Network Working Group M. Degermark
Request for Comments: 2507 Lulea University of Technology/SICS
Category: Standards Track B. Nordgren
Lulea University of Technology/Telia Research AB
S. Pink
Lulea University of Technology/SICS
February 1999

IP Header Compression

Status of this Memo

Copyright Notice

Abstract

This document describes how to compress multiple IP headers and TCP and UDP headers per hop over point to point links. The methods can be applied to of IPv6 base and extension headers, IPv4 headers, TCP and UDP headers, and encapsulated IPv6 and IPv4 headers.

Headers of typical UDP or TCP packets can be compressed down to 4-7 octets including the 2 octet UDP or TCP checksum. This largely removes the negative impact of large IP headers and allows efficient use of bandwidth on low and medium speed links.

The compression algorithms are specifically designed to work well over links with nontrivial packet-loss rates. Several wireless and modem technologies result in such links.

TABLE OF CONTENTS

1. Introduction

There are several reasons to do header compression on low- or medium-speed links. Header compression can

* Improve interactive response time

  For very low-speed links, echoing of characters may take longer than 100-200 ms because of the time required to transmit large headers. 100-200 ms is the maximum time people can tolerate without feeling that the system is sluggish.

* Allow using small packets for bulk data with good line efficiency

  This is important when interactive (for example Telnet) and bulk traffic (for example FTP) is mixed because the bulk data should be carried in small packets to decrease the waiting time when a packet with interactive data is caught behind a bulk data packet.

  Using small packet sizes for the FTP traffic in this case is a global solution to a local problem. It will increase the load on the network as it has to deal with many small packets. A better solution might be to locally fragment the large packets over the slow link.

* Allow using small packets for delay sensitive low data-rate traffic

  For such applications, for example voice, the time to fill a packet with data is significant if packets are large. To get low end-to-end delay small packets are preferred. Without header compression, the smallest possible IPv6/UDP headers (48 octets) consume 19.2 kbit/s with a packet rate of 50 packets/s. 50 packets/s is equivalent to having 20 ms worth of voice samples in each packet. IPv4/UDP headers consumes 11.2 kbit/s at 50 packets/s. Tunneling or routing headers, for example to support mobility, will increase the bandwidth consumed by headers by 10-20 kbit/s. This should be compared with the bandwidth required for the actual sound samples, for example 13 kbit/s with GSM encoding. Header compression can reduce the bandwidth needed for headers significantly, in the example to about 1.7 kbit/s. This enables higher quality voice transmission over 14.4 and 28.8 kbit/s modems.

* Decrease header overhead.

  A common size of TCP segments for bulk transfers over medium-speed links is 512 octets today. When TCP segments are tunneled, for example because Mobile IP is used, the IPv6/IPv6/TCP header is 100

octets. Header compression will decrease the header overhead for IPv6/TCP from 19.5 per cent to less than 1 per cent, and for tunneled IPv4/TCP from 11.7 to less than 1 per cent. This is a significant gain for line-speeds as high as a few Mbit/s.

The IPv6 specification prescribes path MTU discovery, so with IPv6 bulk TCP transfers should use segments larger than 512 octets when possible. Still, with 1400 octet segments (RFC 894 Ethernet encapsulation allows 1500 octet payloads, of which 100 octets are used for IP headers), header compression reduces IPv6 header overhead from 7.1% to 0.4%.

* Reduce packet loss rate over lossy links.

  Because fewer bits are sent per packet, the packet loss rate will be lower for a given bit-error rate. This results in higher throughput for TCP as the sending window can open up more between losses, and in fewer lost packets for UDP.

The mechanisms described here are intended for a point-to-point link. However, care has been taken to allow extensions for multi-access links and multicast.

Headers that can be compressed include TCP, UDP, IPv4, and IPv6 base and extension headers. For TCP packets, the mechanisms of Van Jacobson [RFC-1144] are used to recover from loss. Two additional mechanisms that increase the efficiency of VJ header compression over lossy links are also described. For non-TCP packets, compression slow-start and periodic header refreshes allow minimal periods of packet discard after loss of a header that changes the context. There are hooks for adding header compression schemes on top of UDP, for example compression of RTP headers.

Header compression relies on many fields being constant or changing seldomly in consecutive packets belonging to the same packet stream. Fields that do not change between packets need not be transmitted at all. Fields that change often with small and/or predictable values, e.g., TCP sequence numbers, can be encoded incrementally so that the number of bits needed for these fields decrease significantly. Only fields that change often and randomly, e.g., checksums or authentication data, need to be transmitted in every header.

The general principle of header compression is to occasionally send a packet with a full header; subsequent compressed headers refer to the context established by the full header and may contain incremental changes to the context.

This header compression scheme does not require that all packets in the same stream passes over the compressed link. However, for TCP streams the difference between subsequent headers can become more irregular and the compression rate can decrease. Neither is it required that corresponding TCP data and acknowledgment packets traverse the link in opposite directions.

This header compression scheme is useful on first-hop or last-hop links as well as links in the middle of the network. When many packet streams (several hundred) traverse the link, a phenomenon that could be called CID thrashing could occur, where headers seldom can be matched with an existing context and have to be sent uncompressed or as full headers. It is up to an implementation to use techniques such as hysteresis to ensure that the packet streams that give the highest compression rates keep their context. Such techniques are more likely to be needed in the middle of the network.

2. Terminology

This section explains some terms used in this document.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Subheader

An IPv6 base header, an IPv6 extension header, an IPv4 header, a UDP header, or a TCP header.

Header

A chain of subheaders.

Compress

The act of reducing the size of a header by removing header fields or reducing the size of header fields. This is done in a way such that a decompressor can reconstruct the header if its context state is identical to the context state used when compressing the header.

Decompress

The act of reconstructing a compressed header.

Context identifier (CID)

A small unique number identifying the context that should be used to decompress a compressed header. Carried in full headers and compressed headers.

Context

The state which the compressor uses to compress a header and the decompressor uses to decompress a header. The context is the uncompressed version of the last header sent (compressor) or received (decompressor) over the link, except for fields in the header that are included "as-is" in compressed headers or can be inferred from, e.g., the size of the link-level frame.

The context for a packet stream is associated with a context identifier. The context for non-TCP packet streams is also associated with a generation.

Generation

For non-TCP packet streams, each new version of the context for a given CID is associated with a generation: a small number that is incremented whenever the context associated with that CID changes. Carried by full and compressed non-TCP headers.

Packet stream

A sequence of packets whose headers are similar and share context. For example, headers in a TCP packet stream have the same source and final destination address, and the same port numbers in the TCP header. Similarly, headers in a UDP packet stream have the same source and destination address, and the same port numbers in the UDP header.

Full header (header refresh)

An uncompressed header that updates or refreshes the context for a packet stream. It carries a CID that will be used to identify the context.

Full headers for non-TCP packet streams also carry the generation of the context they update or refresh.

Regular header

A normal, uncompressed, header. Does not carry CID or generation association.

Incorrect decompression

When a compressed and then decompressed header is different from the uncompressed header. Usually due to mismatching context between the compressor and decompressor or bit errors during transmission of the compressed header.

Differential coding

A compression technique where the compressed value of a header field is the difference between the current value of the field and the value of the same field in the previous header belonging to the same packet stream. A decompressor can thus obtain the value of the field by adding the value in the compressed header to its context. This technique is used for TCP streams but not for non-TCP streams.

3. Compression method

Much of the header information stays the same over the life-time of a packet stream. For non-TCP packet streams almost all fields of the headers are constant. For TCP many fields are constant and others change with small and predictable values.

To initiate compression of the headers of a packet stream, a full header carrying a context identifier, CID, is transmitted over the link. The compressor and decompressor store most fields of this full header as context. The context consists of the fields of the header whose values are constant and thus need not be sent over the link at all, or change little between consecutive headers so that it uses fewer bits to send the difference from the previous value compared to sending the absolute value.

Any change in fields that are expected to be constant in a packet stream will cause the compressor to send a full header again to update the context at the decompressor. As long as the context is the same at compressor and decompressor, headers can be decompressed to be exactly as they were before compression. However, if a full header or compressed header is lost during transmission, the context of the decompressor may become obsolete as it is not updated properly. Compressed headers will then be decompressed incorrectly.

IPv6 is not meant to be used over links that can deliver a significant fraction of damaged packets to the IPv6 module. This means that links must have a very low bit-error rate or that link-level frames must be protected by strong checksums, forward error correction or something of that nature. Header compression SHOULD not be used for IPv4 without strong link-level checksums. Damaged

frames will thus be discarded by the link layer. The link layer implementation might indicate to the header compression module that a frame was damaged, but it cannot say what packet stream it belonged to as it might be the CID that is damaged. Moreover, frames may disappear without the link layer implementation's knowledge, for example if the link is a multi-hop link where frames can be dropped due to congestion at each hop. The kind of link errors that a header compression module should deal with and protect against will thus be packet loss.

So a header compression scheme needs mechanisms to update the context at the decompressor and to detect or avoid incorrect decompression. These mechanisms are very different for TCP and non-TCP streams, and are described in sections 3.2 and 3.3.

The compression mechanisms in this document assume that packets are not reordered between the compressor and decompressor. If the link

does reorder, section 11 describes mechanisms for ordering the packets before decompression. It is also assumed that the link-layer implementation can provide the length of packets, and that there is no padding in UDP packets or tunneled packets.

3.1. Packet types

This compression method uses four packet types in addition to the IPv4 and IPv6 packet types. The combination of link-level packet type and the value of the first four bits of the packet uniquely determines the packet type. Details on how these packet types are represented are in section 13.

FULL_HEADER - indicates a packet with an uncompressed header, including a CID and, if not a TCP packet, a generation. It establishes or refreshes the context for the packet stream identified by the CID.

COMPRESSED_NON_TCP - indicates a non-TCP packet with a compressed header. The compressed header consists of a CID identifying what context to use for decompression, a generation to detect an inconsistent context and the randomly changing fields of the header.

COMPRESSED_TCP - indicates a packet with a compressed TCP header, containing a CID, a flag octet indentifying what fields have changed, and the changed fields encoded as the difference from the previous value.

COMPRESSED_TCP_NODELTA - indicates a packet with a compressed TCP header where all fields that are normally sent as the difference to the previous value are instead sent as-is. This packet type is only sent as the response to a header request from the decompressor. It must not be sent as the result of a retransmission.

In addition to the packet types used for compression, regular IPv4 and IPv6 packets are used whenever a compressor decides to not compress a packet. An additional packet type may be used to speed up repair of TCP streams over links where the decompressor can send packets to the compressor.

CONTEXT_STATE - indicates a special packet sent from the decompressor to the compressor to communicate a list of (TCP) CIDs for which synchronization has been lost. This packet is only sent over a single link so it requires no IP header. The format is shown in section 10.2.

3.2. Lost packets in TCP packet streams

Since TCP headers are compressed using the difference from the previous TCP header, loss of a packet with a compressed or full header will cause subsequent compressed headers to be decompressed incorrectly because the context used for decompression was not incremented properly.

Loss of a compressed TCP header will cause the TCP sequence numbers of subsequently decompressed TCP headers to be off by k, where k is the size of the lost segment. Such incorrectly decompressed TCP headers will be discarded by the TCP receiver as the TCP checksum reliably catches "off-by-k" errors in the sequence numbers for plausible k.

TCP’s repair mechanisms will eventually retransmit the discarded segment and the compressor peeks into the TCP headers to detect when TCP retransmits. When this happens, the compressor sends a full header on the assumption that the retransmission was due to mismatching compression state at the decompressor. [RFC-1144] has a good explanation of this mechanism.

The mechanisms of section 10 should be used to speed up the repair of the context. This is important over medium speed links with high packet loss rates, for example wireless. Losing a timeout’s worth of packets due to inconsistent context after each packet lost over the link is not acceptable, especially when the TCP connection is over the wide area.

3.3. Lost packets in UDP and other non-TCP packet streams

Incorrectly decompressed headers of UDP packets and other non-TCP packets are not so well-protected by checksums as TCP packets. There are no sequence numbers that become "off-by-k" and virtually guarantees a failed checksum as there are for TCP. The UDP checksum only covers payload, UDP header, and pseudo header. The pseudo header includes the source and destination addresses, the transport protocol type and the length of the transport packet. Except for those fields, large parts of the IPv6 header are not covered by the UDP checksum. Moreover, other non-TCP headers lack checksums altogether, for example fragments.

In order to safely avoid incorrect decompression of non-TCP headers, each version of the context for non-TCP packet streams is identified by a generation, a small number that is carried by the full headers that establish and refresh the context. Compressed headers carry the generation value of the context that were used to compress them. When a decompressor sees that a compressed header carries a generation value other than the generation of its context for that packet stream, the context is not up to date and the packet must be discarded or stored until a full header establishes correct context.

Differential coding is not used for non-TCP streams, so compressed non-TCP headers do not change the context. Thus, loss of a compressed header does not invalidate subsequent packets with compressed headers. Moreover, the generation changes only when the context of a full header is different from the context of the previous full header. This means that losing a full header will make the context of the decompressor obsolete only when the full header would actually have changed the context.

The generation field is 6 bits long so the generation value repeats itself after 64 changes to the context. To avoid incorrect decompression after error bursts or other temporary disruptions, the compressor must not reuse the same generation value after a shorter time than MIN_WRAP seconds. A decompressor which has been disconnected MIN_WRAP seconds or more must wait for the next full header before decompressing. A compressor must wait at least MIN_WRAP seconds after booting before compressing non-TCP headers. Instead of reusing a generation value too soon, a compressor may switch to another CID or send regular headers until MIN_WRAP seconds have passed. The value of MIN_WRAP is found in section 14.

3.3.1. Compression Slow-Start

To allow the decompressor to recover quickly from loss of a full header that would have changed the context, full headers are sent periodically with an exponentially increasing period after a change in the context. This technique avoids an exchange of messages between compressor and decompressor used by other compression schemes, such as in [RFC-1553]. Such exchanges can be costly for wireless mobiles as more power is consumed by the transmitter and delay can be introduced by switching between sending and receiving. Moreover, techniques that require an exchange of messages cannot be used over simplex links, such as direct-broadcast satellite channels or cable TV systems, and are hard to adapt to multicast over multi-access links.

```
 |.|..|....|........|................|.............................
 ^
 Change   Sent packets: | with full header, . with compressed header
```

The picture shows how packets are sent after change. The compressor keeps a variable for each non-TCP packet stream, F_PERIOD, that keeps track of how many compressed headers may be sent between full headers. When the headers of a non-TCP packet stream change so that its context changes, a full header is sent and F_PERIOD is set to one. After sending F_PERIOD compressed headers, a full header is sent. F_PERIOD is doubled each time a full header is sent during compression slow-start.

3.3.2. Periodic Header Refreshes

To avoid losing too many packets if a receiver has lost its context, there is an upper limit, F_MAX_PERIOD, on the number of non-TCP packets with compressed headers that may be sent between header refreshes. If a packet is to be sent and F_MAX_PERIOD compressed headers have been sent since the last full header for this packet stream was sent, a full header must be sent.

To avoid long periods of disconnection for low data rate packet streams, there is also an upper bound, F_MAX_TIME, on the time between full headers in a non-TCP packet stream. If a packet is to be sent and more than F_MAX_TIME seconds have passed since the last full header was sent for this packet stream, a full header must be sent. The values of F_MAX_PERIOD and F_MAX_TIME are found in section 14.

3.3.3. Rules for sending Full Headers

The following pseudo code can be used by the compressor to determine when to send a full header for a non-TCP packet stream. The code maintains two variables:

```
C_NUM       -- a count of the number of compressed headers sent
               since the last full header was sent.
F_LAST      -- the time of sending the last full header.
```

and uses the functions

```
current_time()       return the current time
min(a,b)             return the smallest of a and b
```

the procedures send_full_header(), increment_generation_value(), and send_compressed_header() do the obvious thing.

```
if ( <this header changes the context> )

    C_NUM := 0;
    F_LAST := current_time();
    F_PERIOD := 1;
    increment_generation_value();
    send_full_header();

elseif ( C_NUM >= F_PERIOD )

    C_NUM := 0;
    F_LAST := current_time();
    F_PERIOD := min(2 * F_PERIOD, F_MAX_PERIOD);
    send_full_header();

elseif ( current_time() > F_LAST + F_MAX_TIME )

    C_NUM := 0;
    F_LAST := current_time();
    send_full_header();

else

    C_NUM := C_NUM + 1
    send_compressed_header();

endif
```

3.3.4. Cost of sending Header Refreshes

If every f'th packet carries a full header, H is the size of a full header, and C is the size of a compressed header, the average header size is

$$(H-C)/f + C$$

For f > 1, the average header size is (H-C)/f larger than a compressed header.

In a diagram where the average header size is plotted for various f values, there is a distinct knee in the curve, i.e., there is a limit beyond which further increasing f gives diminishing returns. F_MAX_PERIOD should be chosen to be a frequency well to the right of the knee of the curve. For typical sizes of H and C, say 48 octets for the full header (IPv6/UDP) and 4 octets for the compressed header, setting F_MAX_PERIOD > 44 means that full headers will contribute less than an octet to the average header size. With a four-address routing header, F_MAX_PERIOD > 115 will have the same effect.

The default F_MAX_PERIOD value of 256 (section 14) puts the full header frequency well to the right of the knee and means that full headers will typically contribute considerably less than an octet to the average header size. For H = 48 and C = 4, full headers contribute about 1.4 bits to the average header size after reaching the steady-state header refresh frequency determined by the default

F_MAX_PERIOD. 1.4 bits is a very small overhead.

After a change in the context, the exponential backoff scheme will initially send full headers frequently. The default F_MAX_PERIOD will be reached after nine full headers and 255 compressed headers have been sent. This is equivalent to a little over 5 seconds for a typical voice stream with 20 ms worth of voice samples per packet.

During the whole backoff period, full headers contribute 1.5 octets to the average header size when H = 48 and C = 4. For 20 ms voice samples, it takes less than 1.3 seconds until full headers contribute less than one octet to the average header size, and during these initial 1.3 seconds full headers add less than 4 octets to the average header size. The cost of the exponential backoff is not great and as the headers of non-TCP packet streams are expected to change seldomly, it will be amortized over a long time.

The cost of header refreshes in terms of bandwidth are higher than similar costs for hard state schemes like [RFC-1553] where full headers must be acknowledged by the decompressor before compressed headers may be sent. Such schemes typically send one full header plus a few control messages when the context changes. Hard state schemes require more types of protocol messages and an exchange of messages is necessary. Hard state schemes also need to deal explicitly with various error conditions that soft state handles automatically, for instance the case of one party disappearing unexpectedly, a common situation on wireless links where mobiles may go out of range of the base station.

The major advantage of the soft state scheme is that no handshakes are needed between compressor and decompressor, so the scheme can be used over simplex links. The costs in terms of bandwidth are higher than for hard state schemes, but the simplicity of the decompressor, the simplicity of the protocol, and the lack of handshakes between compressor and decompressor justifies this small cost. Moreover, soft state schemes are more easily extended to multicast over multi-access links, for example radio links.

4. Grouping packets into packet streams

This section explains how packets MAY be grouped together into packet streams for compression. To achieve the best compression rates, packets SHOULD be grouped together such that packets in the same packet stream have similar headers. If this grouping fails, header compression performance will be bad, since the compression algorithm can rarely utilize the existing context for the packet stream and full headers must be sent frequently.

Grouping is done by the compressor. A compressor may use whatever criterion it finds appropriate to group packets into packet streams. To determine what packet stream a packet belongs to, a compressor MAY

a) examine the compressible chain of subheaders (see section 7),

b) examine the contents of an upper layer protocol header that follows the compressible chain of subheaders, for example ICMP headers, DVMRP headers, or tunneled IPX headers,

c) use information obtained from a resource manager, for example if a resource manager requests compression for a particular packet stream and provides a way to identify packets belonging to that packet stream,

d) use any other relevant information, for example if routes flap and the hop limit (TTL) field in a packet stream changes frequently between n and n+k, a compressor may choose to group the packets into two different packet streams.

A compressor is also free not to group packets into packet streams for compression, letting some packets keep their regular headers and passing them through unmodified.

As long as the rules for when to send full headers for a non-TCP packet stream are followed and subheaders are compressed as specified in this document, the decompressor is able to reconstruct a compressed header correctly regardless of how packets are grouped into packet streams.

4.1 Guidelines for grouping packets

In this section we give OPTIONAL guidelines for how a compressor may group packets into packet streams for compression.

Defining fields

The defining fields of a header should be present and identical in all packets belonging to the same packet stream. These fields are marked DEF in section 7. The defining fields include the flow label, source and destination addresses of IP headers, final destination address in routing headers, the next header fields (for IPv6), the protocol field (IPv4), port numbers (UDP and TCP), and the SPI in authentication and encryption headers.

Fragmented packets

Fragmented and unfragmented packets should never be grouped together in the same packet stream. The Identification field of the Fragment header or IPv4 header should not be used to identify the packet stream. If it was, the first fragment of a new packet would cause a compression slow-start.

No field after a Fragment Header, or an IPv4 header for a fragment, should be used for grouping purposes.

Upper protocol identification

The first next header field identifying a header not described in section 7 should be used for identifying packet streams, i.e., all packets with the same DEF fields and the same upper protocol should be grouped together.

TTL field (Hop Limit field)

A sophisticated implementation might monitor the TTL (Hop Limit) field and if it changes frequently use it as a DEF field. This can occur when there are frequent route flaps so that packets traverse different paths through the internet.

Traffic Class field (IPv6), Type of Service field (IPv4)

It is possible that the Traffic Class field of the IPv6 header and the Type of Service of the IPv4 header will change frequently between packets with otherwise identical DEF fields. A sophisticated implementation should watch out for this and be prepared to use these fields as defining fields.

When IP packets are tunneled they are encapsulated with an additional IP header at the tunnel entry point and then sent to the tunnel endpoint. To group such packets into packet streams, the inner headers should also be examined to determine the packet stream. If this is not done, full headers will be sent each time the headers of the inner IP packet changes. So when a packet is tunneled, the identifying fields of the inner subheaders should be considered in addition to the identifying fields of the initial IP header.

An implementation can use other fields for identification than the ones described here. If too many fields are used for identification, performance might suffer because more CIDs will be used and the wrong CIDs might be reused when new flows need CIDs. If too few fields are used for identification, performance might suffer because there are too frequent changes to the context.

We stress that these guidelines are educated guesses. When IPv6 is widely deployed and IPv6 traffic can be analyzed, we might find that other grouping algorithms perform better. We also stress that if the grouping fails, the result will be bad performance but not incorrect decompression. The decompressor can do its task regardless of how the grouping algorithm works.

5. Size Issues

5.1. Context Identifiers

Context identifiers can be 8 or 16 bits long. Their size is not relevant for finding the context. An 8-bit CID with value two and a 16-bit CID with value two are equivalent.

The CID spaces for TCP and non-TCP are separate, so a TCP CID and a non-TCP CID never identify the same context. Even if they have the same value. This doubles the available CID space while using the same number of bits for CIDs. It is always possible to tell whether a full or compressed header is for a TCP or non-TCP packet, so no mixups can occur.

Non-TCP compressed headers encode the size of the CID using one bit in the second octet of the compressed header. The 8-bit CID allows a minimum compressed header size of 2 octets for non-TCP packets, the CID uses the first octet and the size bit and the 6-bit Generation value fit in the second octet.

For TCP the only available CID size is 8 bits as in [RFC-1144]. 8 bits is probably sufficient as TCP connections are always point-to-point.

The 16 bit CID size may not be needed for point-to-point links; it is intended for use on multi-access links where a larger CID space may be needed for efficient selection of CIDs.

The major difficulty with multi-access links is that several compressors share the CID space of a decompressor. CIDs can no longer be selected independently by the compressors as collisions may occur. This problem may be resolved by letting the decompressors have a separate CID space for each compressor. Having separate CID spaces requires that decompressors can identify which compressor sent the compressed packet, perhaps by utilizing link-layer information as to who sent the link-layer frame. If such information is not available, all compressors on the multi-access link may be enumerated, automatically or otherwise, and supply their number as part of the CID. This latter method requires a large CID space.

5.2. Size of the context

The size of the context SHOULD be limited to simplify implementation of compressor and decompressor, and put a limit on their memory requirements. However, there is no upper limit on the size of an IPv6 header as the chain of extension headers can be arbitrarily long. This is a problem as the context is essentially a stored header.

The configurable parameter MAX_HEADER (see section 14) represents the maximum size of the context, expressed as the maximum sized header that can be stored as context. When a header is larger than MAX_HEADER, only part of it is stored as context. An implementation MUST NOT compress more than the initial MAX_HEADER octets of a header. An implementation MUST NOT partially compress a subheader.

Thus, the part of the header that is stored as context and is compressed is the longest initial sequence of entire subheaders that is not larger than MAX_HEADER octets.

5.3. Size of full headers

It is desirable to avoid increasing the size of packets with full headers beyond their original size, as their size may be optimized for the MTU of the link. Since we assume that the link layer implementation provides the length of packets, we can use the length fields in full headers to pass the values of the CID and the generation to the decompressor.

This requires that the link-layer must not add padding to the payload, at least not padding that can be delivered to the destination link user. It is also required that no extra padding is added after UDP data or in tunneled packets. This allows values of length fields to be calculated from the length of headers and the length of the link-layer frame.

The generation requires one octet and the CID may require up to 2 octets. There are length fields of 2 octets in the IPv6 Base Header, the IPv4 header, and the UDP header.

A full TCP header will thus have at least 2 octets available in the IP header to pass the 8 bit CID, which is sufficient. There will be more than two octets available if there is more than one IP header.

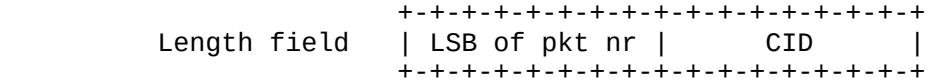
[RFC-1144] uses the 8 bit Protocol field of the IPv4 header to pass the CID. We cannot use the corresponding method as the sequence of IPv6 extension headers is not fixed and CID values are not disjoint from the legal values of Next Header fields.

An IPv6/UDP or IPv4/UDP packet will have 4 octets available to pass the generation and the CID, so all CID sizes may be used. Fragmented or encrypted packet streams may have only 2 octets available to pass the generation and CID. Thus, 8-bit CIDs may be the only CID sizes that can be used for such packet streams. When IPv6/IPv4 or IPv4/IPv6 tunneling is used, there will be at least 4 octets available, and both CID sizes may be used.

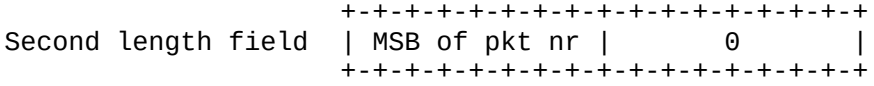
The generation value is passed in the higher order octet of the first length field in the full header. When only one length field is available, the 8-bit CID is passed in the low order octet. When two length fields are available, the lowest two octets of the CID are passed in the second length field and the low order octet of the first length field carries the highest octet of the CID.

5.3.1. Use of length fields in full TCP headers

Use of first length field:



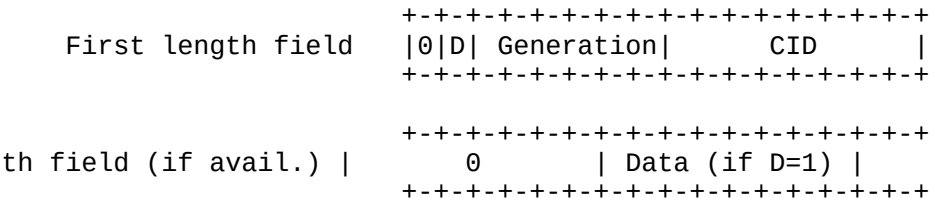
Use of second length field if available:



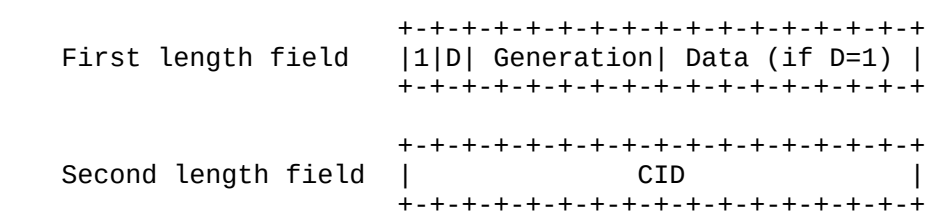
Pkt nr is short for packet sequence number, described in section 11.2.

5.3.2. Use of length fields in full non-TCP headers

Full non-TCP headers with 8-bit CID:



Full non-TCP headers with 16-bit CID:

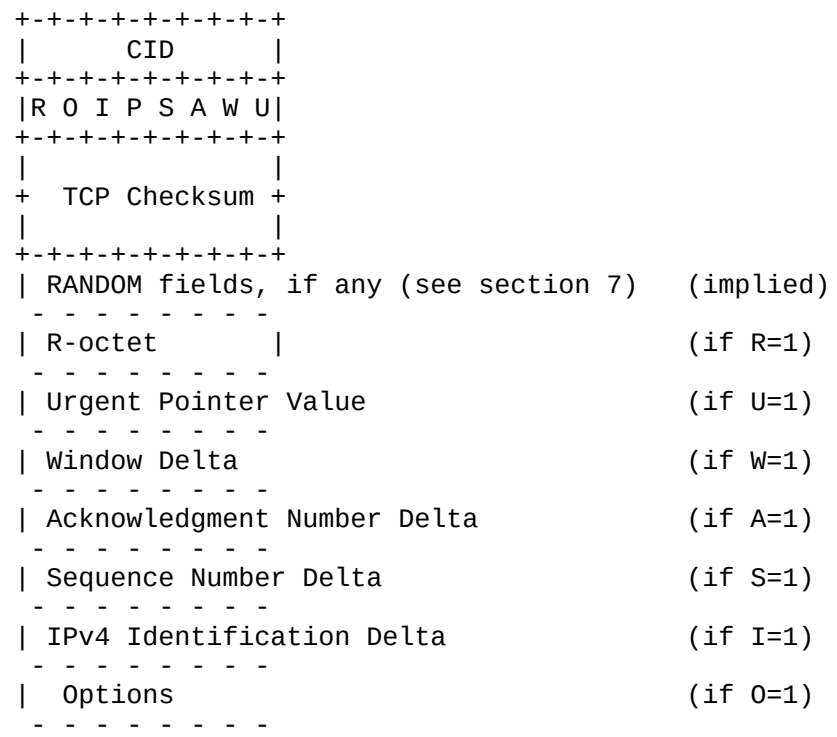


The first bit in the first length field indicates the length of the CID. The Data field is zero if D is zero. The use of the D bit and Data field is explained in section 12.

6. Compressed Header Formats

This section uses some terminology (DELTA, RANDOM) defined in section 7.

a) COMPRESSED_TCP format (similar to [RFC 1144]):



The latter flags in the second octet (IPSAWU) have the same meaning as in [RFC-1144], regardless of whether the TCP segments are carried by IPv6 or IPv4. The C bit has been eliminated because the CID is always present. The context associated with the CID keeps track of the IP version and what RANDOM fields are present. The order between delta fields specified here is exactly as in [RFC-1144]. An implementation will typically scan the context from the beginning and insert the RANDOM fields in order. The RANDOM fields are thus placed before the DELTA fields of the TCP header in the same order as they occur in the original uncompressed header.

The I flag is zero unless an IPv4 header immediately precedes the TCP header. The combined IPv4/TCP header is then compressed as a unit as described in [RFC-1144]. Identification fields in IPv4 headers that are not immediately followed by a TCP header are RANDOM.

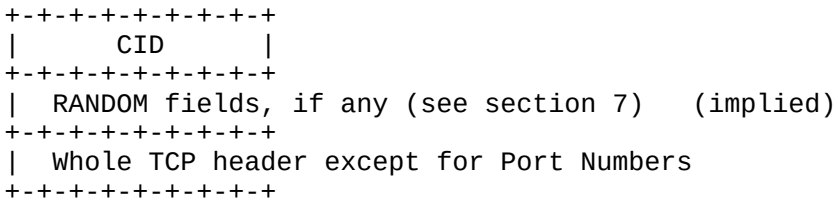
If the O flag is set, the Options of the TCP header were not the same as in the previous header. The entire Option field are placed last in the compressed TCP header.

If the R flag is set, there were differences between the context and the Reserved field (6 bits) in the TCP header or bit 6 or 7 of the TOS octet (Traffic Class octet) in a IPv4 header (IPv6 header) that immediately precedes the TCP header. An octet with the actual values of the Reserved field and bit 6 and 7 of the TOS or Traffic Class field is then placed immediately after the RANDOM fields. Bits 0-5 of the passed octet is the actual value of the Reserved field, and bits 6 and 7 are the actual values of bits 6 and 7 in the TOS or Traffic Class field. If there is no preceding IP header, bits 6 and 7 are 0. The octet passed with the R flag MUST NOT update the context.

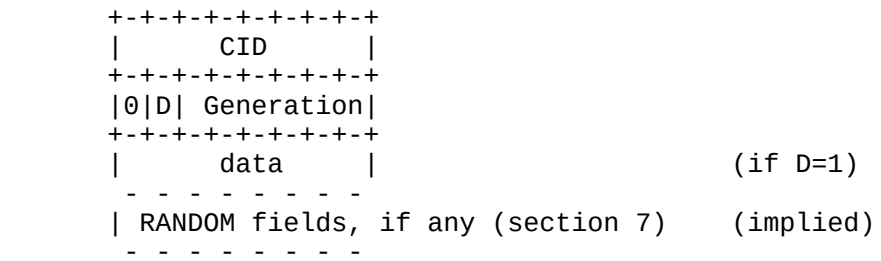
NOTE: The R-octet does not update the context because if it did, the nTCP checksum would not guard the receiving TCP from erroneously decompressed headers. Bits 6 and 7 of the TOS octet or Traffic Class octet is expected to change frequently due to Explicit Congestion Notification.

See section 7.12 and [RFC-1144] for further information on how to compress TCP headers.

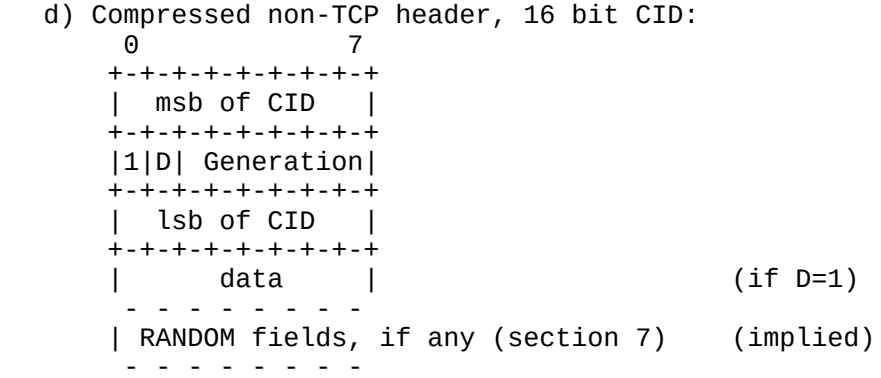
b) COMPRESSED_TCP_NODELTA header format



c) Compressed non-TCP header, 8 bit CID:



d) Compressed non-TCP header, 16 bit CID:



The generation, CID and optional one octet data are followed by relevant RANDOM fields (see section 7) as implied by the compression state, placed in the same order as they occur in the original uncompressed header, followed by the payload.

7. Compression of subheaders

This section gives rules for how the compressible chain of subheaders is compressed. These rules MUST be followed. Subheaders that may be compressed include IPv6 base and extension headers, TCP headers, UDP headers, and IPv4 headers. The compressible chain of subheaders extends from the beginning of the header

a) up to but not including the first header that is not an IPv4 header, an IPv6 base or extension header, a TCP header, or a UDP header, or

b) up to and including the first TCP header, UDP header, Fragment Header, Encapsulating Security Payload Header, or IPv4 header for a fragment,

whichever gives the shorter chain. For example, rules a) and b) both fit a chain of subheaders that contain a Fragment Header and ends at a tunneled IPX packet. Since rule b) gives a shorter chain, the compressible chain of subheaders stops at the Fragment Header.

The following subsections are a systematic classification of how all fields in subheaders are expected to change.

NOCHANGE The field is not expected to change. Any change means that a full header MUST be sent to update the context.

DELTA The field may change often but usually the difference from the field in the previous header is small, so that it is cheaper to send the change from the previous value rather than the current value. This type of compression is only used for TCP packet streams.

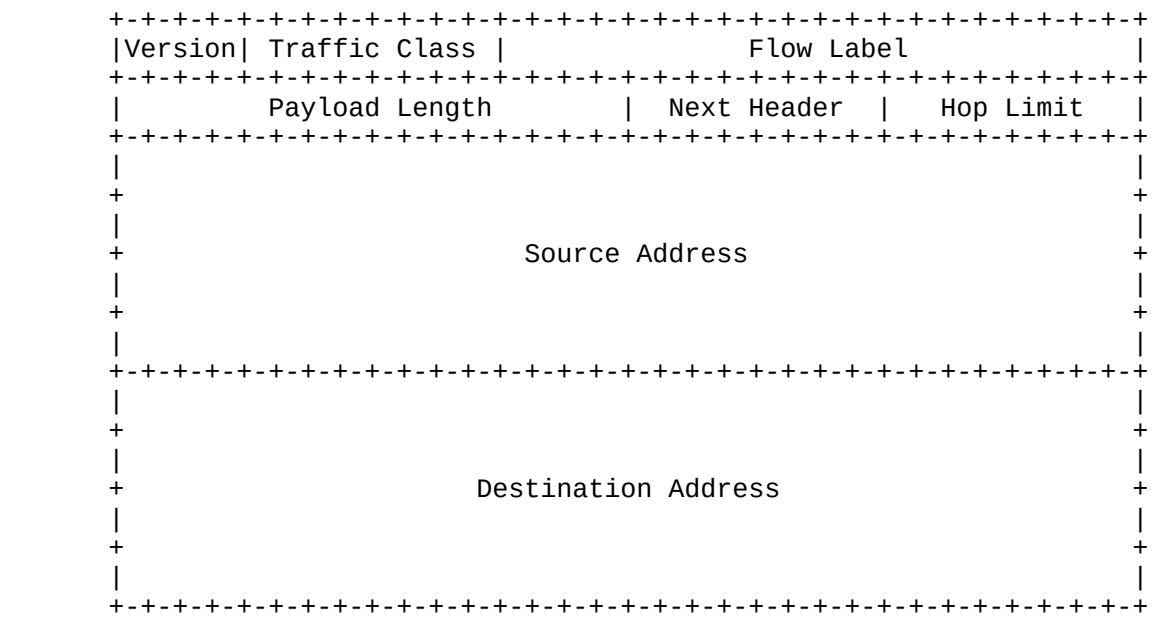
RANDOM The field must be included "as-is" in compressed headers, usually because it changes unpredictably.

INFERRED The field contains a value that can be inferred from other values, for example the size of the frame carrying the packet, and thus must not be included in the compressed header.

The classification implies how a compressed header is constructed. No field that is NOCHANGE or INFERRED is present in a compressed header. A compressor obtains the values of NOCHANGE fields from the context identified by the compression identifier, and obtains the values of INFERRED fields from the link-layer implementation, e.g., from the size of the link-layer frame, or from other fields, e.g., by recalculating the IPv4 header checksum. DELTA fields are encoded as the difference to the value in the previous packet in the same packet stream. The decompressor must update the context by adding the value in the compressed header to the value in its context. The result is the proper value of the field. RANDOM fields must be sent "as-is" in the compressed header. RANDOM fields must occur in the same order in the compressed header as they occur in the full header.

Fields that may optionally be used to identify what packet stream a packet belongs to according to section 4.1 are marked with the word DEF. To a compressor using the optional guidelines from section 4.1, any difference in corresponding DEF fields between two packets implies that they belong to different packet streams. Moreover, if a DEF field is present in one packet but not in another, the packets belong to different packet streams.

7.1. IPv6 Header [IPv6, section 3]



Version	NOCHANGE (DEF)
Traffic Class	NOCHANGE (might be DEF, see sect 4.1) (see also sect 6 a)
Flow Label	NOCHANGE (DEF)
Payload Length	INFERRED
Next Header	NOCHANGE
Hop Limit	NOCHANGE (might be DEF, see sect 4.1)
Source Address	NOCHANGE (DEF)
Destination Address	NOCHANGE (DEF)

The Payload Length field of encapsulated headers must correspond to the length value of the encapsulating header. If not, the header chain MUST NOT be compressed.

NOTE: If this the IP header closest to a TCP header, bit 7 of the Traffic Class field can be passed using the R-flag of the compressed TCP header. See section 6 a).

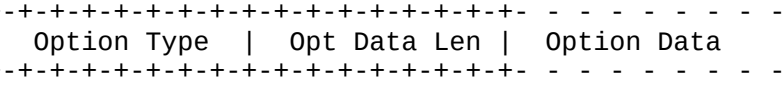
This classification implies that the entire IPv6 base header will be compressed away.

7.2. IPv6 Extension Headers [IPv6, section 4]

What extension headers are present and the relative order of them is not expected to change in a packet stream. Whenever there is a change, a full packet header must be sent. All Next Header fields in IPv6 base header and IPv6 extension headers are NOCHANGE.

7.3. Options [IPv6, section 4.2]

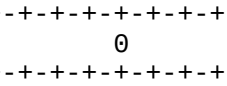
The contents of Hop-by-hop Options and Destination Options extension headers are encoded with TLV "options" (see [IPv6]):



Option Type and Opt Data Len fields are assumed to be fixed for a given packet stream, so they are classified as NOCHANGE. The Option data is RANDOM unless specified otherwise below.

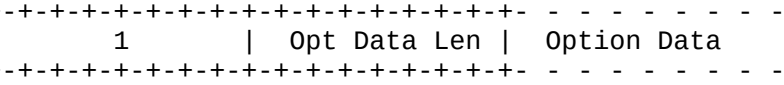
Padding

Pad1 option



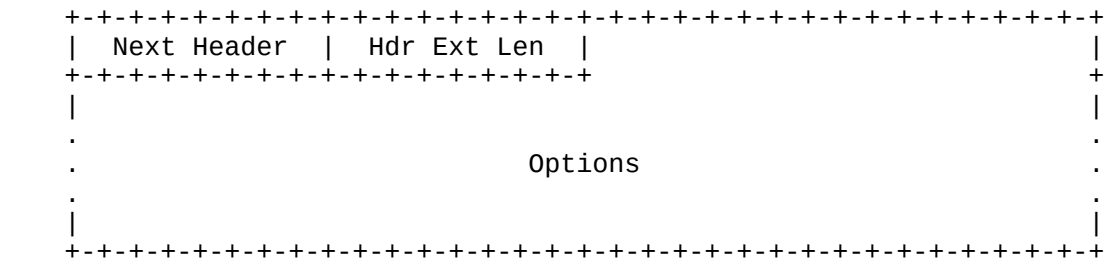
Entire option is NOCHANGE.

PadN option



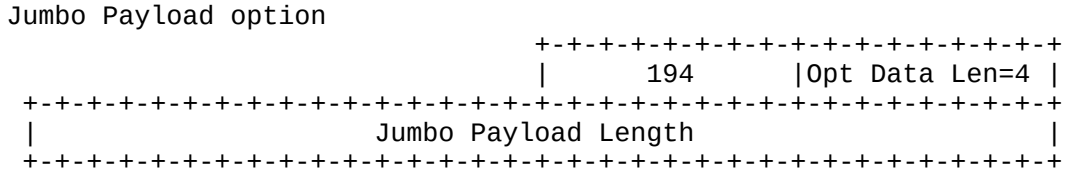
All fields are NOCHANGE.

7.4. Hop-by-Hop Options Header [IPv6, section 4.3]



Next Header	NOCHANGE
Hdr Ext Len	NOCHANGE
Options	TLV coded values and padding. Classified according to 7.3 above, unless being a Jumbo Payload option (see below).

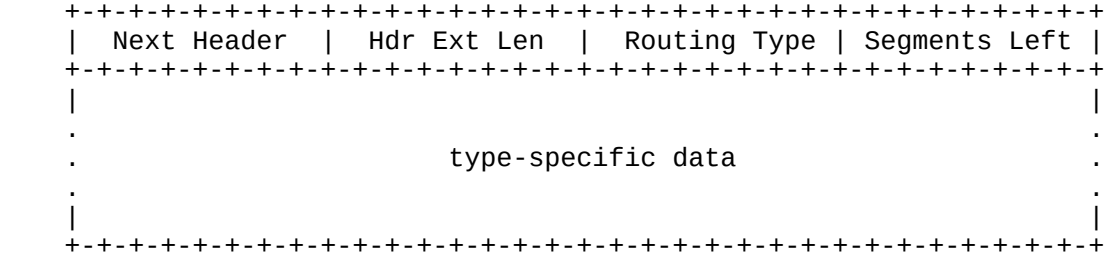
Jumbo Payload option



First two fields are NOCHANGE and Jumbo Payload Length INFERRED. (frame length must be supplied by link layer implementation).

> NOTE: It is silly to compress the headers of a packet carrying a Jumbo Payload Option since the relative header overhead is negligible. Moreover, it is usually a bad idea to send such large packets over low- and medium-speed links.

7.5. Routing Header [IPv6, section 4.4]



All fields of the Routing Header are NOCHANGE.

If the Routing Type is not recognized, it is impossible to determine the final Destination Address unless the Segments Left field has the value zero, in which case the Destination Address is the final Destination Address in the basic IPv6 header.

In the Type 0 Routing Header, the last address is DEF if (Segments Left > 0).

Routing Headers are compressed away completely. This is a big win as the maximum size of the Routing Header is 392 octets. Moreover, Type 0 Routing Headers with one address, size 24 octets, are used by Mobile IP.

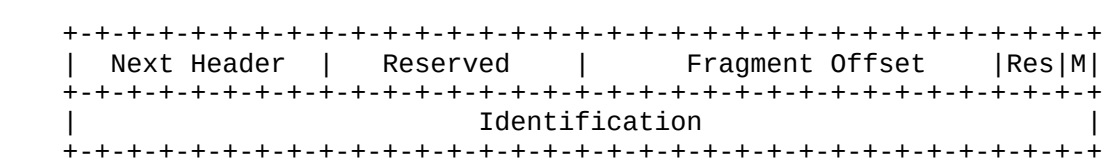
7.6. Fragment Header [IPv6, section 4.5]

The first fragment of a packet has Fragment Offset = 0 and the chain of subheaders extends beyond its Fragment Header. If a fragment is not the first (Fragment Offset not 0), there are no subsequent subheaders (unless the chain of subheaders in the first fragment didn't fit entirely in the first fragment).

Since packets may be reordered before reaching the compression point, and some fragments may follow other routes through the network, a compressor cannot rely on seeing the first fragment before other fragments. This implies that information in subheaders following the Fragment Header of the first fragment cannot be examined to determine the proper packet stream for other fragments.

It is possible to design compression schemes that can compress subheaders after the Fragment Header, at least in the first fragment, but to avoid complicating the rules for sending full headers and the rules for compression and decompression, the chain of subheaders that follow a Fragment Header MUST NOT be compressed.

The fields of the Fragment Header are classified as follows.



Next Header	NOCHANGE
Reserved	NOCHANGE
Res	RANDOM
M flag	RANDOM
Fragment Offset	RANDOM
Identification	RANDOM

This classification implies that a Fragment Header is compressed down to 6 octets. The minimum IPv6 MTU is 1280 octets so most fragments will be at least 1280 octets. Since the 6 octet overhead of the compressed fragment header is amortized over a fairly large packet, the additional complexity of more sophisticated compression schemes is not justifiable.

NOTE: The Identification field is RANDOM instead of NOCHANGE to avoid one compression slow-start per original packet.

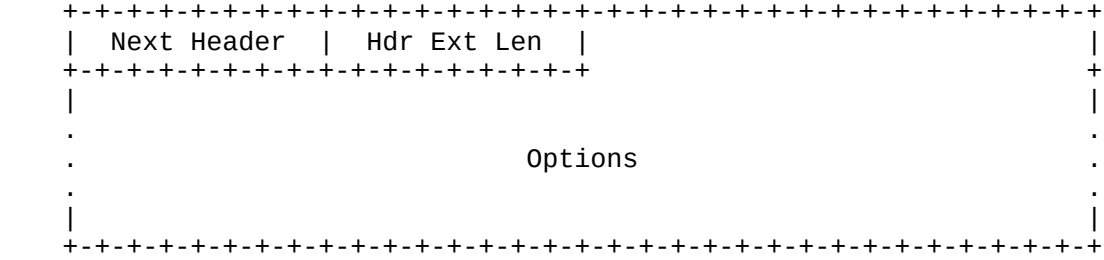
Grouping of fragments according to the optional guidelines in section4.1:

Fragments and unfragmented packets should not be grouped together.

Port numbers cannot be used to identify the packet stream because port numbers are not present in every fragment. To adhere to the uniqueness rules for the Identification value, a fragmented packet stream is identified by the combination of Source Address and (final) Destination Address.

NOTE: The Identification value is NOT used to identify the packet stream. This avoids using a new CID for each packet and saves the cost of the associated compression slow-start. We expect that the unfragmentable part of the headers will not change too frequently, if it does thrashing may occur.

7.7. Destination Options Header [IPv6, section 4.6]



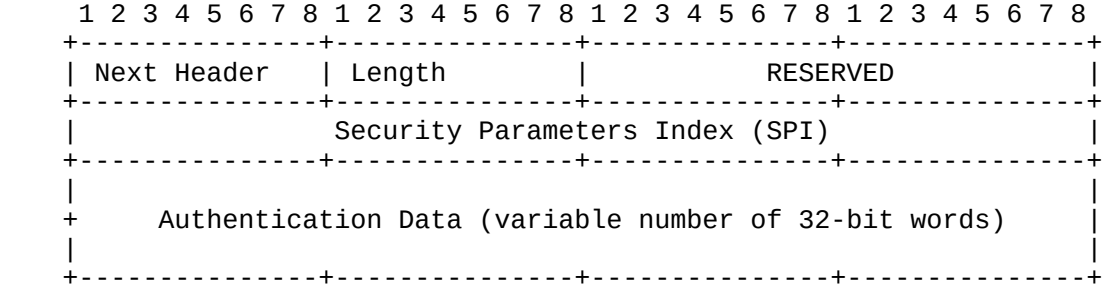
Next Header	NOCHANGE
Hdr Ext Len	NOCHANGE
Options	TLV coded values and padding. Compressed according to 7.3 above.

The only Destination Options defined in [IPv6] are the padding options.

7.8. No Next Header [IPv6, section 4.7]

Covered by rules for IPv6 Header Extensions (7.2).

7.9. Authentication Header [RFC-2402, section 3.2]



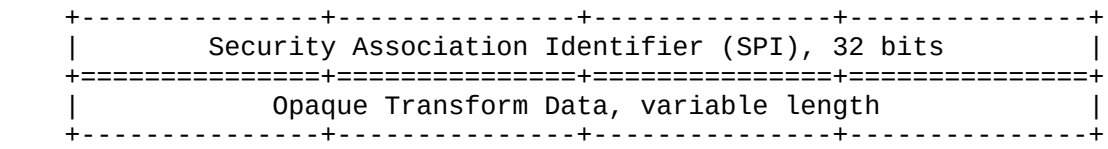
Next Header	NOCHANGE
Length	NOCHANGE
Reserved	NOCHANGE
SPI	NOCHANGE (DEF)
Authentication Data	RANDOM

[RFC-1828] specifies how to do authentication with keyed MD5, the authentication method all IPv6 implementations must support. For this method, the Authentication Data is 16 octets.

7.10. Encapsulating Security Payload Header [RFC-2406, section 3.1]

This header implies that the subsequent parts of the packet are encrypted. Thus, no further header compression is possible on subsequent headers as encryption is typically already performed when the compressor sees the packet.

However, when the ESP Header is used in tunnel mode an entire IP packet is encrypted, and the headers of that packet MAY be compressed before the packet is encrypted at the entry point of the tunnel. This means that it must be possible to feed an IP packet and its length to the decompressor, as if it came from the link-layer. The mechanisms for dealing with reordering described in section 11 MUST also be used, as packets can be reordered in a tunnel.



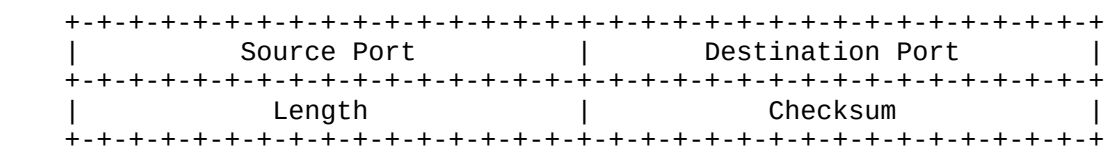
SPI	NOCHANGE (DEF)
Opaque Transform Data	RANDOM

Everything after the SPI is encrypted and is not compressed.

7.11. UDP Header

The UDP header is described in [RFC-768].

The Next Header field (IPv6) or Protocol field (IPv4) in the preceding subheader is DEF.



Source Port	NOCHANGE (DEF)
Destination Port	NOCHANGE (DEF)
Length	INFERRED
Checksum	RANDOM, unless it is zero, in which case it is NOCHANGE.

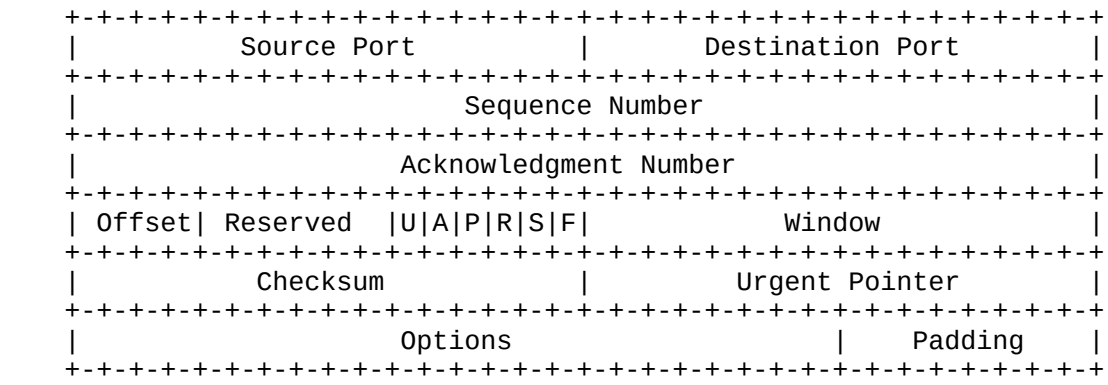
The Length field of the UDP header MUST match the Length field(s) of preceding subheaders, i.e, there must not be any padding after the UDP payload that is covered by the IP Length.

The UDP header is typically compressed down to 2 octets, the UDP checksum. When the UDP checksum is zero (which it cannot be with IPv6), it is likely to be so for all packets in the flow and is defined to be NOCHANGE. This saves 2 octets in the compressed header.

7.12. TCP Header

The TCP header is described in [RFC-793].

The Next Header field (IPv6) or Protocol field (IPv4) in the preceding subheader is DEF.



U, A, P, R, S, and F stands for Urg, Ack, Psh, Rst, Syn, and Fin.

There are two ways to compress the TCP header.

7.12.1. Compressed with differential encoding

Source Port	NOCHANGE (DEF)	
Destination Port	NOCHANGE (DEF)	
Sequence Number	DELTA	
Acknowledgment Number	DELTA	
Offset	NOCHANGE	
Reserved	DELTA	(if differs from context, set R-flag in flag octet and send absolute value as described in 6 a.)
Urg,Psh	RANDOM	(placed in flag octet)
Ack	INFERRED to be 1	
Rst,Syn,Fin	INFERRED to be 0	
Window	DELTA	(if change in Window, set W-flag in flag octet and send difference)
Checksum	RANDOM	
Urgent Pointer	DELTA	(if Urg is set, send absolute value)
Options, Padding	DELTA	(if change in Options, set O-flag and send whole Options, Padding)

A packet with a TCP header compressed according to the above must be indicated to be of type COMPRESSED_TCP. The compressed header is described in section 6.

This method is essentially the differential encoding techniques of Jacobson, described in [RFC-1144], the differences being the placement of the compressed TCP header fields (see section 6), the use of the O-flag, the use of the R-flag, and elimination of the C-flag. The O-flag allows compression of the TCP header when the Timestamp option is used and the Options fields changes with each header.

DELTA values (except for Reserved field and Options, Padding) MUST be coded as in [RFC-1144]. A Reserved field value passed with the R-flag MUST NOT update the context at compressor or decompressor.

7.12.2. Without differential encoding

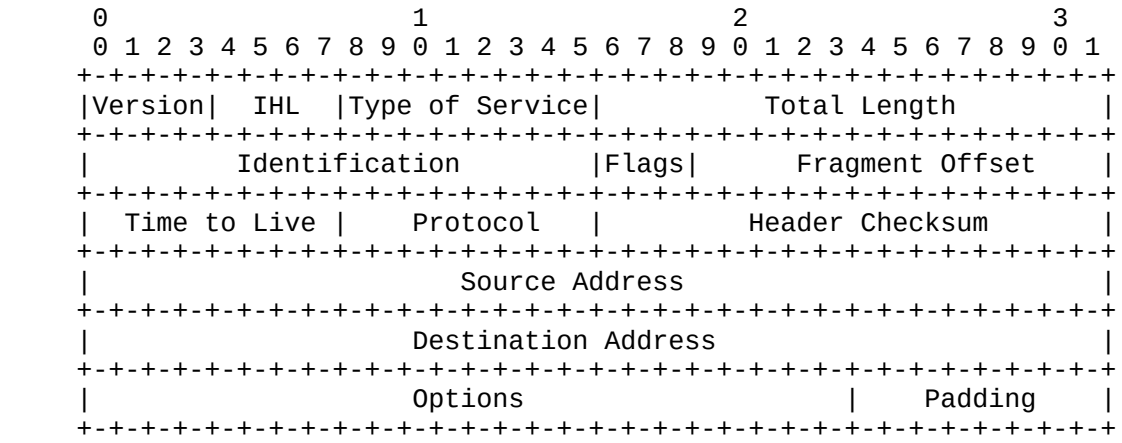
Source Port	NOCHANGE	(DEF)
Destination Port	NOCHANGE	(DEF)
(all the rest)	RANDOM	

The Identification field in a preceding IPv4 header is RANDOM.

A packet with a TCP header compressed according to the above must be indicated to be of type COMPRESSED_TCP_NODELTA. It uses the same CID space as COMPRESSED_TCP packets, and the header MUST be saved as context. The compressed header is described in section 6.

This packet type can be sent as the response to a header request instead of sending a full header, can be used over links that reorder packets, and can be sent instead of a full header when there are changes that cannot be represented by a compressed header. A sophisticated compressor can switch to sending only COMPRESSED_TCP_NODELTA headers when the packet loss frequency is high.

7.13. IPv4 header [RFC-791, section 3.1]



There are two ways to compress the IPv4 header

a) If the IPv4 header is not for a fragment (MF flag is not set and Fragment Offset is zero) and there are no options (IHL is 5), it is classified as follows

Version	NOCHANGE	(DEF)
IHL	NOCHANGE	(DEF, must be 5)
Type of Service	NOCHANGE	(might be DEF, see sect 4.1) (see also 6 a)
Total Length	INFERRED	(from link-layer implementation or encapsulating IP header)
Identification	DELTA/	(If the Protocol field has the (value corresponding to TCP)
	RANDOM	(otherwise)
Flags	NOCHANGE	(MF flag must not be set)
Fragment Offset	NOCHANGE	(must be zero)
Time to Live	NOCHANGE	(might be DEF, see sect 4.1)
Protocol	NOCHANGE	
Header Checksum	INFERRED	(calculated from other fields)
Source Address	NOCHANGE	(DEF)
Destination Address	NOCHANGE	(DEF)
Options, Padding		(not present)

Note: When a TCP header immediately follows, the IPv4 and TCP header MUST be compressed as a unit as described in section 6. Bits 6 and 7 of the Type of Service field (bits 14 and 15 of the first word) can then be passed using the R-flag (see section 6 a).

b) If the IPv4 header is for a fragment (MF bit set or Fragment Offset nonzero), or there are options (IHL > 5), all fields are RANDOM (i.e., if the header is compressed all fields are sent as-is and not compressed). This classification allows compression of the tunnel header, but not the fragment header, when fragments are tunneled. If the IPv4 header is for a fragment it ends the compressible chain of subheaders, i.e., it must be the last subheader to be compressed. If the IPv4 header has options but is not for a fragment it does not end the compressible chain of subheaders, so subsequent subheaders can be compressed.

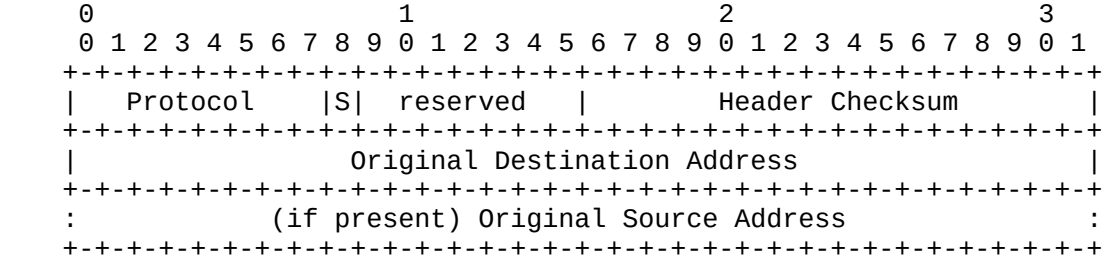
A compressor that follows the optional guidelines of section 4.1 will in case a) use the Version, Source Address and Destination Address to define the packet stream, together with the fact that there are no IPv4 options and that this is not a fragment.

Case b) can define two kinds of packet streams depending on whether the IPv4 header is for a fragment or not.

If the IPv4 header in case b) is for a fragment, a compressor following the optional guidelines will use that fact together with the Version, Source Address, and Destination Address to determine the packet stream.

If the IPv4 header in case b) is not for a fragment, it must have options. A compressor following the optional guidelines will use that fact, but not the size of the options, together with the Version, Source Address, and Destination Address to determine the packet stream.

7.14. Minimal Encapsulation header [RFC-2004, section 3.1]



Protocol	NOCHANGE
Original Source Address Present (S)	NOCHANGE
reserved	NOCHANGE
Header Checksum	INFERRED (calculated from other values)
Original Destination Address	NOCHANGE
Original Source Address	NOCHANGE (present only if S=1)

This header is likely to be used by Mobile IP.

8. Changing context identifiers

On a point-to-point link, the compressor has total knowledge of what CIDs are in use at the decompressor and may change what CID a packet stream uses or reuse CIDs at will.

Each non-TCP CID is associated with a context with a generation value. To avoid too rapid generation wrap-around and potential incorrect decompression, an implementation MUST avoid wrap-around of the generation value in less than MIN_WRAP seconds (see section 14).

To aid in avoiding wrap-around, the generation value associated with a CID MUST NOT be reset when changing to a new packet stream. Instead, a compressor MUST increment the generation value by one when using the CID for a new non-TCP packet stream.

9. Rules for dropping or temporarily storing packets

When a decompressor receives a packet with a compressed TCP header with CID C, it MUST be discarded when the context for C has not been initialized by a full header.

When a decompressor receives a packet with a compressed non-TCP header with CID C and generation G, the header must not be decompressed using the current context when

a) the decompressor has been disconnected from the compressor for more than MIN_WRAP seconds, because the context might be obsolete even if it has generation G.

b) the context for C has a generation other than G.

In case a) and b) the packet may either be

i) discarded immediately, or else

ii) stored temporarily until the context is updated by a packet with a full non-TCP header with CID C and generation G, after which the header can be decompressed.

Packets stored in this manner MUST be discarded when

*) receiving full or compressed non-TCP headers with CID C and a generation other than G,

*) the decompressor has not received packets with CID C in the last MIN_WRAP seconds.

When full headers are lost, a decompressor can receive compressed non-TCP headers with a generation value other than the generation of its context. Rule ii) allows the decompressor to store such headers until they can be decompressed using the correct context.

10. Low-loss header compression for TCP

Since fewer bits are transmitted per packet with header compression, the packet loss rate is lower with header compression than without, for a fixed bit-error rate. This is beneficial for links with high bit-error rates such as wireless links.

However, since TCP headers are compressed using differential encoding, a single lost TCP segment can ruin an entire TCP sending window because the context is not incremented properly at the decompressor. Subsequent headers will therefore be decompressed to be different than before compression and discarded by the TCP receiver because the TCP checksum fails.

A TCP connection in the wide area where the last hop is over a medium-speed lossy link, for example a wireless LAN, will then have poor performance with traditional header compression because the delay-bandwidth product is relatively large and the bit-error rate relatively high. For a 2 Mbit/s wireless LAN and an end-to-end RTT of 200 ms, the delay-bandwidth product is 50 kbyte. That is equivalent to about 97 512-octet segments with compressed headers. Each loss can thus be multiplied by a factor of 100.

This section describes two simple mechanisms for quick repair of the context. With these mechanisms header compression will improve TCP throughput over lossy links as well as links with low bit-error rates.

10.1. The "twice" algorithm

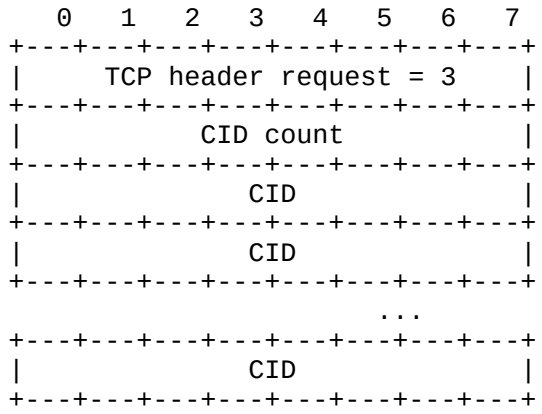
The decompressor may compute the TCP checksum to determine if its context is not updated properly. If the checksum fails, the error is assumed to be caused by a lost segment that did not update the context properly. The delta of the current segment is then added to the context again on the assumption that the lost segment contained the same delta as the current. By decompressing and computing the TCP checksum again, the decompressor checks if the repair succeeded or if the delta should be applied once more.

Analysis of traces of various TCP bulk transfers show that applying the delta of the current segment one or two times will repair the context for between 83 and 99 per cent of all single-segment losses in the data stream. For the acknowledgment stream, the success rate is smaller due to the delayed ack mechanism of TCP. The "twice" mechanism repairs the context for 53 to 99 per cent of the losses in the acknowledgment stream. A sophisticated implementation of this idea would determine whether the TCP stream is an acknowledgment or data stream and determine the segment size by observing the stream of full and compressed headers. Trying deltas that are small multiples of the segment size will result in even higher rates of successful repairs for acknowledgment streams.

10.2. Header Requests

The relatively low success rate for the "twice" algorithm for TCP acknowledgment streams calls for an additional mechanism for repairing the context at the decompressor. When the decompressor fails to repair the context after a loss, the decompressor may optionally request a full header from the compressor. This is possible on links where the decompressor can identify the compressor and send packets to it.

On such links, a decompressor may send a CONTEXT_STATE packet back to the compressor to indicate that one or more contexts are invalid. A decompressor SHOULD NOT transmit a CONTEXT_STATE packet every time a compressed packet refers to an invalid context, but instead should limit the rate of transmission of CONTEXT_STATE packets to avoid flooding the reverse channel. A CONTEXT_STATE packet can indicate that several contexts are out of date, this technique SHOULD be used instead of sending several separate packets. The following diagram shows the format of a CONTEXT_STATE packet.



The first octet is a type code to allow the CONTEXT_STATE packet type to be shared for other compression protocols that are (see [CRTP]) or may be defined in parallel with this one. When used for TCP header requests the type code has the value 3, and the remainder of the packet is a sequence of CIDs preceded by a one-octet count of the number of CIDs.

On receipt of a CONTEXT_STATE packet, the compressor MUST mark the CIDs invalid to ensure that the next packet emitted in those packet streams are FULL_HEADER or COMPRESSED_TCP_NODELTA packets.

Header requests are an optimization, so loss of a CONTEXT_STATE packet does not affect the correct operation of TCP header compression. When a CONTEXT_STATE packet is lost, eventually a new one will be transmitted or TCP will timeout and retransmit. The big advantage of using header requests is that TCP acknowledgment streams can be repaired after a roundtrip-time over the lossy link. This will typically avoid a TCP timeout and unnecessary retransmissions. The lower packet loss rate due to smaller packets will then result in higher throughput because the TCP window can grow larger between losses.

11. Links that reorder packets

Some links reorder packets, for example multi-hop radio links that use deflection routing to route around congested nodes. Packets routed different ways can then arrive at the destination in a different order than they were sent.

11.1. Reordering in non-TCP packet streams

Compressed non-TCP headers do not change the context, and neither do full headers that refresh it. There can be problems only when a full header that changes the context arrives out of order. There are two cases:

- A packet with a full header with generation G arrives *after* a packet with a compressed header with generation G. This case is covered by rule b) ii) in section 9.

- A packet with a full header with generation G arrives *before* a packet with a compressed header with generation G-1 (modulo 64). The decompressor MAY then keep both versions of the context around for a while to be able to decompress subsequent compressed headers with generation G-1 (modulo 64). The old context MUST be discarded after MIN_WRAP seconds.

11.2. Reordering in TCP packet streams

A compressor may avoid sending COMPRESSED_TCP headers and only send COMPRESSED_TCP_NODELTA headers when there is reordering over the link. Compressed headers will typically be 17 octets with that method, significantly larger than the usual 4-7 octets.

To achieve better compression rates the following method, adding only two octets to the compressed header for a total of 6-9 octets, may be used. A packet sequence number, incremented by one for every packet in the TCP stream, is then associated with each compressed and full header. This allows the decompressor to place the packets in the correct sequence and apply their deltas to the context in the correct order. A simple sliding window scheme is used to place the packets in the correct order.

Two octets are needed for the packet sequence numbers. One octet gives only 256 sequence numbers. In a sliding window scheme the window should be no larger than half of the sequence number space, so packets can not arrive more than 127 positions out-of-sequence. This is equivalent to a delay of 260 ms on 2 Mbit/s links with 512 octet segments. Delays of that order are not uncommon over wide-area Internet connections. However, two octets giving 2^16 = 65536 values should be sufficient.

Full TCP/IP headers will only have space for one octet of sequence number when there is no tunneling. It is not feasible to increase the size of full headers since the packet size might be optimized for the MTU of the link. Therefore only the least significant octet of the packet sequence number can be placed in such full headers. We believe

that such full headers can be positioned correctly frequently enough with only the least significant octet of the packet sequence number available.

The packet sequence number zero MUST be skipped over. Avoiding zero takes care of a problem that can occur when the TCP window scale option is used to enlarge the TCP window. When exactly 2^16 octets of TCP data is lost, a compressed header will be decompressed incorrectly without being detected by the TCP checksum. TCP segment sizes are often a power of two. So by using a packet sequence number space that is not a power of two either the TCP sequence number or the packet sequence number will differ when 2^16 octets are lost. Whenever a compressor sees the window scale option on a SYN segment, it MUST use packet sequence numbers when subsequently compressing that packet stream.

In compressed TCP headers the two octet packet sequence number MUST be placed immediately after the TCP Checksum. See section 5.3 for placement of packet sequence numbers in full headers.

12. Hooks for additional header compression

The following hook is supplied to allow additional header compression schemes for headers on top of UDP. The initial chain of subheaders is then compressed as described here, and the other header compression scheme is applied to the header above the UDP header. An example of such additional header compression is Compressed RTP by Casner and Jacobson [CRTP]. To allow some error detection, such schemes typically need a sequence number that may need to be passed in full headers as well as compressed UDP headers.

The D-bit and Data octet (see section 6) provides the necessary mechanism. When a sequence number, say, needs to be passed in a FULL_HEADER or COMPRESSED_NON_TCP header, the D-bit is set and the sequence number is placed in the Data field. The decompressor must then extract and make the Data field available to the additional header compression scheme.

Use of additional header compression schemes like CRTP must be negotiated. The D-bit and Data octet mechanism must automatically be enabled whenever use of additional header compression schemes has been negotiated.

13. Demultiplexing

For each link layer, there must be a document specifying how the various packet types used by IP header compression is indicated. Such a document exists for PPP [PPP-HC]. This section gives OPTIONAL guidelines on how packet types may be indicated by a specific link-layer.

It is necessary to distinguish packets with regular IPv4 headers, regular IPv6 headers, full IPv6 packets, full IPv4 packets, compressed TCP packets, compressed non-TCP packets, and CONTEXT_STATE packets.

The decision to use a distinct ethertype (or equivalent) for IPv6 has already been taken, which means that link-layers must be able to indicate that a packet is an IPv6 packet.

IP header compression requires that the link-layer implementation can indicate four kinds of packets: COMPRESSED_TCP for format a) in section 6, COMPRESSED_TCP_NODELTA for format b), COMPRESSED_NON_TCP for formats c) and d), and CONTEXT_STATE as described in section 11.2. It is also desirable to indicate FULL_HEADERS at the link layer.

Full headers can be indicated by setting the first bit of the Version field in a packet indicated to be an IPv6 packet. In addition, one bit of the Version field is used to indicate if the first subheader is an IPv6 or an IPv4 header, and one bit is used to indicate if this full header carries a TCP CID or a non-TCP CID. The first four bits are encoded as follows:

Version	Meaning
0110	regular IPv6 header
1T*0	T=1 indicates a TCP header, T=0 indicates a non-TCP header
1*V0	V=1 indicates a IPv6 header, V=0 indicates a IPv4 header

If a link-layer cannot indicate the packet types for the compressed headers or CONTEXT_STATE, packet types that cannot be indicated could start with an octet indicating the packet type, followed by the header.

First octet	Type of compressed header
0	COMPRESSED_TCP
1	COMPRESSED_TCP_NODELTA
2	COMPRESSED_NON_TCP
3	CONTEXT_STATE

The currently assigned CONTEXT_STATE type values are

Value	Type	Reference
0	Reserved	-
1	IP/UDP/RTP w. 8-bit CID	[CRTP]
2	IP/UDP/RTP w. 16-bit CID	[CRTP]
3	TCP header request	Section 10.2

14. Configuration Parameters

Header compression parameters are negotiated in a way specific to the link-layer implementation. Such procedures for link-layer xxx needs to be specified in a document "IP header compression over xxx". Such a document exists for PPP [PPP-HC].

The following parameter is fixed for all implementations of this header compression scheme.

MIN_WRAP - minimum time of generation value wrap around

3 seconds.

The following parameters can be negotiated between the compressor and decompressor. If not negotiated their values must be as specified by DEFAULT.

F_MAX_PERIOD - Largest number of compressed non-TCP headers that may be sent without sending a full header.

DEFAULT is 256

F_MAX_PERIOD must be at least 1 and at most 65535.

F_MAX_TIME - Compressed headers may not be sent more than F_MAX_TIME seconds after sending last full header.

DEFAULT is 5

F_MAX_TIME must be at least 1 and at most 255.

NOTE: F_MAX_PERIOD and F_MAX_TIME should be lower when it is likely that a decompressor loses its state.

MAX_HEADER - The largest header size in octets that may be compressed.

DEFAULT is 168 octets, which covers

- Two IPv6 base headers
- A Keyed MD5 Authentication Header
- A maximum-sized TCP header

MAX_HEADER must be at least 60 octets and at most 65535 octets.

TCP_SPACE - Maximum CID value for TCP.

DEFAULT is 15 (which gives 16 CID values)

TCP_SPACE must be at least 3 and at most 255.

NON_TCP_SPACE - Maximum CID value for non-TCP.

DEFAULT is 15 (which gives 16 CID values)

NON_TCP_SPACE must be at least 3 and at most 65535.

EXPECT_REORDERING - The mechanisms in section 11 are used.

DEFAULT no.

15. Implementation Status

A prototype using UDP as the link layer has been operational since March 1996. A NetBSD implementation for PPP has been operational since October 1996.

16. Acknowledgments

This protocol uses many ideas originated by Van Jacobson in the design of header compression for TCP/IP over slow-speed links [RFC-1144]. It has benefited from discussions with Stephen Casner and Carsten Bormann.

We thank Craig Partridge for pointing out a problem that can occur when the TCP window scale option is used. A solution to this problem relying on the packet sequence numbers used for reordering is described in section 11.2.

17. Security Considerations

The compression protocols in this document run on top of a link-layer protocol. The compression protocols themselves introduce no new additional vulnerabilities beyond those associated with the specific link-layer technology being used.

Denial-of-service attacks are possible if an intruder can introduce (for example) bogus Full Header packets onto the link. However, an intruder having the ability to inject arbitrary packets at the link-layer in this manner raises additional security issues that dwarf those related to the use of header compression.

We advise implementors against identifying packet streams with the aid of information that is encrypted, even if such information happens to be available to the compressor. Doing so may expose traffic patterns.

18. Authors' Addresses

Mikael Degermark
Department of Computer Science and Electrical Engineering
Lulea University of Technology
SE-971 87 Lulea, Sweden

Phone: +46 920 91188
Fax: +46 920 72831
Mobile: +46 70 833 8933
EMail: micke@sm.luth.se

Bjorn Nordgren
CDT/Telia Research AB
Aurorum 6
S-977 75 Lulea, Sweden

Phone: +46 920 75400
Fax: +46 920 75490
EMail: bcn@lulea.trab.se, bcn@cdt.luth.se

Stephen Pink
Department of Computer Science and Electrical Engineering
Lulea University of Technology
SE-971 87 Lulea, Sweden

Phone: +46 920 752 29
Fax: +46 920 728 31
Mobile: +46 70 532 0007
EMail: steve@sm.luth.se

19. References

[RFC-768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

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

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

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

[RFC-1553] Mathur, A. and M. Lewis, "Compressing IPX Headers Over WAN Media (CIPX)", RFC 1553, December 1993.

[RFC-1700] Reynolds, J. and J. Postel, "Assigned Numbers", STD 2, RFC 1700, October 1994. See also: http://www.iana.org/numbers.html

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

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

[RFC-1828] Metzger, W., "IP Authentication using Keyed MD5", RFC 1828, August 1995.

[IPv6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[ICMPv6] Conta, A. and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification.", RFC 2463, December 1998.

[RFC-2004] Perkins, C., "Minimal Encapsulation within IP", RFC 2004, October 1996.

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

[PPP-HC] Engan, M., Casner, S. and C. Bormann, "IP Header Compression for PPP", RFC 2509, February 1999.

20. Full Copyright Statement