Network Working Group
Request for Comments: 1692
Category: Standards Track

P. Cameron
Xylogics, International Ltd.
D. Crocker
Silicon Graphics, Inc.
D. Cohen
Myricom
J. Postel
ISI
August 1994

Transport Multiplexing Protocol (TMux)

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

One of the problems with the use of terminal servers is the large number of small packets they can generate. Frequently, most of these packets are destined for only one or two hosts. TMux is a protocol which allows multiple short transport segments, independent of application type, to be combined between a server and host pair.

Acknowledgments

This specification is the result of the merger of two documents: the original TMux proposal which was the result of several discussions and related initiatives through IETF working groups; and IEN 90 [1] originally proposed by Danny Cohen and Jon Postel in May 1979.

Applicability Statement

The TMux protocol is intended to optimize the transmission of large numbers of small data packets that are generated in situations where many interactive Telnet and Rlogin sessions are connected to a few hosts on the network. In these situations, TMux can improve both network and host performance. TMux is not intended for multiplexing long streams composed of large blocks of data that are typically transmitted by such applications as FTP.

The TMux protocol may be applicable to other situations where small packets are generated, but this was not considered in the design.

The use of the TMux protocol in any other situation may require some modification.

1. Introduction

When network designers consider which protocols generate the most load, they naturally tend to consider protocols which transfer large blocks of data (e.g., FTP, NFS). What is often not considered is the load generated by Telnet and Rlogin because of the assumption that users type slowly and the packets are very small. This is a grave underestimation of the load on networks and hosts which have many Telnet and Rlogin ports on multiple terminal servers.

The problem stems from the fact that the work a host must do to process a 1-octet packet is very nearly as much as the work it must do to process a 1500-octet packet. That is, it is the overhead of processing a packet which consumes a host's resources, not the processing of the data.

In particular, communication load is not measured only in bits per seconds but also in packets per seconds, and in many situation the latter is the true performance limit, not the former. The proposed multiplexing is aimed at alleviating this situation.

If one assumes that most users connected to a terminal server will be connecting to only a few hosts, then it should be obvious that the network and host load could be greatly reduced if traffic from multiple users, destined for the same host, could be sent in the same packet.

TMux is designed to improve network utilization and reduce the interrupt load on hosts which conduct multiple sessions involving many short packets. It does this by multiplexing transport traffic onto a single IP datagram [2], thereby resulting in fewer, larger packets. TMux is highly constrained in its method of accomplishing this task, seeking simplicity rather than sophistication.

2. Protocol Design

IP hosts may engage in the use of TMux transparently, and may even switch back and forth between use of TMux and carriage of transport segments in the usual, independent IP datagrams.

TMux operates by placing a set of transport segments into the same IP datagram. Each segment is preceded by a TMux mini-header which specifies the segment length and the actual segment transport protocol. The receiving host demultiplexes the individual transport segments and presents them to the transport layer as if they had been

received in the usual IP/transport packaging. The transport layer is, therefore, unaware of the special encapsulation which was used.

Hence, a TMux message appears as:

| IP hdr | TM hdr | Tport segment | TM hdr | Tport segment| ...|

Where:

TM hdr — is a TMux mini-header and specifies the following Tport segment.

Tport segment — refers to the entire transport segment, including transport headers.

The TMux Protocol is defined to allow the combining of transmission units of different higher level protocols in one transmission unit of a lower level protocol. Only segments with the same Internet Protocol (IP) header, (with the possible exception of the protocol and check-sum fields) may be combined. For example, the segment (H1, B1) and the segment (H2, B2), where Hi and Bi are the headers and the bodies of the segment, respectively, may be combined (multiplexed) only if H=H1=H2. The combined TMux message is either (H, B1, B2) or (H, B2, B1).

The receiver of this combined message should treat it as if the two original segments, (H,B1), and (H,B2), arrived separately. It is recommended, though not a requirement, that the segments in the TMux message should be processed in the same order that they are in the TMux message.

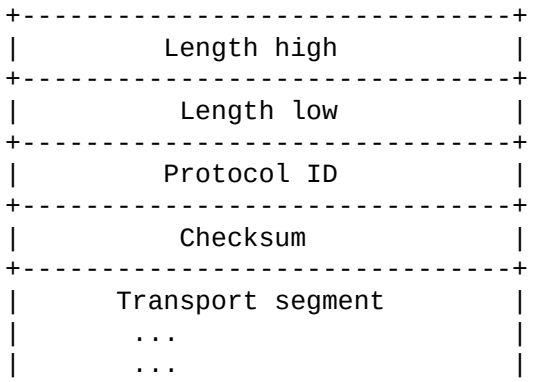
The multiplexing is achieved by combining the individual segments, (H,B1) through (H,Bn), into a single message. This single message has an IP header which is equal to H, but having in the PROTOCOL field the value 18 which is the protocol number of the TMux protocol. This IP header is followed by all the segments, B1 through Bn. Each segment, Bi, is preceded by a 4 octet TMux mini header. This contains the number of the protocol to which this segment is addressed. It also contains the total length of this segment, including this mini header. Since this mini header is not otherwise protected by a check-sum, it also includes a checksum field which just covers this mini header.

2.1. IP Protocol field value

TMux is indicated in an IP datagram by the Protocol (ID) value of 18 (22 octal), see [3].

2.2. Header Format

Each 4 octet TMux mini-header has the following general format:



The LENGTH field specifies the octet count for this mini header and the following transport segment, from 0-65535 octets. Hence, the length field has a minimum value of 4. For segments that are larger than the maximum allowed for TMux (see section 5.1), individual IP datagrams should be sent.

The Protocol ID field contains the value that would normally have been placed in the IP header Protocol field.

The 'Checksum' field is the XOR of the first 3 octets.

To ensure that TCP, UDP and other segments keep their 32 bit alignment, where the segments being multiplexed are not a multiple of 32 bits long, extra octets will be added to re-align the end of the segment, and hence the next segment. These octets will be ignored on input. This padding will not affect the LENGTH field, it will still contain the real length of the segment.

2.3. Sending Data

Host endpoints may choose to use TMux at any time and in either (or both) directions. They also may switch back and forth between use of TMux packaging and the usual individual IP datagrams for individual transport associations. The only barrier to the use of TMux is for the sender to know whether TMux is supported by the receiver. This is important, since early use of TMux is likely to be limited.

The easiest way to detect TMUX support is to only send TMux messages to hosts from which a valid TMux message has already been received. This then leaves the problem of one host starting the TMux connection. This is most easily accomplished by the host sending an IP datagram with no data (i.e., with the IP total length field of 20), but with an IP Protocol field value of 18 for TMux. This is referred to as a TMux ENQ (enquiry) message. The host receiving this message then knows that the originator supports TMux, and can start to send TMux messages. This will in turn cause the originator of the ENQ message to start to use TMux. If for any reason the receiver does not intend to send TMux messages to the originator, but is prepared to accept them, then it can reply with another ENQ message.

If an ENQ message does not get a response, then it is reasonable to resend the ENQ a while later in case the original ENQ message was lost. If this again is lost, the ENQ may be repeated as often as needed, but the time between requests should increase exponentially up to a limit of about 1 hour. Suitable times between ENQs would be 15 seconds, 30 seconds, 60 seconds, 120 seconds etc.

Note that this checking process does not need to impede any of the transport (user) data, which may be sent as convenient, albeit in its less-efficient IP datagram form.

The only problem with this scheme is that a host which supports TMux may stop supporting it, as might happen when the host is re-booted. Other hosts need to learn of this change. The solution to this is to maintain a Time To Live (TTL) value for hosts from which TMux messages have been received. This TTL is a timed TTL, rather than a count as used in the IP TTL field, and this time stamp is updated every time a TMux message is received. This can then be used to expire the information held by TMux on the host after a suitable time, e.g., 1 minute.

This TTL time stamp is used as follows. When TMux is passed a segment to be sent to a host, a check is made to see if the time to live has expired. If the TTL has not expired, the segment is sent in a TMux message as normal. If the TTL has expired, the host is marked as being unable to TMux, but the segment is STILL sent as a TMux message (i.e., with the normal delay to allow other segments to be multiplexed). If the host is really unable to TMux anymore (a rare occurrence) then this segment will be timed out and retried by the transport provider i.e., TCP. Because the host was marked as not able to TMux, the retry will be sent as a normal IP datagram. If the remote host is still able to TMux then it should send back TMux traffic (even if it has been rebooted), typically a TCP window update, and the local host will mark it as able to TMux again. This way of operating removes any performance problem caused by

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.