

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 checksum 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.

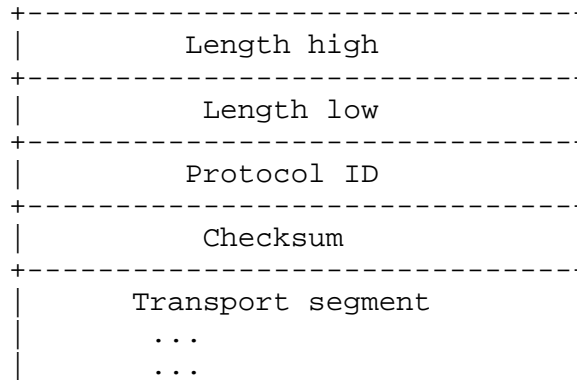
The multiplexing is achieved by combining the individual segments, (H,B1) through (H,Bn), into a single message. This single message has an IP header which is equal to H, but having in the PROTOCOL field the value 18 which is the protocol number of the TMux protocol. This IP header is followed by all 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 checksum, 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

continually dropping out of TMuxing and having to send probe messages. If the IP datagram to be sent is from UDP, then the remote host may not send anything in reply. So for UDP this scheme will not be any better than just stopping sending TMux messages to the host, but it is also no worse.

### 3. Protocol Behavior

#### 3.1. Transport Flow Control

TMux operates as an extension to the IP datagram protocol. Hence, it has no impact on most flow control mechanisms, since they operate at the transport layer -- above TMux.

#### 3.2. Connection Management

The concept of a connection pertains to certain transport protocols, but not to IP or to TMux. Hence, when connection management is required by a transport protocol using TMux, it occurs in the same fashion as it does for IP. In fact, the transport protocol is not to be aware that TMux is being used.

#### 3.3 Multiplexed Message Construction

When a transport provider (e.g., TCP or UDP) sends a segment, TMux first removes the IP header (if present) and adds a TMux mini-header and the segment to the Multiplexed Message under construction for the host specified by the destination address of the segment.

When the first message to be transmitted is placed into the Multiplexed Message under construction, a timer is started. When the timer expires, the Multiplexed Message under construction is transmitted. This ensures that all segments available for sending before the timer expires are sent in a single Multiplexed Message. If, during construction of the Multiplexed Message, the buffer holding the message fills, the Multiplexed Message is transmitted immediately.

The delay time should be user configurable; a reasonable time is 20 to 30 milliseconds. The time period should be large enough to give a reasonable probability of sending multiple segments but not so large that the echo response time becomes a problem. This suggests that the upper limit for the timer is probably 1/10th second. As the cost of using timeouts on many systems is quite large, it is recommended that a single timer be used and that all TMux messages under construction are sent when the timer expires.

Additionally, configuration options may limit the number of included data segments or the maximum size of the Multiplexed Message before it is transmitted. It is also suggested that larger segments (e.g., those over 700 octets) should be sent as standard IP datagrams, and not multiplexed. This is to ensure that the delay caused by the TMux timer does not put a delay on those segments for which it is inadvisable. The size of the largest segments to be multiplexed should (if possible) be configurable.

#### 4. Protocol Example

This example shows a TMux message consisting of three multiplexed segments:

A TCP segment consisting of a 20 octet TCP header, 5 octets of data and 3 octets of padding. Thus the length field is

$$\begin{array}{rcl}
 & \text{Mini header} & + \text{ TCP header} & + \text{ data} \\
 = & 4 & + & 20 & + & 5 \\
 = & 29 & & & & 
 \end{array}$$

The padding is NOT included in the length.

A TCP segment consisting of a 20 octet TCP header, 4 octets of data. This segment does not require padding.

A UDP segment consisting of a 4 octet UDP header, 41 octets of data and 3 octets of padding.

	Length = 29 (2 octets)	
	Protocol ID = 6 (TCP)	
	Checksum	
	TCP Header (20 octets)	
	TCP data (5 octets)	
	Padding (3 octets)	
	Length = 28 (2 octets)	

Protocol ID = 6 (TCP)
Checksum
TCP Header (20 octets)
TCP data (4 octets)
Length = 49 (2 octets)
Protocol ID = 17 (UDP)
Checksum
UDP Header (4 octets)
UDP data (41 octets)
Padding (3 octets)

## 5. Implementation Suggestion

### 5.1 Maximum TMux Message Size

In section 3.3, a note is made about sending messages immediately if the limit on TMux message size is reached. On systems where Path MTU Discovery (as per RFC 1191 [4]) has been implemented this should be used to discover the maximum message size that can be transmitted, and this should be used as the maximum TMux message size.

### 5.2 Deciding Which Segments to Multiplex

It is the responsibility of the sender to decide which segments should be TMux'd and which should not. For example, segments sent by FTP should not normally be multiplexed. In many situations, it may be sensible to restrict the sessions that can be multiplexed to just those involved in interactive traffic (Telnet and Rlogin) by examining the source and destination TCP port numbers. However, if a segment that would not normally be multiplexed is to be sent and a TMux message is already under construction, then the extra segment



can be added to the TMux message under construction, and this complete message should be sent immediately, rather than waiting for the timer to expire.

## 6. Implementation notes

The following notes are the result of experience gained during the testing of early implementations of TMux. Whilst they do not form part of the actual standard, they should be followed if possible to ensure compatibility with other implementations.

Because the TMux mini-header does not contain a TOS field, only segments with the same IP TOS field should be contained in a single TMux message. As most systems do not use the TOS feature, this is not a major restriction. Where the TOS field is used, it may be desirable to hold several messages under construction for a host, one for each TOS value.

Segments containing IP options should not be multiplexed.

Only unicast addresses should be considered for multiplexing.

Segments addressed to the loopback address (127.0.0.1) are not candidates for multiplexing.

Only segments with a source or destination port that is for an interactive session (i.e., Telnet and Rlogin) should be considered for multiplexing using TMux.

If an error is discovered in a checksum of a TMux header, the rest of the message, starting there, is ignored. If an unknown PROTOCOL field is discovered in any TMux header, this segment, and only this one, is ignored.

If the TMux implementation is continually sending TMux messages containing exactly one segment (because there is little traffic to multiplex), then TMux may be turned off. This implies that TMux may be switched off when there is no congestion.

To prevent intermediate nodes from fragmenting and reconstructing TMux frames, implementations may want to set the "do not fragment" flag in the IP datagram of TMux messages.

If host B receives a TMux ENQ message from host A, but does not have any data for host A, then it may also send back an ENQ message. However, host A may send another ENQ message in response to this, so causing B to respond and so on. Thus if this facility is used, code must be included to prevent this looping behavior happening. Sending

an ENQ in response to an ENQ is not recommended, except in special circumstances.

It is recommended that the following aspects of the TMux protocol be user configurable:

The maximum size of a segment that can be multiplexed by TMux.

The delay between the first segment being placed into the message under construction and the message being sent.

## 7. Security Considerations

Because TMux is effectively an extension to IP, it does not have any more impact on site security than does IP. Security should be dealt with by upper layer protocols.

Because some routers filter packets on the TCP port numbers, any segments sent using TMux will not be subject to this filtering as it will obscure the TCP port number. However, larger segments for the same TCP connection will still be sent as IP datagrams, and so will be subject to filtering, thus giving rise to a potential problem. For this reason, any routers that do not support TMux, but which do support this type of filtering should not allow TMux messages through (in either direction). This will cause both hosts to think the other does not support TMux, so all segments will be sent as IP datagrams, thus eliminating this problem.

A better solution to this problem, is for routers to understand the TMux protocol, and to inspect each of the multiplexed segments and remove those segments that fail the filtering.

## 8. References

- [1] Cohen, D., and Postel, J., "Multiplexing Protocol", IEN 90, USC/Information Sciences Institute,, May 1979.
- [2] Postel, J., "Internet Protocol", STD 5, RFC 791, USC/Information Sciences Institute, September 1981.
- [3] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1340, USC/Information Sciences Institute, March 1990.
- [4] Mogul, J., and S. Deering, "Path MTU discovery", RFC 1191, DECWRL and Stanford University, November 1990.

## 9. Authors' Addresses

Peter Cameron  
Xylogics International, Ltd.  
Featherstone Rd.  
Wolverton Mill  
Milton Keynes  
MK12 5RD  
United Kingdom

Phone: +44 908 222112  
Fax: +44 908 222115  
EMail: cameron@xylint.co.uk

David Crocker  
Silicon Graphics, Inc.  
2011 N. Shoreline Blvd.  
P.O. Box 7311  
Mountain View, CA 94039-7311  
USA

Phone: +1 415 390 1804  
Fax: +1 415 962 8404  
EMail: dcrocker@sgi.com

Danny Cohen  
Myricom  
325 N. Santa Anita Ave.  
Arcadia, CA 91006  
USA

Phone: +1 818 821 5555  
EMail: Cohen@myricom.com

Jon Postel  
USC/Information Sciences Institute  
4676 Admiralty Way  
Marina del Rey, CA 90292-6695  
USA

Phone: +1 310 822 1511  
Fax: +1 310 823 6714  
EMail: Postel@ISI.EDU

## 10. Discussion List

There is a discussion list for this protocol, which for historical reasons is called:

`cmp-id@xylint.co.uk`

Requests to join the list should be sent to:

`cmp-id-request@xylint.co.uk`

