Improvements on IP Header Compression

Cédric Westphal
Nokia Research Center, Mountain View, CA
cedric.westphal@nokia.com

***Abstract*—Due to bandwidth constraints on the wireless link in an IP network, it is useful to compress the headers so as to maximize the utilization of the link. Different Header Compression schemes have been proposed. Some make use of the similarity in consecutive headers in a packet flow to compress these headers. These are called here *time* compression, as they acquire the compression information over the time length of a flow. Another scheme introduced recently makes use of the similarity in consecutive flows from or to a given mobile terminal to compress these headers. This we call a *space* compression, as it uses information about the distribution of flows over the destination address space. We present here a comparison of these two classes of header compression algorithm -time and space- and of a third one that combines them. The combined algorithm significantly outperforms traditional header compression algorithm. We evaluate the compression scheme with respect to actual internet data traces.**

Keywords: Header compression, entropy, information theory, encoding, frequency distribution.

I. Introduction

The purpose of IP header compression algorithm is to improve on the ratio of the overhead versus the payload for an IP packet. It is of tremendous importance since the increase of the address space when shifting to IPv6 translates into an increase of the header size. Also, the bandwidth bottleneck in the future mobile Internet is the wireless link.

Header compression (HC) algorithms have been developed for over a decade. Recently, we introduced a different scheme that uses the flow history to compress IP headers [7]. The algorithm uses the frequency of a flow to compress its headers, and is denoted here as a Frequency-Based header compression (FBHC). We recall in the next section the basic concepts of this IP header compression scheme.

While [7] offered a mathematical model to evaluate the performance of the FBHC algorithm, it was not compared to the landmark HC algorithms such as ROHC -RFC3095 [1]- and IPHC[1] -RFC2507 [2]. We fill this gap in this document. Namely, we focus on TCP traffic and thus on RFC2507 as our comparison benchmark. More importantly, we present a way of combining both traditional and new IP header compression algorithms. We see that the combination of the two classes of algorithms yields a significant improvement over either one of the algorithms.

II. Header Compression Framework

First we introduce some terminology to describe Header Compression. Denote by $p^i, i = 1, 2, \ldots$ the sequence of packets sent by some user u. We consider only sent packets to simplify the description. Received packets can be treated in a symmetric fashion.

A packet p^i is composed of an IP header and some data. The IP header is composed of several fields, such as source address (the user u's address), destination address, ports, protocol, and some transport protocol information.

The filter f of a packet is defined here as the IP 5-tuple (source IP address, source port, destination IP address, destination port, protocol). The filter function $\mathcal{F}$ is defined such that: $\mathcal{F}(p^i)$ gives the filter of the packet. Note that the definition of filter could be extended to cover other fields of the IP header.

Internet traffic is composed of microflows, which we define now. Assume a given time threshold τ:

Definition II.1: A microflow m_f is a sequence of packets with the same IP 5-tuple such that two consecutive packets are within τ units of time of each other.

IP headers in a microflow exhibit some similarities: the filter is the same from one packet to the next. Furthermore, the protocol header is highly correlated as well[2]. When a microflow crosses a bandwidth constrained link, the link layer can take advantage of this correlation by compressing the IP header.

Definition II.2: An IP Header Compression algorithm is a device to reduce bandwidth usage on a given link by replacing the IP header by a label (or compressed header) at one end of the link, transmitting the data with the label attached, then replacing the label at the other end of the link by the original (reconstructed) IP header.

HC can be described as two functions, a compressor $\mathcal{C}$ applying on $(p^1, \ldots, p^j)$ and a decompressor $\mathcal{D}$ applying on $(\mathcal{C}(p^1), \ldots, \mathcal{C}(p^j))$ such that, for packets p^j crossing link l;

$$\begin{aligned} \text{size}(\mathcal{C}(p^j)) &< \text{size}(p^j) \\ \mathcal{D}\left(\mathcal{C}(p^j|p^1, \ldots, p^{j-1})|\mathcal{C}(p^1), \ldots, \mathcal{C}(p^{j-1})\right) &= p^j \end{aligned} \quad (1)$$

We denote by $\mathcal{C}(f)$ the compressed filter. Several IP Header Compression schemes provide this functionality, mostly on bandwidth constrained wireless links. The most common schemes, the Van Jacobson (VJ) algorithm [6], RFC2507[2] and the Robust Header Compression (ROHC) algorithm [1] work on the same principles. These schemes make use of the predictable behavior of the header sequence within one microflow. Without entering into the specific technical details, the header is either Full Header (FH), First-Order Header (FO) or second order header (SO).

[1] Even though FBHC or ROHC also are IP header compressions, we use IPHC for RFC2507 as its title is 'IP Header compression'

[2] We do not consider similarities in the data attached to each packet in a microflow, we restrict ourselves to layer 3, IP header compression, and ignore application layer, data compression.

- FH corresponds to the regular transmission of all the information bits that make the IP header.
- FO corresponds to the header without the constant information (the IP 5-tuple, the constant fields in the protocol header...). Changing fields (sequence number, time stamps...) are represented entirely.
- SO corresponds to the header without the constant and the predictably incremental (so called 'delta') information. As for FO, both compressor and decompressor need to acquire first the information before switching to this mode.

These HC schemes can be described as a *compression over time*. The knowledge required to improve the compression factor is acquired over the time length of a microflow: the decompressor has to learn the compression parameters before the compressor can use a new compression state.

However, most microflows today are short lived, with small packets [4], [5]. These connections do not give enough time to the traditional HC engine to acquire the compression state and reach a compressed steady state. To quote from [9]: *Many recent studies have noted that the majority of TCP flows traveling over the wide area internet are very short, with mean sizes around 10KB and median sizes less than 10KB.* This implies most of the internet traffic today is away from IPHC's optimized application domain.

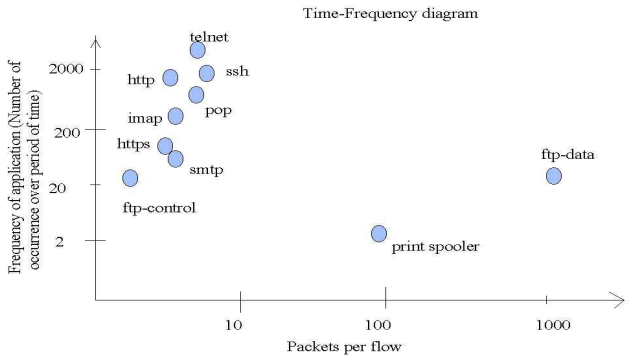


Fig. 1. Frequency (number of occurrences) vs. Time (flow length) for TCP applications

Definition II.3: The frequency of a microflow for a given user is defined as the number of microflows from this user having the same IP 5-tuple divided by the total number of microflows from this user.

In figure 1, we present the distribution of the microflows of a group of users in a time/frequency domain. The x-axis represents the length of a connection[3]. We only plot TCP flows for some IANA 'well-known' ports (see section IV for the measurement data). Longer UDP streams would be on the right-hand side of the graph. The y-axis represents how frequent a connection is with respect to the other connections. The IPHC compression schemes works better with longer connection, so that the compression engine can acquire compression states. This is represented by the area to the right of the graph. The frequency-based header compression (FBHC) introduced in [7] applies to the bulk of the microflows on the left side of the graph. In the next section, we recall the main ideas of the FBHC algorithm.

[3]connection and microflow have the same meaning here

III. FBHC Algorithm

We present now the FBHC algorithm as introduced in [7]. We define the compressor and the decompressor in this section.

A. Filter table

Denote by $\mathcal{M}$ the set of all microflows originating from user u until time t.

Definition III.1: A filter table is a table of elements of the form: (filter, filter count, time of first filter occurrence, filter rate, compressed filter). The compressed filter is also called the code word for the filter f. More precisely, each elements is of the form:

$$(f, fcount, ftime, frate, c) = (f, \Sigma_{p:p\in m, m\in\mathcal{M}} \mathbf{1}_{\{\mathcal{F}(p)=f\}}, \min_{m\in\mathcal{M}}(\mathcal{T}(p) : p \in m \text{ and } \mathcal{F}(p) = f), \frac{fcount}{t - ftime}, \mathcal{C}(f)).$$

A filter table has a finite depth, D which is the number of entries in the table. Since the table contains both f and $\mathcal{C}(f)$, maintaining such a table provides a HC function $c = \mathcal{C}(f)$, as well as the decompression function $f = \mathcal{D}(c)$. c is the compression code for f, and is a function of $frate$ and t.

To define our compression algorithm, and assuming that both $\mathcal{C}$ and $\mathcal{D}$ have the same filter tables available to them, it suffices to describe how this filter table evolves as a function of time.

Consider a filter table T_{freq} with depth D_{freq} and a filter table T_{rec} with depth D_{rec}. Intuitively, T_{freq} is assigned the task of keeping the information for the most frequent microflows, and T_{rec} for the most recent microflows. T_{rec} is ordered in a First-In-First-Out way: the entry on top of the table is the oldest one whereas the on at the bottom is the latest one.

T_{rec} and T_{freq} assign a mapping from f to c, however, they use different coding alphabets: an entry in T_{rec} cannot have the same c as an entry in T_{freq}.

B. Frequency based algorithm

The compressor maintains two tables T_{freq} and T_{rec}, one for frequent flows, and one for recent flows. The compressor receives a full packet p with filter f from user u at time t_p.

- if $\mathcal{F}(p) = f \in T_{freq}$, that is, if the f has an entry in the T_{freq} table, then the compressor:
  - replaces f with the corresponding value c in the T_{freq} table and forwards the compressed packet.
  - updates the value $fcount$ by one.
  - computes the new rates $frate$ using time t_p for all entries in the table.
  - reassigns the codes c based on the new frequency $frate$.
- otherwise, if T_{freq} is not full (has less than D_{freq} entries), then adds the entry corresponding to f in T_{freq}.
- otherwise, if $f \in T_{rec}$, that is, if the filter of p has an entry in the T_{rec} table, then the compressor:

– replaces f with the corresponding value c in the T_{rec} table and forwards the compressed packet.
– updates the value $fcount$ by one.
– computes the new rate $frate(f_i)$ using time t_p for all entries f_i in the table T_{freq} and compares it with $frate(f)$.
– if there exists some value f_j in T_{freq} such that $frate(f_j) \leq frate(f)$, then adds the entry corresponding to f in T_{freq}, removes the entry corresponding to f in T_{rec}, and removes the entry with the lowest $frate(f_i)$ from T_{freq}.

- otherwise, if T_{rec} is full, that is, if it contains D_{rec} entries, then the compressor removes the first entry in the table T_{rec}, moves up all the entries so that the second becomes first, the third second, etc. Adds the entry corresponding to f last in T_{rec}. Forwards p as is.
- otherwise, adds the entry corresponding to f as the last one in T_{rec}, and forwards p as is.

This defines both the compressor, and the decompressor, as it suffices to replace p with the compressed packet, and substitute c and f in the table update process described above. For instance, if the received code c correspond to an entry in T_{freq}, then replace c with its f to recover and forward the initial packet p, then update the frequencies, and compute the new codes.

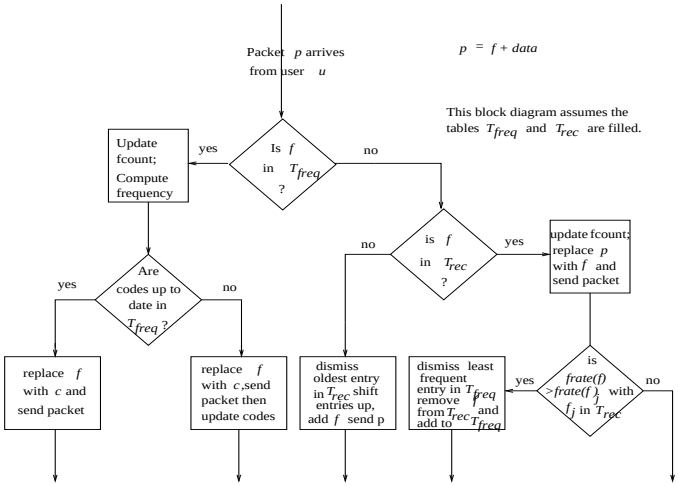


Fig. 2. Block diagram for compression/decompression algorithm

The assumption that the link is perfect ensures that both the compressor $\mathcal{C}$ and the decompressor $\mathcal{D}$ are synchronous, and that each side's copies of the T_{freq} and the T_{rec} are the same. Since the compressor and the decompressor need to be updated at the flow granularity -and not for every packet- the layer 2 or layer 4 reliability mechanisms provide built in synchronization. The loss of a packet will not affect FBHC. On the other hand, the loss of the whole flow is impossible thanks to for instance TCP mechanisms.

IV. Comparison with existing IP HC

An analytical evaluation of FBHC was conducted in [7]. We do not cover this evaluation here. Instead, we focus on a measurement study conducted on the Nokia LAN in the Mountain View campus to perform a comparison with IPHC (ROHC, the other standardized IP header compression algorithm focuses on IP/UDP/RTP, not TCP flows).

For the data collection, a measurement box was inserted between the router and the switch on one of the branches of the intranet of Nokia's campus in Mountain View. All the TCP data flowing in and out 25 users on the branch was logged during 30 non-consecutive days between April 22nd and June 20th 2002.

A. Collected data

In this section, we consider some general quantities based on the collected data.

FBHC keeps some flow level statistics. As stated in II.1, a flow is a sequence of packets with the same source, destination addresses, port numbers and such that the time between two consecutive packets does not exceed a threshold. Typically, IPHC considers a (default) value F_MAX_TIME $= \tau = 5s$ between packets. We consider several thresholds. Whenever comparing with IPHC, we will consider IPHC and FBHC with the same τ thresholds of course.

Different thresholds are considered to see the sensitivity of the FBHC algorithm to the flow definition. IPHC depends on the threshold as the shorter it is, the more likely it is that a flow terminates, and that it has to be re-initiated upon resumption of the flow. This re-establishment cost that has been studied in another document [8].

Figure 3 depicts the dependency of the number of flows with respect to τ. The longer τ, the less flows there are and the more packets in a flow. At the 5s threshold used in IPHC, the average number of packets per flow is 30. An easy way to increase IPHC performance is to increase the value of the threshold τ. We show later (in Figure 6) that the gain in IPHC performance is not too significant. However, this increases at the same time the complexity of the flow state management. In some cellular networks, the node that decompresses the headers can handle several thousands to several million flows concurrently.

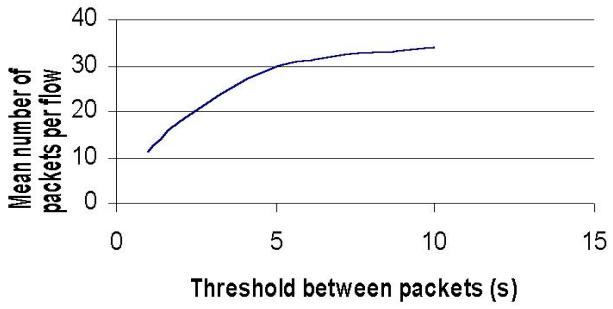


Fig. 3. Dependency of the number of packet vs. flow threshold τ

1) Ports: Another issue of importance is to characterize the traffic per application.

As we deal with per flow HC later in the document, we are interested to see how flows vary depending on the application. Ports that carry the largest number of flows are 23, 20 and 80. We plot in figure 4 the number of packets per flow per application. The application are mapped to a port, and we only picked

the IANA Well Known Ports (ports 1 to 1024) with the most traffic (the other Well Known Ports only saw some marginal traffic).

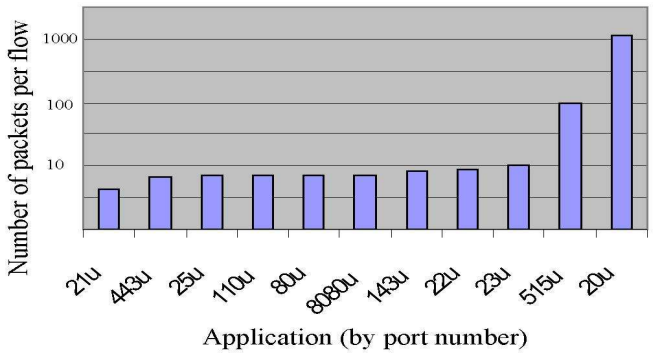


Fig. 4. Number of packets per flow per port

As we can see, most of the ports have around or less than 10 packets per flow, except for telnet (port 20) which has 1200 packets per flow uplink and 750 packets per flow downlink, and for the printer spooler (port 515) which has 50 packets downlink and 100 packets uplink.

The main conclusion we draw is that, except for port 515 and 20, the behavior of the flow is pretty similar independtly of the application for our purpose. Ftp (port 20) and telnet (port 23) traffic are traffic created by the network topology and the distribution of the resources over the network. This traffic is not likely to be found on a mobile device any time soon. The print traffic is not (yet) to be expected either. This has two consequences:

1) for a short term perspective on *mobile users' behavior*, we will restrict the study to the ports that are likely to be used widely soon: namely 80 (http), 443 (https), 143 (imap) and 110 (pop) as web browsing and email are the two main *early* applications using tcp.
2) for a longer term perspective, we may assume that, as least in the corporate world, the traffic measurement of the mobile user will converge to mimic the current traffic patterns. Thus, results on the whole data set could be extracted. However, these results would entail many idiosyncracies of the specific network studied.

In the sequel of this document, we focus on the shorter term perspective, and we consider ports 80, 443, 110 and 143.

B. FBHC algorithm performance on the measured data

We study here the performance gain achieved by the FBHC algorithm on the collected data. To do this, we run the FBHC for each of the 25 users that were monitored. We distinguish in between uplink traffic and downlink traffic for IPHC, as the properties of each direction are quite different usually. However, we do not need this distinction in FBHC, as the frequency of the downlink is comparable to that of the uplink (an uplink flow does translate into a downlink flow for TCP traffic).

In figure 5, we plot the compression achieved for the 25 users with the FBHC algorithm, where the constant fields are removed, the source address is reversed looked up from the MAC layer, and the destination address is compressed using the FBHC tables.

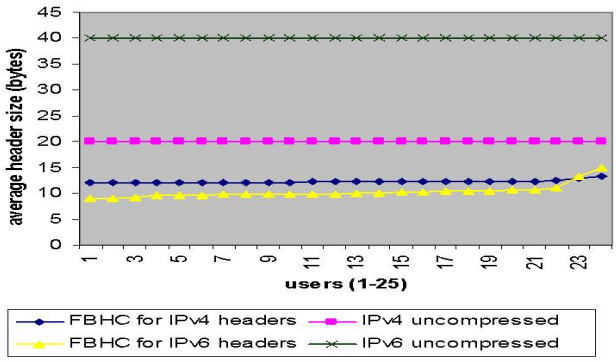


Fig. 5. FBHC compression achieved per user

We also extrapolated the compression achieved if the users were using IPv6 with the same traffic patterns. Figure 5 refers to the whole traffic seen by each user independently of the application. The destination address is compressed on 1 to 1.4 bytes for all but 2 users (1.9 and 2.2 respectively). The mean compression is 1.2 bytes. Recall that 1 byte is the minimal compressed value per our assumption. This translates into a compressed IP header between 12 and 12.4 for all but two users, and 12.2 on average.

IPv6 performs better: the compressed addresses fit on 3 bytes for all but these two users, (again the minimal value is 1 byte) and the average value is about 2 bytes. This translate into an IPv6 address encoded onto 10 bytes on average.

C. Comparison with traditional HC

In this section, we see how FBHC fares when compared with IPHC. The figure 6 we plot is that of IPHC on the whole data we gathered. As we said earlier, this is not very relevant to the task at hand: the structure of the network induces elongated flows that are not likely to be reproduced in a wireless network at least in the short to medium term.

The IPHC performance is evaluated in a very simple manner: the size of the packet depends on the IPHC state. The state is changed deterministically: the number of packets needed to establish the next state is a variable parameter: we considered 2, 3 and 4 packets. 3 is the common value taken for the unidirectional mode, that is 3 is the number of packets sent by the compressor for the compressor to assume that the decompressor has acquired the state. After 2, 3 or 4 FH packets, the compressor sends FO compressed headers. Then, after the same number of FO packets, the compressor moves into the SO state, and sends SO compressed headers. In a reliable mode, the compressor waits for the acknowledgement of the decompressor, and the number of packets sent depends on the round trip time between the compressor and the decompressor. Typically, the number of packets sent would be higher. As we compare TCP flows, the IPHC compression applies only to the fixed parameters in the IP header, and not into the TCP header.

We notice that the threshold τ induces relatively limited changes, whereas the number of packets needed to change

states has a much more important impact. In the remainder of the document, we will consider the impact of the transition levels on the performance of IPHC, and set the value of τ to 1s.

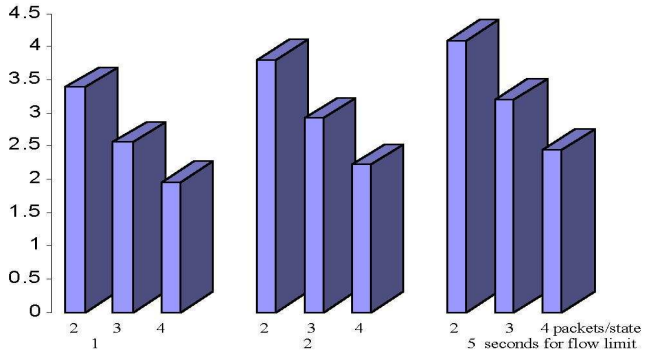


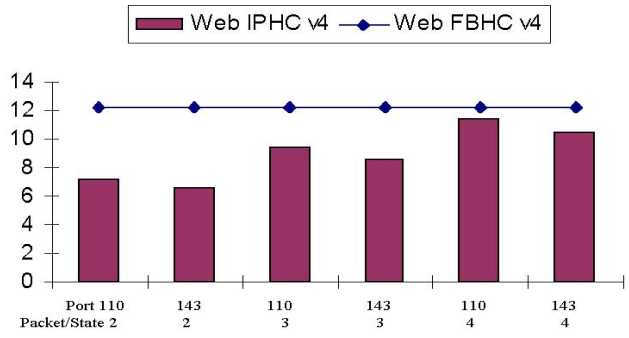
Fig. 6. IPHC compression

IPHC performs well, on average. The performance should be discussed with respect to the application. As discussed earlier, out of the IANA Well Known Ports that see significant traffic, only ftp and the printer spooler ports are long lived. We will see how FBHC compares with IPHC for some of these Well Known Ports in the next section.

1) FBHC vs. IPHC: We compare also the overall bandwidth savings on the overall traffic for both FBHC and IPHC. We study the gain for the applications commonly viewed as the strategic drivers for wireless applications: web browsing (ports 80 and 443) and email (ports 110 and 143) for the 1s thresholds.

Email:

For email, FBHC attains a compressed header size of 12 bytes: the mail port is one of the most frequent, and is always compressed to 1 bytes for the destination address. Figure 7 shows the different header sizes for the different transition levels for IPHC for IPv6. FBHC outperforms IPHC for IPv6 for usual 3 packets per state transitions.



Web browsing:

We ran IPHC and FBHC on the web browsing ports (80 and 443, as 8080 did not yield enough traffic on some monitored nodes to be significant). Figure 7 depicts the behavior of FBHC and IPHC for both port 80 and 443, for different numbers of packets per transition in IPHC. We only show IPv4 here. However, for IPv6, similarly to the mail traffic, FBHC outperforms IPHC for the web traffic.

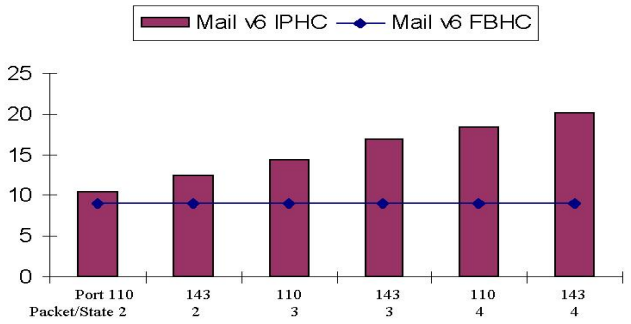


Fig. 7. FBHC and IPHC compression for Mail v6 and Web v4

2) Combining IPHC (or ROHC) and FBHC: FBHC only deals with the information that is external to the flow, whereas IPHC (or ROHC) deals with the internal information within a flow. Both algorithms can be combined. The way we combine them is as follow: we first run FBHC for any new flow. This means that either the flow is recognized by FBHC, and compressed, or let go as is. If it is recognized by FBHC, then its compressed header size is 12 bytes, and 20 if not. After enough packets have been transmitted, IPHC (or ROHC) can compress the flows, whether or not it is affected by FBHC, in effect taking over the compression.

To compute the gain is similar to computing the gain for IPHC with the simple difference: the starting header size is not fixed to 20 anymore, but is determined by FBHC. Figure 8 depicts the synthesis of the different protocols. There are 12 columns on figure 8: they each correspond to a different application (port 80, 110, 143, 443, respectively http, pop, imap and https) and a different transition levels between ROHC states (2, 3 or 4 packets).

Each column is divided into 6 levels: IPHC for IPv4, IPHC for IPv6, FBHC for IPv4, FBHC for IPv6 and the combination of IPHC and FBHC for IPv4 and IPv6.

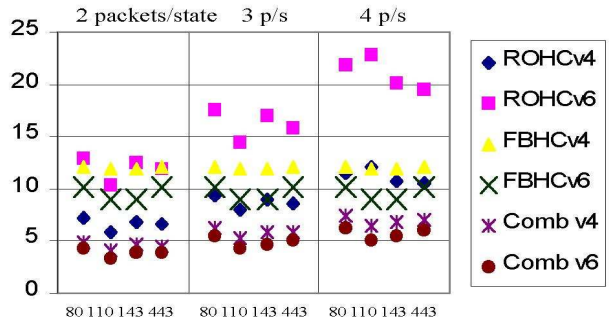


Fig. 8. IPHC, FBHC and their combination

As we can see, the combination of IPHC and FBHC always outperforms either one of these algorithms. The combination of IPHC and FBHC gives a average header length equal to 2/3 of ROHC for IPv4, and 1/3 of IPHC for IPv6. The gain for IPv6 is about 10 bytes per packet over the *IPHC compressed* packets.

As we discussed in [8], IPHC's (or ROHC's) use of bandwidth is actually more than the numbers we just gave. The reason is that the variability of the packet size among a single flow implies that some extra bandwidth has to be allocated to make room for full headers. Some of the compression gain is lost in the allocation procedure. How much compression gain is lost depends on the channel allocation procedure. FBHC does not have this concern as the channel allocation can be made on the actual header size, and the header size is constant throughout the flow.

Furthermore, in a wireless environment, the loss of packets would induce the re-establishment of the IPHC states. IPHC is not able to use Second-Order as much in an error-prone environment. FBHC performance would stay the same, while the performance of IPHC would degrade as the error loss probability increase.

V. Conclusion

We have described here a measurement study of the FBHC algorithm and a comparison with traditional HC algorithms. We have shown that the FBHC algorithm outperforms IPHC for IPv6 mail and web browsing applications.

The main advantage of FBHC over IPHC is that its granularity is at the flow level, and not at the packet level. This simplifies its management. The reader should be convinced that it is easier and more scalable to update state information for flows rather than for each packets.

On the other hand, if the complexity of managing IPHC state is not an issue, then IPHC (or ROHC as well for UDP/RTP flows) and FBHC can be integrated together. We have shown that an extra 10 bytes can be saved off from each IPv6 IPHC compressed headers by combining IPHC with FBHC.

Also, in an error prone environment, having sensitivity only to flow parameters allow to recover from packet losses more easily: all HC schemes are designed for wireless environment. We have not described here any of the complex recovery mechanisms from IPHC or ROHC to compensate from single packet losses. This is not an issue with FBHC.

Further studies will focus on the management complexity of FBHC implementation. Maintaining compression states -and its inherent complexity- is currently one of the issues slowing down a widespread adoption of HC schemes, especially in 3G cellular networks. We will study the architecture issues to integrate FBHC into 3G networks.

Acknowledgments

The author would like to thank Jari Malinen for his help in setting up the measurement tools, and Giao Le, Greg Smith and David Chau for their assistance and their patience with the measurement process.

References

[1] C. Bormann, editor. *Robust Header Compression,* RFC 3095, IETF, July 2001.

[2] M.Degermark, B. Nordgren, S.Pink *IP Header Compression,* RFC 2507, IETF, February 1999.

[3] C. Huitema *IPv6: The New Internet Protocol* Prentice-Hall Inc., Oct. 1997.

[4] Sean McCreary and kc claffy, *Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange.* ITC Specialist Seminar on IP Traffic Measurement, Modeling, and Management, Monterey, California, September 14, 2000.

[5] *Cooperative Association for Internet Data Analysis* http://www.caida.org

[6] V.Jacobson, R. Braden, D.Borman. *Compressing TCP/IP headers for low-speed serial links* IETF, Network working group, RFC 1144, Feb. 1990

[7] C. Westphal *A User-based Frequency-dependent IP Header Compression Architecture* To appear in proceedings of IEEE Globecom, Taiwan, 2002.

[8] Westphal, C., Koodli, R. *IP Header Compression: A Study of Context Establishment* To appear in Proc. of WCNC, New Orleans, March 2003.

[9] Cardwell et al. *Modeling TCP latency* Proc. of IEEE Infocom, Tel Aviv, Israel, March 2000