Network Working Group
Request for Comments: 4413
Category: Informational

M. West
S. McCann
Siemens/Roke Manor Research

March 2006

TCP/IP Field Behavior

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This memo describes TCP/IP field behavior in the context of header compression. Header compression is possible because most header fields do not vary randomly from packet to packet. Many of the fields exhibit static behavior or change in a more or less predictable way. When a header compression scheme is designed, it is of fundamental importance to understand the behavior of the fields in detail. An example of this analysis can be seen in RFC 3095. This memo performs a similar role for the compression of TCP/IP headers.

Table of Contents

1. Introduction

This document describes the format of the TCP/IP header and the header field behavior, i.e., how fields vary within a TCP flow. The description is presented in the context of header compression.

Since the IP header does exhibit slightly different behavior from that previously presented in RFC 3095 [31] for UDP and RTP, it is also included in this document.

This document borrows much of the classification text from RFC 3095 [31], rather than inserting many references to that document.

According to the format presented in RFC 3095 [31], TCP/IP header fields are classified and analyzed in two steps. First, we have a general classification in Section 2, where the fields are classified on the basis of stable knowledge and assumptions. This general classification does not take into account the change characteristics of changing fields, as those will vary more or less depending on the implementation and on the application used. Section 3 considers how field values can be used to optimize short-lived flows. A more detailed analysis of the change characteristics is then done in Section 4. Finally, Section 5 summarizes with conclusions about how the various header fields should be handled by the header compression scheme to optimize compression.

A general question raised by this analysis is: what 'baseline' definition of all possible TCP/IP implementations is to be considered? This review is based on an analysis of currently deployed TCP implementations supporting mechanisms standardised by the IETF.

The general requirement for transparency is also interesting. A number of recent proposals for extensions to TCP use some of the previously 'reserved' bits in the TCP packet header. Therefore, a 'reserved' bit cannot be taken to have a guaranteed zero value; it may change. Ideally, this should be accommodated by the compression profile.

A number of reserved bits are available for future expansion. A treatment of field behavior cannot predict the future use of such bits, but we expect that they will be used at some point. Given this, a compression scheme can optimise for the current situation but should be capable of supporting any arbitrary usage of the reserved bits. However, it is impossible to optimise for usage patterns that have yet to be defined.

2. General classification

The following definitions (and some text) are copied from RFC 3095 [31], Appendix A. Differences of IP field behavior between RFC 3095 [31] (i.e., IP/UDP/RTP behavior for audio and video applications) and this document have been identified.

For the following, we define "session" as a TCP packet stream, being a series of packets with the same IP addresses and port numbers. A packet flow is defined by certain fields (see STATIC-DEF, below) and may be considered a subset of a session. See [31] for a fuller discussion of separation of sessions into streams of packets for header compression.

At a general level, the header fields are separated into 5 classes:

- INFERRED

  These fields contain values that can be inferred from other values (for example, the size of the frame carrying the packet) and thus do not have to be handled at all by the compression scheme.

- STATIC

  These fields are expected to be constant throughout the lifetime of the packet stream. Static information must in some way be communicated once.

- STATIC-DEF

  STATIC fields whose values define a packet stream. They are in general handled as STATIC.

- STATIC-KNOWN

  These STATIC fields are expected to have well-known values and therefore do not need to be communicated at all.

- CHANGING

  These fields are expected to vary randomly within a limited value set or range or in some other manner.

In this section, each of the IP and TCP header fields is assigned to one of these classes. For all fields except those classified as CHANGING, the motives for the classification are also stated. In section 4, CHANGING fields are further examined and classified on the basis of their expected change behavior.

2.1. IP Header Fields

2.1.1. IPv6 Header Fields

Field	Size (bits)	Class
Version	4	STATIC
DSCP*	6	ALTERNATING
ECT flag*	1	CHANGING
CE flag*	1	CHANGING
Flow Label	20	STATIC-DEF
Payload Length	16	INFERRED
Next Header	8	STATIC
Hop Limit	8	CHANGING
Source Address	128	STATIC-DEF
Destination Address	128	STATIC-DEF

* Differs from RFC 3095 [31]. (The DSCP, ECT, and CE flags were amalgamated into the Traffic Class octet in RFC 3095).

Figure 1. IPv6 Header Fields

- Version

  The version field states which IP version is used. Packets with different values in this field must be handled by different IP stacks. All packets of a packet stream must therefore be of the same IP version. Accordingly, the field is classified as STATIC.

- Flow Label

  This field may be used to identify packets belonging to a specific packet stream. If the field is not used, its value should be zero. Otherwise, all packets belonging to the same stream must have the same value in this field, it being one of the fields that define the stream. The field is therefore classified as STATIC-DEF.

- Payload Length

  Information about packet length (and, consequently, payload length) is expected to be provided by the link layer. The field is therefore classified as INFERRED.

- Next Header

  This field will usually have the same value in all packets of a packet stream. It encodes the type of the subsequent header. Only when extension headers are sometimes absent will the field change its value during the lifetime of the stream. The field is therefore classified as STATIC. The classification of STATIC is inherited from RFC 3095 [31]. However, note that the next header field is actually determined by the type of the following header. Thus, it might be more appropriate to view this as an inference, although this depends upon the specific implementation of the compression scheme.

- Source and Destination Addresses

  These fields are part of the definition of a stream and therefore must be constant for all packets in the stream. The fields are therefore classified as STATIC-DEF.

  This might be considered as a slightly simplistic view. In this document, the IP addresses are associated with the transport layer connection and assumed to be part of the definition of a flow. More complex flow-separation could, of course, be considered (see also RFC 3095 [31] for more discussion of this issue). Where tunneling is being performed, the use of the IP addresses in outer tunnel headers is also assumed to be STATIC-DEF.

The total size of the fields in each class is as follows:

Class	Size (octets)
INFERRED	2
STATIC	1.5
STATIC-DEF	34.5
STATIC-KNOWN	0
CHANGING	2

Figure 2: Field sizes

2.1.2. IPv4 Header Fields

Field	Size (bits)	Class
Version	4	STATIC
Header Length	4	STATIC-KNOWN
DSCP*	6	ALTERNATING
ECT flag*	1	CHANGING
CE flag*	1	CHANGING
Packet Length	16	INFERRED
Identification	16	CHANGING
Reserved flag*	1	CHANGING
Don't Fragment flag*	1	CHANGING
More Fragments flag	1	STATIC-KNOWN
Fragment Offset	13	STATIC-KNOWN
Time To Live	8	CHANGING
Protocol	8	STATIC
Header Checksum	16	INFERRED
Source Address	32	STATIC-DEF
Destination Address	32	STATIC-DEF

* Differs from RFC 3095 [31]. (The DSCP, ECT and CE flags were amalgamated into the TOS octet in RFC 3095; the DF flag behavior is considered later; the reserved field is discussed below).

Figure 3. IPv4 Header Fields

- Version

  The version field states which IP version is used. Packets with different values in this field must be handled by different IP stacks. All packets of a packet stream must therefore be of the same IP version. Accordingly, the field is classified as STATIC.

- Header Length

  As long as no options are present in the IP header, the header length is constant and well known. If there are options, the fields would be STATIC, but it is assumed here that there are no options. The field is therefore classified as STATIC-KNOWN.

- Packet Length

  Information about packet length is expected to be provided by the link layer. The field is therefore classified as INFERRED.

- Flags

  The Reserved flag must be set to zero, as defined in RFC 791 [1]. In RFC 3095 [31] the field is therefore classified as STATIC-KNOWN. However, it is expected that reserved fields may be used at some future point. It is undesirable to select an encoding that would preclude the use of a compression profile for a future change in the use of reserved fields. For this reason, the alternative encoding of CHANGING is used. (A compression profile can, of course, still optimise for the current situation, where the field value is known to be 0).

  The More Fragments (MF) flag is expected to be zero because fragmentation is, ideally, not expected. However, it is also understood that some scenarios (for example, some tunnelling architectures) do cause fragmentation. In general, though, fragmentation is not expected to be common in the Internet due to a combination of initial MSS negotiation and subsequent use of path-MTU discovery. RFC 3095 [31] points out that, for RTP, only the first fragment will contain the transport layer protocol header; subsequent fragments would have to be compressed with a different profile. This is also obviously the case for TCP. If fragmentation were to occur, the first fragment, by definition, would be relatively large, minimizing the header overhead. Subsequent fragments would be compressed with another profile. It is therefore considered undesirable to optimise for fragmentation in performing header compression. The More Fragments flag is therefore classified as STATIC-KNOWN.

- Fragment Offset

  Under the assumption that no fragmentation occurs, the fragment offset is always zero. The field is therefore classified as STATIC-KNOWN. Even if fragmentation were to be further considered, only the first fragment would contain the TCP header, and the fragment offset of this packet would still be zero.

- Protocol

  This field will usually have the same value in all packets of a packet stream. It encodes the type of the subsequent header.

Only where the sequence of headers changes (e.g., an extension header is inserted or deleted or a tunnel header is added or removed) will the field change its value. The field is therefore classified as STATIC. Whether such a change would cause the sequence of packets to be treated as a new flow (for header compression) is an issue for profile design. ROHC profiles must be able to cope with extension headers and tunnelling, but the choice of strategy is outside the scope of this document.

o Header Checksum

The header checksum protects individual hops from processing a corrupted header. When almost all IP header information is compressed away, there is no point in having this additional checksum. Instead, it can be regenerated at the decompressor side. The field is therefore classified as INFERRED.

Note that the TCP checksum does not protect the whole TCP/IP header, but only the TCP pseudo-header (and the payload). Compare this with ROHC [31], which uses a CRC to verify the uncompressed header. Given the need to validate the complete TCP/IP header, the cost of computing the TCP checksum over the entire payload, and known weaknesses in the TCP checksum [37], an additional check is necessary. Therefore, it is highly desirable that some additional checksum (such as a CRC) will be used to validate correct decompression.

o Source and Destination Addresses

These fields are part of the definition of a stream and must thus be constant for all packets in the stream. The fields are therefore classified as STATIC-DEF.

The total size of the fields in each class is as follows:

Class	Size (octets)
INFERRED	4
STATIC*	1.5
STATIC-DEF	8
STATIC-KNOWN*	2.25
CHANGING*	4.25

* Differs from RFC 3095 [31]

Figure 4. Field sizes

2.2. TCP Header Fields

Field	Size (bits)	Class
Source Port	16	STATIC-DEF
Destination Port	16	STATIC-DEF
Sequence Number	32	CHANGING
Acknowledgement Num	32	CHANGING
Data Offset	4	INFERRED
Reserved	4	CHANGING
CWR flag	1	CHANGING
ECE flag	1	CHANGING
URG flag	1	CHANGING
ACK flag	1	CHANGING
PSH flag	1	CHANGING
RST flag	1	CHANGING
SYN flag	1	CHANGING
FIN flag	1	CHANGING
Window	16	CHANGING
Checksum	16	CHANGING
Urgent Pointer	16	CHANGING
Options	0(-352)	CHANGING

Figure 5: TCP header fields

- Source and Destination ports

  These fields are part of the definition of a stream and must thus be constant for all packets in the stream. The fields are therefore classified as STATIC-DEF.

- Data Offset

  The number of 4 octet words in the TCP header, indicating the start of the data. It is always a multiple of 4 octets. It can be re-constructed from the length of any options, and thus it is not necessary to carry this explicitly. The field is therefore classified as INFERRED.

2.3. Summary for IP/TCP

Summarizing this for IP/TCP, one obtains the following:

Class \ IP ver	IPv6 (octets)	IPv4 (octets)
INFERRED	2 + 4 bits	4 + 4 bits
STATIC	1 + 4 bits	1 + 4 bits
STATIC-DEF	38 + 4 bits	12
STATIC-KNOWN	-	2 + 2 bits
CHANGING	17 + 4 bits	19 + 6 bits
Totals	60	40

(Excludes options, which are all classified as CHANGING).

Figure 6. Overall field sizes

3. Classification of Replicable Header Fields

Where multiple flows either overlap in time or occur sequentially within a short space of time, there can be a great deal of similarity in header field values. Such commonality of field values is reflected in the compression context. Thus, it should be possible to utilise commonality between fields across different flows to improve the compression ratio. In order to do this, it is important to understand the 'replicable' characteristics of the various header fields.

The key concept is that of 'replication': an existing context is used as a baseline and replicated to initialise a new context. Those fields that are the same are then automatically initialised in the new context. Those that have changed will be updated or overwritten with values from the initialisation packet that triggered the replication. This section considers the commonality between fields in different flows.

Note, however, that replication is based on contexts (rather than on just field values), so compressor-created fields that are part of the context may also be included. These, of course, are dependent upon the nature of the compression protocol (ROHC profile) being applied.

A brief analysis of the relationship of TCP/IP fields among 'replicable' packet streams follows.

'N/A': The field need not be considered in the replication process, as it is inferred or known 'a priori' (and, therefore, does not appear in the context).

'No': The field cannot be replicated since its change pattern between two packet flows is uncorrelated.

'Yes': The field may be replicated. This does not guarantee that the field value will be the same across two candidate streams, only that it might be possible to exploit replication to increase the compression ratio. Specific encoding methods can be used to improve the compression efficiency.

3.1. IPv4 Header (Inner and/or Outer)

Field	Class	Replicable
Version	STATIC	N/A
Header Length	STATIC-KNOWN	N/A
DSCP	ALTERNATING	No (1)
ECT flag	CHANGING	No (2)
CE flag	CHANGING	No (2)
Packet Length	INFERRED	N/A
Identification	CHANGING	Yes (3)
Reserved flag	CHANGING	No (4)
Don't Fragment flag	CHANGING	Yes (5)
More Fragments flag	STATIC-KNOWN	N/A
Fragment Offset	STATIC-KNOWN	N/A
Time To Live	CHANGING	Yes
Protocol	STATIC	N/A
Header Checksum	INFERRED	N/A
Source Address	STATIC-DEF	Yes
Destination Address	STATIC-DEF	Yes

Figure 7: IPv4 header

(1) The DSCP is marked according to the application's requirements. If it can be assumed that replicable connections belong to the same diffserv class, then it is likely that the DSCP will be replicable. The DSCP can be set not only by the sender but by any packet marker. Thus, a flow may have a number of DSCP values at different points in the network. However, header compression

operates on a point-to-point link and so would expect to see a relatively stable value. If re-marking is being done based on the state of a meter, then the value may change mid-flow. Overall, though, we expect supporting replication of the DSCP to be useful for header compression.

(2) It is not possible for the ECN bits to be replicated (note that use of the ECN nonce scheme [19] is anticipated). However, it seems likely that all TCP flows between ECN-capable hosts will use ECN, the use (or not) of ECN for flows between the same end-points might be considered replicable. See also note (4).

(3) The replicable context for this field includes the IP-ID, NBO, and RND flags (as described in ROHC RTP). This highlights that the replication is of the context, rather than just the header field values and, as such, needs to be considered based on the exact nature of compression applied to each field.

(4) Since the possible future behavior of the 'Reserved Flag' cannot be predicted, it is not considered as replicable. However, it might be expected that the behavior of the reserved flag between the same end-points will be similar. In this case, any selection of packet formats (for example) based on this behavior might carry across to the new flow. In the case of packet formats, this can probably be considered as a compressor-local decision.

(5) In theory, the DF bit may be replicable. However, this is not guaranteed and, in practice, it is unlikely to be useful to do this. From the perspective of header compression, having to indicate whether or not a 1-bit flag should be replicated or specified explicitly is likely to require more bits than simply conveying the value of the flag. We do not rule out DF replication.

3.2. IPv6 Header (inner and/or outer)

Field	Class	Replicable
Version	STATIC	N/A
Traffic Class	CHANGING	Yes (1)
ECT flag	CHANGING	No (2)
CE flag	CHANGING	No (2)
Flow Label	STATIC-DEF	N/A
Payload Length	INFERRED	N/A
Next Header	STATIC	N/A
Hop Limit	CHANGING	Yes
Source Address	STATIC-DEF	Yes
Destination Address	STATIC-DEF	Yes

(1) See comment about DSCP field for IPv4, above.
(2) See comment about ECT and CE flags for IPv4, above.

Figure 8. IPv6 Header

3.3. TCP Header

Field	Class	Replicable
Source Port	STATIC-DEF	Yes (1)
Destination Port	STATIC-DEF	Yes (1)
Sequence Number	CHANGING	No (2)
Acknowledgement Number	CHANGING	No
Data Offset	INFERRED	N/A
Reserved Bits	CHANGING	No (3)
Flags		
CWR	CHANGING	No (4)
ECE	CHANGING	No (4)
URG	CHANGING	No
ACK	CHANGING	No
PSH	CHANGING	No
RST	CHANGING	No
SYN	CHANGING	No
FIN	CHANGING	No
Window	CHANGING	Yes
Checksum	CHANGING	No
Urgent Pointer	CHANGING	Yes (5)

Figure 9: TCP Header

(1) On the server side, the port number is likely to be a well-known value. On the client side, the port number is generally selected by the stack automatically. Whether the port number is replicable depends upon how the stack chooses the port number. Whilst most implementations use a simple scheme that sequentially picks the next available port number, it may not be desirable to rely on this behavior.

(2) With the recommendation (and expected deployment) of TCP Initial Sequence Number randomization, defined in RFC 1948 [10], it will be impossible to share the sequence number. Thus, this field will not be regarded as replicable.

(3) See comment (4) for the IPv4 header, above.

(4) See comment (2) on ECN flags for the IPv4 header, above.

(5) The urgent pointer is very rarely used. This means that, in practice, the field may be considered replicable.

3.4. TCP Options

Option	SYN-only (1)	Replicable
End of Option List	No	No (2)
No-Operation	No	No (2)
Maximum Segment Size	Yes	Yes
Window Scale	Yes	Yes
SACK-Permitted	Yes	Yes
SACK	No	No
Timestamp	No	No

Figure 10. TCP Options

(1) This indicates whether the option only appears in SYN packets. Options that are not 'SYN-only' may appear in any packet. Many TCP options are used only in SYN packets. Some options, such as MSS, Window Scale, and SACK-Permitted, will tend to have the same value among replicable packet streams.

Thus, to support context sharing, the compressor should maintain such TCP options in the context (even though they only appear in the SYN segment).

(2) Since these options have fixed values, they could be regarded as replicable. However, the only interesting thing to convey about

these options is their presence. If it is known that such an option exists, its value is defined.

3.5. Summary of Replication

From the above analysis, it can be seen that there are reasonable grounds for exploiting redundancy between flows as well as between packets within a flow. Simply consider the advantage of being able to elide the source and destination addresses for a repeated connection between two IPv6 endpoints. There will also be a cost (in terms of complexity and robustness) for replicating contexts, and this must be considered when one decides what constitutes an appropriate solution.

Finally, note that the use of replication requires that the compressor have a suitable degree of confidence that the source data is present and correct at the decompressor. This may place some restrictions on which of the 'changing' fields, in particular, can be utilised during replication.

4. Analysis of Change Patterns of Header Fields

To design suitable mechanisms for efficient compression of all header fields, their change patterns must be analyzed. For this reason, an extended classification is done based on the general classification in 2, considering the fields that were labeled CHANGING in that classification.

The CHANGING fields are separated into five different subclasses:

- STATIC

  These are fields that were classified as CHANGING on a general basis, but that are classified as STATIC here due to certain additional assumptions.

- SEMISTATIC

  These fields are STATIC most of the time. However, occasionally the value changes but reverts to its original value after a known number of packets.

- RARELY-CHANGING (RC)

  These are fields that change their values occasionally and then keep their new values.

- ALTERNATING

  These fields alternate between a small number of different values.

- IRREGULAR

  These, finally, are the fields for which no useful change pattern can be identified.

To further expand the classification possibilities without increasing complexity, the classification can be done either according to the values of the field and/or according to the values of the deltas for the field.

When the classification is done, other details are also stated regarding possible additional knowledge about the field values and/or field deltas, according to the classification. For fields classified as STATIC or SEMISTATIC, the value of the field could be not only STATIC but also well-KNOWN a priori (two states for SEMISTATIC fields). For fields with non-irregular change behavior, it could be known that changes are usually within a LIMITED range compared to the maximal change for the field. For other fields, the values are completely UNKNOWN.

Figure 11 classifies all the CHANGING fields on the basis of their expected change patterns. (4) refers to IPv4 fields and (6) refers to IPv6.

Field	Value/Delta	Class	Knowledge
DSCP(4) / Tr.Class(6)	Value	ALTERNATING	UNKNOWN
IP ECT flag(4)	Value	RC	UNKNOWN
IP CE flag(4)	Value	RC	UNKNOWN
IP Id(4) Sequential	Delta	STATIC	KNOWN
IP Id(4) Seq. jump	Delta	RC	LIMITED
IP Id(4) Random	Value	IRREGULAR	UNKNOWN
IP DF flag(4)	Value	RC	UNKNOWN
IP TTL(4) / Hop Lim(6)	Value	ALTERNATING	LIMITED
TCP Sequence Number	Delta	IRREGULAR	LIMITED
TCP Acknowledgement Num	Delta	IRREGULAR	LIMITED
TCP Reserved	Value	RC	UNKNOWN
TCP flags			
ECN flags	Value	IRREGULAR	UNKNOWN
CWR flag	Value	IRREGULAR	UNKNOWN
ECE flag	Value	IRREGULAR	UNKNOWN
URG flag	Value	IRREGULAR	UNKNOWN
ACK flag	Value	SEMISTATIC	KNOWN
PSH flag	Value	IRREGULAR	UNKNOWN
RST flag	Value	IRREGULAR	UNKNOWN
SYN flag	Value	SEMISTATIC	KNOWN
FIN flag	Value	SEMISTATIC	KNOWN
TCP Window	Value	ALTERNATING	KNOWN
TCP Checksum	Value	IRREGULAR	UNKNOWN
TCP Urgent Pointer	Value	IRREGULAR	KNOWN
TCP Options	Value	IRREGULAR	UNKNOWN

Figure 11. Classification of CHANGING Fields

The following subsections discuss the various header fields in detail. Note that Table 1 and the discussion below do not consider changes caused by loss or reordering before the compression point.

4.1. IP Header

4.1.1. IP Traffic-Class / Type-Of-Service (TOS)

The Traffic-Class (IPv6) or Type-Of-Service/DSCP (IPv4) field might be expected to change during the lifetime of a packet stream. This analysis considers several RFCs that describe modifications to the original RFC 791 [1].

The TOS byte was initially described in RFC 791 [1] as 3 bits of precedence followed by 3 bits of TOS and 2 reserved bits (defined to be zero). RFC 1122 [21] extended this to specify 5 bits of TOS, although the meanings of the additional 2 bits were not defined. RFC 1349 [23] defined the 4th bit of TOS as 'minimize monetary cost'. The next significant change was in RFC 2474 [14] (obsoleting RFC 1349 [23]). RFC 2474 reworked the TOS octet as 6 bits of DSCP (DiffServ Code Point) plus 2 unused bits. Most recently, RFC 2780 [30] identified the 2 reserved bits in the TOS or traffic class octet for experimental use with ECN.

It is therefore proposed that the TOS (or traffic class) octet be classified as 6 bits for the DSCP and 2 additional bits. These 2 bits may be expected to be zero or to contain ECN data. From a future-proofing perspective, it is preferable to assume the use of ECN, especially with respect to TCP.

It is also considered important that the profile work with legacy stacks, since these will be in existence for some considerable time to come. For simplicity, this will be considered as 6 bits of TOS information and 2 bits of ECN data, so the fields are always considered to be structured the same way.

The DSCP (as for TOS in ROHC RTP) is not expected to change frequently (although it could change mid-flow, for example, as a result of a route change).

4.1.2. ECN Flags

Initially, we describe the ECN flags as specified in RFC 2481 [15] and RFC 3168 [18]. Subsequently, a suggested update is described that would alter the behavior of the flags.

In RFC 2481 [15] there are 2 separate flags, the ECT (ECN Capable Transport) flag and the CE (Congestion Experienced) flag. The ECT

flag, if negotiated by the TCP stack, will be '1' for all data packets and '0' for all 'pure acknowledgement' packets. This means that the behavior of the ECT flag is linked to behavior in the TCP stack. Whether this can be exploited for compression is not clear.

The CE flag is only used if ECT is set to '1'. It is set to '0' by the sender and can be set to '1' by an ECN-capable router in the network to indicate congestion. Thus the CE flag is expected to be randomly set to '1' with a probability dependent on the congestion state of the network and the position of the compressor in the path. Therefore, a compressor located close to the receiver in a congested network will see the CE bit set frequently, but a compressor located close to a sender will rarely, if ever, see the CE bit set to '1'.

A recent experimental proposal [19] suggests an alternative view of these 2 bits. This considers the two bits together to have 4 possible codepoints. Meanings are then assigned to the codepoints:

00 Not ECN capable
01 ECN capable, no congestion (known as ECT(0))
10 ECN capable, no congestion (known as ECT(1))
11 Congestion experienced

The use of 2 codepoints for signaling ECT allows the sender to detect when a receiver is not reliably echoing congestion information.

For the purposes of compression, this update means that ECT(0) and ECT(1) are equally likely (for an ECN capable flow) and that '11' will be seen relatively rarely. The probability of seeing a congestion indication is discussed above in the description of the CE flag.

It is suggested that, for the purposes of compression, ECN with nonces be assumed as the baseline, although the compression scheme must be able to compress the original ECN scheme transparently.

4.1.3. IP Identification

The Identification field (IP ID) of the IPv4 header identifies which fragments constitute a datagram, when fragmented datagrams are reassembled. The IPv4 specification does not specify exactly how this field is to be assigned values, only that each packet should get an IP ID that is unique for the source-destination pair and protocol for the time during which the datagram (or any of its fragments) could be alive in the network. This means that assignment of IP ID values can be done in various ways, which we have separated into three classes:

- Sequential jump

  This is the most common assignment policy in today's IP stacks. A single IP ID counter is used for all packet streams. When the sender is running more than one packet stream simultaneously, the IP ID can increase by more than one between packets in a stream. The IP ID values will be much more predictable and will require fewer bits to transfer than random values, and the packet-to-packet increment (determined by the number of active outgoing packet streams and sending frequencies) will usually be limited.

- Random

  Some IP stacks assign IP ID values by using a pseudo-random number generator. There is thus no correlation between the ID values of subsequent datagrams. Therefore, there is no way to predict the IP ID value for the next datagram. For header compression purposes, this means that the IP ID field needs to be sent uncompressed with each datagram, resulting in two extra octets of header. IP stacks in cellular terminals that need optimum header compression efficiency should not use this IP ID assignment policy.

- Sequential

  This assignment policy keeps a separate counter for each outgoing packet stream, and thus the IP ID value will increment by one for each packet in the stream, except at wrap around. Therefore, the delta value of the field is constant and well known a priori. This assignment policy is the most desirable for header compression purposes. However, its usage is not as common as it perhaps should be.

  In order to avoid violating RFC 791 [1], packets sharing the same IP address pair and IP protocol number cannot use the same IP ID values. Therefore, implementations of sequential policies must make the ID number spaces disjoint for packet streams of the same IP protocol going between the same pair of nodes. This can be done in a number of ways, all of which introduce occasional jumps and thus make the policy less than perfectly sequential. For header compression purposes, less frequent jumps are preferred.

Note that the ID is an IPv4 mechanism and is therefore not a problem for IPv6. For IPv4, the ID could be handled in three different ways. First, we have the inefficient but reliable solution where the ID field is sent as-is in all packets, increasing the compressed headers by two octets. This is the best way to handle the ID field if the sender uses random assignment of the ID field. Second, there can be

solutions with more flexible mechanisms that require fewer bits for the ID handling as long as sequential jump assignment is used. Such solutions will probably require even more bits if random assignment is used by the sender. Knowledge about the sender's assignment policy could therefore be useful when choosing between the two solutions above. Finally, even for IPv4, header compression could be designed without any additional information for the ID field included in compressed headers. To use such schemes, it must be known which assignment policy for the ID field is being used by the sender. That might not be possible to know, which implies that the applicability of such solutions is very uncertain. However, designers of IPv4 stacks for cellular terminals should use an assignment policy close to sequential.

With regard to TCP compression, the behavior of the IP ID field is essentially the same. However, in RFC 3095 [31], the IP ID is generally inferred from the RTP Sequence Number. There is no obvious candidate in the TCP case for a field to offer this 'master sequence number' role.

Clearly, from a busy server, the observed behavior may well be quite erratic. This is a case where the ability to share the IP compression context between a number of flows (between the same end-points) could offer potential benefits. However, this would only have any real impact where there is a large number of flows between one machine and the server. If context sharing is being considered, then it is preferable to share the IP part of the context.

4.1.4. Don't Fragment (DF) flag

Path-MTU discovery (RFC 1191 for IPv4 [6] and RFC 1981 for IPv6 [11]) is widely deployed for TCP, in contrast to little current use for UDP packet streams. This employs the DF flag value of '1' to detect the need for fragmentation in the end-to-end path and to probe the minimum MTU along the network path. End hosts using this technique may be expected to send all packets with DF set to '1', although a host may end PMTU discovery by clearing the DF bit to '0'. Thus, for compression, we expect the field value to be stable.

4.1.5. IP Hop-Limit / Time-To-Live (TTL)

The Hop-Limit (IPv6) or Time-To-Live (IPv4) field is expected to be constant during the lifetime of a packet stream or to alternate between a limited number of values due to route changes.

4.2. TCP Header

Any discussion of compressability of TCP fields borrows heavily from RFC 1144 [22]. However, the premise of how the compression is performed is slightly different, and the protocol has evolved slightly in the intervening time.

4.2.1. Sequence Number

Understanding the sequence and acknowledgement number behavior is essential for a TCP compression scheme.

At the simplest level, the behavior of the sequence number can be described relatively easily. However, there are a number of complicating factors that also need to be considered.

For transferring in-sequence data packets, the sequence number will increment for each packet by between 0 and an upper limit defined by the MSS (Maximum Segment Size) and, if it is being used, by Path-MTU discovery.

There are common MSS values, but these can be quite variable and unpredictable for any given flow. Given this variability and the range of window sizes, it is hard (compared with the RTP case, for example) to select a 'one size fits all' encoding for the sequence number. (The same argument applies equally to the acknowledgement number).

Note that the increment of the sequence number in a packet is the size of the data payload of that packet (including the SYN and FIN flags). This is, of course, exactly the relationship that RFC 1144 [22] exploits to compress the sequence number in the most efficient case. This technique may not be directly applicable to a robust solution, but it may be a useful relationship to consider.

However, at any point on the path (i.e., wherever a compressor might be deployed), the sequence number can be anywhere within a range defined by the TCP window. This is a combination of a number of values (buffer space at the sender; advertised buffer size at the receiver; and TCP congestion control algorithms). Missing packets or retransmissions can cause the TCP sequence number to fluctuate within the limits of this window.

It is desirable to be able to predict the sequence number with some regularity. However, this also appears to be difficult to do. For example, during bulk data transfer, the sequence number will tend to go up by 1 MSS per packet (assuming no packet loss). Higher layer values have been seen to have an impact as well, where sequence

number behavior has been observed with an 8 kbyte repeating pattern -- 5 segments of 1460 bytes followed by 1 segment of 892 bytes. The implementation of TCP and the management of buffers within a protocol stack can affect the behavior of the sequence number.

It may be possible to track the TCP window by the compressor, allowing it to bound the size of these jumps.

For interactive flows (for example, telnet), the sequence number will change by small, irregular amounts. In this case, the Nagle algorithm [3] commonly applies, coalescing small packets where possible in order to reduce the basic header overhead. This may also mean that predictable changes in the sequence number are less likely to occur. The Nagle algorithm is an optimisation and is not required to be used (applications can disable its use). However, it is turned on by default in all common TCP implementations.

Note also that the SYN and FIN flags (which have to be acknowledged) each consume 1 byte of sequence space.

4.2.2. Acknowledgement Number

Much of the information about the sequence number applies equally to the acknowledgement number. However, there are some important differences.

For bulk data transfers, there will tend to be 1 acknowledgement for every 2 data segments. The algorithm is specified in RFC 2581 [16]. An ACK need not always be sent immediately on receipt of a data segment, but it must be sent within 500ms and should be generated for at least every second full-size segment (MSS) of received data. It may be seen from this that the delta for the acknowledgement number is roughly twice that of the sequence number. This is not always the case, and the discussion about sequence number irregularity should be applied.

As an aside, a common implementation bug is ’stretch ACKs’ [33] (acknowledgements may be generated less frequently than every two full-size data segments). This pattern can also occur following loss on the return path.

Since the acknowledgement number is cumulative, dropped packets in the forward path will result in the acknowledgement number remaining constant for a time in the reverse direction. Retransmission of a dropped segment can then cause a substantial jump in the acknowledgement number. These jumps in acknowledgement number are bounded by the TCP window, just as for the jumps in sequence number.

In the acknowledgement case, information about the advertised received window gives a bound to the size of any ACK jump.

4.2.3. Reserved

This field is reserved, and it therefore might be expected to be zero. This can no longer be assumed, due to future-proofing. It is only a matter of time before a suggestion for using the flag is made.

4.2.4. Flags

- ECN-E (Explicit Congestion Notification)

  '1' to echo CE bit in IP header. It will be set in several consecutive headers (until 'acknowledged' by CWR). If ECN nonces are used, then there will be a 'nonce-sum' (NS) bit in the flags, as well. Again, transparency of the reserved bits is crucial for future-proofing this compression scheme. From an efficiency/compression standpoint, the NS bit will either be unused (always '0') or randomly changing. The nonce sum is the 1-bit sum of the ECT codepoints, as described in [19].

- CWR (Congestion Window Reduced)

  '1' to signal congestion window reduced on ECN. It will generally be set in individual packets. The flag will be set once per loss event. Thus, the probability of its being set is proportional to the degree of congestion in the network, but it is less likely to be set than the CE flag.

- ECE (Echo Congestion Experience)

  If 'congestion experienced' is signaled in a received IP header, this is echoed through the ECE bit in segments sent by the receiver until acknowledged by seeing the CWR bit set. Clearly, in periods of high congestion and/or long RTT, this flag will frequently be set to '1'.

  During connection open (SYN and SYN/ACK packets), the ECN bits have special meaning:

  * CWR and ECN-E are both set with SYN to indicate desire to use ECN.

* CWR only is set in SYN-ACK, to agree to ECN.

  (The difference in bit-patterns for the negotiation is such that it will work with broken stacks that reflect the value of reserved bits).

- URG (Urgent Flag)

  '1' to indicate urgent data (which is unlikely with any flag other than ACK).

- ACK (Acknowledgement)

  '1' for all except the initial 'SYN' packet.

- PSH (Push Function Field)

  Generally accepted to be randomly '0' or '1'. However, it may be biased more to one value than the other (this is largely caused by the implementation of the stack).

- RST (Reset Connection)

  '1' to reset a connection (unlikely with any flag other than ACK).

- SYN (Synchronize Sequence Number)

  '1' for the SYN/SYN-ACK, only at the start of a connection.

- FIN (End of Data: FINished)

  '1' to indicate 'no more data' (unlikely with any flag other than ACK).

4.2.5. Checksum

Carried as the end-to-end check for the TCP data. See RFC 1144 [22] for a discussion of why this should be carried. A header compression scheme should not rely upon the TCP checksum for robustness, though, and should apply appropriate error-detection mechanisms of its own. The TCP checksum has to be considered to be randomly changing.

4.2.6. Window

This may oscillate randomly between 0 and the receiver's window limit (for the connection).

In practice, the window will either not change or alternate between a relatively small number of values. Particularly when the window is closing (its value is getting smaller), the change in window is likely to be related to the segment size, but it is not clear that this necessarily offers any compression advantage. When the window is opening, the effect of 'Silly-Window Syndrome' avoidance should be remembered. This prevents the window from opening by small amounts that would encourage the sender to clock out small segments.

When thinking about what fields might change in a sequence of TCP segments, one should note that the receiver can generate 'window update' segments in which only the window advertisement changes.

4.2.7. Urgent Pointer

From a compression point of view, the Urgent Pointer is interesting because it offers an example where 'semantically identical' compression is not the same as 'bitwise identical'. This is because the value of the Urgent Pointer is only valid if the URG flag is set.

However, the TCP checksum must be passed transparently, in order to maintain its end-to-end integrity checking property. Since the TCP checksum includes the Urgent Pointer in its coverage, this enforces bitwise transparency of the Urgent Pointer. Thus, the issue of 'semantic' vs. 'bitwise' identity is presented as a note: the Urgent Pointer must be compressed in a way that preserves its value.

If the URG flag is set, then the Urgent Pointer indicates the end of the urgent data and thus can point anywhere in the window. It may be set (and changing) over several segments. Note that urgent data is rarely used, since it is not a particularly clean way of managing out-of-band data.

4.3. Options

Options occupy space at the end of the TCP header. All options are included in the checksum. An option may begin on any byte boundary. The TCP header must be padded with zeros to make the header length a multiple of 32 bits.

Optional header fields are identified by an option kind field. Options 0 and 1 are exactly one octet, which is their kind field. All other options have their one-octet kind field, followed by a one-octet length field, followed by length-2 octets of option data.

4.3.1. Options Overview

The IANA provides the authoritative list of TCP options. Figure 12 describes the current allocations at the time of publication. Any new option would have a 'kind' value assigned by IANA. The list is available at [20]. Where applicable, the associated RFC is also cited.

Kind	Length octets	Meaning	RFC	Use
0	-	End of Option List	RFC 793	*
1	-	No-Operation	RFC 793	*
2	4	Maximum Segment Size	RFC 793	*
3	3	WSopt - Window Scale	RFC 1323	*
4	2	SACK Permitted	RFC 2018	*
5	N	SACK	RFC 2018	*
6	6	Echo (obsoleted by option 8)	RFC 1072	
7	6	Echo Reply (obsoleted by option 8)	RFC 1072	
8	10	TSopt - Time Stamp Option	RFC 1323	*
9	2	Partial Order Connection Permitted	RFC 1693	
10	3	Partial Order Service Profile	RFC 1693	
11	6	CC	RFC 1644	
12	6	CC.NEW	RFC 1644	
13	6	CC.ECHO	RFC 1644	
14	3	Alternate Checksum Request	RFC 1146	
15	N	Alternate Checksum Data	RFC 1146	
16		Skeeter		
17		Bubba		
18	3	Trailer Checksum Option		
19	18	MD5 Signature Option	RFC 2385	
20		SCPS Capabilities		
21		Selective Negative Acks		
22		Record Boundaries		
23		Corruption experienced		
24		SNAP		
25		Unassigned (released 12/18/00)		
26		TCP Compression Filter		

Figure 12. Common TCP Options

The 'use' column is marked with '*' to indicate options that are most likely to be seen in TCP flows. Also note that RFC 1072 [4] has been obsoleted by RFC 1323 [7], although the original bit usage is defined only in RFC 1072.

4.3.2. Option Field Behavior

Generally speaking, all option fields have been classified as changing. This section describes the behavior of each option referenced within an RFC, listed by 'kind' indicator.

0: End of Option List

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not at the end of each option, and it need only be used if the end of the options would not otherwise coincide with the end of the TCP header. Defined in RFC 793 [2].

There is no data associated with this option, so a compression scheme must simply be able to encode its presence. However, note that since this option marks the end of the list and the TCP options are 4-octet aligned, there may be octets of padding (defined to be '0' in [2]) after this option.

1: No-Operation

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary RFC 793 [2]. There is no data associated with this option, so a compression scheme must simply be able to encode its presence. This may be done by noting that the option simply maintains a certain alignment and that compression need only convey this alignment. In this way, padding can just be removed.

2: Maximum Segment Size

If this option is present, then it communicates the maximum segment size that may be used to send a packet to this end-host. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed RFC 793 [2].

This option is very common. The segment size is a 16-bit quantity. Theoretically, this could take any value; however there are a number of values that are common. For example, 1460 bytes is very common for TCP/IPv4 over Ethernet (though with the increased prevalence of tunnels, for example, smaller

values such as 1400 have become more popular). 536 bytes is the default MSS value. This may allow for common values to be encoded more efficiently.

3: Window Scale Option (WSopt)

This option may be sent in a SYN segment by the TCP end-host
(1) to indicate that the sending TCP end-host is prepared to perform both send and receive window scaling, and
(2) to communicate a scale factor to be applied to its receive window.

The scale factor is encoded logarithmically as a power of 2 (presumably to be implemented by binary shifts). Note that the window in the SYN segment itself is never scaled (RFC 1072 [4]). This option may be sent in an initial segment (i.e., in a segment with the SYN bit on and the ACK bit off). It may also be sent in later segments, but only if a Window Scale option was received in the initial segment. A Window Scale option in a segment without a SYN bit should be ignored. The Window field in a SYN segment itself is never scaled (RFC 1323 [7]).

The use of window scaling does not affect the encoding of any other field during the lifetime of the flow. Only the encoding of the window scaling option itself is important. The window scale must be between 0 and 14 (inclusive). Generally, smaller values would be expected (a window scale of 14 allows for a 1Gbyte window, which is extremely large).

4: SACK-Permitted

This option may be sent in a SYN by a TCP that has been extended to receive (and presumably to process) the SACK option once the connection has opened RFC 2018 [12]. There is no data in this option all that is required is for the presence of the option to be encoded.

5: SACK

This option is to be used to convey extended acknowledgment information over an established connection. Specifically, it is to be sent by a data receiver to inform the data transmitter of non-contiguous blocks of data that have been received and queued. The data receiver awaits the receipt of data in later retransmissions to fill the gaps in sequence space between these blocks. At that time, the data receiver acknowledges the data, normally by advancing the left window edge in the

Acknowledgment Number field of the TCP header. It is important to understand that the SACK option will not change the meaning of the Acknowledgment Number field, whose value will still specify the left window edge, i.e., one byte beyond the last sequence number of fully received data (RFC 2018 [12]).

If SACK has been negotiated (through an exchange of SACK-Permitted options), then this option may occur when dropped segments are noticed by the receiver. Because this identifies ranges of blocks within the receiver's window, it can be viewed as a base value with a number of offsets. The base value (left edge of the first block) can be viewed as offset from the TCP acknowledgement number. There can be up to 4 SACK blocks in a single option. SACK blocks may occur in a number of segments (if there is more out-of-order data 'on the wire'), and this will typically extend the size of or add to the existing blocks.

Alternative proposals such as DSACK RFC 2883 [17] do not fundamentally change the behavior of the SACK block, from the point of view of the information contained within it.

6: Echo

This option carries information that the receiving TCP may send back in a subsequent TCP Echo Reply option (see below). A TCP may send the TCP Echo option in any segment, but only if a TCP Echo option was received in a SYN segment for the connection. When the TCP echo option is used for RTT measurement, it will be included in data segments, and the four information bytes will define the time at which the data segment was transmitted in any format convenient to the sender (see RFC 1072 [4]).

The Echo option is generally not used in practice -- it is obsoleted by the Timestamp option. However, for transparency it is desirable that a compression scheme be able to transport it. (However, there is no benefit in attempting any treatment more sophisticated than viewing it as a generic 'option').

7: Echo Reply

A TCP that receives a TCP Echo option containing four information bytes will return these same bytes in a TCP Echo Reply option. This TCP Echo Reply option must be returned in the next segment (e.g., an ACK segment) that is sent. If more than one Echo option is received before a reply segment is sent, the TCP must choose only one of the options to echo,

ignoring the others; specifically, it must choose the newest segment with the oldest sequence number (see RFC 1072 [4]).

The Echo Reply option is generally not used in practice -- it is obsoleted by the Timestamp option. However, for transparency it is desirable that a compression scheme be able to transport it. (However, there is no benefit in attempting any more sophisticated treatment than viewing it as a generic 'option').

8: Timestamps

This option carries two four-byte timestamp fields. The Timestamp Value field (TSval) contains the current value of the timestamp clock of the TCP sending the option. The Timestamp Echo Reply field (TSecr) is only valid if the ACK bit is set in the TCP header; if it is valid, it echoes a timestamp value that was sent by the remote TCP in the TSval field of a Timestamps option. When TSecr is not valid, its value must be zero. The TSecr value will generally be from the most recent Timestamp option that was received; however, there are exceptions that are explained below. A TCP may send the Timestamps option (TSopt) in an initial segment (i.e., a segment containing a SYN bit and no ACK bit), and it may send a TSopt in other segments only if it received a TSopt in the initial segment for the connection (see RFC 1323 [7]). Timestamps are quite commonly used. If timestamp options are exchanged in the connection set-up phase, then they are expected to appear on all subsequent segments. If this exchange does not happen, then they will not appear for the remainder of the flow.

Because the value being carried is a timestamp, it is logical to expect that the entire value need not be carried. There is no obvious pattern of increments that might be expected, however.

An important reason for using the timestamp option is to allow detection of sequence space wrap-around (Protection Against Wrapped Sequence-number, or PAWS, see RFC 1323 [7]). It is not expected that this is a serious concern on the links on which TCP header compression would be deployed, but it is important that the integrity of this option be maintained. This issue is discussed in, for example, RFC 3150 [32]. However, the proposed Eifel algorithm [35] makes use of timestamps, so it is currently recommended that timestamps be used for cellular-type links [34].

With regard to compression, note that the range of resolutions for the timestamp suggested in RFC 1323 [7] is quite wide (1ms to 1s per 'tick'). This (along with the perhaps wide variation in RTT) makes it hard to select a set of encodings that will be optimal in all cases.

9: Partial Order Connection (POC) permitted

This option represents a simple indicator communicated between the two peer transport entities to establish the operation of the POC protocol. See RFC 1693 [9].

The Partial Order Connection option sees little (or no) use in the current Internet, so the only requirement is that the header compression scheme be able to encode it.

10: POC service profile

This option serves to communicate the information necessary to carry out the job of the protocol -- the type of information that is typically found in the header of a TCP segment. The Partial Order Connection option sees little (or no) use in the current Internet, so the only requirement is that the header compression scheme be able to encode it.

11: Connection Count (CC)

This option is part of the implementation of TCP Accelerated Open (TAO) that effectively bypasses the TCP Three-Way Handshake (3WHS). TAO introduces a 32-bit incarnation number, called a "connection count" (CC), that is carried in a TCP option in each segment. A distinct CC value is assigned to each direction of an open connection. The implementation assigns monotonically increasing CC values to successive connections that it opens actively or passively (see RFC 1644 [8]). This option sees little (or no) use in the current Internet, so the only requirement is that the header compression scheme be able to encode it.

12: CC.NEW

Correctness of the TAO mechanism requires that clients generate monotonically increasing CC values for successive connection initiations. Receiving a CC.NEW causes the server to invalidate its cache entry and to do a 3WHS. See RFC 1644 [8]. This option sees little (or no) use in the current Internet, so the only requirement is that the header compression scheme be able to encode it.

13: CC.ECHO

When a server host sends a segment, it echoes the connection count from the initial in a CC.ECHO option, which is used by the client host to validate the segment (see RFC 1644 [8]). This option sees little (or no) use in the current Internet, so the only requirement is that the header compression scheme be able to encode it.

14: Alternate Checksum Request

This option may be sent in a SYN segment by a TCP to indicate that the TCP is prepared to both generate and receive checksums based on an alternate algorithm. During communication, the alternate checksum replaces the regular TCP checksum in the checksum field of the TCP header. Should the alternate checksum require more than 2 octets to transmit, either the checksum may be moved into a TCP Alternate Checksum Data Option and the checksum field of the TCP header be sent as zero, or the data may be split between the header field and the option. Alternate checksums are computed over the same data as the regular TCP checksum; see RFC 1146 [5].

This option sees little (or no) use in the current Internet, so the only requirement is that the header compression scheme be able to encode it. It would only occur in connection set-up (SYN) packets. Even if this option were used, it would not affect the handling of the checksum, since this should be carried transparently in any case.

15: Alternate Checksum Data

This field is used only when the alternate checksum that is negotiated is longer than 16 bits. These checksums will not fit in the checksum field of the TCP header and thus at least part of them must be put in an option. Whether the checksum is split between the checksum field in the TCP header and the option or the entire checksum is placed in the option is determined on a checksum-by-checksum basis. The length of this option will depend on the choice of alternate checksum algorithm for this connection; see RFC 1146 [5].

If an alternative checksum was negotiated in the connection set-up, then this option may appear on all subsequent packets (if needed to carry the checksum data). However, this option is in practice never seen, so the only requirement is that the header compression scheme be able to encode it.

16 - 18:

These non-RFC option types are not considered in this document.

19: MD5 Digest

Every segment sent on a TCP connection to be protected against spoofing will contain the 16-byte MD5 digest produced by applying the MD5 algorithm to a concatenated block of data [13].

Upon receiving a signed segment, the receiver must validate it by calculating its own digest from the same data (using its own key) and comparing the two digests. A failing comparison must result in the segment's being dropped and must not produce any response back to the sender. Logging the failure is probably advisable.

Unlike other TCP extensions (e.g., the Window Scale option [7]), the absence of the option in the SYN-ACK segment must not cause the sender to disable its sending of signatures. This negotiation is typically done to prevent some TCP implementations from misbehaving upon receiving options in non-SYN segments. This is not a problem for this option, since the SYN-ACK sent during connection negotiation will not be signed and will thus be ignored. The connection will never be made, and non-SYN segments with options will never be sent. More importantly, the sending of signatures must be under the complete control of the application, not at the mercy of a remote host not understanding the option. MD5 digest information should, like any cryptographically secure data, be incompressible. Therefore the compression scheme must simply transparently carry this option, if it occurs.

20 - 26;

Thse non-RFC option types are not considered in this document. This only means that their behavior is not described in detail, as a compression scheme is not expected to be optimised for these options. However, any unrecognised option must be carried by a TCP compression scheme transparently, in order to work efficiently in the presence of new or rare options.

The above list covers options known at the time of writing. Other options are expected to be defined. It is important that any future options can be handled by a header compression scheme. The processing of as-yet undefined options cannot be optimised but, at the very least, unknown options should be carried transparently.

The current model for TCP options is that an option is negotiated in the SYN exchange and used thereafter, if the negotiation succeeds. This leads to some assumptions about the presence of options (being only on packets with the SYN flag set, or appearing on every packet, for example). Where such assumptions hold true, it may be possible to optimise compression of options slightly. However, it is seen as undesirable to be so constrained, as there is no guarantee that option handling and negotiation will remain the same in the future. Also note that a compressor may not process the SYN packets of a flow and cannot, therefore, be assumed to know which options have been negotiated.

5. Other Observations

5.1. Implicit Acknowledgements

There may be a small number of cues for 'implicit acknowledgements' in a TCP flow. Even if the compressor only sees the data transfer direction, for example, seeing a packet without the SYN flag set implies that the SYN packet has been received.

There is a clear requirement for the deployment of compression to be topologically independent. This means that it is not actually possible to be sure that seeing a data packet at the compressor guarantees that the SYN packet has been correctly received by the decompressor (as the SYN packet may have taken an alternative path).

However, there may be other such cues, which may be used in certain circumstances to improve compression efficiency.

5.2. Shared Data

It can be seen that there are two distinct deployments (i) where the forward (data) and reverse (ACK) path are both carried over a common link, and (ii) where the forward (data) and reverse (ACK) path are carried over different paths, with a specific link carrying packets corresponding to only one direction of communication.

In the former case, a compressor and decompressor could be colocated. It may then be possible for the compressor and decompressor at each end of the link to exchange information. This could lead to possible optimizations.

For example, acknowledgement numbers are generally taken from the sequence numbers in the opposite direction. Since an acknowledgement cannot be generated for a packet that has not passed across the link, this offers an efficient way of encoding acknowledgements.

5.3. TCP Header Overhead

For a TCP bulk data-transfer, the overhead of the TCP header does not form a large proportion of the data packet (e.g., < 3% for a 1460 octet packet), particularly compared to the typical RTP voice case. Spectral efficiency is clearly an important goal. However, extracting every last bit of compression gain offers only marginal benefit at a considerable cost in complexity. This trade-off, of efficiency and complexity, must be addressed in the design of a TCP compression profile.

However, in the acknowledgement direction (i.e., for 'pure' acknowledgement headers), the overhead could be said to be infinite (since there is no data being carried). This is why optimizations for the acknowledgement path may be considered useful.

There are a number of schemes for manipulating TCP acknowledgements to reduce the ACK bandwidth. Many of these are documented in [33] and [32]. Most of these schemes are entirely compatible with header compression, without requiring any particular support. While it is not expected that a compression scheme will be optimised for experimental options, it is useful to consider these when developing header compression schemes, and vice versa. A header compression scheme must be able to support any option (including ones as yet undefined).

5.4. Field Independence and Packet Behavior

It should be apparent that direct comparisons with the highly 'packet'-based view of RTP compression are hard. RTP header fields tend to change regularly per-packet, and many fields (IPv4 IP ID, RTP sequence number, and RTP timestamp, for example) typically change in a dependent manner. However, TCP fields, such as sequence number tend to change more unpredictably, partly because of the influence of external factors (size of TCP windows, application behavior, etc.). Also, the field values tend to change independently. Overall, this makes compression more challenging and makes it harder to select a set of encodings that can successfully trade off efficiency and robustness.

5.5. Short-Lived Flows

It is hard to see what can be done to improve performance for a single, unpredictable, short-lived connection. However, there are commonly cases where there will be multiple TCP connections between the same pair of hosts. As a particular example, consider web browsing (this is more the case with HTTP/1.0 [25] than with HTTP/1.1 [26]).

When a connection closes, either it is the last connection between that pair of hosts or it is likely that another connection will open within a relatively short space of time. In this case, the IP header part of the context (i.e., those fields characterised in Section 2.1) will probably be almost identical. Certain aspects of the TCP context may also be similar.

Support for context replication is discussed in more detail in Section 3. Overall, support for sub-context sharing or initializing one context from another offers useful optimizations for a sequence of short-lived connections.

Note that, although TCP is connection oriented, it is hard for a compressor to tell whether a TCP flow has finished. For example, even in the 'bi-directional' link case, seeing a FIN and the ACK of the FIN at the compressor/decompressor does not mean that the FIN cannot be retransmitted. Thus, it may be more useful to think about initializing a new context from an existing one, rather than re-using an existing one.

As mentioned previously in Section 4.1.3, the IP header can clearly be shared between any transport-layer flows between the same two end-points. There may be limited scope for initialisation of a new TCP header from an existing one. The port numbers are the most obvious starting point.

5.6. Master Sequence Number

As pointed out earlier, in Section 4.1.3, there is no obvious candidate for a 'master sequence number' in TCP. Moreover, it is noted that such a master sequence number is only required to allow a decompressor to acknowledge packets in bi-directional mode. It can also be seen that such a sequence number would not be required for every packet.

While the sequence number only needs to be 'sparse', it is clear that there is a requirement for an explicitly added sequence number. There are no obvious ways to guarantee the unique identity of a packet other than by adding such a sequence number (sequence and acknowledgement numbers can both remain the same, for example).

5.7. Size Constraint for TCP Options

As can be seen from the above analysis, most TCP options, such as MSS, WSopt, or SACK-Permitted, may appear only on a SYN segment. Every implementation should (and we expect that most will) ignore unknown options on SYN segments. TCP options will be sent on non-SYN segments only when an exchange of options on the SYN segments has

indicated that both sides understand the extension. Other TCP options, such as MD5 Digest or Timestamp, also tend to be sent when the connection is initiated (i.e., in the SYN packet).

The total header size is also an issue. The TCP header specifies where segment data starts with a 4-bit field that gives the total size of the header (including options) in 32-bit words. This means that the total size of the header plus option must be less than or equal to 60 bytes. This leaves 40 bytes for options.

6. Security Considerations

Since this document only describes TCP field behavior, it raises no direct security concerns.

This memo is intended to be used to aid the compression of TCP/IP headers. Where authentication mechanisms such as IPsec AH [24] are used, it is important that compression be transparent. Where encryption methods such as IPsec ESP [27] are used, the TCP fields may not be visible, preventing compression.

7. Acknowledgements

Many IP and TCP RFCs (hopefully all of which have been collated below), together with header compression schemes from RFC 1144 [22], RFC 3544 [36], and RFC 3095 [31], and of course the detailed analysis of RTP/UDP/IP in RFC 3095, have been sources of ideas and knowledge. Further background information can also be found in [28] and [29].

This document also benefited from discussion on the ROHC mailing list and in various corridors (virtual or otherwise) about many key issues; special thanks go to Qian Zhang, Carsten Bormann, and Gorry Fairhurst.

Qian Zhang and Hongbin Liao contributed the extensive analysis of shareable header fields.

Any remaining misrepresentation or misinterpretation of information is entirely the fault of the authors.

8. References

8.1. Normative References

[1] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[2] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[3] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.

[4] Jacobson, V. and R. Braden, "TCP extensions for long-delay paths", RFC 1072, October 1988.

[5] Zweig, J. and C. Partridge, "TCP alternate checksum options", RFC 1146, March 1990.

[6] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[7] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[8] Braden, B., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994.

[9] Connolly, T., Amer, P., and P. Conrad, "An Extension to TCP: Partial Order Service", RFC 1693, November 1994.

[10] Bellovin, S., "Defending Against Sequence Number Attacks", RFC 1948, May 1996.

[11] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[12] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[13] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998.

[14] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[15] Ramakrishnan, K. and S. Floyd, "A Proposal to add Explicit Congestion Notification (ECN) to IP", RFC 2481, January 1999.

[16] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[17] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.

[18] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[19] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

8.2. Informative References

[20] IANA, "IANA", IANA TCP options, February 1998, <http://www.iana.org/assignments/tcp-parameters>.

[21] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[22] Jacobson, V., "Compressing TCP/IP headers for low-speed serial links", RFC 1144, February 1990.

[23] Almquist, P., "Type of Service in the Internet Protocol Suite", RFC 1349, July 1992.

[24] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[25] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

[27] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[26] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[28] Degermark, M., Nordgren, B., and S. Pink, "IP Header Compression", RFC 2507, February 1999.

[29] Casner, S. and V. Jacobson, "Compressing IP/UDP/RTP Headers for Low-Speed Serial Links", RFC 2508, February 1999.

[30] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, March 2000.

[31] Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L-E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T., and H. Zheng, "RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed", RFC 3095, July 2001.

[32] Dawkins, S., Montenegro, G., Kojo, M., and V. Magret, "End-to-end Performance Implications of Slow Links", BCP 48, RFC 3150, July 2001.

[33] Balakrishnan, Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", RFC 3449, December 2002.

[34] Inamura, H., Montenegro, G., Ludwig, R., Gurtov, A., and F. Khafizov, "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks", RFC 3481, February 2003.

[35] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.

[36] Engan, M., Casner, S., Bormann, C., and T. Koren, "IP Header Compression over PPP", RFC 3544, July 2003.

[37] Karn, P., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, July 2004.

Authors' Addresses

Mark A. West
Siemens/Roke Manor Research
Roke Manor Research Ltd.
Romsey, Hants SO51 0ZN
UK

Phone: +44 (0)1794 833311
EMail: mark.a.west@roke.co.uk
URI: http://www.roke.co.uk

Stephen McCann
Siemens/Roke Manor Research
Roke Manor Research Ltd.
Romsey, Hants SO51 0ZN
UK

Phone: +44 (0)1794 833341
EMail: stephen.mccann@roke.co.uk
URI: http://www.roke.co.uk

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).