Soft State Header Compression for Wireless Networks[*]

Mikael Degermark[†]

Stephen Pink[†‡]

[†]CDT/Department of Computer Science
Luleå University
S-971 87 Luleå, Sweden
`Mikael.Degermark@sm.luth.se`

[‡]Swedish Institute of Computer Science
PO box 1263
S-164 28 Kista, Sweden
`Stephen.Pink@sics.se`

Abstract

Low-speed wireless is becoming a popular way to connect mobile computers to the Internet and other networks. Users will desire the same services on low speed links as they already have on higher speed networks. Since bandwidth will probably always be a limitation on wide area wireless links, protocol mechanisms need to be developed to use the available bandwidth on low speed networks efficiently. In the face of growing address sizes, such as those in IPv6, we offer a header compression scheme that enables efficient use of bandwidth when packets are small. The scheme uses soft state*, works over simplex links, supports multicast transmission for real time applications, and besides providing more usable bandwidth on wireless links, allows better utilization of gateways and servers attached to wireless networks.*

1 Introduction

In the future, many end systems will be attached to the global Internet over relatively low-speed wireless links. With the advent of the Mbone as well as some commercial real time applications, audio and video are becoming important applications in the Internet. It is thus important to utilize such links efficiently to allow audio over low speed wireless links. When bandwidth is very low, video may not be feasible. Audio applications, however, can have sufficiently low data rates to be viable over low-speed links. Delay sensitive applications with low data rates, such as interactive voice conversations, are best served by small packets that are filled quickly. The bandwidth consumed by packet headers is relatively large when packet sizes are small. This can be prohibitive on low-speed links especially with the coming of IPv6 which increases the IP address size from 4 to 16 bytes per address.

[*]This work was supported by grants from the Centre for Distance Spanning Technology (CDT), Luleå, Sweden, and Ericsson Radio Systems AB.

Header compression can reduce the bandwidth required for headers by an order of magnitude, and, since header compression allows smaller packet sizes, the end-to-end delay can be significantly lower. In addition, as the overall bandwidth efficiency of the link is increased, other applications will also benefit from header compression. By paying processing time and header storage, bandwidth is saved. Thus, it is a bad trade-off to do header compression in the high-speed backbone, and a good trade-off on low-speed links.

The efficiency of our header compression scheme is based on there being consecutive headers belonging to the same packet flow that are identical or change very little during the life of the flow. This allows the upstream node to send a short index identifying a previously sent header stored as state in the downstream node, instead of sending the full header.

A header compression scheme must be able to detect when stored headers in the upstream and downstream nodes become inconsistent due to loss. Not doing so would mean decompressing packet headers incorrectly. It is desirable to update the downstream headers quickly to avoid long periods of loss. We present a method of doing this that requires no upstream messages, and thus can be used over simplex links. The scheme makes use of *soft state*, a form of state storage introduced by Clark [2] and used, for example, in the RSVP resource reservation protocol [10] to make efficient use of Internet router and link resources. In our header compression scheme, soft state promotes better resource utilization on wireless and other low-bandwidth links and gateways.

2 Related work

Jacobson [6] has developed a header compression scheme for TCP streams over point-to-point links. The scheme uses incremental encoding, and can rely on the TCP checksum to discard incorrectly decompressed packets. The compressor peeks into the TCP headers to determine when TCP retransmits. Whenever it does, a full header is sent to update the stored header at the decompressor on the assumption that the loss was due to mismatching stored headers.

The methods presented in this paper are different as they aim at compressing headers of unreliable flows that use, for example, UDP. Jacobson's methods cannot be applied to such flows because there are no retransmissions, and because the flows are not necessarily point-to-point, i.e., they may be multicast flows. Moreover, since the headers typically do not contain sequence numbers, the UDP checksum is less likely to detect incorrectly decompressed headers. A goal of our scheme is to do efficient header compression for flows that use IP multicast over shared media networks. A complete header compression module would use Jacobson's methods for TCP streams and our methods for non-TCP streams.

There are header compression schemes, for example a scheme for IPX [7], where compressor and decompressor perform a handshake whenever the stored header must change. Such methods cannot be used over simplex links and are hard to adapt for multicast over shared media.

3 Wireless multimedia

Multimedia makes special demands on network protocols, e.g., reserving resources, providing bounded end-to-end delays and delay variations. Networks of the future will often have, on the edges, slow speed wireless links that provide, among other things, mobile support for real time applications. Many wireless links, especially wide area networks, are of relatively low bandwidth, making bandwidth a scarce end-to-end resource. Network protocols, therefore, need to deal with low as well as high bandwidth links to provide multimedia services.

At the same time as wireless links are becoming popular, the Internet is being used increasingly to transmit multimedia including delay sensitive traffic such as real time audio and video. One example of this is the Mbone which uses IP multicast. Other examples include new commercial applications that attempt to imitate telephony service between users on dial up Internet connections.

The end-to-end delay budget for interactive voice conversations can be as low as 150 ms. The propagation delay in the links consumes part of this; ideally it is about 20 ms across the US (coast to coast) and 100 ms to the farthest point in a global network. To allow for processing time and queuing delay in network elements, the time spent in end systems should be short.

Audio and video applications typically sample a certain amount of data before placing it in a packet and sending it off. When the data rate is low, the time consumed filling a packet is significant and small packets are preferred when low total end-to-end delay is required. However, when packets are small the amount of bandwidth consumed by packet headers increases.

For example, if an audio application reduces its packet size to contain 20 ms of audio samples instead of 80 ms, IPv4/UDP headers (28 byte) consume 11.2 kbit/s instead of 2.8 kbit/s and IPv6/UDP headers (48 byte) consume 19.2 kbit/s instead of 4.8 kbit/s. When packets are tunneled across a wireless network, e.g., to support mobility [1, 8, 9], or when extra routing information is added to an IPv6 header [4, section 4.4], the bandwidth needed for headers alone can approach 50 kbit/s. With header compression, IPv6/UDP headers can be reduced to 4 bytes and 20 ms packets would only require 1.6 kbit/s for the headers.

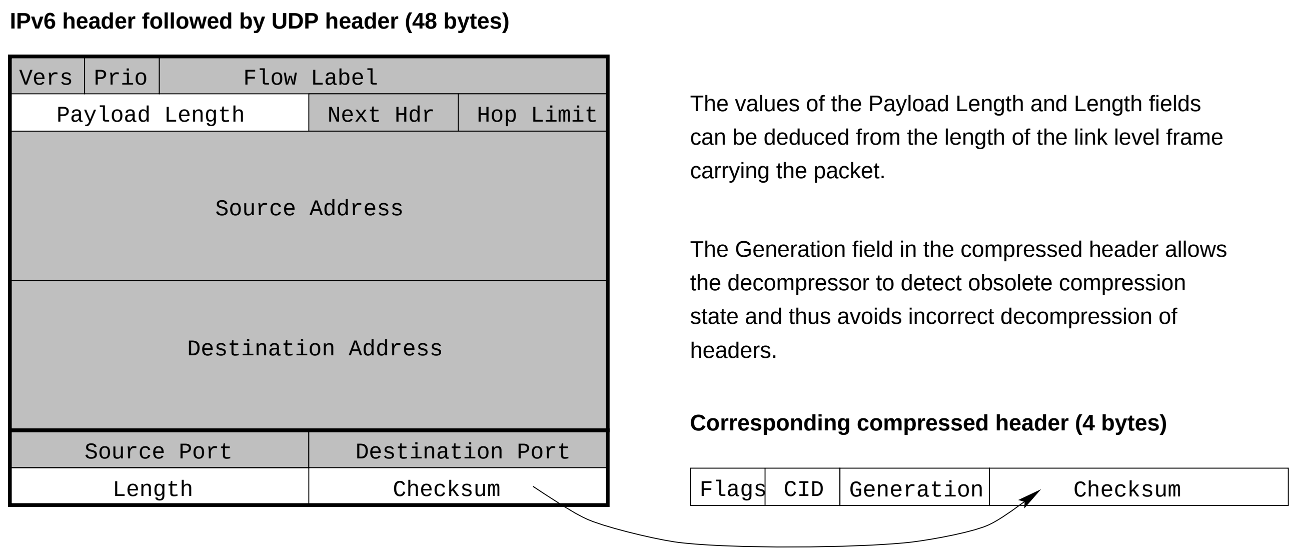
Table 1 shows the required header bandwidth for various headers and times between packets. *Tunnel* means a IPv6/UDP header encapsulated in an IPv6 header. *Routing* means an IPv6/UDP header with a four address routing header. *Compr* means the compressed version of IPv6/UDP, *tunnel*, or *routing*. For comparison, the bandwidth needed for the actual audio samples is somewhere between 10 kbit/s for GSM quality to 128 kbit/s for CD quality.

	Time between packets		
	80 ms	40 ms	20 ms
IPv4/UDP	2.8	5.6	11.2
IPv6/UDP	4.8	9.6	19.2
tunnel	8.8	17.6	35.2
routing	12.0	24.0	48.0
compr	0.4	0.8	1.6

Table 1: Required bandwidth for headers, kbit/s

4 Compression and state

Header compression relies on consecutive headers belonging to the same flow being identical or changing seldomly. Figure 1 shows a 48 byte IPv6/UDP header with the fields expected to stay the same in grey. It also shows a typical compressed header. To compress headers, the upstream node can send a full header at the beginning and otherwise refer to the last full header sent in the flow, the *compression state* for that flow, with a small identifier, the compression



The values of the Payload Length and Length fields can be deduced from the length of the link level frame carrying the packet.

The Generation field in the compressed header allows the decompressor to detect obsolete compression state and thus avoids incorrect decompression of headers.

Figure 1: Headers. Grey fields are not expected to change between consecutive headers in a flow, and are stored as compression state in compressor and decompressor.

state identifier (CID), on subsequent packets. The downstream node stores the full header and uses it to expand compressed headers.

Whenever there is a change in the grey fields between consecutive headers of a flow, the next packet is sent with a full header to update the compression state. This is straightforward in the ideal case of a lossless link. When a packet is lost, however, the compression state in the upstream and downstream nodes may diverge so that the downstream compression state needs to be refreshed by resending a full header (see section 6).

The CIDs should be kept small to improve the compression rate. Since they are a scarce resource, they must be managed carefully. On a point-to-point link this is straightforward as the compressor knows what CIDs are in use at the decompressor. On shared media networks, the CID space is shared among several compressors, so the compressor does not know which CIDs are in use. We use link level addresses in conjunction with the CID to allow the decompressor to have a separate CID space for each compressor.

The situation is even more complicated for multicast flows over shared media, as the compressor does not know who, or even how many, the decompressors are. In addition, the receivers of a multicast flow may be heterogeneous in the sense that some may not be capable of decompression, and the compressor needs a method to determine this. By a utilizing an existing mechanism (IGMP [3]) for determining if there are members of a multicast group on a shared medium, this problem can be solved. If all receivers must be able to decompress, the mechanisms for dealing with heterogeneity are not needed and the ability to use simplex links is preserved.

5 Soft state

The state in our header compression is “soft” in the sense that it needs to be refreshed periodically or is garbage collected away. Soft state, used in RSVP [10] to setup resource reservations on behalf of receivers, has advantages also for mobile users on wireless links. Soft state can provide better utilization of scarce resources in a mobile server that functions as the gateway from the wireline Internet to a wireless link.

In such cases, mobile clients communicating with the server set up soft compression state that needs to be refreshed by full headers periodically. If a mobile client goes out of range of the server, something that can happen frequently, the server can reclaim the compression state after the soft state timeout without waiting for an explicit close from the client. This increases the server’s capacity to serve other mobile clients. When the first mobile client returns into range of the server, state can be set up again and header compression over the wireless link resumed.

We are able to get soft state by trading off some header compression. A hard-state based scheme does not send refresh messages and so will get more compression. The amount of compression lost in our soft state approach, however, is minimal. Figure 2 shows the average header size when full headers of size H are

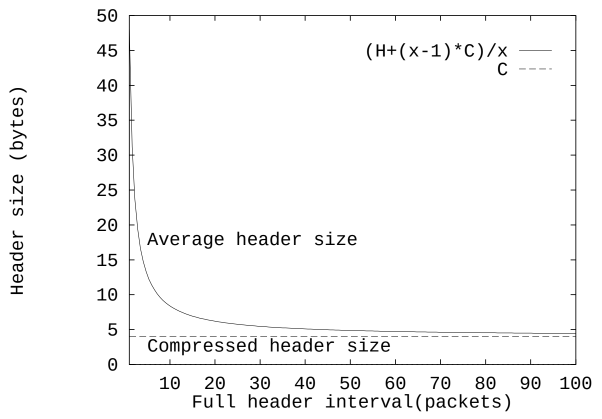


Figure 2: Average header size. $H = 48$, $C = 4$.

sent every xth packet, and the others have compressed headers of size C. For comparison, the diagram also shows the size of the compressed header. The values used for H and C are typical for UDP/IPv6. It is clear from figure 2 that if the header refresh frequency is increased past the knee of the curve, the size of the average header is very close to the size of the compressed header. For example, if we decide to send 256 compressed headers for every full header, roughly corresponding to a full header every five seconds when there are 20 ms between packets, the average header is 1.4 *bits* larger than the compressed header.

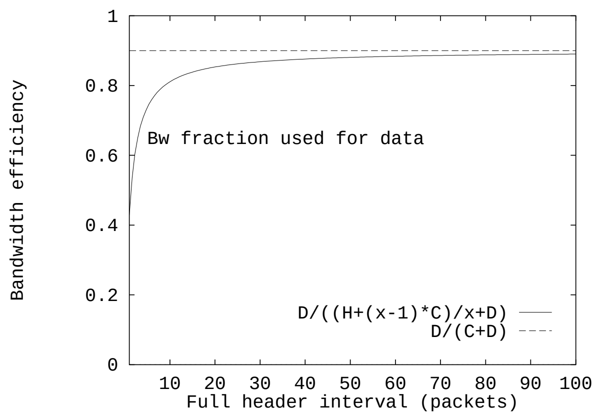


Figure 3: Bandwidth Efficiency. $H = 48$, $C = 4$, $D = 36$.

Figure 3 shows the bandwidth efficiency, i.e., the fraction of the consumed bandwidth used for actual data. The bandwidth efficiency when all headers are compressed is shown for comparison. The size of the data, D, is 36 bytes, which corresponds to 20 ms of GSM encoded audio samples.

Figures 2 and 3 show that, when operating to the right of the knee of the curve, the *size* of the compressed header is more important than *how often* the occasional full header is sent due to soft state refreshes or changes in the header. The cost is slightly higher than for handshake-based schemes, but we think that is justified by the ability of our scheme to compress on simplex links and compress multicast packets. Moreover, we saw above that scarce resources on a mobile gateway or server can be shared more efficiently when a soft state approach is taken.

An important parameter for soft state is the setting of the timer that triggers garbage collection of the compression state. If the timer is set for too short a period, the state can be dropped just before a refresh message arrives and delay can ensue as a result of having to setup the compression state again. If the timer is set for too long a period, the mobile gateway could reject the entrance of new mobile hosts wanting compression on the wireless link while the gateway waits for a host that has gone permanently out of range. The timer should be set to go off after a few header refreshes are missing in order to avoid unnecessarily removing the state if only a single header refresh is lost.

6 Compression slow-start

Whenever a full header would have changed the compression state and the packet carrying that full header is lost, subsequent compressed headers will be decompressed incorrectly due to inconsistent compression state in compressor and decompressor. Thus, those packets carrying subsequent compressed headers must be dropped by or stored in the decompressor until the compression state is refreshed. Without extra mechanisms, the only protection against incorrectly decompressed headers is the transport layer checksum, which does not cover the whole IP header. For example, in figure 1 only the addresses, port numbers, next header field, and UDP Length are protected by the UDP checksum. IPv6 extension headers are not protected by any checksum at all.

Because of this lack of protection, our header compression scheme identifies each version of the compression state with a number, the *generation* of that compression state. Each compressed header contains the generation number of the compression state that should be used to decompress it. Full headers that change the compression state contain a new generation number and full headers that refresh it repeat the last generation number. This allows reliable detection of headers that would be decompressed incorrectly due to inconsistent compression state, since the generation number in those headers will be different from the generation of the compression state.

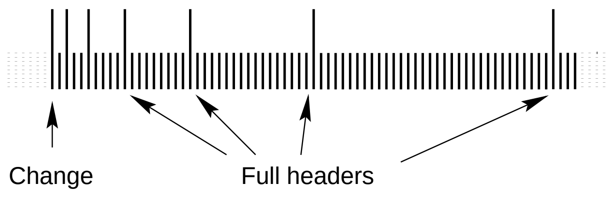


Figure 4: Compression slow-start after header change. All refresh headers carry the same generation number.

To avoid long periods of discarded headers when full headers are lost, the refresh period should be short. To get high compression rates, however, the refresh period should be long. To achieve both these goals, the compressor starts with a short refresh period (one compressed header) when compression begins and when a header changes. The refresh period is then exponentially increased in size with each refresh until the soft state refresh period is reached. We call this *compression slow-start*, and figure 4 illustrates this mechanism. Tall lines represent packets with full headers and short lines packets with compressed headers. If the first packet is lost, the compression state will be synchronized by the third packet and only a single packet with a compressed header must be discarded or stored temporarily. If the first three packets are lost, two additional packets must be discarded or stored, etc. We see that if x packets are lost in an error burst, at most $x - 1$ packets are discarded or stored temporarily due to obsolete compression state.

7 Resource management

Header compression virtually increases the bandwidth of the link. A resource setup protocol like RSVP should be aware of header compression mechanisms to be able to make correct admission control decisions. Setup protocols like RSVP should also be able to request header compression for flows that must have it in order to get the required bandwidth and/or service quality.

The compressor must somehow group packets together into packet streams that share compression state. To get good compression rates, the headers of the resulting packet streams should change seldomly. If they change often, there will be repeated compression slow-starts as the existing compression state can seldomly be used and the compression rate will suffer as full headers are sent frequently.

To determine which packet stream a packet belongs to, a compressor can

- examine the headers of the packet, for example addresses, port numbers, etc,
- 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,
- use other relevant information, for example recent traffic patterns may reveal that there are route flaps such that the TTL often switches between two distinct values, so that using two packet streams is better than using one.

Since CID space is limited and switching between CIDs involves sending more full headers initially because of compression slow start, the compressor must decide to which packet streams the CIDs should be allocated and when to change the allocation. Ideally, the CIDs should be allocated to the streams with the highest packet rates.

8 Reduced packet loss rate

Header compression reduces the number of bits that are transmitted over a link. So for a given bit-error rate the number of transmitted packets containing bit errors is reduced by header compression. This implies that header compression may improve the quality of service over wireless links with high bit-error rates, especially when packets are small, so that the header is a significant fraction of the whole packet.

Header compression, however, may cause the error model for packet streams to change. Without header compression a bit error damages only the packet containing the bit-error. When header compression is used and the bit-error occurs in a full header, the single error could cause loss of subsequent packets. This is because the bit-error might be stored as compression state and when subsequent headers are expanded using that compression state they will contain the same bit-error. Care must be taken to prevent this phenomenon.

It is sufficient for compression state to be installed properly in the decompressor if one full header is transmitted undamaged over the link. What is needed is a way to detect bit-errors in full headers since IPv6 lacks a header checksum. The compressor can extend the UDP checksum to cover the whole full header, and not just part of it, and the decompressor can check the checksum before storing a header as compression state. In this manner erroneous compression state will not be installed in the decompressor and no headers will be expanded to contain bit-errors. The decompressor must restore the original UDP checksum before passing the packet up to IP.

If the decompressor temporarily stores packets for which it does not have proper compression state and expands their headers when a matching full header arrives, no extra packet loss will occur due to header compression. The stored packets will be delayed, however, and hard real-time applications may not be able to utilize them, although adaptive applications might. Once the compression state is installed, the loss rate will decrease.

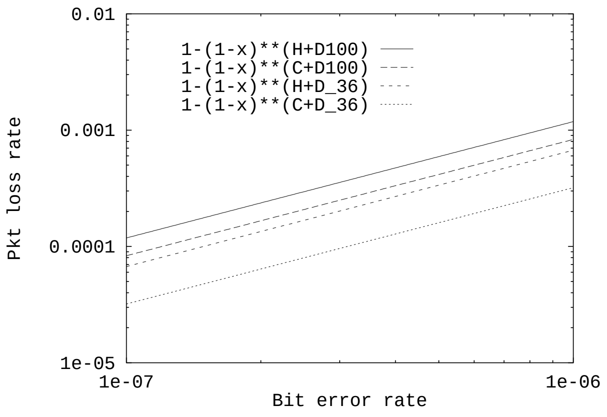


Figure 5: Packet loss rate as a function of a Poisson distributed bit-error rate, with and without header compression and for payloads of 36 and 100 bytes.

Figure 5 shows the packet loss rate as a function of the bit-error rate of the media with and without header compression. The packet loss rates for compressed packets assume that the compression state has been successfully installed. Compressed headers, C, are 4 bytes, full and regular headers, H, are 48 bytes (IPv6/UDP), and the header refresh interval is 256. D is the size of the payload. Bit-errors are Poisson distributed. The packet loss rate is decreased in direct proportion to the decrease in packet size due to header compression. For the 36 byte payload the packet loss rate is decreased by 52% and for the 100 byte payload by 30%.

9 Implementation and standardization status

We currently have a prototype implementation of our header compression scheme for IPv6 and IPv4. The compressor is about 400 lines of C-code and the decompressor about 300 lines. We use our prototype to compress *vat* audio.

The header compression scheme will enable reasonable real-time audio over 14.4 kbit/s modems. With a 28.8 kbit/s modem there will be bandwidth to spare or room for higher quality audio. We are now implementing our header compression scheme in NetBSD, and are planning an implementation for SOLARIS.

In addition, we have submitted our header compression scheme as a draft [5] to the IPng working group of the Internet Engineering Task Force.

10 Conclusion

We have shown how to compress packet headers sent over slow-speed lossy links such as wireless. Our methods enable efficient use of scarce bandwidth and help to decrease end-to-end delay for applications that use digital audio, for example, as using small packets becomes feasible.

We use a method we call *compression slow-start*, where full headers are sent repeatedly with exponentially increasing intervals when the compression state changes, to achieve high compression rates and to recover rapidly from inconsistent compression state between senders and receivers due to lost headers. We rely on the soft state method of periodic refreshes and timer-based garbage collection to allow efficient resource management when mobile computers enter and leave a wireless link, to allow establishment of compression state without extra messages, to support compression on simplex links, and compression of multicast packets.

Acknowledgments

We would like to thank Björn Nordgren for helpful discussions and for implementing the prototype.

References

[1] Mary G. Baker, Xinhua Zhao, Stuart Cheshire, Jonathan Stone: *Supporting Mobility in MosquitoNet.* Proc. 1996 USENIX Technical Conference, San Diego, CA, January 1996.

[2] David D. Clark: *The Design Philosophy of the DARPA Internet Protocols.* Proc. SIGCOMM '88, Computer Communication Review Vol. 18, No. 4, August, 1988, pp. 106–114. Also in Computer Communication Review Vol. 25, No. 1, January, 1995, pp. 102–111.

[3] Steve Deering: *Host Extensions for IP Multicasting.* Request For Comment 1112, August, 1989. `ftp://ds.internic.net/rfc/rfc1112.{ps,txt}`

[4] Steve Deering, Robert Hinden: *Internet Protocol, Version 6 (IPv6) Specification.* Request For Comment 1883, December, 1995. `ftp://ds.internic.net/rfc/rfc1883.txt`

[5] Mikael Degermark, Björn Nordgren, Stephen Pink: *Header Compression for IPv6.* Internet Engineering Task Force, Internet Draft (work in progress), February, 1996. `draft-degermark-ipv6-hc-00.txt`

[6] Van Jacobson: *Compressing TCP/IP Headers for Low-Speed Serial Links.* Request For Comment 1144, February, 1990. `ftp://ds.internic.net/rfc/rfc1144.{ps,txt}`

[7] A. Mathur, M. Lewis: *Compressing IPX Headers Over WAN Media (CIPX).* Request For Comment 1553, December, 1993. `ftp://ds.internic.net/rfc/rfc1553.txt`

[8] Charlie Perkins: *IP Mobility Support.* Internet Engineering Task Force, Internet Draft (work in progress), July 8, 1995.

[9] Charlie Perkins, David B. Johnson: *Mobility Support in IPv6.* Internet Engineering Task Force, Internet Draft (work in progress), July 8, 1995.

[10] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala: *RSVP: A New Resource ReSerVation Protocol.* IEEE Network Magazine, pp. 8-18, September, 1993.