Integrating Compression into the TCP stack, for enhanced performance in a satellite network

D V Ramana, Mohsin Meghji, Varun Uppal
Inmarsat, London

1 Introduction

Real-time, packet by packet compression was a standard feature in 1st and 2nd generation wireless networks. For these networks, air link resources were scarce and the additional CPU cost of compressing individual packets was justified in the gain to capacity. The V.42bis standard became fairly widespread in this regard as it has a good trade-off of compression against CPU load. However, starting with 3rd generation systems (UMTS, etc.), compression has gradually been moved out of the radio network into the edge. There are many reasons for this

1. Compression at the edge is much more efficient, since it can use larger block sizes
2. Type specific compression techniques can be used, thus increasing the compression gain
3. As radio-technology has advanced, the gain due to compression is overtaken by the requirement for fast packet processing in order to fully utilize the high capacity radio channels

Edge supported compression is primarily available in file servers and web-servers. Many web-servers pre-compress their data using LZ77[1] before delivery when the web-browser is able to take advantage of this feature. However, this is still not very widespread.

There are still radio networks where the bandwidth conservation is a priority. Satellite networks fall into this category. While bandwidths and capacities available in satellite networks are gradually increasing, there is still an unfulfilled demand for bandwidth and every bit of conservation is valuable. With the availability of cheap, powerful hardware, the additional cost of processing is diminishing in importance. Most VSAT manufacturers offer some form of compression on their products, mostly based on either V.42bis, or some proprietary protocol.

1.1 Inmarsat BGAN

The Inmarsat BGAN is a geo-satellite based broadband packet data personal communication system. It offers individual terminals packet switched data transmission to upto 512 kb/s, with full portability. A number of features have been built into the Inmarsat BGAN system to increase the user experience and enhance the system capacity.
2 Performance Enhancement Proxy using Indirect TCP

One key feature is a built-in performance enhancement proxy, specifically designed to improve the performance of TCP connections over the long-delay satellite link. The Performance Enhancement Proxy (PEP) differs from those deployed in standard satellite systems in two significant ways. One, it uses the indirect TCP [2] approach, splitting up the TCP link using a PEP gateway. Over the split over-the-air link, instead of using proprietary protocols, the BGAN PEP uses TCP-Vegas[3] stack. This retains the TCP protocol structure and packet syntax, but makes the usage of the air link more efficient by the TCP. The net result is much faster convergence to “true” bandwidth, ability to withstand occasional packet drops and resilience to sudden bandwidth changes. Figure 1 shows the architecture of this system. The TCP PEP is placed between the SGSN and the GGSN and splits the end-to-end TCP connection into two parts. Another TCP PEP is loaded in the endpoint terminal equipment. The two PEPs operate more or less independently; each works on enhancing the flow of data towards the radio link.

2.1 Estimation of true bandwidth

All TCP congestion control [4] works by estimating the true bandwidth available in the network. The true bandwidth is the number of packets per second that the network can delivery for this TCP. The true bandwidth is limited by processing speeds of routers, bandwidth available in intermediate links and various other factors, depending on the nature of the network. The TCP Vegas is a modified version of TCP which tries to improve the estimation of the true bandwidth by separating the queueing delay and the TCP connection latency. The former is treated as an indication of the true bandwidth, whereas the latter is treated as a physical constant, which must be taken into account when computing relative variations. It works by measuring the variation in the round trip time as seen by consecutive acknowledgements received by the transmitter. An increase in the round-trip time (for the same transmission rate) is tantamount to a increase in the queueing delay as seen by outgoing packets (it is assumed that the return link, carrying only acknowledgements, has no queueing delay), whereas a decrease in the round-trip time is seen as an indication that there is bandwidth available in the system, since the queue in the network is shortening. This allows a TCP Vegas to achieve equilibrium transmission rate (the transmission rate at which a constant queue is maintained in the network) faster than that of a standard TCP New Reno implementation in a high delay satellite network [5].

Our implementation had some modifications over the standard TCP Vegas. A key feature of TCP Vegas is to be able to measure the network latency, the basic minimum delay across the link in the face of no queueing. However, in a constantly changing environment, no TCP connection sees a queue less network, since pre-existing TCPs will have outstanding queues. This leads to a situation where a TCP connection entering a congested network will significantly
overestimate the delay and consequently act much more aggressively than pre-
existing TCPs. The problem has been discussed to some extent in [6]. We had
to solve the problem by adjusting the latency estimation algorithm.

A second change that was brought about was to change the starting point.
Standard TCP always starts at a low transmission rate and gradually increases
it. In our case, the starting rate is a function of the QoS and customers with
higher QoS will start at a higher transmission rate. This means that the initial
transmission rate may be larger than the true bandwidth and thus have to adjust
downwards as well as upwards.

Fig. 1. BGAN Network Architecture

3 Compression in TCP PEP

The compression feature built into the TCP PEP has two basic stages. In the
first step, it is negotiated and in the second step it is implemented on a packet
by packet basis.

3.1 Activating compression

Unlike TCP Vegas, which only requires a change in the TCP algorithm and essen-
tially leaves the rest of the TCP end-to-end essentially intact (packet structures,
formats, and so forth), the compression feature cannot be implemented transpar-
ently. Thus, we need to start by negotiating compression end to end between the
two TCP PEPs. If there is no TCP PEP at the ends, then compression cannot be enabled.

The diagram 2 below shows how end-to-end compression is negotiated. As can be seen, it allows up to three PEPs to be part of the chain; the internal PEP recognizes that there are two PEPs at either side and does not interfere with the compression process.

To signal that the TCP is capable of compression, we created a new option TCPOPT\_COMPRESS with value 17. This option value is currently not registered for any purpose by the IETF. The originating TCP will insert the TCPOPT\_COMPRESS option as a TCP option in the first SYN packet transmitted. If the recipient is a standard TCP, it will ignore the option as per protocol [7]. On the other hand, if the recipient is a compression capable TCP PEP, it will send back the TCPOPT\_COMPRESS option in the SYN ACK packet, thus acknowledging and okaying the compression. The SYN packet would be forwarded with the option extracted. It has been planned to extend the use of the TCPOPT\_COMPRESS option to negotiate more details, such as the compression algorithm, the compression parameters, etc.

### 3.2 The choice of the compression algorithm

The TCP PEP uses LZ77 as the compression technique. In the selection procedure, a large number of compression techniques were tested on a selection of real life data and the Calgary Compression Corpus. Since the PEP data is mostly mildly compressible, a good tradeoff was required between the compression achieved and the CPU required. Based on the final result sets, LZ77 as supported by the slib library was selected for compression. In the future, more types may be added, based on further testing.

The compression algorithm had to be modified so as to operate on a packet by packet basis. Compression algorithms are implemented for file compression; thus they do not require to store internal state and can work with arbitrarily large buffer sizes, tuned for optimal performance. On the other hand, for our purpose, we need to be able to work in incremental mode, as individual packets arrive. Since different packets belong to different end-points (and thus may be serviced by different PEPs), compression has to be done on a stream basis.

### 3.3 Impact on TCP and the Vegas algorithm

The implementation of packet based compression within the TCP algorithm has several repercussions on the way TCP operates. These have to do with the handling of sequence numbers, the handling of retransmissions and the handling of the Tcp window.

The diagram in Fig. 3 shows a TCP packet (in simplified form). The header contains one specific field, the sequence number, which identifies the number of the first byte in the payload relative to the initial sequence number. To compute the length sequence number of any other byte, the TCP simply takes the sequence number provided and adds the distance (in bytes) between this byte
Fig. 2. Compression negotiation
and the start of the payload. This, thus, provides a way for identifying each byte separately and the technique is used for acknowledgements.

![Fig. 3. TCP Packet Structure](image)

**Sequence number handling** The problem for the originating PEP is that packets are received uncompressed and sent out compressed \(^1\). There are cases when the events on the two sides have to be correlated. Consider, for example, when an uncompressed packet is received, and we have to check whether it is in sequence or not. To verify this, we need to match its sequence number against the sequence number of the last byte of the previous packet. For the sake of matching events on either side, it is infeasible to modify sequence numbers. However, when compressed packets are sent out, their length is reduced, so the computation of sequence number on the receiving PEP is affected.

To solve this, the PEP handles sequence numbers as follows:

1. The compressing PEP does not modify the sequence number in the outgoing packet.
2. The receiving PEP does all manipulations after it decompresses the packet successfully.

When an individual packet is uncompressable, it is simply transmitted in its original form. The packet is passed to the decompressor which returns it with a format error, in which case the receiver assumes that the packet was not compressed.

**TCP Window management** TCP maintains two windows. One is the normal sliding window, which ensures the number of outstanding bytes is not larger than can be acknowledged using the defined protocol mechanisms. The second is the congestion window. This measures the estimate of the transmitting TCP of the number of packets it can deliver over the link without destabilizing the network.

\(^1\) Vice versa for the receiving PEP
In most networks, the sliding window is kept to a very high value; in other words, it is the congestion window which restricts the rate of transmission.

Since TCP assumes that packets will be transmitted using the maximum number of bytes available (the link MTU), the congestion window is measured in packets. This is a problem; since we are not reconstructing packets per se in the PEP, just replacing full size packets by small ones, the Vegas will restrict the transmission rate to \( cwnd \) packets per second, regardless of the packet size. This means that even after compression, the effective bandwidth as seen by the end-user will not change.

To solve this problem, we inflate the computed value of \( cwnd \) by the measured effectiveness of the compression as follows:

\[
\begin{align*}
icg &= \frac{osz(p)}{csz(p)} \\
\cg &= \cg \times 0.75 + \icg \times 0.25 \\
cwnd^* &= \cg 
\end{align*}
\]

where \( \icg \) and \( \cg \) are the individual (for the current packet \( p \)) and running compression gains, and the functions \( osz() \), \( csz() \) give the original (pre-compressed) and compressed sizes for the packet \( p \).

4 Experimental results and conclusions

The figure 4 shows the performance improvement due to compression for different file-sizes. As the file size increases, the performance figure increases smoothly, for all kinds of media.

5 Acknowledgments

The authors gratefully acknowledge the contributions made by Abheek Saha (Hughes Systique Corporation) and Ashish Batra (Aricent) for this paper.

References

