Performance Measurement of TCP/IP Header Compression

Sakti Debbarma

Computer Center, Tripura University, Tripura, India
E-mail: ersakti03@yahoo.com

Abstract

Compressed Header TCP/IP has recently become a research goal. Different compression schemes of header are found in literature. In this paper optimum size of packets traversing over the Internet is investigated. A suitable parameter is defined to study the benefits of Header Compression. A modified header compression scheme is proposed and its benefits are studied.

Keywords: Data Overhead, payload, packet header, Compressor, decompressor.

Need for Header Compression

Internet works on many different protocols like Transmission Control Protocol (TCP), Users’ Datagram Protocol (UDP) and Internet Protocol (IP). In TCP, UDP, RTP, IPv4 and IPv6, the minimum header bytes are respectively 20, 8, 12, 20 and 40. Header bytes are those protocol bytes which are communicated for a message, that do not represent the data bits of the message. These headers bytes leads to overhead. This overhead can be termed as Data Overhead. Data Overhead (DO) can be computed as DO= S / D-1 where: DO is Data Overhead. D is the amount of Data that should be transmitted by the protocol (measured in bytes). This is the payload of the protocol and is often referred to as user data. S is the amount of data the protocol actually Sent, to transmit the payload D to the receiver (measured in bytes). This includes retransmitted packets, packet headers and acknowledgement.
Studies shows [Fig 1] that in the core networks about 55% of packets traversing in the backbone are less than 100 bytes in size. The graph [fig.1] shows an optimal network throughput for packet size 28. For packets smaller than 28, the network throughput drops due to the Data overhead. For packets larger than the optimum, the network throughput becomes worse due to network congestion.

**Measuring Header Compression**

Consider the following case

When TCP/IP is used to send 1000 bytes in one packet, that would generate one 1040 byte packet (40 bytes header and 1000 bytes payload) and one 40 byte acknowledgement. That would result in:

\[
\begin{align*}
D &= 1000 \\
S &= 1040 + 40 = 1080 \\
DO &= \frac{1080}{1000-1} = 8\%
\end{align*}
\]

Again we take packet size 100, that would generate one 140 byte packet (40 bytes header and 100 bytes payload) and one 40 byte acknowledgement. That would result in:

\[
\begin{align*}
D &= 100 \\
S &= 140 + 40 = 180 \\
DO &= \frac{180}{100-1} = 80\%.
\end{align*}
\]
So it is seen that data overhead becomes very large when packet size is low.

![Graph showing transmission overhead against packet size.](image)

**Figure 2:** The transmission overhead against packet size.

Fig. 2 shows that data overhead increases as packet size decreases. So there is need to reduce data overhead. Data overhead can be reduced by header Compression technique. Header Compression technique actually compresses the excess header fields. This is possible in header compression due to redundant header fields of same packet or the consecutive packets of a stream. The compression of outgoing data grams must be done before any security processing like encryption, authentication and digest; and before any fragmentation of IP data grams.

**Scheme of Header Compression**

Header Compression scheme relies on knowledge of the TCP/IP headers. The algorithm first classifies packets into individual flows (i.e. packets that share the same set of {IP addresses, IP protocol type, and TCP port numbers}). State (a context) is then created for each flow and a Context ID (CID) assigned to identify the flow at the compressor and decompressor. The sender then omits fields in the header that remain unchanged between successive packets in an IP flow (these may be deduced by using the CID in each packet to refer to the context state at the decompressor). Within an IP flow, more then half of the fields are unchanged between successive packets. The “total length” and “identification” field are expected to be handled by link framing protocols. The “IP checksum” can also be re-calculated at the receiver. By suppressing these fields at the compressor and restoring them at the decompressor, header compression can significantly reduce data overhead. The remaining fields
usually change only by a small amount in successive packets. A further saving can be achieved by transmitting the difference (i.e. differential encoding) in the value of the field rather than the entire fields. This reduces compressed header to 3 to 4 bytes. Header Compression relies on two types of error detection: A CRC at the link layer (to detect corruption of the compressed packet) and the TCP checksum at the transport layer (to detect corruption in the compressed packet). When errors are detected, the receiver discards the erroneous packet. This creates another problem in the decompression process. Since the differential compression techniques are applied, the receiver also looses the context state. The next following packet after the discarded packet cannot therefore be decompressed correctly. It must also be discarded. All subsequent packets will therefore be discarded until the next synchronisation (i.e. uncompressed packet is received, restoring the context state). This problem becomes prominent in noisy channel like wireless network where bit error rate is very high. To overcome from this error propagation efficiently, I propose modification in compression technique. A parity compressed header will be generated by combining 3 to 4 compressed header using XOR operation at the compressor. That generated parity packet will be transmitted to decompressor soon after generation. At the decompressor corresponding received compressed headers are used to generate parity compressed header. If there is any error in transmission of header compressed packet the packet will be discarded but compressed header information will be obtained by doing XOR operation among received parity packet and corresponding stored compressed header at the receiver. Stored compressed headers are cleared from receiver buffer after receiving 3 to 4 packets. This is demonstrated in fig.3. and fig.4.

| HC1  | 10101101 |
| HC2  | 11001100 |
| HC3  | 10001001 |
| HC4  | 11111010 |
| Header compressed parity packet | 00010010 |

**Figure 3:** Compressed header in Compressor

| HC1  | 10101101 |
| HC2  | 11001100 |
| HC3  | 10001001 |
| HC4  | Error packet |
| Header compressed parity packet | 00010010 |
| HC4= | Hc1 XOR HC2 XOR HC3 XOR Parity Packet=11111010 |

**Figure 4:** Compressed header in Decompressor.
In such case there is no need of refreshing by transmitting full header at regular interval. In case of refreshing with full header at every $f$th packet, Average header size is $AH_1 = (H-C)/f + C$, where $H$ is size of full header, $C$ is size of Compressed header. In case of sending parity compressed header at regular $f$th packet, Average header size is $AH_2 = (H-C)/P + C$ where $P$ is the size of Header compressed parity packet. As $f >> P$, $AH_2$ much smaller than $AH_1$. So there is saving of bandwidth in proposed scheme.

**Conclusion**

Size of most of the packet in the Internet is made very small to avoid network congestion and improve throughput. But protocol header makes transmission of small packet inefficient. This inefficiency can be resolved by header compression. The parameter Data Overhead can be used to quantify the benefits of header compression. The scheme proposed to overcome desynchronization of context state due to packet loss in noisy channel saves bandwidth.

**References**

[15] [Paul T Ammann, Managing Dynamic IP Networks, Tata McGrawhill, 200
[16] Peterson and Davie, Computer Networks, Asis Harcourt Ltd’ 2001
[17] C T Bhunia, Information Technology, Network and Internet, New Age International Publishers, New Delhi, 2005
[18] C T Bhunia, CSIC, Computer Society of India, Mumbai,
[23] Nimrod Arbel et al, IP Header compression for satellite (First Report), http://www.opalsof.net/qos/VoIP.htm
[29] S L Casner et al, Compressing IP/UDP/RTP headers for low speed serial links, RFC 2508, 1999