

A Lossless Compression Method for Internet Packet Headers

Raimir Holanda and Jorge García
Computer Architecture Dept.
Technical University of Catalonia
Barcelona, Spain
Email: {rholanda,jorge}@ac.upc.edu

Abstract—A critical requirement for performance evaluation and design of network elements is the availability of realistic traffic traces. There are, however, several reasons that makes it difficult to have access to them. Firstly, Internet providers are usually reluctant to make real traces public, secondly, hardware for collecting traces at high speed is usually expensive, and finally, with the increase of link rates, the required storage for packet traces of meaningful duration becomes too large. In this paper we address the problem of compression of these potentially huge packet traces. We propose a novel packet header compression, focused not on the problem of reducing transmission bandwidth or latency, but on the problem of saving storage space. As far as we know, ours is the first method specifically oriented to this goal. With our proposed method, storage size requirements for *.tsh* packet headers are reduced to 16% of its original size. The compression proposed here is more efficient than any other existing method and simple to implement. Others known methods have their compression ratio bounded to 50% and 32%.

I. INTRODUCTION

A critical requirement for performance evaluation and design of network elements is the availability of realistic traffic traces. Network traffic traces can be obtained by several methods. A popular scheme is to collect real traces from routers for extended periods of time [1]. These traces represent the mix of traffic flowing through a router and are collected on one of the input or output links. There are, however, several reasons that makes it difficult to have access to them. Firstly, Internet providers are usually reluctant to make public real traces captured in their networks. Moreover, when these traffic traces are made public [2], they are delivered after some transformations, such as sanitization [3], which modify some basic semantic properties (such as IP address structure). Secondly, there are others problems which arise due to the increasing speed of Internet routers. Hardware for collecting traces at high speed (e.g. to link rates of 2.5 Gbps, 10 Gbps or even 40 Gbps) is usually expensive. Moreover, with the increase of link rates, the required storage for packet traces of meaningful duration becomes too large.

As an example, let us consider the problem of storing a 11 days (one hour per day) trace taken from a link at 10 Gbps and considering 20% of link utilization. Storing the full content of the traffic would require an storage of 900 Gbytes per day. If we only store the 40 bytes TCP/IP headers, together with timing information, we would require a storage capacity of around 45 Gbytes per day and 495 Gbytes along 11

days (assuming a mean packet length of 920 bytes). Similar storage requirements are found, for instance, in [2]. In this case the stored traces were collected with a NLANR PMA OC192MON located on SDSC Tera Grid Cluster.

In this paper we address the problem of the compression of these potentially huge packet traces.

A first approach to cope with the huge storage needs is to use a standard compression method. Content compress can be as simple as removing all extra space characters, inserting a single repeat character to indicate a string of repeated characters, and substituting smaller bit strings for frequently occurring characters. The compression is performed by algorithms which determine how to compress and decompress. Some of the most popular compress algorithms are the Huffman coding [4], LZ77 [5], and deflate [6]. Those specifications define lossless compressed data formats. From our measurements, using these methods on files containing packet headers, we can expect a compression ratio of around 50%.

The previous methods do not take into account the specific properties of the data to be compressed. There are compression techniques developed for the specific case of packet headers. As far as we know, all of them have been developed for saving transmission bandwidth on channels such as wireless and slow point-to-point links. The original scheme proposed for TCP/IP header compression in the context of transmission of Internet traffic through low speed serial links is Van Jacobson's header compression algorithm [7]. The method is based on the fact that in TCP connections, the content of many TCP/IP header fields of consecutive packets of a flow can be usually predicted. As we will show, the achievable compression rate using this method is around 32%. Since then, specifications for the compression of a number of other protocols have been written. Degermark proposed additional compression algorithms for UDP/IP and TCP/IPv6 [8]. Detailed specifications for compressing these protocols, as well as others such as RTP, were described in subsequent RFC's [9] and [10]. Each of these descriptions specify a solution for a given protocol. For multimedia services in wireless environments ROHC (Robust Header Compression) was introduced. ROHC was standardized in [11] and is an integral part of the 3GPP-UMTS specification [12]. Equally for wireless environments, another scheme that makes use of the similarity in consecutive

Taking into account the joint behavior of $F(1)$ and $\Delta F(i)$, we have created four categories of fields. In the first category, are placed the fields whose $F(1)$ values are predictable and $\Delta F(i)$ values are constant or predictable through a flow:

$$\begin{aligned} & ((F(1) \text{ Not Random}) \text{ AND } (F(1)\text{-predictable})) \\ & \text{AND} \\ & ((\Delta F(i) == 0) \text{ OR } (\Delta F(i)\text{-predictable})) \end{aligned}$$

The fields that agree with those constraints are: *interface, version, IHL, type of service, flags, fragment offset, protocol, destination port* for Web servers, *data offset, reserved, and control bits*. This set of fields shows a high similarity within consecutive packets belonging to the same flow and in particular between m-packets flows (flows with m packets).

In the second category are included the fields whose $F(1)$ values are not predictable and $\Delta F(i)$ values are constant or predictable:

$$\begin{aligned} & ((F(1)\text{-Not Random}) \text{ AND } (F(1)\text{-Not predictable})) \\ & \text{AND} \\ & ((\Delta F(i) == 0) \text{ OR } (\Delta F(i)\text{-predictable})) \end{aligned}$$

According with those constraints, we have the following fields: *TTL, source address, and destination address*. For these fields, storage needs are restricted to the first packet of each flow.

The third category incorporates the fields that are hard to predict or calculate and we can not assign random values to them:

$$(\Delta F(i)\text{-Not predictable})$$

In this case, storage needs are extended over all packets. These fields are: *timestamp, total length, header checksum, acknowledgment number, and window*.

Finally, the last category groups the fields whose initial value $F(1)$ is random and the increments $\Delta F(i)$ can be calculated: *identification, source port for Web clients, and sequence number*.

III. FLOW CLUSTERING

In [16] a novel flow characterization that incorporates a specific set of packet characteristics such as TCP structures, inter packet time, and payload size, was proposed. This flow characterization was used in the context of a lossy compression method for packet header traces [17]. To provide a lossless compression an extended approach should be adopted. In this section we summarize the main ideas behind the flow characterization and clustering that we apply in the lossless compression method proposed in this paper.

We start our flow characterization defining a packet flow as a sequence of packets in which each packet has the same value for a 5-tuple of source and destination IP address, protocol number, and source and destination port number.

Let P_i^m be the packet header of the i -th packet of a flow consisting of m packets. $P_i^m(j)$ is a selected header field of P_i^m . For each field $P_i^m(j)$, a function χ_j performs a mapping

into an integer value $F_i^m(j)$:

$$F_i^m(j) = \chi_j(P_i^m(j)) \quad (1)$$

For each packet, let

$$F_i^m = (F_i^m(1), F_i^m(2), \dots) \quad (2)$$

denote a vector of integers, where we include the selected fields. For the complete flow we can define:

$$P^m = (P_1^m, P_2^m, \dots, P_m^m) \quad (3)$$

and

$$F^m = (F_1^m, F_2^m, \dots, F_m^m). \quad (4)$$

Note that the vector F^m can be viewed as a numerical representation of the m packet headers, as we substitute some selected packet header fields by integers.

From the flow classification described in Section II, we have selected 12 fields to study their diversity among Web flows in Internet links. The shaded boxes in Figure 2 depict those selected fields. Using the flow characterization described above, in a high-speed link, we can find potentially a large variety of Web flows. However, looking into the flows, we can see that they are not very different from each other. To study the variety among them, we have used an approach based on clustering, a classical technique used for workload characterization [18]. The basic idea of clustering is to partition the components into groups so the members of a group are as similar as possible and different groups are as dissimilar as possible. From each cluster, we generate a flow template. The clustering methodology starts from a real trace, converting each flow in a F^m vector. Each new flow is compared against the previously generated templates. To be able to do that, we calculate the Euclidian distance between them. In the case of lossless compression methods the maximum distance is zero, but for lossy methods, a small distance can be admitted. Whenever a match is not possible, a new template is generated as a center of a new cluster.

We have applied our methodology to traces from different available packet traces [2], [19]. We concluded that behind the great number of Web flows in a high speed link, many of them have identical or very similar F^m vectors and they could be grouped into few clusters.

IV. PACKET TRACE COMPRESSION

The main reason why header compression can be done at all is the fact that there is significant redundancy between header fields, both within consecutive packets belonging to the same flow but in particular between flows. The big gain of our proposed method comes from the observation that, for a set of selected header fields, the flows traveling in an Internet link are very similar. By utilizing a set of pre-computed templates of flows and predictability for other fields, the header size can be significantly reduced. Hence, we have embarked upon the development of a new header compression scheme for packet header files that reduces drastically storage requirements. This section provides the details of how the

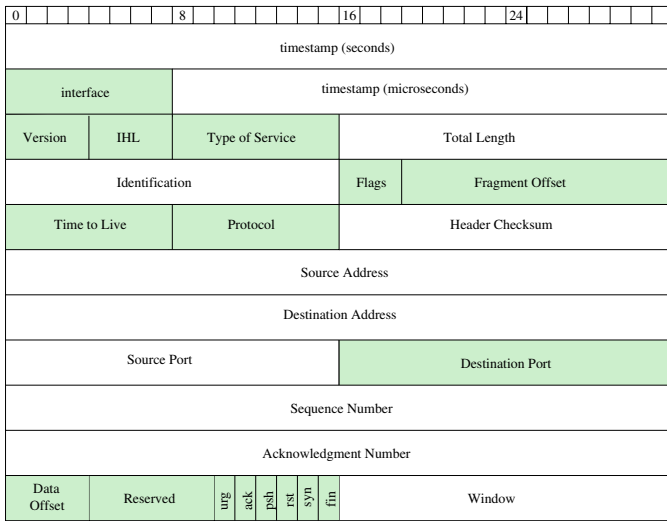


Fig. 2: Fields used to flow clustering

method works, focusing on the fact that the decompressed header is functionally identical to the original header.

Our compression method uses the pre-computed Flow Clustering dataset described in Section III as the main input to compress a TSH packet header (Figure 3). It works by finding F^m vectors that match with one of the templates and starts looking into the 5-tuple of fields (source and destination IP address, source and destination port number, and protocol number) to identify each new connection.

Whenever a packet carrying a new flow is found, a new node is inserted at the end of a temporary data structure (Figure 4). This data structure is implemented by a linked list and stores the packet headers of n connections.

If packets belonging to this same flow are found, we store only a subset of these fields: *timestamp*, *IHL*, *total length*, *Flags*, *Fragment offset*, *time to live*, *header checksum*, *acknowledgment number*, *data offset*, *control bits*, and *window*. When a *Fin* or *Rst* TCP flag arises in a packet, the flow status field is updated, to indicate that this flow has been completed.

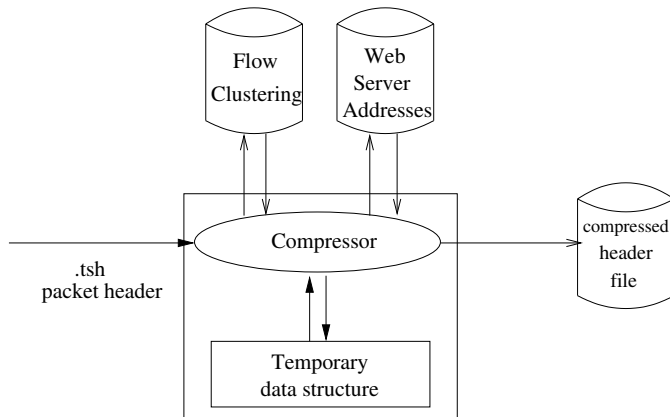


Fig. 3: Compression model

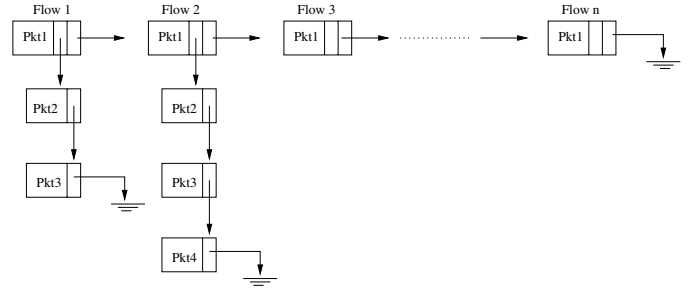


Fig. 4: Temporary data structure

When the head of the linked list reaches a *completed* flow status, the compressor algorithm, examines the number of inserted nodes associated to this flow and searches for identical sequence of packets characteristics into the Flow Clustering dataset. Considering that we are implementing a lossless compressor, the maximum inter flow distance admitted is zero. In the case that a match is not possible, a new record is inserted in this dataset. This new F^m vector will constitute a new template and the center of a new cluster.

After the template searching, the compressor algorithm starts to write into the compressed header file. For many fields (see Figure 2), the storage is reduced to a template identifier, which is the most important realization of our proposed method.

However, for other fields, which predictability is not possible, the carried information requires to be stored. Here, is important to consider that, for some of these fields, which the value is likely to stay constant over the life of a flow, the storage is required only once per each flow. However, for the remainder fields, the storage is required for all packets.

Figure 5 shows the compressed data format for the set of fields, which the values are stored once per each flow. The first field, is a flag to identify the direction of the flow: from or to a Web server (1 bit). The inter-flow time (second field) stores the elapsed time between two consecutive flows (15 bits). The initial window field, stores the initial value assigned to the window field (16 bits). The fourth field is the Web client address (32 bits). In the case of packets flowing to Web servers, this field stores the source address, otherwise, it stores the IP destination address. The next field stores an index to a Web server address dataset (16 bits). According with [20], the best-known characteristics of Web reference streams is their highly skewed popularity distributions, The practical implication of these distributions for reference streams is that most references are concentrated among a small fraction of all of the objects referenced. Hence based on this property, we have seen that using the strategy of store separately the Web server addresses, we increase the compression ratio. The sixth address stores an index to a specific template position into the template dataset (16 bits). Finally, the last field stores the value of the TTL of the first packet of each flow (8 bits). In total, we need of 13 bytes per each flow to store this set of fields.

For the set of fields whose information assume different

(we have increased it from 1 byte to 3 bytes). Moreover, we assume that a time stamp (3 bytes) is added to each header. As a result we assume that minimal encoded headers becomes 8 bytes in the best case and 21 bytes in the worst case.

To estimate the compression ratio for the Jacobson we must use flow-length distribution measured in the available packet traces. We will call P_m as the probability that a Web flow has m packets.

With the changes we have explained before and for a TSH packet header (44 bytes), the compression ratio for m-packet flows using the Van Jacobson method is bounded by:

$$f^{VJ}(m) = \frac{44 + 8(m - 1)}{44m}, \quad (5)$$

obtaining thus a compression ratio given by:

$$C_{Ratio}^{VJ} = \sum_m P_m^{VJ} f^{VJ}(m) \quad (6)$$

The P_m distribution obtained from several traces shows that most flows today are short lived, with small number of packets [23], [24], [25]. Using this distribution, we conclude that the compression rate of the Van Jacobson method reaches 32% in the best case.

In the proposed compression method 13 bytes for each new flow and 7 bytes per packet are sufficient to represent each flow of m packets. There are some data structures with information related to the clusters of flows that are also needed. However these additional data structures are almost constant with the packet trace length. Then for large packet traces, the compression ratio for m-packet flows is given by:

$$f(m) = \frac{13 + 7m}{44m}, \quad (7)$$

obtaining thus a compression ratio of:

$$C_{Ratio} = \sum_m P_m f(m) \quad (8)$$

which results in a compression ratio of around 16%. Figure 8 shows the file size of the original trace and the correspondent file sizes for the three compression methods under analysis.

VII. CONCLUSION

In this paper, we have introduced a novel lossless packet header compression method based on TCP flow clustering. Using the semantic similarities among Web flows and the TCP/IP functionalities, we have grouped many packet streams in few templates.

With our proposed method, storage size requirements for *.tsh* packet headers traces are reduced to 16% of its original size. The compression proposed here is more efficient than any other existing method and simple to implement. Others known methods have their compression ratio bounded to 50% (GZIP) and 32% (Van Jacobson method), pointing out the effectiveness of our method.

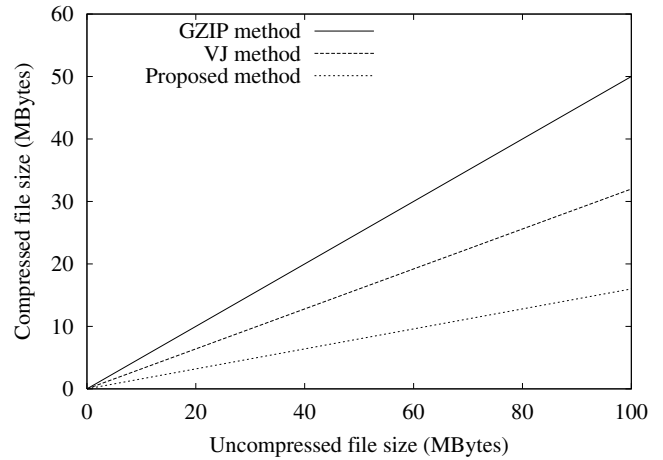


Fig. 8: File size comparison

In future works, we intend to extend the applicability of the method to other packet header formats and other emerging applications like P2P. Moreover, applying the same methodology and preserving the most important statistical properties present into the Internet traffic, some lossy methods can be developed to reach a more effective compression rate.

ACKNOWLEDGMENT

This work was supported by CAPES-Brazil, by the Ministry of Science and Technology of Spain under contract TEC-2004-06437-C05-05, and under grants VI FP project EuroNGI.

REFERENCES

- [1] CAIDA. The Cooperative Association for Internet Data Analysis. In www.caida.org.
- [2] NLANR. National Laboratory for Applied Network Research. In <http://www.nlanr.net>.
- [3] R. Pang and V. Paxson. A High-Level Programming Environment for Packet Trace Anonymization and Transformation. In *Proceedings of ACM SIGCOMM Conference*, August 2003.
- [4] D. E. Knuth. Dynamic Huffman coding. In *Journal of Algorithms*, 6:163-180, June 1985.
- [5] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. In *IEEE Transactions on Information Theory*, Vol. 23, n 3, pp. 337-343.
- [6] DEFLATE. Compressed data format specification. In *Available in ftp://ds.internic.net/rfc/rfc1951.txt*.
- [7] Van Jacobson. Compressing TCP/IP Headers for Low-Speed Serial Links. In *RFC 1144*, February 1990.
- [8] M. Degermark, M. Engan, B. Nordgren, and S. Pink. Low-loss TCP/IP Header Compression for Wireless Networks. In *Proc. MOBICOM*, Rye, NY, November 1996.
- [9] M. Degermark, B. Nordgren, S. Pink. IP Header Compression. In *Internet Engineering Task Force, RFC-2507*, February 1999.
- [10] S. Casner and V. Jacobson. Compressing IP/UDP/RTP Headers for Low-Speed Serial Links. In *Internet Engineering Task Force, RFC-2508*, February 1999.
- [11] C. Bormann et al. RObust Header Compression ROHC: Framework and four profiles: RTP, UDP, ESP, and uncompressed. In *Request for Comments 3095*, July 2001.
- [12] 3rd Generation Partnership Project. Radio Access Bearer Support Enhancements. In *3GPP, Tech. Rep.*, 2002.
- [13] C. Westphal. Improvements on IP Header Compression In *GLOBECOM 2003 - IEEE Global Telecommunications Conference*, vol. 22, no. 1, pp. 676-681, December 2003.
- [14] TSH format. In <http://pma.nlanr.net/Traces/tsh.format.html>.

- [15] R. S. Tomlinson. Selecting Sequence Numbers. In *Proc. ACM SIGCOMM/SIGOPS Interprocess Communications Workshop*, pp. 11–23, 1975.
- [16] R. Holanda, J. Garcia, and V. Almeida. Flow Clustering: a New Approach to Semantic Traffic Characterization. In *12th Conference on Measuring, Modelling, and Evaluation of Computer and Communication Systems*. Dresden Germany, September 2004.
- [17] R. Holanda, J. Verdu, J. Garcia, and M. Valero. Performance Analysis of a New Packet Trace Compressor based on TCP Flow Clustering. In *IEEE International Symposium on Performance Analysis of Systems and Software - ISPASS 2005*, Austin Texas, March 2005.
- [18] R. Jain. The Art of Computer Systems Performance Analysis. In *John Wiley Sons, Inc., New York*, 1991.
- [19] RedIRIS. Spanish National Research Network. In <http://www.rediris.es>.
- [20] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana Oliveira. Characterizing reference locality in the WWW. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems (PDIS96)*, December 1996.
- [21] J. -L. Gailly and M. Adler. GZIP documentation and sources. In <ftp://prep.ai.mit.edu/pub/gnu/>.
- [22] J. -L. Gailly and M. Adler. ZLIB documentation and sources. In <ftp://ftp.uu.net/pub/archiving/zip/doc/>.
- [23] L. Guo and I. Matta. The War Between Mice and Elephants. In *Technical Report BU-CS-2001-005*, Boston University, Computer Science Department, May 2001.
- [24] N. Brownlee and K. Claffy. Understanding Internet traffic streams: Dragonflies and tortoises. In *IEEE Communications Magazine*, 40(10):110–117, October 2002.
- [25] S. McCreary and K. Claffy. Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange. In *ITC Specialist Seminar on IP Traffic Measurement, Modeling, and Management*, Monterey, California, September 2000.