Introduction to Stream Control Transmission Protocol

linuxjournal.com/article/9748

HOWTOs

by Jan Newmarch

on September 1, 2007

Most people who have written networking software are familiar with the TCP and UDP protocols. These are used to connect distributed applications and allow messages to flow between them. These protocols have been used successfully to build Internet applications as we know them: e-mail, HTTP, name services and so forth. But, these protocols are more than 20 years old, and over time, some of their deficiencies have become well known. Although there have been many attempts to devise new general-purpose transport protocols above the IP layer, only one so far has received the blessing of the IETF: SCTP (Stream Control Transmission Protocol). The central motivation behind SCTP is to provide a more reliable and robust protocol than either TCP or UDP that can take advantage of features such as multihoming.

SCTP is not a radical departure from TCP or UDP. It borrows from both but is most similar to TCP. It is a reliable session-oriented protocol, like TCP. It adds new features and options and allows finer control over the transport of packets. In all but the “edge” cases, it can be used as a drop-in in place of TCP. This means that TCP applications often can be ported trivially to SCTP. Of course, to benefit properly from the new features of SCTP, you need to use the additional API calls for SCTP.

The first additional feature in SCTP is better support for multihomed devices—that is, computers with more than one network interface. At one time this meant only routers and bridges connecting different parts of the Internet, but now even computers on the edges of the network can be multihomed. Most laptops have built-in Ethernet cards and Wi-Fi cards, and many have Bluetooth cards as well (which have IP support through the Bluetooth PPP stack). Some laptops now are shipping with WiMAX cards, and it even is possible to run IP over the infrared port! So, the standard laptop is at least dual-homed, with possibly up to five distinct IP network interfaces.

TCP and UDP allow use of only one or all of the interfaces. But, what if you are running your laptop as a peer in, say, a file-sharing service? It probably would be silly to use the Bluetooth and infrared interfaces. WiMAX can be very expensive to shift large amounts of data. But, it would make sense to use both the

Ethernet and Wi-Fi interfaces. SCTP can support this selective choosing of interfaces. Some implementations even can add and drop interfaces dynamically, so as you unplug your laptop and move out of the house, an application can switch to the WiMAX interface if you want.

The second main new feature is multistreaming—that is, one “association” (which is renamed from “connection” from TCP) can support multiple data streams. It is no longer necessary to open up multiple sockets; instead, a single socket can be used for multiple streams to a connected host. Several TCP applications could benefit from this. For example, FTP (the major file transfer protocol) uses two streams: one on port 21 for control messages and another on port 20 for data. This caused problems with firewalls in place. A client could connect to a server through a firewall, but the server could not connect to the client for data transfer because of the firewall. The FTP protocol had to be extended to allow for “passive” connections to overcome this. There would be no need for such an extension under SCTP—simply send the data on a separate stream in an association established by a client.

The X Window System also uses multiple sockets on multiple ports. Although it is not common, a computer can have multiple display devices. Typically, the first is on port 6000, the second on port 6001 and so on. Under SCTP, these could all be separate streams on a single association. HTML documents often contain embedded references to image files, and to display a page properly requires downloading the original page and all of these images (or embedded frames too). HTTP originally used a separate TCP connection per downloaded URL, which was expensive and time consuming. HTTP 1.1 brought in “persistent connections”, so that a single socket could be reused for all of these sequential downloads. Under SCTP, the separate images could be downloaded concurrently in separate streams on a single association.

There are even more subtle uses of SCTP multiple streams. An MPEG movie consists of different types of frames: I frames, P frames and B frames. I frames encode complete images, and the other two types measure differences between frames. Typically, there is an I frame every ten frames, with the others “predicted” from these. It is critical that the I frames be delivered, but less so for the P and B frames. Although SCTP is not designed as a Quality-of-Service protocol, it does allow different delivery parameters on different streams within an association, so that the I frames can be delivered more reliably.

SCTP has many more features, such as:

- TCP is a byte-oriented protocol, and UDP is message-oriented. The majority of applications are message-oriented, and applications using TCP have to jump through hoops, such as sending the message length as a first parameter. SCTP is message-oriented, so such tricks are not so necessary.
- A single socket can support multiple associations—that is, a computer can use a single socket to talk to more than one computer. This is not multicast, but it could be useful in peer-to-peer situations.
- SCTP has no “out of band” messages, but a large number of events can be interleaved onto a single association, so that an application can monitor the state of the association (for example, when the other end adds another interface to the association).

- The range of socket options is greater than TCP or UDP. These also can be used to control individual associations or individual streams within a single association. For example, messages on one stream can be given a longer time-to-live than messages on other streams, increasing the likelihood of their delivery.

Availability of SCTP

The SCTP Web site (www.sctp.org) has a list of implementations of SCTP. There are implementations for BSD and Windows, and since 2001, there has been a Linux kernel project at sourceforge.net/projects/lksctp. At present, SCTP is not in any Microsoft release, so applications running on Windows need to install one of the available stacks.

SCTP is included in the Linux kernel as an experimental network protocol. SCTP is normally built as a module. It may be necessary to load the module using `modprobe sctp`. To build user applications, you may need to install the SCTP tools—in Fedora Core 6, these are in the RPM packages lksctp-tools-1.0.6-1.fc6.i386.rpm and lksctp-tools-devel-1.0.6-1.fc6.i386.rpm. On Fedora Core 6, I also had to add a symbolic link from /usr/lib/libsctp.so to /usr/lib/libsctp.so.1.

The lksctp-tools package contains the libraries to run SCTP applications. It also contains a program called checksctp, which tells you if your kernel has support for SCTP. When you run this program, it prints either "SCTP supported" or an error message.

The devel package contains the sctp.h header file, so you can compile and build your own applications, and man pages for the SCTP function calls.

Firewalls

Most firewalls can be configured to deal with SCTP packets, but the documentation for each firewall may not mention SCTP explicitly. For example, the man page for iptables says, "The specified protocol [in a rule] can be one of tcp, udp, icmp, or all...". But, it then goes on to say, "A protocol name from /etc/protocols is also allowed", and in that file, we find that protocol 132 is sctp. So, rules for SCTP can be added to iptables in the same way as TCP and UDP rules.

For example, an iptables rule to accept SCTP connections to port 13 would be:

```
-A INPUT -p sctp -m sctp -i eth0 --dport 13 -j ACCEPT
```

Webmin is a popular administration tool for managing things like iptables rules. Unfortunately, as of version 1.340, it could not accept this rule, because it is hard-wired to accept port numbers only for TCP and UDP, not realising that SCTP also uses port numbers. Such a rule would need to be entered by hand into the iptables configuration file /etc/sysconfig/iptables. This will be fixed in later versions of Webmin after I logged a bug report, but similar problems may occur in other tools.

One-to-One Socket API

As with TCP and UDP, SCTP provides a socket API for applications. A server creates a socket bound to a port and then uses this to accept a connection from a client. A client also creates a socket and then connects to a server. Both then use the socket file descriptor to read and write messages. SCTP is not a superset of TCP. Nevertheless, when restricted to a similar style of connection as TCP, there are sufficient similarities that an SCTP socket often can be used as a drop-in replacement for a TCP socket. When used in this way, SCTP sockets are called one-to-one sockets, as they simply connect one host to a single other host.

To create a TCP socket, use the system call:

```
sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
```

This creates an IPv4 socket. To create an IPv6 socket, replace the first parameter with AF_INET6. The last parameter often is given as zero, meaning “use the only protocol value in the family”. It is better to use IPPROTO_TCP explicitly, because SCTP introduces another possible value.

To create an SCTP one-to-one socket, simply replace IPPROTO_TCP with IPPROTO_SCTP:

```
sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_SCTP)
```

and that (in many cases) is it! The client or server is now talking the SCTP protocol instead of TCP.

To see this in action, Listings 1 (echo_client.c) and 2 (echo_server.c) give a simple echo-client and server, where the server returns a string sent to it when a client connects to it. Only the line above needs to change in both the client and the server (with also an extra include file, sctp.h).

Listing 1. echo_client.c

```
#define USE_SCTP

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#ifdef USE_SCTP
#include <netinet/sctp.h>
#endif

#define SIZE 1024
char buf[SIZE];
char *msg = "hello\n";
#define ECHO_PORT 2013

int main(int argc, char *argv[]) {
        int sockfd;
        int nread;
        struct sockaddr_in serv_addr;
        if (argc != 2) {
                fprintf(stderr, "usage: %s IPaddr\n", argv[0]);
                exit(1);
        }
        /* create endpoint using TCP or SCTP */
        sockfd = socket(AF_INET, SOCK_STREAM,
#ifdef USE_SCTP
                        IPPROTO_SCTP
#else
                        IPPROTO_TCP
#endif
                );
        if (sockfd < 0) {
                perror("socket creation failed");
                exit(2); }
        /* connect to server */
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
        serv_addr.sin_port = htons(ECHO_PORT);
        if (connect(sockfd,
                    (struct sockaddr *) &serv_addr,
                    sizeof(serv_addr)) < 0) {
                perror("connect to server failed");
                exit(3);
        }
        /* write msg to server */
        write(sockfd, msg, strlen(msg) + 1);
        /* read the reply back */
        nread = read(sockfd, buf, SIZE);
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase®
Smarter legal research.