To: IANA
From: The MITRE Corporation
Subject: Request for TCP Option number assignments

I Introduction

This is to request that IANA assign TCP option numbers in support of research that we are conducting. We are experimenting with extensions to TCP to support its operation in stressed environments that may be characterized by one or more of the following:

- Highly asymmetric bandwidth allocation
- Large delays
- Link outages
- A high degree of corruption-related loss

This research began as part of the Space Communications Protocol Specification (SCPS) work, however we believe that the options developed have wider, more general-purpose applicability.

The four TCP options for which we are requesting assignment of option numbers are:

1. SCPS-capabilities option. This option allows endpoints to agree/decline to use the following TCP extensions: Best Effort Transport Service (BETS, a set of partial-reliability extensions for TCP), Selective Negative Acknowledgments (SNACK), loss-tolerant header compression, and network-layer timestamps.
2. Selective Negative Acknowledgments (SNACK) option. This option is set on ACK segments to indicate the presence and location(s) of holes in the receiver's out-of-sequence queue.
3. Record boundaries option. This option is set on data packets containing bytes that have been logically marked as being record boundaries.
4. Corruption experienced option. This option can exist on both data and ACK packets to indicate that a link in a path is experiencing corruption.

Section II describes the options for which we are requesting assignment of option numbers. Section III contains descriptions of those options that can be invoked via the SCPS-capabilities option but which do not themselves require separate option number assignment.

II TCP Options for which we are requesting option number assignment.

1. SCPS-Capabilities Option

The SCPS-capabilities option provides an efficient and convenient way for endpoints to agree on which of a set of TCP extensions they want to use during a connection, and allows specification of a connection ID for use when our loss-tolerant header compression scheme is enabled. The SCPS-Capabilities Option shall be located in the options area of the TCP SYN segment and shall contain the following fields, all of which are mandatory.

SCPS-Capabilities Option Type	1 Octet containing the value to be assigned as the SCPS-capabilities option number
SCPS-Capabilities Option Length	1 Octet containing the value 4.
SCPS-Capabilities Bit Vector	1 Octet containing a bit vector of “can-do” options. The format of this field is described below
Connection ID	1 Octet containing a 1-byte connection identifier; a non-zero value indicates the sender's desire to send compressed headers, while a zero value indicates that the sender will not compress outgoing headers.

The SCPS capabilities bit vector is 1 octet long; its bits have the following interpretation:

Bit	**Meaning if 0 “Not OK”**	**Meaning if 1 “OK”**
BETS 0x80	Connection may not operate in BETS mode.	Sender is willing to operate in BETS mode[1].
SNACK1 0x40	Do not send short form (length=4) SNACK option.	OK to send short form of SNACK option. This bit must be set to 1 if the SNACK2 bit is set to 1.
SNACK2 0x20	Do not send long form (length>4) SNACK option.	OK to send long form of SNACK option. If the SNACK2 bit is set to 1 then the SNACK1 bit must also be set to 1.
Com 0x10	Do not compress TCP headers.	Sender is willing to accept compressed headers[2].
NL TS 0x08	Network layer timestamps are unavailable or are unsuitable for compressing timestamps.	Network layer timestamps are available and a timestamp accompanies this segment. If received, suitable, and available at both ends, use to compress the TCP timestamp option[3].
Reserved 0x04 0x02 0x01	Reserved for future use.	

Notes

- If both TCP endpoints send the BETS bit set to "OK," the connection will operate in BETS mode.
- The semantics for the combination of the "Com" bit and the connection identifier are as follows:

Com	**ConnectionID**	**Meaning**
0	0	Will not send or accept compressed headers.
0	X≠0	Will not accept compressed headers. Would like to send compressed headers and, if peer will accept them, will do so using connection id X.
1	0	Willing to accept compressed headers; will not send compressed headers.
1	X≠0	Willing to accept compressed headers; would like to send compressed headers and, if peer will accept them, will use connection id X.

- If compressed headers are in use and both TCP endpoints indicate that use of Network Layer Timestamps (NL TS) is acceptable, then outbound timestamps shall be carried in the timestamps field of the network-layer header. If there is no timestamp field in the network layer header, the use of network layer timestamps shall be deemed unsuitable.

[1] Best Effort Transport Service (BETS) mode provides a means for applications to allow the transport layer to declare a particular sequence number range undeliverable after some period of time and to move on with the transmission. BETS can only be activated via the SCPS capabilities option and, unlike SNACK, no "BETS" option accompanies data segments. Thus BETS does not require option number assignment from IANA. The operation of BETS mode is outlined in section III; for more information on BETS mode, see section III of this document and [2].

[2] Our header compression scheme provides loss-tolerant (i.e. not differentially encoded) compression of TCP headers and can only be invoked via the SCPS-capabilities option. The transport protocol number for the compressed header protocol, as previously assigned by IANA, is 105 decimal. For more information on the compression scheme and the format of the compressed headers, see section III of this document and [2].

[3] Network layer timestamps are only used when our compressed headers are implemented and running over a suitable network layer protocol. In such cases, the presence/absence of network layer timestamps is indicated in the compressed header; we thus are not requesting assignment of an option number for network layer timestamps. Network layer timestamps were developed as part of the SCPS project for use in extremely bandwidth-constrained environments.

From the above, the full SCPS-Capabilities Option format is:

0	1	2	3	4	5	6	7	
Option Type = "SCPS-Capabilities" (number to be assigned)								Octet 1
Option Length = 4								Octet 2
BETS	SN1	SN2	Com	NL TS	Reserved			Octet 3
Connection ID								Octet 4

2. Selective Negative Acknowledgment Option

We are experimenting with the use selective negative acknowledgments, or SNACKs. The receiving endpoint may invoke the SNACK Option by sending an appropriately formed SNACK Option on an ACK segment whenever an out-of-sequence queue forms or a new hole in the out-of-sequence queue forms, provided that the sender advertised its ability to receive by setting the SNACK1 or SNACK2 bits in the SCPS-Capabilities option during the SYN exchange. The format for a SNACK option is:

SNACK Option Type	1 Octet containing the value to be assigned as the SNACK option number.
SNACK Option Length	1 Octet containing the length of this option. There are two forms of the SNACK option; the short form occupies exactly 6 bytes; the long form is variable length, occupying at least 7 bytes.
Hole1 Offset	2 Octets containing an offset from the current acknowledgment number of the first hole being reported for this Option.
Hole1 Size	2 Octets
SNACK Bit-Vector (used only in the long form)	Variable-Length of at least one octet

The hole1 offset is calculated by subtracting the acknowledgment number from the offset sequence number, dividing the difference by the amount of user data carried in a maximum-sized segment, and rounding the result down. The Hole1 Size value is calculated by dividing the hole size (including any remainder from the division used to calculate the hole offset) by the amount of user data in a maximum-sized segment and rounding up.

The SNACK bit-vector field is used only when the long form of the SNACK option has been authorized during the SYN exchange and, if present, shall occupy contiguous octets immediately after the Hole1 Size field. The SNACK bit-vector field maps the sequence space of the receiver's buffer into MSS-sized blocks beginning one octet beyond the end of the block specified by the Hole1 Offset and Size fields. Each "0" in the SNACK bit-vector signifies missing data in the corresponding MSS-sized block in the receiver's resequencing queue. The SNACK bit-vector is right-padded with zeros as necessary to ensure that it ends on a byte boundary. Zeros to the right of the last "1" in the SNACK bit-vector are NOT interpreted as indicating missing data at the receiver.

Upon receipt of a SNACK Option, the data-sender shall retransmit all segments necessary to fill the signaled holes. It is strongly recommended that these retransmissions occur in the order of ascending sequence numbers.

3. Record Boundary Option

The Record Boundary Option provides a way for applications to send and receive record boundary indicators that are preserved across transmission (and possible retransmission).

The Record Boundary Option shall be located in the options area of the TCP header and shall contain the following fields, both of which are mandatory:

Record Boundary Option Type	1 Octet containing the value to be assigned as the Record Boundary option number
Record Boundary Option Length	1 Octet containing the value 2.

When using record boundaries, the sending application indicates, via an Application Programming Interface (API) specific method to the transport layer, that the final octet of data in a particular request represents the end of a record. The final octet of the request is then logically marked by the transport layer with the end-of-record mark, and this logical marking is preserved across transmission and possible retransmission of the marked byte. Further, the sequence number of the byte associated with the record marker shall always be the highest sequence number in the segment carrying the Record Boundary Option (i.e. the byte marked as a record boundary shall be the final octet in the segment). A read operation at the receiver will associate the end of record marker with the marked octet. The requirement that bytes associated with record boundaries always appear as the last bytes of segments interacts with the Nagle algorithm and constrains the transport protocol in performing repacketization during retransmission.

The octet associated with the Record Boundary Option shall not be changed by the Transport Service Provider (TSP); applications at both ends of a prospective connection must determine that a Transport service provides record boundary capability before connection establishment (e.g., by requesting a particular socket option – beyond the scope of this document).

Note that there is a potential interaction between the record boundary option and connections using the BETS partial-reliability capabilities. Specifically, if an application requests both record boundaries and best effort service (BETS), bytes marked as record boundaries may not be delivered to the receiving application. It is the application developer's responsibility to ensure that the application is robust against the possibility of such losses.

4. Corruption Experienced Option

The Corruption Experienced Option provides a means for a TCP connection to react to data loss caused by corruption differently than to loss caused by congestion. Currently, TCP assumes that all loss is a result of congestion. When losses are due to corruption, TCP’s congestion response is inappropriate, and has a negative effect on throughput. Use of this option is only appropriate in networks where congestion is explicitly signaled. In the absence of an explicit corruption signal, loss is assumed to be due to congestion, and congestion recovery mechanisms are invoked. There remains a further issue that is the subject of ongoing research, which is the coincidence of corruption in one portion of the network and severe congestion in another. The case in which (all) congestion signals fail either due to congestion collapse or corruption of the congestion signals is under study.

Corruption signaling is accomplished at both the network control (ICMP) layer and within TCP. A router declares one of its inbound interfaces to be corrupted in response to a signal from the link layer entity, by polling inbound error counts, or by other means. The router uses a least-recently-used queue approach to inform recent near-side users of that link that it is corrupted, and of the source-destination address pairs traversing the corrupted link. This information is carried to the near-side users via a network layer control message (e.g. a new ICMP message type). The receiving host responds to this message by sending the “corruption experienced” option to its peer.

The rationale for using a combination of network layer signaling and a TCP option is as follows. We need to inform both endpoints of the connection that the link is corrupted, but it is a router on the receiving side of the link that has this information. If the link is corrupted in both directions and the data and acknowledgment streams both traverse the link, fine; the routers on either side of the corrupted link will inform their respective endpoints via network-layer signaling. Note that in this case, neither router has much hope of sending an ICMP-type message across the corrupted link to inform the endpoint on the *other* side of the corruption. If only one of the [data, acknowledgment] streams traverses the corrupted link, then the most reliable means of notifying the endpoint on the sending side of that link is by having the endpoint on the receiving side of the link set an option in the TCP header of an outbound packet, and allowing the packet to traverse the non-corrupted path back to the sending side.

In response to the corruption experienced option and in the absence of any direct or indirect indication of congestion, the sending TCP may choose not to cut its transmission rate in response to loss. The format of a "Corruption Experienced" option is:

Corruption Experienced Option Type	1 Octet containing the value to be assigned as the Corruption Experienced option number
Corruption Experienced Option Length	1 Octet containing the value 2.

When a connection receives an indication that corruption has been experienced:

a) the sending TCP shall send the Corruption Experienced Option to the receiving TCP at an implementation-defined rate not to exceed once per round-trip time;
b) the sending TCP shall continuing sending the Corruption Experienced Option for no more than two round-trip times after the previous indication of corruption was received;
c) upon receipt of the Corruption Experienced Option from a sending TCP, the receiving TCP shall not send a corresponding Corruption Experienced Option to its peer.

When a sending TCP receives evidence that packets need to be retransmitted (via duplicate acknowledgments, a SNACK, or a retransmission timeout) it checks to see if the path is marked as corrupted. If the path is not marked as corrupted, the sending TCP updates its transmission policy according to the rules of the congestion control algorithm in use. If the path is marked as corrupted, the sending TCP may choose to not modify its cwnd and ssthresh variables (i.e. it may choose not to cut its transmission rate). Additionally, when there is evidence that data loss is due to corruption rather than congestion, the sending TCP may choose not to use the "exponential backoff" algorithm to increase the time between successive retransmissions.

Any indication of congestion, explicitly signaled or implicit via other information overrides the corruption experienced option and invokes the congestion response.

III Options That Can Be Invoked Via the SCPS-Capabilities Option Which Do Not Themselves Require Option Number Assignment

Best Effort Transport Service (BETS) Option

The Best Effort Transport Service (BETS) Option provides a means for applications to allow the transport layer to declare a particular segment undeliverable after some period of time and to move on with transmission. BETS uses the R1 and R2 thresholds of RFC 1122, interpreting them as counts of transmissions (not time nor retransmissions).

The value of R1 shall be set by the sending TCP, and the value of R2 shall set by the sending application via a Transport-interface option. The value of R2 shall be interpreted by the sending TCP as the threshold at which attempted retransmission of a segment is discontinued. If the value of R2 is set to a non-zero positive number greater than one, then the value of R1 shall be set to a value less than that of R2.

For this section, the following definitions apply:

- SND.UNA is the sequence number of the first unacknowledged octet.
- SND.NXT is the sequence number of the next octet to be sent.
- SEG.SEQ is the sequence number of the first octet in a segment.
- SND.WND is the number of octets of unacknowledged data that the sender is authorized by the receiver to have outstanding.

When the number of transmissions of the same segment reaches or exceeds the value of R1, the sending TCP shall operate as it would in fully reliable mode. If the value of R1 is forced to zero because R2 is set to one,

– no action shall be taken in response to the R1 threshold's being exceeded;
– segments shall be discarded after being initially transmitted (rather than being queued for retransmission).

When either the R2 limit is reached (or exceeded) for a segment, or SND.NXT = SND.UNA + SND.WND, the sending TCP shall behave exactly as if it had received a positive acknowledgment that advances SND.UNA to SEG.SEQ of the next segment in the retransmission queue and that makes no change to SND.WND (refer to 3.9 of RFC 793 and 4.2.2.20 of RFC 1122). That is

1) R2 Reached or Exceeded Or 2) SND.NXT = SND.UNA + SND.WND	$\Rightarrow$	Sender behaves as if it had received a positive acknowledgment that advances SND.UNA to SEG.SEQ of the next segment in the retransmission queue. No changes are made to SND.WND

The use of BETS over a shared network is highly experimental. It is possible to violate congestion control using BETS, and certain settings can be particularly dangerous. For example, if R2 is set to 1, the sending TCP will attempt to send each packet once, will immediately treat the packet as acknowledged, and will move on to the next packet (without waiting for feedback from the receiving TCP). This case is applicable to the original SCPS research, but requires other means to ensure that the sending TCP does not congest the network. One example of such means is a combination of rate control and a simple network where congestion is not an issue, such as a spacecraft and a groundstation connected via a single, low-rate link. Setting R2 to 1 is almost surely inappropriate when running over a shared network.

If R2 is greater than 1, the sender expects a stream of acknowledgments (else it will continually time out). For R2 greater than 1, the acknowledgments (or lack thereof) will trigger normal congestion control responses from the sender.

If R2 is set to zero by an application, it is an escape value that indicates that the TCP should not break a connection due to excessive retransmissions, nor should it invoke the Best Effort Transport Service. Rather, it is a request to retransmit an "infinite" number of Times. In this case the value of R1 shall be determined by the implementor.

For SYN segments:

1. an R2-SYN shall be defined and shall be able to be set independently of R2 (in an implementation-dependent manner, a socket option with the Reference Implementation);
2. R2-SYN shall be greater than or equal to R2;
3. if R2 is increased to a value greater than R2-SYN, then R2-SYN shall be increased to match the value of R2 (independent of user input).

Missing ACK Segments

The sending application may query the sending TCP in an API-specific manner to determine the location and length of segments that were not acknowledged by the receiver. The sending TCP shall provide the sending application with a means to determine which data elements may not have been acknowledged by the receiver. The sending TCP shall specify the missing data elements by reporting a pair of numbers for each missing data element:

– the first number in the pair shall be the octet offset from the start of the connection to the first octet of unacknowledged data;
– the second number in the pair shall be the octet offset from the start of the connection to the last octet of unacknowledged data.

Once read by the sending application, the missing-data-element specification pair shall be removed from the list. System-dependent limits on the amount of information that can be retained by the sending TCP shall be documented for the implementation.

NOTE – The TCP implementation may restrict the amount of memory that is available for storing these pairs of numbers and, if so, may treat the available memory as a circular buffer. If an application does not request a report of missing data elements before the available memory fills, information could be lost.

For TCP implementations that support the Record Boundary Option, the implementation may specify missing data elements by reporting an additional pair of numbers for each missing data element:

a) the first number of the additional pair shall be the record offset from the start of the connection to the beginning of the unacknowledged data;
b) the second number of the additional pair shall be the record offset from the start of the connection to the end of the unacknowledged data.

Missing Data Segments

In Best Effort mode, the receiving TCP shall operate in the same manner as in fully reliable mode until an out-of-sequence segment (i.e., a segment having a higher-than-expected sequence number) arrives. The receiving TCP shall store out-of-sequence segments in an out-of-sequence queue. When an out-of-sequence queue is formed, the receiving TCP shall start an interval timer. The receiving TCP shall use two thresholds:

a) a size-based threshold, BE1, defined as a value in octets that corresponds to a locally administered percentage of the receiver’s buffer space;
b) a time-based threshold, BE2, defined as the interval in locally sized clock ticks after which out-of-sequence data will be delivered to the user.

The receiving TCP shall provide the receiving application with a means to set BE1 and BE2.

When the size of the out-of-sequence queue reaches or exceeds BE1 or the interval BE2 elapses, the receiving TCP shall

a) issue an acknowledgment that acknowledges the data in the missing segment plus any data in the out-of-sequence queue that would be acknowledgeable were the missing segment received;
b) issue an error, warning or advisory to the receiving application identifying the size of the missing segment(s) in octets;
c) deliver the subsequent in-sequence data to the receiving application.

NOTE – If missing and acknowledged segments arrive after RCV.NXT has advanced, the receiving TCP may discard them. Alternatively, the receiving TCP may store them via some out-of-band storage means for off-line merging with the rest of the data. However, this requires the receiving TCP to maintain the sequence number and size of each area of missing data.

The BE2 timer shall be used as follows:

a) the receiving TCP shall start the BE2 timer when all in-sequence data have been read by the application and out of sequence data exist in the receive queue;
b) when the hole at SND.UNA is closed (either by receipt of the missing data, exceeding the BE1 threshold, or expiration of the BE2 timer), the receiving TCP shall cancel the BE2 timer;
c) if additional holes in the out of sequence queue exist, the receiving TCP shall restart the BE2 timer as described above.

Loss-Tolerant Header Compression[4]

Standard Van-Jacobson header compression[4] is extremely efficient in reducing the size of a TCP/IP header. Part of its efficiency comes from delta-encoding, whereby values in the fields of the N+1st TCP header are sent not as absolutes, but are sent in the compressed header as their difference from their values in the Nth header. A drawback of this approach is that the loss of a single packet generally incurs a retransmission timeout in order to resynchonize transmitter and receiver. Some studies have been done[4] where, when a packet with a delta-encoded compressed header cannot be decompressed at the receiver, the receiver guesses that a packet has been lost, assumes values for the fields in the lost packet based on the history of the data stream, and tries to uncompress the received packet, using these guesses. This works well when the data stream is sufficiently predictable that adequate guesses can be made about missed packets (specifically the size of the data portion of the packet).

We are experimenting with a different form of header compression that is loss-tolerant. That is, the receiver can decompress any correctly received packet without having received the previous packet in the sequence. We call this compression scheme Loss-Tolerant TCP header compression. Though not as efficient as Van Jacobson header compression, it does not use delta-encoding and hence is robust against the loss of a single packet. (Of course, the lost packet will still have to be retransmitted. The issue here is that packets received out-of-sequence after the lost packet can be decompressed and stored in an out-of-sequence buffer).

The loss-tolerant TCP compressed header shall contain some or all of the following fields (the numbers after the fields give their length in octets). Fields designated "mandatory" are required for all compressed headers; the presence of other fields depends on the contents of the uncompressed TCP header.

- Connection Identifier (mandatory) 1
- Compressed Header Bit-Vector (mandatory) 1-2
- Urgent Pointer 2
- Window 2
- ACK Number 4
- Sequence Number 4
- Outbound Timestamp format dependent
- Echo Reply Timestamp format dependent
- Options Length 1
- Options data dependent
- Pad Field 1
- Checksum (mandatory) 2

COMPRESSED HEADER FIELDS

Connection Identifier

The Connection Identifier field is mandatory for all loss-tolerant compressed headers and shall occupy the first octet of the compressed header. The Connection Identifier field shall contain the Connection Identifier established during the SYN-segment exchange of the SCPS-capabilities Option.

[4] This is the header compression scheme for which we applied and were granted protocol number 105 decimal.

Compressed Header Bit-Vector

The Compressed Header Bit-Vector field is mandatory for all loss-tolerant compressed headers and shall occupy at least one and no more than two octets beginning with the first octet following the Connection Identifier field. The Compressed Header Bit-Vector field shall contain information necessary for decompressing the compressed header, as detailed below.

Bit Name	Meaning when set to "1"
More	Compressed Header Bit-Vector is 16 bits long rather than 8 bits long.
TS1	TCP Timestamp Option is present.
TS2	A timestamp reply (TS Echo Reply) appears in the compressed header.
RB	The last octet of data accompanying this segment is the end of a user-defined record.
P	The Push bit from the uncompressed TCP header is set.
S	The compressed header contains a 4-octet sequence number.
A	The compressed header contains a 2-octet window specification and a 4-octet acknowledgment number.
Opts	The compressed header contains uncompressed options.
Pad	The compressed header contains one octet of padding.
URG	The URG bit from the uncompressed TCP header is set.
AckR	The ACK bit from the uncompressed TCP header is set (this field is only valid when the RST bit is set).
RST	The RST bit from the uncompressed TCP header is set.
FIN	The FIN bit from the uncompressed TCP header is set.

Urgent Pointer

The Urgent Pointer field shall be included if the URG flag is set in the TCP header and shall occupy two octets immediately following the Compressed Header Bit-Vector field. The Urgent Pointer field shall contain the unmodified urgent pointer value from the TCP header.

Window

The Window field shall be included if either the window value of the acknowledgment number has changed from the last segment sent on the connection, or if nothing has changed from the last segment sent on the connection, and shall occupy two octets immediately following the location for the Urgent Pointer field. The Window field shall contain the unmodified window value from the TCP header.

ACK Number

The ACK Number field shall be included if either the window value of the acknowledgment number has changed from the last segment sent on the connection, or if nothing has changed from the last segment sent on the connection, and shall occupy four octets immediately following the Window field. The ACK Number field shall contain the unmodified ACK number from the TCP
header.

Sequence Number

The Sequence Number field shall be included if the segment is retransmittable (i.e., user data is included or the FIN flag is set), and shall occupy four octets immediately following the location for the ACK Number field. The Sequence Number field shall contain the unmodified sequence number from the TCP header.

Outbound Timestamp

The Outbound Timestamp (TS1) field shall be included if

- a TCP Timestamps Option is present; **and**
- the SCPS Capabilities Option negotiation indicated that NL Ts were *not* available;

and shall occupy one or more octets immediately following the location for the Sequence Number field. The TS1 field shall contain the timestamp value to be echoed.

Echo Reply Timestamp
The Echo Reply Timestamp (TS2) field shall be included if a TCP Timestamps Option is present and shall occupy one or more octets immediately following the location for the TS1 field. The TS2 field shall contain the echo reply timestamp value. Because separate “TS1” and “TS2” bits are used in the Compressed Header Bit-Vector, the equivalent of RFC 1072 timestamps may be compressed, if desired.

TCP Options Length
The TCP Options Length field shall be included if any TCP options remain after header compression and shall occupy one octet immediately following the location for the TS2 field. The TCP Options Length field shall contain the length in octets of the remaining TCP options. Note that the Record Boundary flag, the TS1 and TS2 flags, and the SNACK flag in the Compressed Header Bit-Vector field exist for the purpose of compressing TCP options.

TCP Options
The TCP Options field shall be included if any TCP options remain after header compression and shall occupy one or more octets immediately following the TCP Options Length field. The TCP Options field shall contain any TCP options that have not been compressed.

Pad
The Pad field may be included to ensure the compressed header ends on an even octet boundary and shall occupy one octet immediately following the location for the TCP Options field. If included, the Pad field shall have a value of zero. A Pad field shall not be included if its inclusion necessitates adding a second octet to a single-octet Compressed Header Bit-Vector field. Note that a zero-value second octet of the Compressed Header Bit-Vector field may also be used for padding a compressed header to an even octet boundary.

Checksum
The Checksum field is mandatory for all loss-tolerant compressed headers and shall occupy the two final octets of the compressed header. The Checksum field shall contain the value obtained using the standard TCP checksum algorithm for the contents of the compressed header, the user data, and the TCP pseudo-header.

The loss-tolerant compressed segment format is illustrated below. The fields shown with dashed outlines (after the first octet of the Compressed Header Bit- Vector but before the checksum) are only included when necessary. Their presence or absence is indicated by the corresponding bits in the Compressed Header Bit-Vector. In the figure, each optional field shows three elements of information: the bit of the Compressed Header Bit-Vector that signals its presence, the name of the field, and the length of the field in octets. These are shown in the format “Bit: Name (Length)”.

0	1	2	3	4	5	6	7	
Connection ID								Octet 1
More	TS1	TS2	RB	SNACK	Push	S	A	Octet 2
Opts	Pad	URG	ACKR	Res	RST	Res	FIN	Octet 3
								Octet 4
A: Ack Number (4 octets)								
S: Sequence Number (4 octets)								
TS1: Outbound timestamp (format-dependent)								
TS2: Echo reply timestamp (format-dependent)								
Opts: Uncompressed TCP Options Length (1 octet)								
Opts: Uncompressed TCP Options (data-dependent)								
Pad: Optional Pad (1 octet)								
Checksum Octet 1								
Checksum Octet 2								
Data								First Octet of data begins at offset ≤ n ≤ 64

The Loss-Tolerant Header Compression algorithm supports “piggy-backing” of acknowledgments on data-carrying segments (as in uncompressed TCP), but *it is an implementation option* whether to exercise this ability. The compressor for a particular implementation may send acknowledgments (and other information not directly related to the data being transferred) separately from the data in order to ensure a constant header size for data-carrying segments. (This can aid in packing fixed-length lower-layer frames when bulk data is to be transferred.) In accordance with the robustness principle stated in RFC 1122 and in TCP, a decompressor must be prepared to accept piggy-backed acknowledgments even if the compressor in that implementation does not generate them. (The robustness principle roughly states “be generous in what you accept and conservative in what you send,” and is intended to promote interoperability.)

DECOMPRESSOR PROCESSING

Upon receipt of a segment of type "Loss-Tolerantly Compressed TCP" from the Network Layer, along with relevant Network-layer information, such as the length of the packet, incoming source timestamp, and the source and destination addresses:

a) the decompressor shall use the Connection Identifier and network addresses to find the TCP endpoint with which this packet should be associated;
b) if the endpoint is not located, the decompressor shall discard the segment and log an error;
c) if the endpoint is located, the decompressor shall use the information from the TCP pseudo header to verify the checksum in the compressed segment;
d) if the checksum fails, the decompressor shall discard the segment and log an error;
e) the decompressor shall reconstruct the TCP header using a template created from the uncompressed header of the previous segment received on the connection with all fields except the port information initialized to zero;
f) the decompressor shall save the reconstructed header for use in decompressing subsequent compressed packets;
g) the decompressor shall output the decompressed segment for immediate processing by TCP or insertion into an out-of-sequence queue.

IV References:

[1] J. Postel. *Transmission Control Protocol*. IAB STD 7. RFC 793, September 1, 1981. <URL: http://ds.internic.net/rfc/rfc793.txt>.

[2] *Space Communications Protocol Specification (SCPS)—Transport Protocol (SCPS-TP)*. Draft Recommendation for Space Data System Standards, CCSDS 714.0-R-3. Washington, D.C.: CCSDS, September 1997. <URL: ftp://ftp.ccsds.org/pub/ccsds/pdf/CCSDS-714.0-R-3.pdf>

[3] The SCPS web page, <URL: http://www.scps.org>

[4] V. Jacobson, *Compressing TCP/IP Headers for Low-Speed Serial Links*, RFC 1144, February 1990.

[5] Mikael Degermark, Bjorn Nordgren, and Stephen Pink. *IP Header Compression*, December 1997. Internet-Draft draft-degermark-ipv6-hc-08.txt (work in progress).

Filename:	IANA TP Option Request1.doc
Directory:	C:\EUDORA\Attach
Template:	C:\Program Files\Microsoft Office 97\Templates\NORMAL.DOT
Title:	To:
Subject:	
Author:	Keith Scott
Keywords:	
Comments:	
Creation Date:	12/21/98 5:31 PM
Change Number:	137
Last Saved On:	12/22/98 3:29 PM
Last Saved By:	Keith Scott
Total Editing Time:	1,062 Minutes
Last Printed On:	12/22/98 4:26 PM
As of Last Complete Printing	
Number of Pages:	13
Number of Words:	4,984 (approx.)
Number of Characters:	28,410 (approx.)