Mobile Gateway

Mikael Degermark*, Björn Nordgren*
Internet Research Institute
HP Laboratories Bristol
HPL-IRI-98-004
September, 1998

IP performance, TCP, UDP, mobile gateway, network performance, wireless network, header compression, snoop protocol, fast handoff

We present an architecture that addresses a number of issues that would otherwise severely hamper network performance in a wireless network. By combining a number of techniques that improve IP performance, on low bandwidth links, both throughput and bandwidth utilization are increased. Special attention has been devoted to ensure good performance of TCP but other protocols such as UDP will also benefit from deployment of the Mobile Gateway techniques.

Internal Accession Date Only

*Swedish Institute of Computer Science, Box 1263, SE-164 29 Kista, Sweden

1 Introduction

Wireless links are becoming more popular as a means of accessing corporate Intranets and the Internet. As computers get smaller, portable computers are becoming more popular which is fueling a demand for users to be able to access a network from more locations than just their desktop. Wireless technology is also becoming a promising way to network various appliances. The recent release of new radio spectrum at 5GHz by the FCC for unlicensed in-building use is both a response to these demands and a fuel for the rapid development and deployment of high speed wireless links. In addition to providing convenient Intranet access, such links could very well replace existing telephony infrastructure in many corporate environments. Wireless links have different characteristics from wired links. These differences can cause a serious loss of performance when a protocol, for example TCP, which was designed to run over fast, relatively reliable links, are run over a wireless link.

2 Mobile Gateway

The project combines several methods for improving performance when transmitting TCP/IP traffic over a wireless network with frequent movement between base stations. The methods originate from several researchers [1, 2, 5], but have never been combined before. Together they improve wireless performance for both bulk transfers and audio/video traffic in the presence of bit-errors on the wireless link and frequent handovers between base stations.

The means for achieving mobility at the network protocol level is Mobile IP. In conjunction with using Mobile IP, we have combined

- header compression - which reduces the header size and thus the overhead and packet-error rate.
- the snoop protocol - a scheme for local retransmissions that snoops into TCP headers to deduce the state of TCP connections and thus can make better decisions on what and when to retransmit.
- fast handoff - a mechanism that avoids exposing local movements of a mobile host. This avoids wide-area signaling and reduces processing at corresponding hosts.

These are combined such that the performance of header compression and the snoop protocol are maintained across handoffs.

3 Mobile IP

Mobile IP is a routing protocol operating at the IP level (level-3). It allows mobile hosts to retain their IP address when changing their point-of-attachment to the Internet. That would otherwise be impossible because each IP address encodes a specific point-of-attachment. Two important consequences of using Mobile IP are that a mobile host, MH, can be reached with its usual IP address regardless of where it happens to be attached to the Internet, and that sessions involving the mobile host can be maintained across movements. The latter is beneficial for mobile hosts that use wireless technologies and during movement may attach to several different subnets.

There are Mobile IP specifications for both IPv4 and IPv6. Unless stated otherwise, the short description given here is for IPv4. Mobile IP for IPv6 is simpler in several ways because IPv6 is a newer protocol with more functionality, where mobility was a goal from the start.

Mobile IP introduces two new architectural entities, the *home agent, HA,* and the *foreign agent, FA.* The home agent is attached to the home network of the mobile host, i.e., the network where packets sent to its *home address* end up. When the mobile host is away from its home network, the home agent captures packets sent to the home address and *tunnels* them to where the mobile host happens to be at the time.

Tunneling means that the original packet is encapsulated with an additional IP header whose destination address is the end-point of the tunnel. At the tunnel end-point, the extra header is stripped off and the packet is forwarded to the mobile host.

A *foreign agent, FA,* serves as the tunnel end-point in Mobile IP. In addition to decapsulating tunneled packets, it allows the MH to obtain a *care-of address* which belongs to the foreign network, i.e., packets with that address are routed there. The MH contacts the HA and registers the care-of address with it. It is the care-of address that serves as the tunnel destination address. In addition, the FA may participate in handovers to a new FA by forwarding packets that arrive after the MH has left. This requires that the MH informs its old FA of its new care-of address.

To summarize, a packet sent to the mobile host will be routed to the home agent which tunnels it to the foreign agent which forwards it to the mobile host. This detour can be avoided by a technique called *route optimization.* Route optimization requires that the *corresponding host, CH,* the host that sends packets to the MH, has a binding cache in which it can remember bindings between home addresses and care-of addresses. The MH can then inform the CH of its care-of address and the CH can tunnel packets directly to the FA without involving the HA. With route optimization, the MH needs to inform its HA, the old FA, and all its CHs of its new care-of address whenever it moves.

There are really two kinds of users of mobile IP: those who are attached to the wired network and those who use a wireless link layer. For the former user mobile IP works well, but for the latter a number of challenges arise that are not explicitly addressed by mobile IP.

The bandwidth limitations of some wireless links are causing problems. This is further accentuated by the larger headers imposed by mobile IP. Consider the case when two mobile nodes are sending packets to each other, for example during a phone conversation, and route optimization is being used. The outgoing packets going over the wireless links will all have an extra IP header (IPv4) or routing header (IPv6). The header is likely to bel larger than the actual audio data in the payload, an obvious waste of bandwidth. For IPv4 there is a standardized way to reduce the inner header of an encapsulated IP header from 20 bytes to 8-12 bytes [9], that can reduce the overhead somewhat on the wireless link and over the wired Internet. Although this technique reduces the header from 48 bytes to 32-40 bytes, the large headers are still a problem.

Wireless user roam more than their wired counterparts. This becomes a problem when you realize that all movement is exposed to the home agent.[1] Apart from causing excess control traffic over the Internet the potentially large distances between home agents and foreign agents produces latency and packet loss during the handoff phase.

Wireless media typically also has bit-error rates orders of magnitude higher than wired media. While this is not caused by mobile IP it will noticeably degrade the performance of TCP, as TCP was designed under the assumption that packet loss is caused by congestion and will reduce its sending rate whenever it detects that a packet is lost.

4 Header Compression

To battle the ill effects of increasing headers and limited bandwidth we deploy a method known as header compression, originally invented by Van Jacobson [7]. Header compression is described in [6] and [5]. The major benefit with our scheme compared to Van Jacobsons is that his scheme handled only TCP/IPv4 whereas our copes with both TCP and non TCP traffic, most notably UDP. Our scheme also handles all kinds of tunneling[2].

Header compression is different from general compression algorithms such as Lempel Ziv compression. The header compression algorithm makes use of detailed knowledge of how

[1] There is work within IETF that addresses this problem but to the best of our knowledge there is nothing standardized yet.

[2] We are by no means trying to indicate that Van Jacobson has made a mistake. The reason he did not include support for other things than TCP/IPv4 in his scheme is merely that there were no incentive for doing so at that point in time.

the headers are composed and how the different header fields change between successive headers. The key observation that allows efficient header compression is that in a *packet stream*, most fields are identical in headers of consecutive packets. For example, figure 1 show a UDP/IPv6 header with the fields expected to stay the same colored grey. As a first approximation, you may think of a packet stream as all packets sent from a particular source address and port to a particular destination address and port using the same transport protocol.

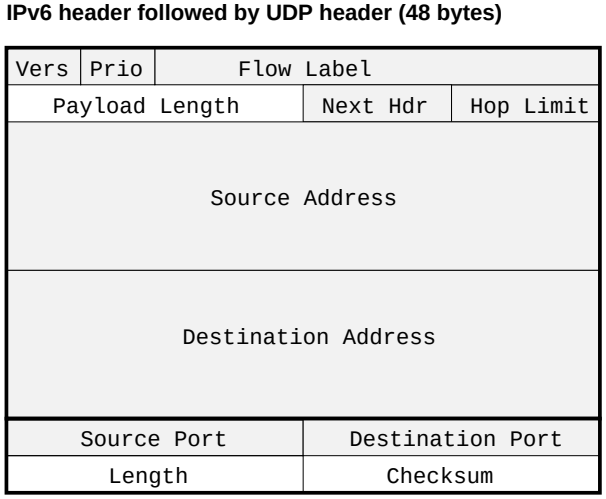

Figure 1: Unchanging fields of UDP/IPv6 packet.

With this definition of packet stream, in figure 1 addresses and port numbers will clearly be the same in all packets belonging to the same stream. The IP version is 6 for IPv6 and the Next Hdr field will have the value representing UDP. If the Flow Label field is nonzero, the Prio field should by specification not change frequently. If the Flow Label field is zero, it is *possible* for the Prio field to change frequently, but if it does, the definition of what a packet stream is can be changed slightly so that packets with different values of the Prio field belong to different packet streams. The Hop Limit field is initialized to a fixed value at the sender and is decremented by one by each router forwarding the packet. Because packets usually follow the same path through the network, the value of the field will change only when routes change.

The Payload length and Length fields give the size of the packet in bytes. Those fields are not really needed since that information can be deduced from the size of the link-level frame carrying a packet, provided there is no padding of that frame.

The only remaining field is the UDP checksum. It covers the payload and the pseudo header, the latter consisting of the Nxt Hdr field, the addresses, the port numbers and

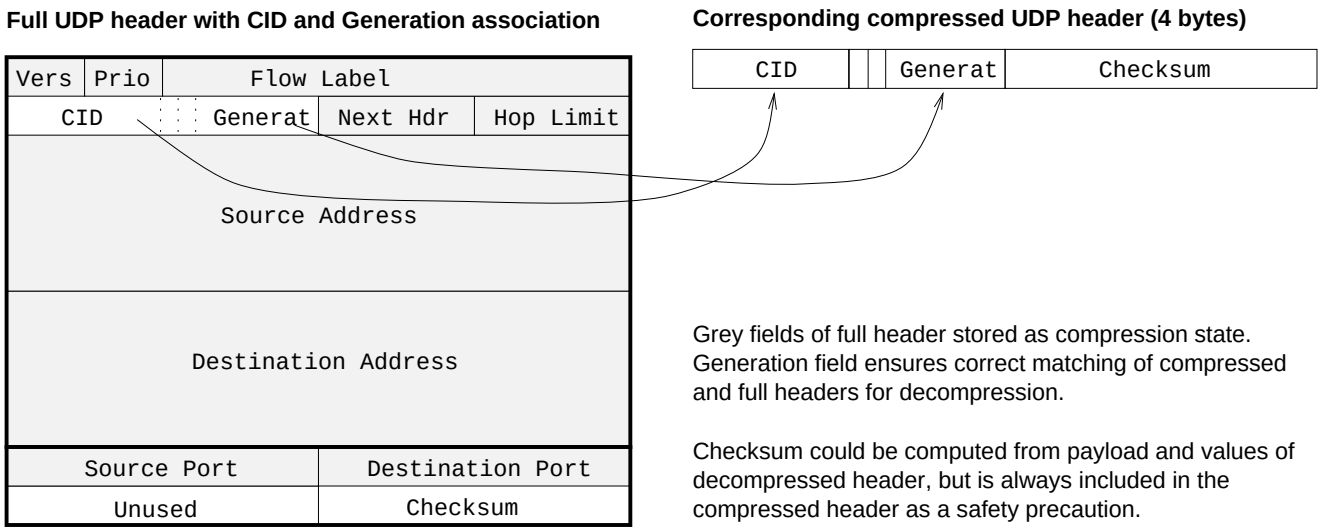

Figure 2: Full and compressed headers.

the UDP Length. Because the checksum field is computed from the payload, it will change from packet to packet.

To compress the headers of a packet stream a compressor sends a packet with a *full header*, essentially a regular header establishing an association between the non-changing fields of the header and a *compression identifier*, CID, a small unique number also carried by compressed headers. The full header is stored as *compression state* by the decompressor. The CIDs in compressed headers are used to lookup the appropriate compression state to use for decompression. In a sense, all fields in the compression state are replaced by the CID. Figure 2 shows full and compressed headers. The size of a packet might be optimized for the MTU[3] of the link, to avoid increasing the packet size for full headers, the CID is carried in length fields. Full UDP headers also contain a generation field used for detection of obsolete compression state (see section 4.1).

All fields in headers can be classified into one of the following four categories depending on how they are expected to change between consecutive headers in a packet stream. [6] provides such classifications for IPv6 basic and extension headers, IPv4, TCP, and UDP headers.

nochange The field is not expected to change. Any change means that a full header must be sent to update the compression state.

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 need not be included in

[3]Maximum Transmission Unit, maximum size of packets transmitted over the link.

compressed headers.

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 used for fields in TCP headers only.

random The field is included *as-is* in compressed headers, usually because it changes unpredictably.

Because a full header must be sent whenever there is a change in **nochange** fields, it is essential that packets are grouped into packet streams such that changes occur seldomly within each packet stream.

The compression method outlined above would work very well in the ideal case of a lossless link. In the real world bit-errors will result in lost packets and the loss of a full header can cause inconsistent compression state at compressor and decompressor, resulting in *incorrect decompression*, expanding headers to be different than they were before compressing them. A header compression method needs mechanisms to avoid incorrect decompression due to inconsistent compression state and it needs to update the compression state if it should become inconsistent. Our scheme use different mechanisms for UDP and TCP, covered in sections 4.1 and 4.2.

If header compression would result in significantly increased loss rates, the gains from the reduced header size could be less than the reduced throughput due to loss. All in all, header compression would then decrease throughput. In the following, we show how this can be avoided and the potential gain from header compression can be realized even over lossy links.

4.1 UDP header compression

For UDP packet streams the compressor will send full headers periodically to refresh the compression state. If not refreshed, the compression state is garbage collected away. This is an application of the *soft state* principle introduced by Clark [3] and used for example in the RSVP [13] resource reservation setup protocol, and the PIM [4] multicast routing protocol.

The periodic refreshes of soft state provide the following advantages.

- If the first full header is lost, the decompressor can install proper compression state when a refreshing header arrives. This is also true when there is a change in a **nochange** field and the resulting full header is lost.

- When a decompressor is temporarily disconnected from the compressor, a common situation for wireless, it can install proper compression state when the connection is resumed and a refresh header arrives.
- In multicast groups, periodic refreshes allow new receivers to install compression state without explicit communication with the compressor.
- The scheme can be used over simplex links as no upstream messages are necessary.

4.1.1 Header Generations

We do not use incremental encoding of any header fields that can be present in the header of a UDP packet. This means that loss of a compressed header will not invalidate the compression state. It is only loss of a full header that would change the compression state that can result in inconsistent compression state and incorrect decompression.

To avoid such incorrect decompression, each version of the compression state is associated with a *generation*, represented by a small number, carried by full headers that install or refresh that compression state and in headers that were compressed using it. Whenever the compression state changes, the generation number is incremented. This allows a decompressor to detect when its compression state is out of date by comparing its generation to the generation in compressed headers. When the compression state is out of date, the decompressor may drop or store packets until a full header installs proper compression state.

4.1.2 Compression Slow-Start

To avoid long periods of packet discard when full headers are lost, the refresh interval should be short. To get high compression rates, however, the refresh interval should be long. We use a new mechanism we call *compression slow-start* to achieve both these goals. The compressor starts with a very short interval between full headers, one packet with a compressed header, when compression begins and when a header changes. The refresh interval is then exponentially increased in size with each refresh until the steady state refresh period is reached. Figure 3 illustrates the slow-start mechanism, tall lines represents 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 when the full header that updates the compression state after a change is lost in an error burst of x packets, at most $x - 1$ packets are discarded or stored temporarily due to obsolete compression state.

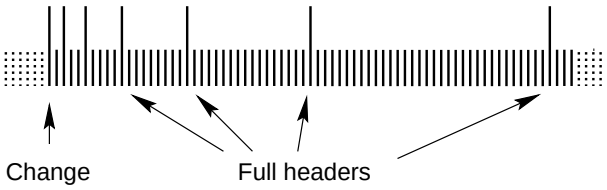

Figure 3: Compression slow-start after header change. All refresh headers carry the same generation number.

With the slow-start mechanism, choosing the interval between header refreshes becomes a tradeoff between the desired compression rate and how long it is acceptable to wait before packets start coming through after joining a multicast group or coming out from a radio shadow. We propose a time limit of at most 5 seconds between full headers and a maximum number of 256 compressed headers between full headers. These limits are approximately equal when packets are 20 ms apart.

4.1.3 Soft-state

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 4 shows the

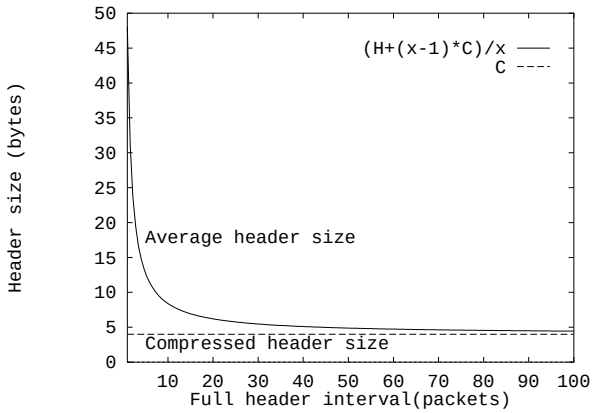

Figure 4: Average header size. $H = 48$, $C = 4$.

average header size when full headers of size H are 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 4 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.

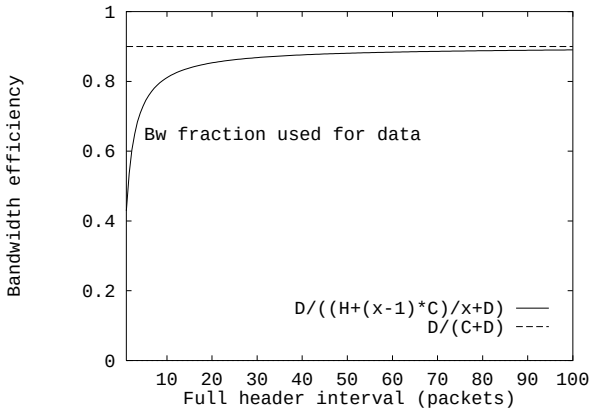

Figure 5: Bandwidth Efficiency. $H = 48$, $C = 4$, $D = 36$.

Figure 5 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 4 and 5 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 on multi-access links.

4.1.4 Error-free compression state

Header compression 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 bit-errors occur in a full header, a 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.

If the link-level framing protocol uses a strong checksum, this will never happen because frames with bit-errors will be discarded before reaching the decompressor. However, some framing protocols, for example SLIP [10], lack strong checksums. PPP[11] has a strong checksum if HDLC-like framing [12] is used, but that is not required.

IPv6 must not be operated over links that can deliver a significant fraction of corrupted packets. This means that when IPv6 is run over a lossy wireless link the link layer must have a strong checksum or error correction. Thus, the rest of this discussion about how to protect against bit-errors in the compression state is not applicable to IPv6. These mechanisms are justified only when used for protocols where a significant fraction of corrupted packets can be delivered to the compressor.

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. The compressor extends the UDP checksum to cover the whole full header rather than just the *pseudo-header* since the pseudo-header doesn't cover all the fields in the IP header. The decompressor then performs 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 restores the original UDP checksum before passing the packet up to IP.

Once the compression state is installed, there will be no extra packet losses with UDP header compression. 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, there will be no packet loss related 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.

4.1.5 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 will 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.

Figure 6 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

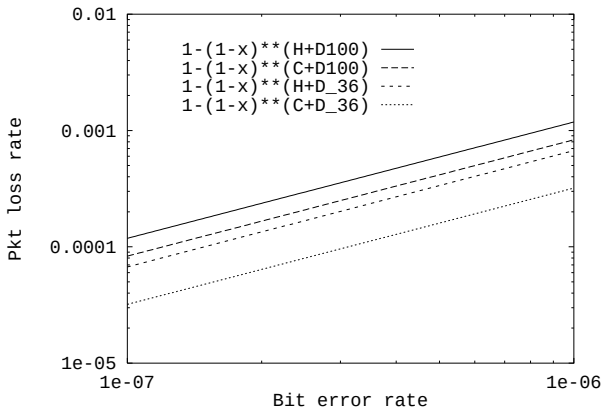

Figure 6: Packet loss rate as a function of bit-error rate, with and without header compression and for payloads of 36 and 100 bytes.

bytes, full and regular headers, H, are 48 bytes (IPv6/UDP). D is the size of the payload.

Thus, our header compression scheme for UDP/IP in addition to decreasing the required header bandwidth, also reduces the rate of packet loss. 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%. With tunneling, the packet loss rate decreases by 68% and 45%, respectively.

If bit-errors occur in bursts whose length is of the same order as the packet size, there will be little or no improvement in the packet loss frequency because of header compression. The numbers above assume uniformly distributed bit-errors.

4.2 TCP header compression

The currently used header compression method for TCP/IPv4 is by Jacobson [7], and is known as VJ header compression. Jacobson carefully analyzes how the various fields in the TCP header change between consecutive packets in a TCP connection. Utilizing this knowledge, his method can reduce the size of a TCP/IPv4 header to 3–6 bytes.

It is straightforward to extend VJ header compression to TCP/IPv6. It is important to do this since not only are the base headers in IPv6 larger than IPv4, multiple headers needed to support Mobile IPv6[8], i.e., routing headers with 16 byte addresses tunneled to the mobile host, will produce a large overhead on wireless networks.

4.2.1 Compression of TCP header

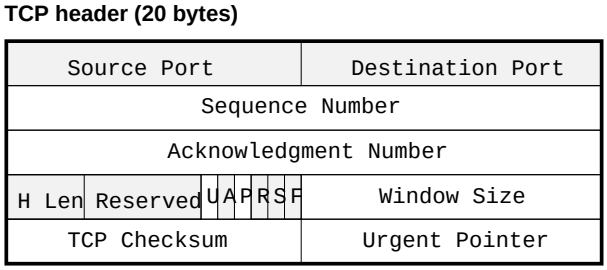

Figure 7: TCP header. Grey fields usually do not change.

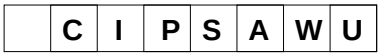

Figure 8: Flag byte of compressed TCP header.

Most fields in the TCP header are transmitted as the difference from the previous header. The changes are usually by small positive numbers and the difference can be represented using fewer bits than the absolute value. Differences of 1-255 are represented by one byte and differences of 0 or 256-65535 are represented by three bytes.

A flag byte, see figure 8, encodes the fields that have changed. Thus no values need to be transmitted for fields that do not change. The S, A, and W bits of the flag byte corresponds to the Sequence Number, Acknowledgment Number, and Window Size fields of the TCP header. The I bit is associated with an identification field in the IPv4 header, encoded in the same way as the previously mentioned fields. The U and P bits in the flag byte are copies of the U and P flags in the TCP header. The Urgent Pointer field is transmitted only when the U bit is set. Finally, the C bit allows the 8-bit CID to be compressed away when several consecutive packets belong to the same TCP connection. If the C bit is zero, the CID is the same as on the previous packet. The TCP checksum is transmitted unmodified.

VJ header compression recognizes two special cases that are very common for the data stream of bulk data transfers and interactive remote login sessions, respectively. Using special encodings of the flag byte, the resulting compressed header is then four bytes, one byte for the flag byte, one byte of the CID, and the two byte TCP checksum.

4.2.2 Updating TCP compression state

VJ header compression uses a differential encoding technique called *delta encoding* which means that differences in the fields are sent rather than the fields themselves. Using delta encoding implies that the compression state stored in the decompressor changes for each header. When a header is lost, the compression state of the decompressor is not incremented properly and the compressor and decompressor will have inconsistent state. This is different from UDP where loss of compressed headers do not make the state inconsistent. Inconsistent compression state for TCP/IP streams will result in a situation where sequence numbers and/or acknowledgment numbers of decompressed headers are off by some number k, typically the size of the missing segment. The TCP receiver (sender) will compute the TCP checksum which reliably detects such errors and the segment (acknowledgment) will be discarded by the TCP receiver (sender).

TCP receivers do not send acknowledgments for discarded segments, and TCP senders do not use discarded acknowledgments, so the TCP sender will eventually get a timeout signal and retransmit. The compressor peeks into TCP segments and acknowledgments and detects when TCP retransmits, and then sends a full header. The full header updates the compression state at the decompressor and subsequent headers are decompressed correctly.

The problem with this scenario is that it completely disables TCPs fast repair mechanism and forces TCP into slow start every time a packet is lost. Unless this problem is properly dealt with header compression will actually lower the throughput over a link with high bit error rate. The methods used by the mobile gateway to re-synchronize the compression state are known as "the twice mechanism" and "header request", both of which are explained in great detail in [5].

5 Snoop

To improve TCP performance over the wireless link we use the Snoop protocol [1]. Snoop is an alternative to using a link-layer protocol that deals with packet loss by local retransmission and acknowledgments. The Snoop protocol is a protocol booster for TCP, it enhances TCP performance but does not change the semantics of TCP. Snoop understands some of the semantics of TCP and tries to avoid triggering TCP congestion control mechanisms that would otherwise reduce TCP throughput.

Snoop buffers TCP segments passing by and performs local retransmissions whenever it detects that a TCP segment has been lost on the wireless link. Provided the link RTT is small compared to the end-to-end RTT, this avoids triggering TCPs end-to-end retrans-

mission policies. Snoop also filters duplicate acknowledgments to avoid triggering TCPs fast repair mechanisms when the segment was lost was over the wireless link and is being retransmitted locally. Snoop avoids transmitting a TCP segment multiple times, when the segment has reached the mobile node. The latter two functions are not performed by a naive link retransmission protocol. For some scenarios they will improve the performance significantly compared to such protocols.

6 Mobile IP hierarchy

As mentioned earlier, mobile IP does not behave well when mobile hosts roam extensively. Among the first to draw attention to this shortcoming was the authors of [2]. In their article they propose a hierarchical structure for supporting movement of the mobile host. The purpose of the hierarchy is as indicated above to hide local movement from the higher levels of the hierarchy. This results in faster handoffs and less signaling over the Internet.

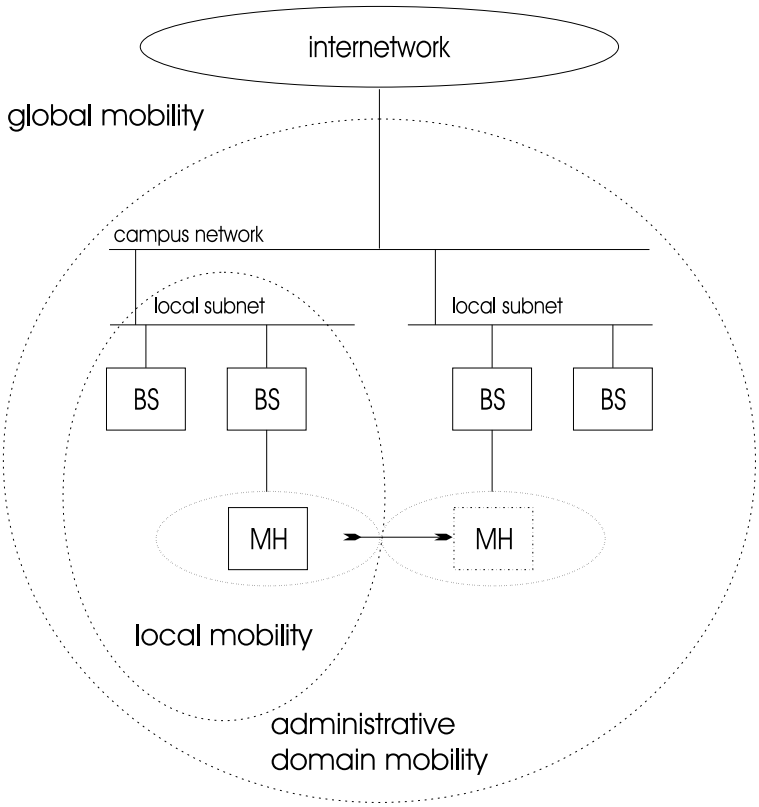

Figure 9: The different levels of the Mobile IP hierarchy.

The hierarchy has three levels:

- Local mobility, supports movement between base stations attached to the same subnet.
- Administrative domain mobility, supports movement between base stations within the same administrative domain
- Global mobility, supports movement between different administrative domains

In the mobile gateway project we have implemented the lowest level, the local handoff, of this hierarchy. This is sufficient to verify that our augmented handoff mechanisms work.

7 Handoffs revisited

This section describes the handoff that occurs when the mobile host moves between two mobile gateways that are on the same local area network. The local handoff is designed to minimize the service disruption the mobile user experiences during a handoff while maintaining the performance of header compression and the snoop protocol. This is an augmented version of the handoff described in [2]. The scenario is depicted in figure 10. Figure 11 shows the messages exchanged during the handoff. Each base station periodically sends beacons on its wireless interface. The mobile host uses the beacons to determine out which BS that is best to use as gateway. The decision to switch to a new BS is made solely by the mobile host and is based on a metric provided by the beacon. Apart from giving the mobile host a measure of the performance of the connection to the BS the beacon also contains the IP and MAC addresses of the wireless interface of the sending BS.

Once the mobile host has decided to switch to a new base station the following message exchange occur:

- The MH sends a *Greet* message to the new BS. The *Greet* message contains the IP address of the MH, the MAC (Ethernet) address of the corresponding interface and the IP address of the wired interface of the old BS. It also makes the new BS it default gateway. If header compression is in use the MH migrates its compression state, see section 8. Furthermore the *Greet* message tells the new BS if the MH wants to use header compression and if so the size of the compression tables.
- The new BS creates a routing table entry for the MH so that packets are be forwarded to the MH. It also responds to the *Greet* message by sending a *Greet Ack* message to the MH.

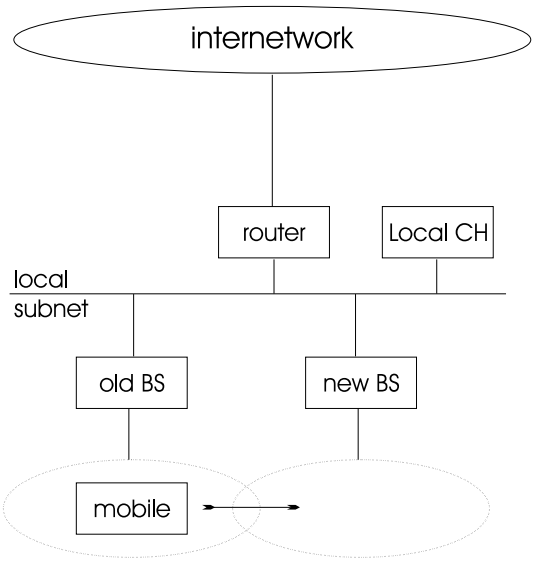

Figure 10: The mobile host moves from the old BS to the new BS

- The new BS sends a *Notify* message to the old BS. This message informs the old BS that it is relieved of its packet forwarding duties for MH. It also conveys the wired IP address of the new BS.
- The old BS removes its routing table entry for MH. It then sends the compression state tables associated with MH to the new BS. The old BS then sends a *Notify Ack* message. The last thing sent by the old BS to the new BS is the contents of its snoop buffers (for TCP), if any, and the contents of its retransmission buffers if any.
- When the new BS has received the compression state tables it sends a *Comp Ack* message to the MH.
- The new BS then broadcasts a redirect message on the local subnet so that other nodes are notified of the the whereabouts of the MH.

8 Migrating compression state

In order to save bandwidth on the wireless media the mobile gateway will migrate compression state during a handoff. The purpose of this is to avoid sending many full headers between the MH and the new BS immediately after a handoff.

The delta encoding of the compressed TCP headers makes it difficult to compute the

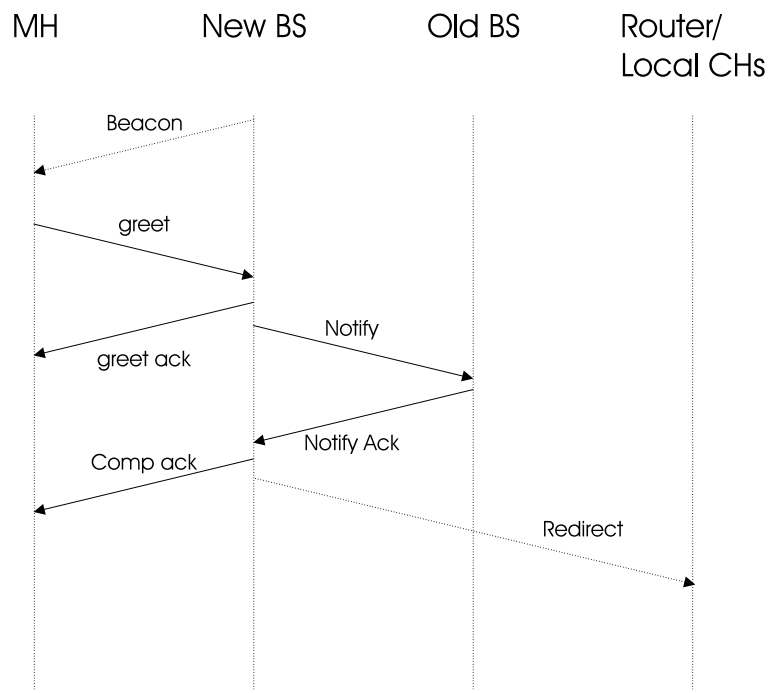

Figure 11: Messages exchanged during handoff.

correct compression state after a handoff, since any number of packets could have been lost. As compression state for TCP needs only one full header to synchronize there is little point in migrating compression state for TCP.

For UDP on the other hand there is a strong case for migrating compression state. The UDP headers change infrequently and the compressed headers are not delta encoded. There is also a larger penalty involved here since each UDP flow would trigger a *compression slow-start*, see section 4.1.2. It is important to note that the state of the *slow-start* needs to be migrated as well as the actual compression state.

When doing header compression over point-to-point links the compression state tables are associated with an interface. However, when doing header compression over shared media there has to be separate compression state tables for each peer on the shared media. In other words, both the interface and the MAC address of the peer are needed to index the compression state tables.

When migrating compression state the MH takes the UDP compression state tables associated with the old BS and makes them UDP compression state tables for the new BS. It then clears the TCP compression state tables for the new BS. The MH performs these actions after sending the *Greet* message. The new BS will receive the necessary UDP compression state tables from the old BS in conjunction with the *Notify Ack*. It also clears the

TCP compression state tables. The MH should refrain from sending compressed headers using the newly migrated compression state tables until receiving a *Comp Ack* message indicating that the new BS has the corresponding tables installed. The old compression state tables will time out and be available for garbage collection.

9 Memory requirements and CPU usage

Nothing is free, at least not protocol boosters. The tribute that has to be payed in this case is increased cpu usage and memory consumption.

The size of the compression state table for a host using 16 compression states and a maximum header size of 128 bytes is at most

$$4 \times 16 \times 128 = 8192 \text{ bytes.}$$

The memory consumption for the Snoop booster is approximately

$$2 \times \text{RTT} \times \text{bandwidth.}$$

Where RTT is the round trip time on the wireless link. On a wireless lan where the RTT is 5 ms and bandwidth is 2 Mbit/s the memory consumption is approximately 2500 bytes per active TCP connection. On a 28.8 modem with RTT equal to 120 ms we would have 432 bytes per active TCP connection.

Assuming a buffer of four[4] packets of size 100 bytes each for UDP packet retransmissions during handoff gives us 400 bytes per active UDP connection. A busy mobile host using header compression that has ten active TCPs and ten active UDP connections would then use about 37K bytes of memory on the BS. The memory usage on the MH is limited to the 8K bytes required for the compression state table.

Currently we have no reliable measurements of how much cpu power is required for each booster. However, our implementation has shown no problems doing header compression at full speed over a WaveLAN using a 90Mhz Pentium running NetBSD.

[4]Four packets is the recommended buffer size according to [2].

10 Implementation issues and status

It may appear to be an overwhelming amount of processing that has to be performed within a BS before actually sending a packet to the MH. But the situation is not as severe as it may appear at first. The key observation here is that you only have to parse each packet header once. After parsing the packet header you will know which compression state it matches, if any, and you will also know whether to put it in the snoop buffer (TCP) or if it should go into the retransmission buffer (UDP). Thus all three protocol boosters can be implemented using a single lookup/parsing.

When implementing the mobile gateway we have discovered a serious flaw in BSD networking code. It turns out that later versions of BSD for various reasons has integrated the ARP lookup into routing lookup routines. Albeit beneficial in many ways this approach disables BSD from correctly handling situations where one IP address may have different MAC addresses depending on the interface it is accessed through. The BS needs to speak to the mobile host on one shared media interface (the wireless link) and proxy-arp for it on another (the local subnet). This means that the ARP lookup must return different MAC addresses for different interfaces. The most common case for proxy-arping is when the machine you are proxy-arping for is on a point-to-point interface, typically PPP. This case is handled gracefully by BSD since point-to-point links does not have MAC addresses and thus there is only one ARP entry for that particular IP address. A workaround that temporarily solves the problem is to have another host on your local area network doing the proxy-arping on behalf of the mobile gateway. The mobile gateway then controls the proxy-arping host by sending gratuitous arps that forces the proxy-arping host to update its arp cache contents accordingly. Although this workaround solves our problem it has some drawbacks: it adds complexity, it requires the presence of another host that is not running a mobile gateway, a dropped proxy-arp packet causes the proxy-arping host to continue to advertise the old MAC address. We are currently looking into if its reasonable to modify the BSD kernel

Our current implementation handles header compression over the wireless media and supports the augmented handoffs. The testbed consists of two machines, running NetBSD, acting as base stations and two laptops, running FreeBSD, acting as mobile hosts. We are currently performing measurements to determine and tune the performance of the mobile gateway.

11 Further work

The current implementation of header compression is based on the point-to-point specification. This is sufficient for testing purposes. However, a specification for shared media is

needed for realistic deployment of the mobile gateway. We are finalizing a draft specifying header compression over multi-access links.

12 Acronyms

BS base station — a router connected to both the wired network and the wireless network, routing packets to and from MHs.

CH corresponding host — a host communicating with the mobile host.

CID compresssion state identifier

FA foreign agent

HA home agent

IETF Internet Engineering Task Force

IP Internet Protocol

MH mobile host

MG mobile gateway — a piece of software that runs on the BS.

PPP Point-to-point Protocol

RTT round trip time

TCP Transmisssion Control Protocol

UDP User Datagram Protocol

References

[1] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz. Improving tcp/ip performance over wireless networks. In *Proceedings of First Annual International Conference on Mobile Computing and Networking (Mobi Com '95)*, pages 1–11, New York, Nov. 1995. IEEE/ACM, ACM Press.

[2] R. Caceres and V. Padmanabhan. Fast and scalable handoffs for wireless internetworks. In *Proceedings 2Nd Annual International Conference on Mobile Computing and Networking (Mobi Com '96)*, pages 56–66, New York, Nov. 1996. IEEE/ACM, ACM Press.

[3] D. D. Clark. The Design Philosophy of the DARPA Internet Protocols. *ccr*, 18(4):106–114, Aug. 1988. *Proceedings ACM SIGCOMM '88 Symposium.* Also in *Computer Communication Review*, 25(1):102–111.

[4] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei. An Architecture for Wide-Area Multicast Routing. *ccr*, 24(4):126–135, Oct. 1994. *Proceedings ACM SIGCOMM '94 Conference.*

[5] M. Degermark, M. Engan, B. Nordgren, and S. Pink. Low-loss TCP/IP header compression for wireless networks. In *Proceedings 2Nd Annual International Conference on Mobile Computing and Networking (Mobi Com '96)*, pages 1–14, New York, Nov. 1996. IEEE/ACM, ACM Press.

[6] M. Degermark, B. Nordgren, and S. Pink. Header compression for ipv6. Internet draft (work in progress), Internet Engineering Task Force, Dec. 1997.

[7] V. Jacobson. Compressing TCP/IP headers for low-speed serial links. Request for Comments (Proposed Standard) RFC 1144, Internet Engineering Task Force, Feb. 1990.

[8] C. Perkins. IP mobility support. Request for Comments (Proposed Standard) 2002, Internet Engineering Task Force, Oct. 1996.

[9] C. Perkins. Minimal encapsulation within IP. Request for Comments (Proposed Standard) 2004, Internet Engineering Task Force, Oct. 1996.

[10] J. Romkey. Nonstandard for transmission of IP datagrams over serial lines: SLIP. Request for Comments (Standard) RFC 1055, Internet Engineering Task Force, June 1988.

[11] W. Simpson. The point-to-point protocol (PPP). Request for Comments (Standard) STD 51, RFC 1661, Internet Engineering Task Force, July 1994.

[12] W. Simpson. PPP in HDLC-like framing. Request for Comments (Standard) STD 51, RFC 1662, Internet Engineering Task Force, July 1994.

[13] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A New Resource ReSerVation Protocol. *IEEE Network Magazine*, 7(5):8–18, Sept. 1993.