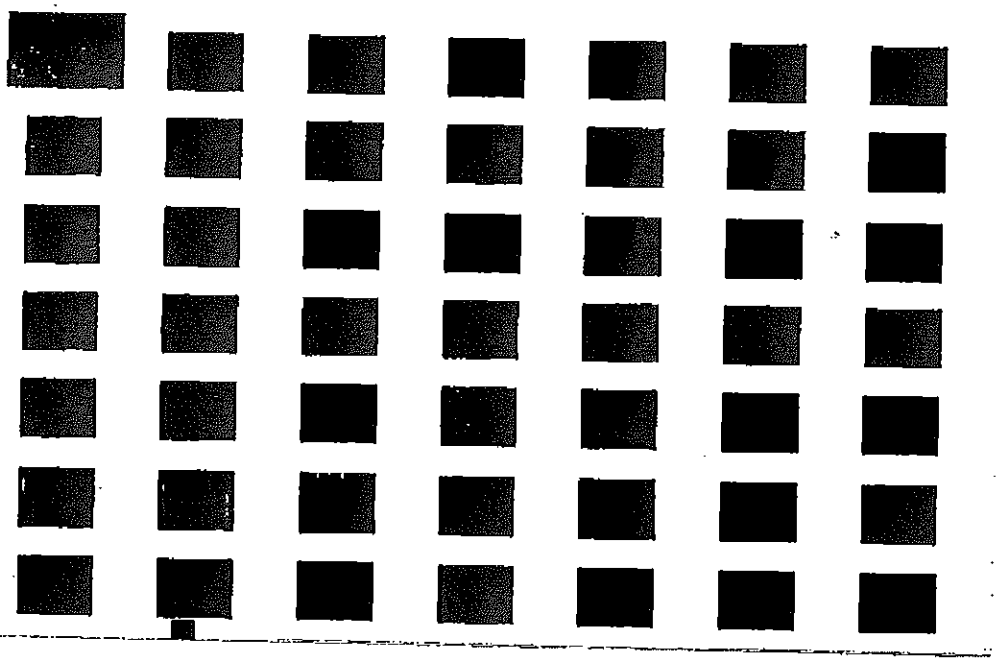
```
                        mib_tcp_seg_defs->tcpSegFrgmtHighThld;
    tcp_seg_ptr->frgmtRate.high_thld =
                        mib_tcp_seg_defs->tcpSegFrgmtRateHighThld;

    tcp_seg_ptr->rexmts.high_thld =
                        mib_tcp_seg_defs->tcpSegRexmtHighThld;
    tcp_seg_ptr->rexmtRate.high_thld =
                        mib_tcp_seg_defs->tcpSegRexmtRateHighThld;

/******* REMOVE UNTIL SEG DEFAULTS HDR IS UPDATED *****/
/*
    tcp_seg_ptr->rexmtBytes.high_thld =
                        mib_tcp_seg_defs->tcpSegRexmtByteHighThld;
    tcp_seg_ptr->rexmtByteRate.high_thld =
                        mib_tcp_seg_defs->tcpSegRexmtByteRateHighThld;
*/

    tcp_seg_ptr->hdrBytes.high_thld =
                        mib_tcp_seg_defs->tcpSegHdrByteHighThld;
    tcp_seg_ptr->hdrByteRate.high_thld =
                        mib_tcp_seg_defs->tcpSegHdrByteRateHighThld;

    tcp_seg_ptr->rsts.high_thld =
                        mib_tcp_seg_defs->tcpSegRstHighThld;
    tcp_seg_ptr->rstRate.high_thld =
                        mib_tcp_seg_defs->tcpSegRstRateHighThld;

    tcp_seg_ptr->successfulConnections.high_thld =
                        mib_tcp_seg_defs->tcpSegSuccessfulConnectionHighThld;
    tcp_seg_ptr->successfulConnectionRate.high_thld =
                        mib_tcp_seg_defs->tcpSegSuccessfulConnectionRateHighThld;

    tcp_seg_ptr->connectionRetries.high_thld =
                        mib_tcp_seg_defs->tcpSegConnectionRetryHighThld;
    tcp_seg_ptr->connectionRetryRate.high_thld =
                        mib_tcp_seg_defs->tcpSegConnectionRetryRateHighThld;
```

```
        tcp_seg_ptr->failedConnections.high_thld =
                            mib_tcp_seg_defs->topSegFailedConnectionHighThld;
        tcp_seg_ptr->failedConnectionRate.high_thld =
                            mib_tcp_seg_defs->tcpSegFailedConnectionRateHighThld;
}
```

```
/*
 *  stats_udp_a.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_udp_a.c
 *
 *  Date:       6/5/91
 *
 *  Revision:   1.5
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *
 *   06-05-91  DPD       Fixed aging bug.
 *   06-01-91  DPD       Fixed rate bug.
 *
 */

static char stats_udp_a_c [] = "@(#)stats_udp_a.c 1.5";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
#include "mib_defs.h"
#include "mib_ip.h"
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerard White

13



NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
#include <sys/types.h>
#include <sys/socket.h>
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "protocols.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_udp.h"
#include "stats_mib.h"
#include "mib_udp.h"
#include "snmpd.h"
#include "em_ctrl.h"


/*****************************************************************************
 *
 * Given a UDP dialog, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsUdpAgeDialog (dialog_addr_ptr)

     StatsAddrEntry      *dialog_addr_ptr;

{
     register Uint32                  xudp_dialog_hash;
     register StatsAddrEntry       *xudp_dialog_hash_link;
```

```
        register StatsAddrEntry       *xudp_previous_dialog_hash_link;

        register StatsDialogEntry     *dialog_stats_ptr;
        register StatsDialogLink *dialog_link_ptr;
        register StatsAddrEntry       *addr_record_ptr;
        register StatsUdpAddr         *udp_stats_ptr;
        register StatsUdpSocket       *udp_socket_stats_ptr;
        struct timeval                current_time;
        Uint32                             i;

    if (monCtrl.udpDialogAgeTimer == 0)
        return;

    if (dialog_addr_ptr == NULL)
        return;

    dialog_stats_ptr = (StatsDialogEntry *) dialog_addr_ptr->stats_ptr;

    mon_gettimeofday (&current_time, 0);

    if (current_time.tv_sec - dialog_addr_ptr->lastTime < monCtrl.udpDialogAgeTimer)
        return;

    /*
     * Time to chuck this dialog.  Remove it from the dialog hash table.
     * Look up the dialog in order to set udp_dialog_hash, udp_previous_dialog_hash_link
     * and udp_dialog_hash_link.
     */
    addr_record_ptr = stats_udp_lookup_dialog (
        &dialog_addr_ptr->address.netAddress1.u.ipAddress,
        dialog_addr_ptr->address.port1,
        &dialog_addr_ptr->address.netAddress2.u.ipAddress,
        dialog_addr_ptr->address.port2);

    if (addr_record_ptr != dialog_addr_ptr)
        {
        mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsUdpAgeDialog: ptrs not equal");
```

```
        return;
        }

    xudp_dialog_hash = udp_dialog_hash;
    xudp_dialog_hash_link = udp_dialog_hash_link;
    xudp_previous_dialog_hash_link = udp_previous_dialog_hash_link;

    if ( (xudp_previous_dialog_hash_link == NULL) && (xudp_dialog_hash_link->hash_link ==
    NULL) )
        udp_dialog_hash_table[xudp_dialog_hash] = NULL;
    else
        {
        if (xudp_previous_dialog_hash_link != NULL)
            xudp_previous_dialog_hash_link->hash_link = xudp_dialog_hash_link->hash_link;
        else
            if (xudp_dialog_hash_link->hash_link != NULL)
                udp_dialog_hash_table[xudp_dialog_hash] = xudp_dialog_hash_link->hash_link;

        }

    /*
     * Remove the dialog from the dialogQ for each of the udp stats structures for the
     * ip addresses and deallocate the structure that was on the dialogQ which pointed
     * to the dialog.
     */
    addr_record_ptr = stats_udp_lookup_addr
    (&dialog_addr_ptr->address.netAddress1.u.ipAddress);

    for (i=0; i<2; i++)
        {
        if (addr_record_ptr != NULL)
            {
            udp_stats_ptr = (StatsUdpAddr *) addr_record_ptr->stats_ptr;
            if (udp_stats_ptr != NULL)
                {
                dialog_link_ptr = (StatsDialogLink *) udp_stats_ptr->dialogQ.pFlink;
                while (dialog_link_ptr != NULL)
```

```
                    {
                    if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                        break;
                    dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
                    }
               if (dialog_link_ptr != NULL)
                    {
                    FBRemqm (&udp_stats_ptr->dialogQ, (PFBQentry_type) dialog_link_ptr);
                    stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                    }
               }
          addr_record_ptr = stats_udp_lookup_addr (
               &( dialog_addr_ptr->address.netAddress2.u.ipAddress);
     }


/*
 * Remove the dialog from the dialogQ for each of the sockets
 * and deallocate the structure that was on the dialogQ which pointed
 * to the dialog.
 */
addr_record_ptr = stats_udp_lookup_socket (
     &dialog_addr_ptr->address.netAddress1.u.ipAddress,
     dialog_addr_ptr->address.port1);

for (i=0; i<2; i++)
     {
     if (addr_record_ptr != NULL)
          {
          udp_socket_stats_ptr = (StatsUdpSocket *) addr_record_ptr->stats_ptr;
          if (udp_socket_stats_ptr != NULL)
               {
               dialog_link_ptr = (StatsDialogLink *) udp_socket_stats_ptr->dialogQ.pFlink;

               while (dialog_link_ptr != NULL)
                    {
```

```
                    if (dialog_link_ptr->dialog_addr_ptr == dialog_addr_ptr)
                        break;
                    dialog_link_ptr = (StatsDialogLink *) dialog_link_ptr->link.pFlink;
                    }
                if (dialog_link_ptr != NULL)
                    {
                    FBRemqn (&udp_socket_stats_ptr->dialogQ, (PFBQentry_type)
dialog_link_ptr);
                    stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
                    }
                }
            }
            addr_record_ptr = stats_udp_lookup_socket (
                &dialog_addr_ptr->address.netAddress2.u.ipAddress,
                dialog_addr_ptr->address.port2);
        }


/*
 * Remove the dialog from the statsUdpDialogQ and deallocate the structures
 * associated with the dialog.
 */
FBRemqm (&statsUdpDialogQ, (PFBQentry_type) dialog_addr_ptr);

stats_deallocate (dialog_stats_ptr, sizeof(StatsDialogEntry) );
stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );

}


/*****************************************************************************
 *
 * Given a socket, determine if it should be aged out, and, if so,
 * get rid of it.
 */
```

```
void statsUdpAgeAddr (udp_addr_ptr)

     StatsAddrEntry      *udp_addr_ptr;

{
     register StatsDialogLink *dialog_link_ptr;
     register StatsAddrEntry       *addr_record_ptr;
     register StatsUdpAddr         *udp_stats_ptr;
     struct timeval                current_time;

if (monCtrl.udpNodeAgeTimer == 0)
     return;

if (udp_addr_ptr == NULL)
     return;

udp_stats_ptr = (StatsUdpAddr *) udp_addr_ptr->stats_ptr;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - udp_addr_ptr->lastTime < monCtrl.udpNodeAgeTimer)
     return;

/*
 * Time to chuck this socket.  Remove it from the hash table.
 * Look up the ip address in order to set udp_hash, udp_previous_hash_link
 * and udp_hash_link.
 */
addr_record_ptr = stats_udp_lookup_addr (
     &udp_addr_ptr->address.netAddress1.u.ipAddress);

if (addr_record_ptr != udp_addr_ptr)
     {
     mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsUdpAgeAddr: ptrs not equal");
     return;
     }
```

```
if ( (udp_previous_hash_link == NULL) && (udp_hash_link->hash_link == NULL) )
     udp_hash_table[udp_hash] = NULL;
else
     {
     if (udp_previous_hash_link != NULL)
          udp_previous_hash_link->hash_link = udp_hash_link->hash_link;
     else
          if (udp_hash_link->hash_link != NULL)
               udp_hash_table[udp_hash] = udp_hash_link->hash_link;
     }

/*
 * Remove the dialog links from the socket stats and deallocate the stats structure.
 */
if (udp_stats_ptr != NULL)
     {
     dialog_link_ptr = (StatsDialogLink *) FBRemqh (&udp_stats_ptr->dialogQ);
     while (dialog_link_ptr != NULL)
          {
          stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
          dialog_link_ptr = (StatsDialogLink *) FBRemqh (&udp_stats_ptr->dialogQ);
          }
     stats_deallocate (udp_stats_ptr, sizeof(StatsUdpAddr) );
     }


/*
 * Remove the address from the statsUdpAddrQ and deallocate the address structure.
 */
FBRemqm (&statsUdpAddrQ, (PFBQentry_type) udp_addr_ptr);
stats_deallocate (udp_addr_ptr, sizeof(StatsAddrEntry) );

}
```

```
/*****************************************************************************
 *
 * Given a socket, determine if it should be aged out, and, if so,
 * get rid of it.
 */

void statsUdpAgeSocket (socket_addr_ptr)

     StatsAddrEntry      *socket_addr_ptr;

{
     register StatsDialogLink *dialog_link_ptr;
     register StatsAddrEntry       *addr_record_ptr;
     register StatsUdpAddr         *socket_stats_ptr;
     struct timeval                current_time;

if (monCtrl.udpWellKnownAgeTimer == 0)
     return;

if (socket_addr_ptr == NULL)
     return;

socket_stats_ptr = (StatsUdpAddr *) socket_addr_ptr->stats_ptr;

mon_gettimeofday (&current_time, 0);

if (current_time.tv_sec - socket_addr_ptr->lastTime < monCtrl.udpWellKnownAgeTimer)
     return;

/*
 * Time to chuck this socket.  Remove it from the socket hash table.
 * Look up the socket in order to set udp_socket_hash, udp_previous_socket_hash_link
 * and udp_socket_hash_link.
 */
addr_record_ptr = stats_udp_lookup_socket (
     &socket_addr_ptr->address.netAddress1.u.ipAddress,
```

```
     socket_addr_ptr->address.port1);

if (addr_record_ptr != socket_addr_ptr)
     {
     mon_panic (EM_task, STATS_PTRS_NOT_EQUAL, "statsUdpAgeSocket: ptrs not equal");
     return;
     }

if ( (udp_previous_socket_hash_link == NULL) && (udp_socket_hash_link->hash_link == NULL)
)
     udp_socket_hash_table[udp_socket_hash] = NULL;
else
     {
     if (udp_previous_socket_hash_link != NULL)
          udp_previous_socket_hash_link->hash_link = udp_socket_hash_link->hash_link;
     else
          if (udp_socket_hash_link->hash_link != NULL)
               udp_socket_hash_table[udp_socket_hash] = udp_socket_hash_link->hash_link;
     }

/*
 * Remove the dialog links from the socket stats and deallocate the stats structure.
 */
if (socket_stats_ptr != NULL)
     {
     dialog_link_ptr = (StatsDialogLink *) FBRemqh (&socket_stats_ptr->dialogQ);
     while (dialog_link_ptr != NULL)
          {
          stats_deallocate (dialog_link_ptr, sizeof(StatsDialogLink));
          dialog_link_ptr = (StatsDialogLink *) FBRemqh (&socket_stats_ptr->dialogQ);
          }
     stats_deallocate (socket_stats_ptr, sizeof(StatsUdpAddr) );
     }


/*
 * Remove the socket from the statsUdpSocketQ and deallocate the address structure.
```

```
 */
FBRemqm (&statsUdpSocketQ, (PFBQentry_type) socket_addr_ptr);
stats_deallocate (socket_addr_ptr, sizeof(StatsAddrEntry) );

}


/*****************************************************************************
 *
 * UDP rate routines
 */

void statsUdpProtocolRate (udp_protocol_ptr, udp_addr_ptr, rate_type, time)
     StatsProtocolEntry  *udp_protocol_ptr;
     StatsAddrEntry                *udp_addr_ptr;
     Uint32                             rate_type;
     Uint32                             time;
{
     AlarmUserData       udp_alarm_data;

     while (udp_protocol_ptr != NULL)
     {
          /*
           * Pass the protocol as alarm data in case an alarm occurs
           */
          udp_alarm_data.length = 4;
          bcopy (&udp_protocol_ptr->protocol, udp_alarm_data.data, 4);

          statsCalcRates(&udp_protocol_ptr->frameRate, NULL,
                         rate_type, (StatsAddrEntry *)udp_addr_ptr,
                         UDP_PROTOCOL, AL_FRAMES, time, NULL);
          udp_protocol_ptr = udp_protocol_ptr->link;
     }

}
```

```
void statsUdpSegRate (rate_type)
     Uint32                    rate_type;
{
     register StatsUdpSegment *udp_seg_ptr;
     register StatsAddrEntry       *udp_seg_addr_ptr;
     struct timeval                current_time;
     register StatsProtocolEntry   *udp_protocol_ptr;

     udp_seg_addr_ptr = (StatsAddrEntry *) statsUdpSegQ.pFlink;

     while (udp_seg_addr_ptr != NULL)
     {
          udp_seg_ptr = (StatsUdpSegment *) udp_seg_addr_ptr->stats_ptr;
          if (udp_seg_ptr != NULL)
          {
               /* get the current time */
               mon_gettimeofday(&current_time, 0);

               statsCalcRates(&udp_seg_ptr->frameRate, &udp_seg_ptr->frameBuckets,
                              rate_type, (StatsAddrEntry *)udp_seg_addr_ptr,
UDP_SEGMENT,
                              AL_FRAMES, current_time.tv_sec, NULL);
               statsCalcRates(&udp_seg_ptr->byteRate, &udp_seg_ptr->byteBuckets,
                              rate_type, (StatsAddrEntry *)udp_seg_addr_ptr,
UDP_SEGMENT,
                              AL_BYTES, current_time.tv_sec, NULL);
               statsCalcRates(&udp_seg_ptr->errorRate, &udp_seg_ptr->errorBuckets,
                              rate_type, (StatsAddrEntry *)udp_seg_addr_ptr,
UDP_SEGMENT,
                              AL_ERRORS, current_time.tv_sec, NULL);
               statsCalcRates(&udp_seg_ptr->rcvOffSegRate, &udp_seg_ptr->rcvOffSegBuckets,

                              rate_type, (StatsAddrEntry *)udp_seg_addr_ptr,
UDP_SEGMENT,
                              AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
```

```
                statsCalcRates(&udp_seg_ptr->xmtOffSegRate, &udp_seg_ptr->xmtOffSegBuckets,
                                        rate_type, (StatsAddrEntry *)udp_seg_addr_ptr,
UDP_SEGMENT,
                                        AL_XMT_OFF_SEG, current_time.tv_sec, NULL);
                statsCalcRates(&udp_seg_ptr->transitRate, &udp_seg_ptr->transitBuckets,
                                        rate_type, (StatsAddrEntry *)udp_seg_addr_ptr,
UDP_SEGMENT,
                                        AL_TRANSIT, current_time.tv_sec, NULL);
                statsCalcRates(&udp_seg_ptr->flowCtrlRate, &udp_seg_ptr->flowCtrlBuckets,
                                        rate_type, (StatsAddrEntry *)udp_seg_addr_ptr,
UDP_SEGMENT,
                                        AL_FLOW_CTRLS, current_time.tv_sec, NULL);
                statsCalcRates(&udp_seg_ptr->frgmtRate, &udp_seg_ptr->frgmtBuckets,
                                        rate_type, (StatsAddrEntry *)udp_seg_addr_ptr,
UDP_SEGMENT,
                                        AL_FRAGMENTS, current_time.tv_sec, NULL);

                /* calculate the rates on the protocolQ */
                udp_protocol_ptr = (StatsProtocolEntry *) udp_seg_ptr->protocolQ.pFlink;
                statsUdpProtocolRate (udp_protocol_ptr, udp_seg_addr_ptr,
                                                rate_type, current_time.tv_sec);

                statsUdpDialogRate (rate_type);

                if ((current_time.tv_sec - udp_seg_addr_ptr->seconds_start_time) >=
nonCtrl.rateTimer)
                        udp_seg_addr_ptr->seconds_start_time = current_time.tv_sec;
                }
                udp_seg_addr_ptr = (StatsAddrEntry *) udp_seg_addr_ptr->link.pFlink;
        }
}


void statsUdpRate (udp_common_ptr, udp_entry_ptr, rate_type, current_time)
        StatsUdpCommon      *udp_common_ptr;
```

```
    StatsAddrEntry      *udp_entry_ptr;
    Uint32                  rate_type;
    struct timeval      current_time;
{
    statsCalcRates(&udp_common_ptr->frameRate, &udp_common_ptr->frameBuckets,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_FRAMES, current_time.tv_sec, NULL);
    statsCalcRates(&udp_common_ptr->rcvFrameRate, NULL,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_RCV_FRAMES, current_time.tv_sec, NULL);
    statsCalcRates(&udp_common_ptr->xmtFrameRate, NULL,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_XMT_FRAMES, current_time.tv_sec, NULL);

    statsCalcRates(&udp_common_ptr->byteRate, &udp_common_ptr->byteBuckets,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_BYTES, current_time.tv_sec, NULL);
    statsCalcRates(&udp_common_ptr->rcvByteRate, NULL,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_RCV_BYTES, current_time.tv_sec, NULL);
    statsCalcRates(&udp_common_ptr->xmtByteRate, NULL,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_XMT_BYTES, current_time.tv_sec, NULL);

    statsCalcRates(&udp_common_ptr->errorRate, &udp_common_ptr->errorBuckets,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_ERRORS, current_time.tv_sec, NULL);
    statsCalcRates(&udp_common_ptr->rcvErrorRate, NULL,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_RCV_ERRORS, current_time.tv_sec, NULL);
    statsCalcRates(&udp_common_ptr->xmtErrorRate, NULL,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_XMT_ERRORS, current_time.tv_sec, NULL);

    statsCalcRates(&udp_common_ptr->rcvOffSegRate, &udp_common_ptr->rcvOffSegBuckets,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_RCV_OFF_SEG, current_time.tv_sec, NULL);
```

```
    statsCalcRates(&udp_common_ptr->xmtOffSegRate, &udp_common_ptr->xmtOffSegBuckets,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_XMT_OFF_SEG, current_time.tv_sec, NULL);

    statsCalcRates(&udp_common_ptr->flowCtrlRate, &udp_common_ptr->flowCtrlBuckets,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_FLOW_CTRLS, current_time.tv_sec, NULL);

    statsCalcRates(&udp_common_ptr->frgmtRate, &udp_common_ptr->frgmtBuckets,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_FRAGMENTS, current_time.tv_sec, NULL);
    statsCalcRates(&udp_common_ptr->rcvFrgmtRate, NULL,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_RCV_FRAGMENTS, current_time.tv_sec, NULL);
    statsCalcRates(&udp_common_ptr->xmtFrgmtRate, NULL,
                   rate_type, (StatsAddrEntry *)udp_entry_ptr,
                   UDP_NODE, AL_XMT_FRAGMENTS, current_time.tv_sec, NULL);
}

void statsUdpAddrRate (rate_type)
    Uint32                  rate_type;
{
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
     register StatsAddrEntry        *udp_entry_ptr;
     register StatsUdpAddr          *udp_addr_ptr;
     register StatsUdpCommon        *udp_common_ptr;
     register Uint32                     loop_count;
     struct timeval                 current_time;
     register StatsProtocolEntry    *udp_protocol_ptr;

     /* get the current time */
     mon_gettimeofday(&current_time, 0);

     loop_count = 0;

     /***** AGING *****/
```

```
        if (statsNextUdpAddrEntry != NULL)
             entry_ptr = statsNextUdpAddrEntry;      /* pick up where we left off */
        else
        {
             /* Only age structures if we've processed them all */
             entry_ptr = statsUdpAddrQ.pFlink;
             while (entry_ptr != NULL)
             {
                  next_entry_ptr = entry_ptr->pFlink;
                  statsUdpAgeAddr ((StatsAddrEntry *) entry_ptr);
                  entry_ptr = next_entry_ptr;
             }

             entry_ptr = statsUdpAddrQ.pFlink;       /* start at the beginning */
        }
        /***** AGING *****/

        while ((entry_ptr != NULL) && (loop_count < stats_q_count))
        {
             udp_entry_ptr = (StatsAddrEntry *)entry_ptr;
             if (((udp_entry_ptr->em_control & rate_type) != 0)
                                  && (udp_entry_ptr->stats_ptr != NULL))
             {
                  if ((current_time.tv_sec - udp_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
                  {
                  udp_addr_ptr = (StatsUdpAddr *)udp_entry_ptr->stats_ptr;
                  udp_common_ptr = (StatsUdpCommon *) &udp_addr_ptr->common;
                  statsUdpRate (udp_common_ptr, udp_entry_ptr, rate_type, current_time);

                  /* calculate the rates on the protocolQ */
                  udp_protocol_ptr = (StatsProtocolEntry *) udp_addr_ptr->protocolQ.pFlink;
                  statsUdpProtocolRate (udp_protocol_ptr, udp_entry_ptr,
                                        rate_type, current_time.tv_sec);

                  udp_entry_ptr->seconds_start_time = current_time.tv_sec;
                  loop_count++;
```

```
            }
        }
        entry_ptr = entry_ptr->pFlink;
    }
    statsNextUdpAddrEntry = entry_ptr;      /* pick up where we left off */

    if (statsNextUdpAddrEntry != NULL)
        statsRatesNotDone |= STATS_UDP_ADDR_RATE;
}

void statsUdpSocketRate (rate_type)
    Uint32                  rate_type;
{
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
     register StatsAddrEntry        *udp_entry_ptr;
     register StatsUdpSocket        *udp_socket_ptr;
     register StatsUdpCommon        *udp_common_ptr;
     register Uint32                     loop_count;
     struct timeval                 current_time;
     register StatsProtocolEntry    *udp_protocol_ptr;

    /* get the current time */
    mon_gettimeofday(&current_time, 0);

    loop_count = 0;

    /***** AGING *****/
    if (statsNextUdpSocketEntry != NULL)
        entry_ptr = statsNextUdpSocketEntry;        /* pick up where we left off */
    else
    {
        /* Only age structures if we've processed them all */
        entry_ptr = statsUdpSocketQ.pFlink;
        while (entry_ptr != NULL)
        {
            next_entry_ptr = entry_ptr->pFlink;
```

```
                statsUdpAgeSocket ((StatsAddrEntry *) entry_ptr);
                entry_ptr = next_entry_ptr;
            }

            entry_ptr = statsUdpSocketQ.pFlink;          /* start at the beginning */
        }
        /***** AGING *****/

        while ((entry_ptr != NULL) && (loop_count < stats_q_count))
        {
            udp_entry_ptr = (StatsAddrEntry *)entry_ptr;
            if (((udp_entry_ptr->en_control & rate_type) != 0)
                                    && (udp_entry_ptr->stats_ptr != NULL))
            {
                if ((current_time.tv_sec - udp_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
                {
                udp_socket_ptr = (StatsUdpSocket *)udp_entry_ptr->stats_ptr;
                udp_common_ptr = (StatsUdpCommon *) &udp_socket_ptr->common;
                statsUdpRate (udp_common_ptr, udp_entry_ptr, rate_type, current_time);

                udp_entry_ptr->seconds_start_time = current_time.tv_sec;
                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }
        statsNextUdpSocketEntry = entry_ptr;         /* pick up where we left off */

        if (statsNextUdpSocketEntry != NULL)
            statsRatesNotDone |= STATS_UDP_SOCKET_RATE;
}

void statsUdpDialogRate (rate_type)
        Uint32                  rate_type;
{
        register StatsDialogEntry       *udp_dialog_ptr;
```

```
    register FBQentry_type          *entry_ptr;
    register FBQentry_type          *next_entry_ptr;
     register StatsAddrEntry        *udp_entry_ptr;
     register Uint32                     loop_count;
     struct timeval                 current_time;

     /* get the current time */
     mon_gettimeofday(&current_time, 0);

     if (statsNextUdpPairEntry != NULL)
          entry_ptr = statsNextUdpPairEntry;      /* pick up where we left off */
     else
     {
          /* Only age structures if we've processed then all */
          entry_ptr = statsUdpDialogQ.pFlink;
          while (entry_ptr != NULL)
          {
               next_entry_ptr = entry_ptr->pFlink;
               statsUdpAgeDialog ((StatsAddrEntry *) entry_ptr);
               entry_ptr = next_entry_ptr;
          }

          entry_ptr = statsUdpDialogQ.pFlink;          /* start at the beginning */
     }

     loop_count = 0;
     while ((entry_ptr != NULL) && (loop_count < stats_g_count))
     {
          udp_entry_ptr = (StatsAddrEntry *)entry_ptr;
          if (((udp_entry_ptr->em_control & rate_type) != 0)
                              && (udp_entry_ptr->stats_ptr != NULL))
          {
               if ((current_time.tv_sec - udp_entry_ptr->seconds_start_time) >=
monCtrl.rateTimer)
               {
               udp_dialog_ptr = (StatsDialogEntry *)udp_entry_ptr->stats_ptr;
```

```
                statsCalcRates(&udp_dialog_ptr->packetRate, NULL,
                               rate_type, (StatsAddrEntry *)udp_entry_ptr,
                               UDP_PAIR, AL_FRAMES, current_time.tv_sec, NULL);

                statsCalcRates(&udp_dialog_ptr->byteRate, NULL,
                               rate_type, (StatsAddrEntry *)udp_entry_ptr,
                               UDP_PAIR, AL_BYTES, current_time.tv_sec, NULL);

                statsCalcRates(&udp_dialog_ptr->errorRate, NULL,
                               rate_type, (StatsAddrEntry *)udp_entry_ptr,
                               UDP_PAIR, AL_ERRORS, current_time.tv_sec, NULL);

                statsCalcRates(&udp_dialog_ptr->fragmentRate, NULL,
                               rate_type, (StatsAddrEntry *)udp_entry_ptr,
                               UDP_PAIR, AL_FRAGMENTS, current_time.tv_sec, NULL);

                statsCalcRates(&udp_dialog_ptr->rexmtRate, NULL,
                               rate_type, (StatsAddrEntry *)udp_entry_ptr,
                               UDP_PAIR, AL_REXMTS, current_time.tv_sec, NULL);

                statsCalcRates(&udp_dialog_ptr->flowCtrlRate, NULL,
                               rate_type, (StatsAddrEntry *)udp_entry_ptr,
                               UDP_PAIR, AL_FLOW_CTRLS, current_time.tv_sec, NULL);

                udp_entry_ptr->seconds_start_time = current_time.tv_sec;
                loop_count++;
                }
            }
            entry_ptr = entry_ptr->pFlink;
        }
        statsNextUdpPairEntry = entry_ptr;

        if (statsNextUdpPairEntry != NULL)
            statsRatesNotDone |= STATS_UDP_PAIR_RATE;
}
```

```
/*
 *  stats_udp_p.c
 *
 *   [description]
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                   All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_udp_p.c
 *
 *  Date:       6/19/91
 *
 *  Revision:   1.5
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO       Description of change. (latest first)
 *   --------  ---       ---------------------------------------------
 *   06-14-91  KR        Fixed set of applicationProtocol in dialog
 *
 */

static char stats_udp_p_c [] = "@(#)stats_udp_p.c 1.5";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
#include "mib_defs.h"
#include "mib_ip.h"
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <bsd43/sys/time.h>
#include "util.h"
#include "kuser.h"
#include "mbuf.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "mtm_in.h"
#include "rtp.h"
#include "alarms.h"
#include "protocols.h"
#include "stats.h"
#include "stats_ip.h"
#include "stats_udp.h"
#include "stats_mib.h"
#include "mib_udp.h"


/*
 * Global Data Structures
 */

StatsAddrEntry          *udp_this_seg_addr_ptr;
StatsUdpSegment              *udp_this_seg_stats_ptr;
StatsProtocolEntry      *udp_this_seg_protocol_ptr;

StatsAddrEntry          *udp_src_seg_addr_ptr, *udp_dst_seg_addr_ptr;
StatsUdpSegment              *udp_src_seg_stats_ptr, *udp_dst_seg_stats_ptr;
StatsProtocolEntry      *udp_src_seg_protocol_ptr, *udp_dst_seg_protocol_ptr;

StatsAddrEntry          *udp_src_node_addr_ptr, *udp_dst_node_addr_ptr;
StatsUdpAddr            *udp_src_node_stats_ptr, *udp_dst_node_stats_ptr;

StatsAddrEntry         *udp_src_socket_addr_ptr, *udp_dst_socket_addr_ptr;
StatsUdpSocket         *udp_src_socket_stats_ptr, *udp_dst_socket_stats_ptr;
```

```
StatsProtocolEntry      *udp_src_node_protocol_ptr, *udp_dst_node_protocol_ptr;

StatsAddrEntry          *udp_dialog_addr_ptr;
StatsDialogEntry        *udp_dialog_stats_ptr;


/*
 * Local data structures
 */

 StatsAddrEntry  *udp_hash_table[UDP_HASH_TABLE_SIZE];
 Uint32          udp_hash;
 StatsAddrEntry  *udp_hash_link;
 StatsAddrEntry  *udp_previous_hash_link;

 StatsAddrEntry  *udp_socket_hash_table[UDP_SOCKET_HASH_TABLE_SIZE];
 Uint32          udp_socket_hash;
 StatsAddrEntry  *udp_socket_hash_link;
 StatsAddrEntry  *udp_previous_socket_hash_link;

 StatsAddrEntry  *udp_dialog_hash_table[UDP_DIALOG_HASH_TABLE_SIZE];
 Uint32          udp_dialog_hash;
 StatsAddrEntry  *udp_dialog_hash_link;
 StatsAddrEntry  *udp_previous_dialog_hash_link;


/*****************************************************************************
 *
 * Look for the segment address structure.
 * If no match is found, NULL is returned.
 */

 StatsAddrEntry *stats_udp_lookup_segment (segment)
```

```
    Uint32                  segment;
{
    register StatsAddrEntry       *seg_addr_ptr;

seg_addr_ptr = (StatsAddrEntry *) statsUdpSegQ.pFlink;

while (seg_addr_ptr != NULL)
    {
    if (seg_addr_ptr->address.segment1 == segment)
        break;
    seg_addr_ptr = (StatsAddrEntry *) seg_addr_ptr->link.pFlink;
    }

    return (seg_addr_ptr);
}


/*****************************************************************************
 *
 * Find the structure for keeping Udp segment statistics for the given
 * segment.  If one is not found, attempt to allocate one.
 */

    StatsAddrEntry *stats_udp_get_segment (segment)

    Uint32    segment;
{
    register StatsAddrEntry       *seg_addr_ptr;
    register StatsUdpSegment *seg_stats_ptr;

 seg_addr_ptr = stats_udp_lookup_segment (segment);

/*
 * If not found, try to allocate a structure for this segment.
 * If a structure can't be obtained, count this as a drop.
 */
```

```
 if (seg_addr_ptr == NULL)
     {
     seg_addr_ptr = (StatsAddrEntry *) stats_allocate ( sizeof(StatsAddrEntry) );
     if (seg_addr_ptr != NULL)
          {
          seg_addr_ptr->address.addressType = MibSegment1;
          seg_addr_ptr->address.segment1 = segment;
          seg_addr_ptr->parse_control = PARSE_CONTROL_DEFAULT;
          seg_addr_ptr->startTime = stats_start_time.tv_sec;
          seg_addr_ptr->lastTime = stats_start_time.tv_sec;

          FBInsqt (&statsUdpSegQ, (PFBQentry_type) seg_addr_ptr);
          }
     else
          {
          stats_mon_udp_dropped;
          return (NULL);
          }
     }

/*
 * There's an address structure. See if there's a statistics structure
 * for Udp attached, if not and parse control allows, allocate one.
 */
seg_stats_ptr = (StatsUdpSegment *) seg_addr_ptr->stats_ptr;
if (seg_stats_ptr == NULL)
     {
     /*
      * If Udp parsing is enabled, allocate Udp statistics structure if
      * one has not already been allocated.  If a structure can't be obtained,
      * count this as a drop.
      */
     if (seg_addr_ptr->parse_control & MibParseUdp)
          {
          seg_stats_ptr = (StatsUdpSegment *) stats_allocate (sizeof (StatsUdpSegment) );

          if (seg_stats_ptr != NULL)
```

```
                {
                seg_addr_ptr->stats_ptr = (Uint32 *) seg_stats_ptr;
                seg_stats_ptr->frameRate.type                    = STATS_RATE_10S;
                seg_stats_ptr->byteRate.type                     = STATS_RATE_10S;
                seg_stats_ptr->errorRate.type                    = STATS_RATE_10S;
                seg_stats_ptr->rcvOffSegRate.type                = STATS_RATE_10S;
                seg_stats_ptr->xmtOffSegRate.type                = STATS_RATE_10S;
                seg_stats_ptr->transitRate.type                      = STATS_RATE_10S;
                seg_stats_ptr->flowCtrlRate.type                 = STATS_RATE_10S;
                seg_stats_ptr->frgmtRate.type                    = STATS_RATE_10S;

                setUdpSegDflts (&mibSegDefaults.mibUdpSegDefaults, seg_stats_ptr);
                }
            else
                stats_mon_udp_dropped;
            }
        }

return (seg_addr_ptr);
}



/*************************************************************************
 *
 *  Find the udp ip address record.
 *  A hash is done on the ip address.  A pointer to the first
 *  StatsAddrEntry with the same hash is found.  Structures with the
 *  same hash are linked.  The link must be walked and the ip address
 *  compared with the those in each of the structures until a match
 *  is found.  If no match is found, NULL is returned.
 */
StatsAddrEntry *stats_udp_lookup_addr (ip_addr)

    Uint32          *ip_addr;
{
```

```
    Uint32          i;

/*
 * Compute ip address hash to get index into hash table
 */
udp_hash = (*ip_addr & 0x0000ffff);
udp_hash = ((udp_hash + ((udp_hash & 0xff00) >> 8)) & (UDP_HASH_TABLE_SIZE - 1));

udp_previous_hash_link = NULL;
udp_hash_link = (StatsAddrEntry *) udp_hash_table[udp_hash];

/*
 * Walk linked list for exact entry
 */
while (udp_hash_link != NULL)
    {
    if (udp_hash_link->address.netAddress1.u.ipAddress == *ip_addr)
        return (udp_hash_link);
    else
        {
        udp_previous_hash_link = udp_hash_link;
        udp_hash_link = udp_hash_link->hash_link;
        }
    }

/*
 * No entry found.
 */
return (NULL);
}



/*****************************************************************************
 *
```

```
 * Allocate a stats structure if parse control is turned on.
 *
 */

 Uint32 stats_udp_get_stats (udp_addr_record_ptr)
     StatsAddrEntry     *udp_addr_record_ptr;
 {
     register StatsUdpAddr         *udp_addr_stats_ptr;

     if ((udp_addr_record_ptr != NULL) && (udp_addr_record_ptr->stats_ptr == NULL))
     {
          /*
           * If parse control is turned on for Udp, allocate a structure for Udp
           * address statistics and initialize it.  If parse control is turned on,
           * but a structure can't be obtained, return FALSE
           */
          if (udp_addr_record_ptr->parse_control & MibParseUdp)
          {
               udp_addr_record_ptr->stats_ptr = (Uint32 *) stats_allocate (sizeof
(StatsUdpAddr) );
               udp_addr_stats_ptr = (StatsUdpAddr *) udp_addr_record_ptr->stats_ptr;
               if (udp_addr_stats_ptr != NULL)
               {
               udp_addr_stats_ptr->common.frameRate.type                    =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.rcvFrameRate.type            =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.xmtFrameRate.type            =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.byteRate.type                    =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.rcvByteRate.type                 =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.xmtByteRate.type                 =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.errorRate.type                   =
STATS_RATE_10S;
```

```
                    udp_addr_stats_ptr->common.rcvErrorRate.type            =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.xmtErrorRate.type            =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.rcvOffSegRate.type               =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.xmtOffSegRate.type               =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.rcvFrgmtRate.type            =
STATS_RATE_10S;
                    udp_addr_stats_ptr->common.xmtFrgmtRate.type            =
STATS_RATE_10S;

                    InitqH   (&udp_addr_stats_ptr->protocolQ);
                    FBInitqh (&udp_addr_stats_ptr->dialogQ);

                    setUdpAddrDflts (&mibNodeDefaults.mibUdpAddrDefaults,
&udp_addr_stats_ptr->common);
                }
                else
                    return (FALSE);
            }
        }
        return (TRUE);
    }

/*************************************************************************
 *
 * Allocate a socket stats structure if parse control is turned on.
 *
 */

Uint32 stats_udp_get_socket_stats (udp_addr_record_ptr)
    StatsAddrEntry     *udp_addr_record_ptr;
{
    register StatsUdpSocket       *udp_socket_stats_ptr;
```

```
    if ((udp_addr_record_ptr != NULL) && (udp_addr_record_ptr->stats_ptr == NULL))
    {
        /*
         * If parse control is turned on for Udp, allocate a structure for Udp
         * socket statistics and initialize it.  If parse control is turned on,
         * but a structure can't be obtained, return FALSE
         */
        if (udp_addr_record_ptr->parse_control & MibParseUdp)
        {
            udp_addr_record_ptr->stats_ptr = (Uint32 *) stats_allocate (sizeof
(StatsUdpSocket) );
            udp_socket_stats_ptr = (StatsUdpSocket *) udp_addr_record_ptr->stats_ptr;
            if (udp_socket_stats_ptr != NULL)
            {
            udp_socket_stats_ptr->common.frameRate.type                =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.rcvFrameRate.type             =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.xmtFrameRate.type             =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.byteRate.type                 =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.rcvByteRate.type              =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.xmtByteRate.type              =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.errorRate.type                =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.rcvErrorRate.type             =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.xmtErrorRate.type             =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.rcvOffSegRate.type            =
STATS_RATE_10S;
                udp_socket_stats_ptr->common.xmtOffSegRate.type            =
STATS_RATE_10S;
```

```
                    udp_socket_stats_ptr->common.rcvFrgmtRate.type              =
STATS_RATE_10S;
                    udp_socket_stats_ptr->common.xmtFrgmtRate.type              =
STATS_RATE_10S;

                    FBInitqh (&udp_socket_stats_ptr->dialogQ);

                    setUdpAddrDflts (&mibNodeDefaults.mibUdpAddrDefaults,
&udp_socket_stats_ptr->common);
               }
               else
                    return (FALSE);
          }
     }
     return (TRUE);
 }



/*****************************************************************************
 *
 * Find the structure for keeping udp address statistics for the given
 * ip address.  If one not found, attempt to allocate one.
 */

 StatsAddrEntry *stats_udp_get_addr (ip_addr)

     Uint32              *ip_addr;
{
     register StatsAddrEntry       *udp_addr_record_ptr;
     register StatsUdpAddr         *udp_addr_stats_ptr;
     register StatsAddrEntry       *ip_addr_record_ptr;


 udp_addr_record_ptr = stats_udp_lookup_addr (ip_addr);
```

```
 if (udp_addr_record_ptr == NULL)
      {
      /*
       * Try to allocate a statistics structure for this address.
       * If udp is turned on for this ip address, but a structure can't
       * be obtained, count this as a drop.  If udp is not turned on, just
       * return NULL.
       */
      ip_addr_record_ptr = stats_ip_lookup_addr (ip_addr);
      if (ip_addr_record_ptr == NULL)
           return (NULL);

      if ((ip_addr_record_ptr->parse_control & MibParseUdp) != MibParseUdp)
           return (NULL);

      udp_addr_record_ptr = (StatsAddrEntry *)stats_allocate (sizeof (StatsAddrEntry) );
      if (udp_addr_record_ptr == NULL)
           {
           stats_mon_udp_dropped;
           return (NULL);
           }
      else
           {
           udp_addr_record_ptr->hash_link                                    = NULL;
           udp_addr_record_ptr->startTime                                    =
stats_start_time.tv_sec;
           udp_addr_record_ptr->lastTime                                     =
stats_start_time.tv_sec;
           udp_addr_record_ptr->address.addressType                    = MibNetAddress1 |
MibSegment1;
           udp_addr_record_ptr->address.netAddress1.netAddressType = NetTcpIp;
           udp_addr_record_ptr->address.netAddress1.length         = 4;
           udp_addr_record_ptr->address.netAddress1.u.ipAddress    = *ip_addr;

           udp_addr_record_ptr->parse_control = ip_addr_record_ptr->parse_control;
           udp_addr_record_ptr->address.segment1 = ip_addr_record_ptr->address.segment1;
```

```
            FBInsqt (&statsUdpAddrQ, (PFBQentry_type) udp_addr_record_ptr);

            /*
             * Allocate a stats structure if parse control is turned on.
             * stats_udp_get_stats will return FALSE, so count this as a drop.
             */
            if (stats_udp_get_stats (udp_addr_record_ptr) == FALSE)
                stats_mon_udp_dropped;
            }

        /*
         * Put this udp address record into the hash table.
         * Variable hash was set when stats_udp_lookup_addr was executed.
         */
        if ( udp_hash_table[udp_hash] == NULL)
            udp_hash_table[udp_hash] = udp_addr_record_ptr;
        else
            {
            /*
             * Find the last structure of the hash link
             */
            udp_hash_link = udp_hash_table[udp_hash];

            while (udp_hash_link->hash_link != NULL)
                udp_hash_link = udp_hash_link->hash_link;

            udp_hash_link->hash_link = udp_addr_record_ptr;
            }
        }

return (udp_addr_record_ptr);
}
```

```
/*****************************************************************************
 *
 *  Find the udp socket record.
 *  A hash is done on the ip address and port number.  A pointer to the first
 *  StatsAddrEntry with the same hash is found.  Structures with the
 *  same hash are linked.  The link must be walked and the ip address and port
 *  compared with the those in each of the structures until a match
 *  is found.  If no match is found, NULL is returned.
 */
StatsAddrEntry *stats_udp_lookup_socket (ip_addr, port)

     Uint32          *ip_addr;
     Uint32          port;
{
     Uint32          i;

/*
 * Compute ip address hash to get index into hash table
 */
udp_socket_hash = (*ip_addr & 0x0000ffff) + ((port & 0x0000ffff) << 16);
udp_socket_hash = ((udp_socket_hash + ((udp_socket_hash & 0xff00) >> 8)) &
                   (UDP_SOCKET_HASH_TABLE_SIZE - 1));


udp_previous_socket_hash_link = NULL;
udp_socket_hash_link = (StatsAddrEntry *) udp_socket_hash_table[udp_socket_hash];

/*
 * Walk linked list for exact entry
 */
while (udp_socket_hash_link != NULL)
     {
     if ((udp_socket_hash_link->address.port1 == port) &&
          (udp_socket_hash_link->address.netAddress1.u.ipAddress == *ip_addr) )
          return (udp_socket_hash_link);
     else
          {
```

```
            udp_previous_socket_hash_link = udp_socket_hash_link;
            udp_socket_hash_link = udp_socket_hash_link->hash_link;
            }
        }


/*
 * No entry found.
 */
return (NULL);
}



/*************************************************************************
 *
 * Find the structure for keeping udp statistics for the given
 * ip address and port.  If one not found, attempt to allocate one.
 */

StatsAddrEntry *stats_udp_get_socket (ip_addr, port)

    Uint32              *ip_addr;
    Uint32              port;
{
    register StatsAddrEntry     *udp_socket_addr_ptr;
    register StatsUdpAddr       *udp_addr_stats_ptr;
    register StatsAddrEntry     *ip_addr_record_ptr;

/*
 * Don't bother keeping stats for ports that aren't well-known yet.
 */
if (!stats_well_known_port (port) )
    return (NULL);

udp_socket_addr_ptr = stats_udp_lookup_socket (ip_addr, port);
```

```
    if (udp_socket_addr_ptr == NULL)
        {
        /*
         * Try to allocate a statistics structure for this address.
         * If udp is turned on for this ip address, but a structure can't
         * be obtained, count this as a drop.  If udp is not turned on, just
         * return NULL.
         */
        ip_addr_record_ptr = stats_ip_lookup_addr (ip_addr);
        if (ip_addr_record_ptr == NULL)
            return (NULL);
        if ((ip_addr_record_ptr->parse_control & MibParseUdp) == NULL)
            return (NULL);

        udp_socket_addr_ptr = (StatsAddrEntry *)stats_allocate (sizeof (StatsAddrEntry) );
        if (udp_socket_addr_ptr == NULL)
            {
            stats_mon_udp_dropped;
            return (NULL);
            }
        else
            {
            udp_socket_addr_ptr->hash_link = NULL;
            udp_socket_addr_ptr->startTime = stats_start_time.tv_sec;
            udp_socket_addr_ptr->lastTime = stats_start_time.tv_sec;

            udp_socket_addr_ptr->address.addressType = MibNetAddress1 | MibSegment1 |
MibPort1;
            udp_socket_addr_ptr->address.netAddress1.netAddressType = NetTcpIp;
            udp_socket_addr_ptr->address.netAddress1.length = 4;
            udp_socket_addr_ptr->address.netAddress1.u.ipAddress = *ip_addr;
            udp_socket_addr_ptr->address.port1 = port;

            udp_socket_addr_ptr->parse_control = ip_addr_record_ptr->parse_control;
            udp_socket_addr_ptr->address.segment1 = ip_addr_record_ptr->address.segment1;
```

```
            FBInsqt (&statsUdpSocketQ, (PFBQentry_type) udp_socket_addr_ptr);

          /*
             * Allocate a stats structure if parse control is turned on.
             * stats_udp_get_socket_stats will return FALSE, so count this as a drop.
             */
             if (stats_udp_get_stats (udp_socket_addr_ptr) == FALSE)
               stats_mon_udp_dropped;
          }

       /*
        * Put this udp socket into the hash table.
        * Variable udp_socket_hash was set when stats_udp_lookup_socket was executed.
        */
       if ( udp_socket_hash_table[udp_socket_hash] == NULL)
            udp_socket_hash_table[udp_socket_hash] = udp_socket_addr_ptr;
       else
            {
            /*
             * Find the last structure of the hash link
             */
            udp_socket_hash_link = udp_socket_hash_table[udp_socket_hash];

            while (udp_socket_hash_link->hash_link != NULL)
                 udp_socket_hash_link = udp_socket_hash_link->hash_link;

            udp_socket_hash_link->hash_link = udp_socket_addr_ptr;
            }
       }

return (udp_socket_addr_ptr);
}
```

```
/************************ stats_udp_lookup_dialog ******************************
 *
 * Find a udp dialog given 2 ip addresses.
 * If the dialog is not found, NULL is returned.
 */

StatsAddrEntry *stats_udp_lookup_dialog (ip_addr1, port1, ip_addr2, port2)

     Uint32                    *ip_addr1, *ip_addr2;
     Uint32                    port1, port2;
{
     register Uint32                xudp_dialog_hash;
     register StatsAddrEntry        *xudp_dialog_hash_link;
     register StatsAddrEntry        *xudp_previous_dialog_hash_link;

     register StatsAddrEntry        *dialog_addr_ptr;
/*
 * Compute hash function on both sockets
 */
xudp_dialog_hash = (*ip_addr1 & 0x0000ffff) + ((port1 & 0x0000ffff) << 16) +
                   (*ip_addr2 & 0x0000ffff) + ((port2 & 0x0000ffff) << 16);
xudp_dialog_hash = ((xudp_dialog_hash + ((xudp_dialog_hash & 0xff00) >> 8))
                     & (UDP_DIALOG_HASH_TABLE_SIZE - 1));

xudp_previous_dialog_hash_link = NULL;
xudp_dialog_hash_link = (StatsAddrEntry *) udp_dialog_hash_table[xudp_dialog_hash];

/*
 * Walk linked list for exact entry
 */
while (xudp_dialog_hash_link != NULL)
     {
     if (((xudp_dialog_hash_link->address.port1 == port1) &&
          (xudp_dialog_hash_link->address.port2 == port2) &&
          (xudp_dialog_hash_link->address.netAddress1.u.ipAddress == *ip_addr1) &&
          (xudp_dialog_hash_link->address.netAddress2.u.ipAddress == *ip_addr2) ) ||
```

```
            ((xudp_dialog_hash_link->address.port1 == port2) &&
             (xudp_dialog_hash_link->address.port2 == port1) &&
             (xudp_dialog_hash_link->address.netAddress1.u.ipAddress == *ip_addr2) &&
             (xudp_dialog_hash_link->address.netAddress2.u.ipAddress == *ip_addr1) ))
                 {
                 break;
                 }
            else
                 {
                 xudp_previous_dialog_hash_link = xudp_dialog_hash_link;
                 xudp_dialog_hash_link = xudp_dialog_hash_link->hash_link;
                 }
        }

    udp_dialog_hash = xudp_dialog_hash;
    udp_dialog_hash_link = xudp_dialog_hash_link;
    udp_previous_dialog_hash_link = xudp_previous_dialog_hash_link;

    return (xudp_dialog_hash_link);
    }


    /************************ stats_udp_get_dialog ******************************
     *
     * Find or allocate an udp dialog given 2 sockets.
     * If the dialog is not found, attempt to allocate a structure.
     */
    StatsAddrEntry *stats_udp_get_dialog (ip_addr1, port1, ip_addr2, port2)

        Uint32                  *ip_addr1, *ip_addr2;
        Uint32                  port1, port2;
    {
        register StatsAddrEntry       *dialog_addr_ptr;
        register StatsDialogEntry     *dialog_stats_ptr;
        register StatsDialogLink *dialog_link;
        register StatsAddrEntry       *udp_addr1_record_ptr, *udp_addr2_record_ptr;
```

```
    register StatsUdpAddr          *udp_addr1_stats_ptr, *udp_addr2_stats_ptr;
    register StatsAddrEntry        *udp_socket1_record_ptr, *udp_socket2_record_ptr;
    register StatsUdpSocket        *udp_socket1_stats_ptr, *udp_socket2_stats_ptr;
    register StatsAddrEntry        *ip_addr1_record_ptr, *ip_addr2_record_ptr;
    register Uint32                 parse_control;


dialog_addr_ptr = stats_udp_lookup_dialog (ip_addr1, port1, ip_addr2, port2);
if (dialog_addr_ptr != NULL)
    return (dialog_addr_ptr);

/*
 * Check parse control for the ip addresses.
 */
ip_addr1_record_ptr = stats_ip_lookup_addr (ip_addr1);
ip_addr2_record_ptr = stats_ip_lookup_addr (ip_addr2);

if ( (ip_addr1_record_ptr == NULL) || (ip_addr2_record_ptr == NULL) )
    return (NULL);

if (ip_addr1_record_ptr != NULL)
    parse_control = ip_addr1_record_ptr->parse_control;
if (ip_addr2_record_ptr != NULL)
    parse_control |= ip_addr2_record_ptr->parse_control;


/*
 * If udp is turned on for these ip addresses, allocate structures
 * for the dialog.  If can't get them, count this as a drop.
 */
if ((parse_control & MibParseUdp) != MibParseUdp)
    return (NULL);

/*
 * Try to allocate structures for this dialog.
 * If they can't be obtained, count this as a drop.
 */
```

```
dialog_addr_ptr = (StatsAddrEntry *) stats_allocate (sizeof (StatsAddrEntry) );
if (dialog_addr_ptr == NULL)
     {
     stats_mon_udp_dropped;
     return (NULL);
     }

dialog_stats_ptr = (StatsDialogEntry *) stats_allocate (sizeof (StatsDialogEntry) );
if (dialog_stats_ptr == NULL)
     {
     stats_deallocate (dialog_addr_ptr, sizeof(StatsAddrEntry) );
     stats_mon_udp_dropped;
     return (NULL);
     }


/*
 * Initialize the structures
 */
dialog_addr_ptr->hash_link                                    = NULL;
dialog_addr_ptr->address.addressType                  = MibNetAddress1 | MibPort1 |
                                                                MibNetAddress2 |
NibPort2;
dialog_addr_ptr->address.netAddress1.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress1.length       = 4;
dialog_addr_ptr->address.netAddress1.u.ipAddress  = *ip_addr1;
dialog_addr_ptr->address.port1                                = port1;
dialog_addr_ptr->address.netAddress2.netAddressType = NetTcpIp;
dialog_addr_ptr->address.netAddress2.length       = 4;
dialog_addr_ptr->address.netAddress2.u.ipAddress  = *ip_addr2;
dialog_addr_ptr->address.port2                                = port2;
dialog_addr_ptr->parse_control                                = parse_control;
dialog_addr_ptr->startTime                                    = stats_start_time.tv_sec;
dialog_addr_ptr->lastTime                                     = stats_start_time.tv_sec;
dialog_addr_ptr->stats_ptr                                    = (Uint32 *) dialog_stats_ptr;
```

```
FBInsqt (&statsUdpDialogQ, (PFBQentry_type) dialog_addr_ptr);

dialog_stats_ptr->packetRate.type  = STATS_RATE_10S;
dialog_stats_ptr->byteRate.type    = STATS_RATE_10S;
dialog_stats_ptr->errorRate.type   = STATS_RATE_10S;
dialog_stats_ptr->fragmentRate.type = STATS_RATE_10S;
dialog_stats_ptr->rexmtRate.type   = STATS_RATE_10S;
dialog_stats_ptr->flowCtrlRate.type = STATS_RATE_10S;

dialog_stats_ptr->transport.transportProtocol = UDP_PROTOCOL;
dialog_stats_ptr->transport.applicationProtocol = udp_protocol;
dialog_stats_ptr->transport.initiator = ConnectionInitiatorUnknown;
dialog_stats_ptr->transport.state = ConnectionStateUnknown;
dialog_stats_ptr->transport.closeReason = ConnectionCloseUnknown;

setUdpAddrPairDflts (&mibNodeDefaults.mibUdpAddrPairDefaults, dialog_stats_ptr);

/*
 * Link the dialog statistics into the dialog queue for each udp address stats.
 */
udp_addr1_record_ptr = stats_udp_lookup_addr (ip_addr1);
udp_addr2_record_ptr = stats_udp_lookup_addr (ip_addr2);

if (udp_addr1_record_ptr != NULL)
     {
     if (udp_addr1_record_ptr->stats_ptr != NULL)
          {
          dialog_addr_ptr->address.addressType = MibSegment1;
          dialog_addr_ptr->address.segment1  = udp_addr1_record_ptr->address.segment1;
          udp_addr1_stats_ptr                = (StatsUdpAddr *)
udp_addr1_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&udp_addr1_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
               }
```

```
          }
     }
if (udp_addr2_record_ptr != NULL)
     {
     if (udp_addr2_record_ptr->stats_ptr != NULL)
          {
          dialog_addr_ptr->address.addressType = MibSegment2;
          dialog_addr_ptr->address.segment2  = udp_addr2_record_ptr->address.segment1;
          udp_addr2_stats_ptr                      = (StatsUdpAddr *)
udp_addr2_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&udp_addr2_stats_ptr->dialogQ, (PFBQentry_type)dialog_link);
               }
          }
     }


/*
 * Link the dialog statistics into the dialog queue for each of
 * the sockets.
 */
udp_socket1_record_ptr = stats_udp_lookup_socket (ip_addr1, port1);
udp_socket2_record_ptr = stats_udp_lookup_socket (ip_addr2, port2);

if (udp_socket1_record_ptr != NULL)
     {
     if (udp_socket1_record_ptr->stats_ptr != NULL)
          {
          udp_socket1_stats_ptr =(StatsUdpSocket*)udp_socket1_record_ptr->stats_ptr;
          dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
          if (dialog_link != NULL)
               {
               dialog_link->dialog_addr_ptr = dialog_addr_ptr;
               FBInsqt (&udp_socket1_stats_ptr->dialogQ,(PFBQentry_type)dialog_link);
```

```
                }
            }
        }
    if (udp_socket2_record_ptr != NULL)
        {
        if (udp_socket2_record_ptr->stats_ptr != NULL)
            {
            udp_socket2_stats_ptr =(StatsUdpSocket*)udp_socket2_record_ptr->stats_ptr;
            dialog_link = (StatsDialogLink *)stats_allocate(sizeof(StatsDialogLink));
            if (dialog_link != NULL)
                {
                dialog_link->dialog_addr_ptr = dialog_addr_ptr;
                FBInsqT (&udp_socket2_stats_ptr->dialogQ,(PFBQentry_type)dialog_link);
                }
            }
        }

    /*
     * Put the new dialog address structure in the udp dialog hash table.
     * Variable hash was set up when stats_udp_lookup_dialog was executed.
     */
    if (udp_dialog_hash_table[udp_dialog_hash] == NULL)
        udp_dialog_hash_table[udp_dialog_hash] = dialog_addr_ptr;
    else
        {
        /*
         * Find the last structure of the hash link.
         */
        udp_dialog_hash_link = udp_dialog_hash_table[udp_dialog_hash];
        while (udp_dialog_hash_link->hash_link != NULL)
            udp_dialog_hash_link = udp_dialog_hash_link->hash_link;

        udp_dialog_hash_link->hash_link = dialog_addr_ptr;
        }

    return (dialog_addr_ptr);
    }
```

```
void setUdpAddrDflts (mib_udp_addr_defs, udp_stats_common_ptr)
     MibUdpAddrDefaults  *mib_udp_addr_defs;
     StatsUdpCommon      *udp_stats_common_ptr;
{
     udp_stats_common_ptr->rcvFrames.high_thld =
                              mib_udp_addr_defs->udpAddrRcvPktHighThld;
     udp_stats_common_ptr->rcvFrameRate.high_thld =
                              mib_udp_addr_defs->udpAddrRcvPktRateHighThld;

     udp_stats_common_ptr->rcvBytes.high_thld =
                              mib_udp_addr_defs->udpAddrRcvByteHighThld;
     udp_stats_common_ptr->rcvByteRate.high_thld =
                              mib_udp_addr_defs->udpAddrRcvByteRateHighThld;

     udp_stats_common_ptr->rcvErrors.high_thld =
                              mib_udp_addr_defs->udpAddrRcvErrorHighThld;
     udp_state_common_ptr->rcvErrorRate.high_thld =
                              mib_udp_addr_defs->udpAddrRcvErrorRateHighThld;

     udp_stats_common_ptr->xmtFrames.high_thld =
                              mib_udp_addr_defs->udpAddrXmtPktHighThld;
     udp_stats_common_ptr->xmtFrameRate.high_thld =
                              mib_udp_addr_defs->udpAddrXmtPktRateHighThld;

     udp_stats_common_ptr->xmtBytes.high_thld =
                              mib_udp_addr_defs->udpAddrXmtByteHighThld;
     udp_stats_common_ptr->xmtByteRate.high_thld =
                              mib_udp_addr_defs->udpAddrXmtByteRateHighThld;

     udp_stats_common_ptr->xmtErrors.high_thld =
                              mib_udp_addr_defs->udpAddrXmtErrorHighThld;
     udp_stats_common_ptr->xmtErrorRate.high_thld =
                              mib_udp_addr_defs->udpAddrXmtErrorRateHighThld;

     udp_stats_common_ptr->rcvOffSegs.high_thld =
```

```
                                mib_udp_addr_defs->udpAddrRcvOffSegHighThld;
        udp_stats_common_ptr->rcvOffSegRate.high_thld =
                                mib_udp_addr_defs->udpAddrRcvOffSegRateHighThld;

        udp_stats_common_ptr->xmtOffSegs.high_thld =
                                mib_udp_addr_defs->udpAddrXmtOffSegHighThld;
        udp_stats_common_ptr->xmtOffSegRate.high_thld =
                                mib_udp_addr_defs->udpAddrXmtOffSegRateHighThld;

        udp_stats_common_ptr->rcvFrgmts.high_thld =
                                mib_udp_addr_defs->udpAddrRcvFrgmtHighThld;
        udp_stats_common_ptr->rcvFrgmtRate.high_thld =
                                mib_udp_addr_defs->udpAddrRcvFrgmtRateHighThld;

        udp_stats_common_ptr->xmtFrgmts.high_thld =
                                mib_udp_addr_defs->udpAddrXmtFrgmtHighThld;
        udp_stats_common_ptr->xmtFrgmtRate.high_thld =
                                mib_udp_addr_defs->udpAddrXmtFrgmtRateHighThld;
}

void setUdpAddrPairDflts (mib_udp_addr_pair_defs, dialog_ptr)
        MibUdpAddrPairDefaults    *mib_udp_addr_pair_defs;
        StatsDialogEntry          *dialog_ptr;
{
        dialog_ptr->packets.high_thld =
                                mib_udp_addr_pair_defs->udpAddrPairPktHighThld;
        dialog_ptr->packetRate.high_thld =
                                mib_udp_addr_pair_defs->udpAddrPairPktRateHighThld;

        dialog_ptr->bytes.high_thld =
                                mib_udp_addr_pair_defs->udpAddrPairByteHighThld;
        dialog_ptr->byteRate.high_thld =
                                mib_udp_addr_pair_defs->udpAddrPairByteRateHighThld;

        dialog_ptr->errors.high_thld =
                                mib_udp_addr_pair_defs->udpAddrPairErrorHighThld;
```

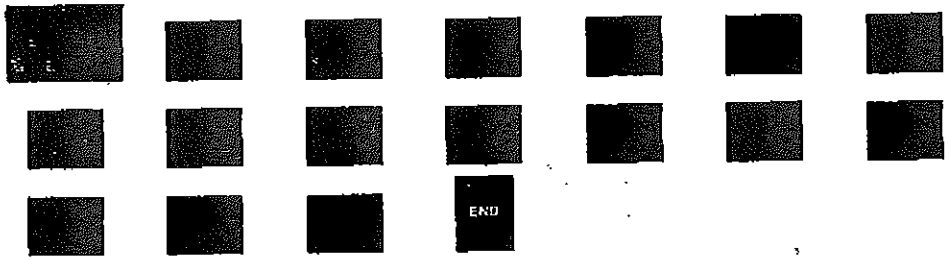
```
        dialog_ptr->errorRate.high_thld =
                        mib_udp_addr_pair_defs->udpAddrPairErrorRateHighThld;

        dialog_ptr->fragments.high_thld =
                        mib_udp_addr_pair_defs->udpAddrPairFrgmtHighThld;
        dialog_ptr->fragmentRate.high_thld =
                        mib_udp_addr_pair_defs->udpAddrPairFrgmtRateHighThld;

        dialog_ptr->rexmts.high_thld =
                        mib_udp_addr_pair_defs->udpAddrPairRexmtHighThld;
        dialog_ptr->rexmtRate.high_thld =
                        mib_udp_addr_pair_defs->udpAddrPairRexmtRateHighThld;

        dialog_ptr->flowCtrls.high_thld =
                        mib_udp_addr_pair_defs->udpAddrPairFlowCtrlHighThld;
        dialog_ptr->flowCtrlRate.high_thld =
                        mib_udp_addr_pair_defs->udpAddrPairFlowCtrlRateHighThld;
}

void setUdpSegDflts (mib_udp_seg_defs, udp_seg_ptr)
        MibUdpSegDefaults    *mib_udp_seg_defs;
        StatsUdpSegment          *udp_seg_ptr;
{
        udp_seg_ptr->frames.high_thld =
                        mib_udp_seg_defs->udpSegPktHighThld;
        udp_seg_ptr->frameRate.high_thld =
                        mib_udp_seg_defs->udpSegPktRateHighThld;

        udp_seg_ptr->bytes.high_thld =
                        mib_udp_seg_defs->udpSegByteHighThld;
        udp_seg_ptr->byteRate.high_thld =
                        mib_udp_seg_defs->udpSegByteRateHighThld;

        udp_seg_ptr->errors.high_thld =
                        mib_udp_seg_defs->udpSegErrorHighThld;
        udp_seg_ptr->errorRate.high_thld =
```

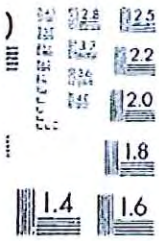
```
                        mib_udp_seg_defs->udpSegErrorRateHighThld;

        udp_seg_ptr->rcvOffSegs.high_thld =
                        mib_udp_seg_defs->udpSegRcvOffSegHighThld;
        udp_seg_ptr->rcvOffSegRate.high_thld =
                        mib_udp_seg_defs->udpSegRcvOffSegRateHighThld;

        udp_seg_ptr->xmtOffSegs.high_thld =
                        mib_udp_seg_defs->udpSegXmtOffSegHighThld;
        udp_seg_ptr->xmtOffSegRate.high_thld =
                        mib_udp_seg_defs->udpSegXmtOffSegRateHighThld;

        udp_seg_ptr->transits.high_thld =
                        mib_udp_seg_defs->udpSegTransitHighThld;
        udp_seg_ptr->transitRate.high_thld =
                        mib_udp_seg_defs->udpSegTransitRateHighThld;

        udp_seg_ptr->frgmts.high_thld =
                        mib_udp_seg_defs->udpSegFrgntHighThld;
        udp_seg_ptr->frgmtRate.high_thld =
                        mib_udp_seg_defs->udpSegFrgmtRateHighThld;
}
```

```
/*
 *  stats_p.c
 *
 *   (description)
 *
 *
 *        Copyright (c) 1991 Concord Communications Inc.
 *                     All rights reserved.
 *
 *
 *  Path:       /home/hawk4/malibu/trakker_db/monitor/stats/SCCS/s.stats_p.c
 *
 *  Date:       8/8/91
 *
 *  Revision:   1.9
 *
 *
 *  Changes:
 *
 *   MM-DD-YY  WHO        Description of change. (latest first)
 *   --------  ---        ---------------------------------------------
 *
 *   07-22-91  KR         Added stats_program and stats_port_to_protocol
 *   07-16-91  DPD        Changed stats_check_for_duplicate_ip to return Uint32
 *   06-07-91  KR         Added more well known ports
 */

static char stats_p_c [] = "@(#)stats_p.c    1.9";

#include <stdio.h>

#include <cci_std.h>

#include "system.h"
#include "address.h"
#include <sys/types.h>
#include <sys/socket.h>
```

NETWORK MONITORING
Ferdinand Engel, Kendall S. Jones,
Mary Robertson, David M. Thompson and
Gerard White

14





NETWORK MONITORING

Ferdinand Engel, Kendall S. Jones,
Kary Robertson, David M. Thompson and
Gerald White

```
#include <bsd43/sys/time.h>
#include </usr/include/bsd43/time.h>
#include "util.h"
#include "kuser.h"
#include "lan_82596.h"
#include "lan_i596_driver.h"
#include "drivers.h"
#ifdef unix
#include <sys/cci.h>
#endif
#include "lanutil.h"
#include "rtp.h"
#include "stats.h"
#include "stats_dll.h"
#include "stats_ip.h"
#include "stats_icmp.h"
#include "stats_tcp.h"
#include "stats_udp.h"
#include "stats_rpc.h"
#include "stats_nfs.h"
#include "stats_pmap.h"
#include "stats_mib.h"
#include "mib_alarms.h"
#include "alarms.h"
#include "mbuf.h"
#include "em_ctrl.h"


struct timeval start_time;
struct timeval end_time;


/*
 * Global data structures
 */
struct timeval      stats_start_time;
```

```
Uint32                  stats_tiny_size;
Uint32                  stats_very_small_size;
Uint32                  stats_small_size;
Uint32                  stats_medium_size;
Uint32                  stats_large_size;
Uint32                  stats_very_large_size;

Qhead_type              statsTinyQ;
Qhead_type              statsVerySmallQ;
Qhead_type              statsSmallQ;
Qhead_type              statsMediumQ;
Qhead_type              statsLargeQ;
Qhead_type              statsVeryLargeQ;

StatsAddrEntry      statsMonitorAddr;
StatsMonitorType    statsMonitor;
FBQhead_type        statsDllSegQ;
FBQhead_type        statsIpSegQ;
FBQhead_type        statsIcmpSegQ;
FBQhead_type        statsUdpSegQ;
FBQhead_type        statsTcpSegQ;
FBQhead_type        statsNfsSegQ;

FBQhead_type        statsMacAddrQ;
FBQhead_type        statsIpAddrQ;
FBQhead_type        statsIcmpAddrQ;
FBQhead_type        statsUdpAddrQ;
FBQhead_type        statsTcpAddrQ;
FBQhead_type        statsNfsClientQ;

FBQhead_type        statsUdpSocketQ;
FBQhead_type        statsTcpSocketQ;
FBQhead_type        statsNfsServerQ;

FBQhead_type        statsDllDialogQ;
FBQhead_type        statsIpDialogQ;
```

```
FBQhead_type        statsIcmpDialogQ;
FBQhead_type        statsUdpDialogQ;
FBQhead_type        statsTcpDialogQ;
FBQhead_type        statsNfsDialogQ;

FBQhead_type        statsNfsFileSystemQ;

MibAddress          statsDfltAddr;
MibSegmentDefaults  mibSegDefaults;
MibNodeDefaults          mibNodeDefaults;
VarBind                  *statsVbPtr;

Uint32                   mySegmentId;


/*************************************************************************
 *
 * Routine to update counters of type StatsCount32
 */
void stats_count (counter_ptr, rate_ptr, structure_ptr, alarm_type, alarm_number,
user_ptr)

     StatsCount32        *counter_ptr;
     StatsRollingRate    *rate_ptr;
     StatsAddrEntry      *structure_ptr;
     Uint32                   alarm_type;
     Uint32                   alarm_number;
     Byte                *user_ptr;
{

if (structure_ptr == 0)
     return;

counter_ptr->count++;
counter_ptr->running_count++;
```

```
if (rate_ptr != NULL)
     rate_ptr->count++;

/*
 * If the threshold has been exceeded, send a trap
 */
if (counter_ptr->high_thld != 0)
     {
     if (counter_ptr->running_count >= counter_ptr->high_thld)
          {
          stats_alarm_cntr (counter_ptr, structure_ptr,
                            alarm_type, alarm_number, (AlarmUserData *)user_ptr);
          counter_ptr->running_count = 0;
          }
     }

/*
 * Queue to Event Manager Task
 */


return;
}


/*************************************************************************
 *
 * Routine to deorement counters of type StatsCount32
 */
void stats_decr  (counter_ptr, rate_ptr, structure_ptr, alarm_type, alarm_number,
user_ptr)

     StatsCount32        *counter_ptr;
     StatsRollingRate    *rate_ptr;
     StatsAddrEntry      *structure_ptr;
     Uint32                   alarm_type;
```

```
     Uint32                    alarm_number;
     Byte                 *user_ptr;
{
if (structure_ptr == 0)
     return;

counter_ptr->count--;
counter_ptr->running_count--;

if (rate_ptr != NULL)
     rate_ptr->count++;

return;
}



/*************************************************************************/
/*
 * Routine to update byte counters of type StatsCount32
 */
void stats_count_bytes (counter_ptr, rate_ptr, length, structure_ptr, alarm_type,
alarm_number,
              user_ptr)

     StatsCount32         *counter_ptr;
     StatsRollingRate     *rate_ptr;
     Uint32                    length;
     StatsAddrEntry       *structure_ptr;
     Uint32                    alarm_type;
     Uint32                    alarm_number;
     Byte                 *user_ptr;
 {
  if (structure_ptr == 0)
     return;
```

```
    counter_ptr->count = counter_ptr->count + length;
    counter_ptr->running_count = counter_ptr->running_count + length;

    if (rate_ptr != NULL)
       rate_ptr->count += length;

    /* If threshold comparision is enabled, signal */
    if (counter_ptr->high_thld != 0)
      {
      if (counter_ptr->running_count >= counter_ptr->high_thld)
        {
         stats_alarm_entr (counter_ptr, structure_ptr,
                           alarm_type, alarm_number, (AlarmUserData *)user_ptr);
        counter_ptr->running_count = 0;
        }
      }

    return;
    }


/*****************************************************************************
 *
 * Given the pointer to the protocol qhead, find the protocol statistics that
 * correspond to the requested protocol.  If no match is found, NULL is returned.
 */

StatsProtocolEntry *stats_lookup_protocol (protocolQ_ptr, protocol)

     Qhead_type               *protocolQ_ptr;
     Uint32         protocol;
{
     register StatsProtocolEntry        *protocol_ptr;

if (protocolQ_ptr == NULL)
     return (NULL);
```

```
protocol_ptr = (StatsProtocolEntry *) protocolQ_ptr->pFlink;

while (protocol_ptr != NULL)
     {
     if (protocol != protocol_ptr->protocol)
          protocol_ptr = protocol_ptr->link;
     else
          break;
     }

return (protocol_ptr);
}



/*****************************************************************************
 *
 * Given a pointer to the protocol qhead, find the protocol statistics that
 * correspond to the requested protocol.  If no match is found, attempt to allocate one.
 */

StatsProtocolEntry *stats_get_protocol (protocolQ_ptr, protocol)

     Qhead_type               *protocolQ_ptr;
     Uint32                   protocol;
{

     register StatsProtocolEntry        *protocol_ptr;

if (protocolQ_ptr == NULL)
     return (NULL);

protocol_ptr = stats_lookup_protocol (protocolQ_ptr, protocol);

if (protocol_ptr != NULL)
     return (protocol_ptr);
```

```
    /*
     * Protocol not in list, try to allocate a structure, initialize it, and
     * put it on the protocol queue.
     */
    protocol_ptr = (StatsProtocolEntry *)stats_allocate (sizeof (StatsProtocolEntry) );
    if (protocol_ptr != NULL)
         {
         protocol_ptr->protocol = protocol;
         protocol_ptr->frameRate.type = STATS_RATE_10S;
         Insqt (protocolQ_ptr, (PQentry_type) protocol_ptr);
         }
    return (protocol_ptr);
    }



    /*************************************************************************
     *
     * Save the protocol id in the dialog statistics
     */

    void stats_save_protocol_in_dialog (dialog_ptr, protocol)

         StatsDialogEntry    *dialog_ptr;
         Uint32                   protocol;

    {
         register Uint32                  i;
         register Bool               store_protocol;

    if (dialog_ptr == NULL)
         return;

    store_protocol = TRUE;

    for (i=0; i<dialog_ptr->protocolEntries; i++)
```

```
        {
        if (protocol == dialog_ptr->protocols[i])
             {
             store_protocol = FALSE;
             break;
             }
        }


   if (store_protocol == TRUE)
        {
        if (dialog_ptr->protocolEntries < MIB_PROTOCOLS_PER_DIALOG)
             i = dialog_ptr->protocolEntries;
        else
             i = MIB_PROTOCOLS_PER_DIALOG; /* just keep storing over the last one */

        dialog_ptr->protocols[i] = protocol;
        dialog_ptr->protocolEntries++;
        }
   }


   /*************************************************************************
    *
    * Translate a port number to a protocol number
    */

   Uint32 stats_port_to_protocol (ip_addr_ptr, port, transport_protocol)

        StatsAddrEntry          *ip_addr_ptr;
        Uint32                       port, transport_protocol;
   {
        register StatsPmapping  *pmapping_ptr;

   if (stats_well_known_port (port) )
        return (port);
```

```
pmapping_ptr = stats_pmap_lookup (ip_addr_ptr, port, transport_protocol);
if (pmapping_ptr)
     return (pmapping_ptr->protocol);

return (UNKNOWN_PORT);

}


/*************************************************************************
 *
 * Determine if a port is well known
 */

Bool stats_well_known_port (port)

     Uint32          port;
{
     register Uint32          i;

if ( (port > 0) && (port <= MAX_PORT) )
     return (TRUE);
else
     {
     i = 0;
     while (i != statsWellKnownPorts.count)
          {
          if (port == statsWellKnownPorts.port[i])
               return (TRUE);
          else
               i++;
          }
     }

return (FALSE);
}
```

```
/*****************************************************************************
 *
 * Determine if a program number is in table.  Returns protocol number or 0.
 */

Uint32 stats_program (program_number, version)

     Uint32          program_number, version;
{
     register Uint32          i;

i = 0;
while (i != statsPrograms.count)
     {
     if ( (program_number == statsPrograms.entry[i].program_number) &&
          (version == statsPrograms.entry[i].version) )
          return (statsPrograms.entry[i].protocol);
     else
          i++;
     }

return (0);
}


/*****************************************************************************
 *
 * Allocate a statistics structure
 * A structure is allocated from one of six queues depending on the
 * size requested.  If a structure of the appropriate size cannot be
 * allocated, a larger structure WILL NOT be allocated.
 */

PQentry_type stats_allocate (size)
```

```
    Uint32              size;

{
    register PQentry_type    entry_ptr;

if (size <= stats_tiny_size)
    {
    entry_ptr = (PQentry_type) Remqh (&statsTinyQ);
    if (entry_ptr != NULL)
        bzero (entry_ptr, size);
    return (entry_ptr);
    }

if (size <= stats_very_small_size )
    {
    entry_ptr = (PQentry_type) Remqh (&statsVerySmallQ);
    if (entry_ptr != NULL)
        bzero (entry_ptr, size);
    return (entry_ptr);
    }

if (size <= stats_small_size )
    {
    entry_ptr = (PQentry_type) Remqh (&statsSmallQ);
    if (entry_ptr != NULL)
        bzero (entry_ptr, size);
    return (entry_ptr);
    }

if (size <= stats_medium_size )
    {
    entry_ptr = (PQentry_type) Remqh (&statsMediumQ);
    if (entry_ptr != NULL)
        bzero (entry_ptr, size);
    return (entry_ptr);
    }
```

```
if (size <= stats_large_size)
     {
     entry_ptr = (PQentry_type) Remqh (&statsLargeQ);
     if (entry_ptr != NULL)
          bzero (entry_ptr, size);
     return (entry_ptr);
     }

if (size <= stats_very_large_size)
     {
     entry_ptr = (PQentry_type) Remqh (&statsVeryLargeQ);
     if (entry_ptr != NULL)
          bzero (entry_ptr, size);
     return (entry_ptr);
     }

#ifdef DEBUG
     printf ("stats_allocate: size = %d", size);
#endif DEBUG

return (NULL);
}



/*****************************************************************************
 *
 * Deallocate a statistics structure
 *
 * The structure is returned to the appropriate queue based on the
 * size.  If the algorithm is changed so that a larger structure may
 * be used in stats_allocate to hold a structure that would have fit
 * into a smaller structure, this routine must change.
 */

void stats_deallocate (entry_ptr, size)
```

```
        ntry_type       entry_ptr;
        32              size;

{
if (entry_ptr == NULL)
     {

#ifdef DEBUG
printf ("stats_deallocate: size = %d, entry_ptr is NULL", size);
#endif DEBUG

     return;
     }


if (size <= stats_tiny_size)
     {
     Insqt (&statsTinyQ, entry_ptr);
     return;
     }

if (size <= stats_very_small_size)
     {
     Insqt (&statsVerySmallQ, entry_ptr);
     return;
     }

if (size <= stats_small_size)
     {
     Insqt (&statsSmallQ, entry_ptr);
     return;
     }

if (size <= stats_medium_size)
     {
     Insqt (&statsMediumQ, entry_ptr);
```

```
        return;
        }

    if (size <= stats_large_size)
        {
        Insqt (&statsLargeQ, entry_ptr);
        return;
        }

    if (size <= stats_very_large_size)
        {
        Insqt (&statsVeryLargeQ, entry_ptr);
        return;
        }
    }


/*****************************************************************************
 *
 * Check for a duplicate ip address
 */

Uint32 stats_check_for_duplicate_ip (mib_address, new_mac_addr)
        MibAddress          *mib_address;
        MacAddress          *new_mac_addr;
{
        register Uint32             i;
        register Uint32             new_mac;

        new_mac = FALSE;
        if (mib_address->addressType & MibMacAddress1)
        {

            for (i = 0; i < 6; i++)
            {
                if (mib_address->macAddress1.bytes[i] != new_mac_addr->bytes[i])
```

```
                    new_mac = TRUE;
               }

               if (new_mac == TRUE)
                    stats_alarm_duplicate_ip(mib_address, new_mac_addr);
          }

          return (new_mac);
     }
```

OmniSys Corporation

211 SECOND AVENUE WALTHAM, MA. 02154

END