Return Link Optimization for Internet Service Provision Using DVB-S Networks

Nihal K. G. Samaraweera[1]

News Digital Systems Ltd, Stoneham Lane, Eastleigh,

Hampshire, S050 9NW, United Kingdom

+44(0) 1703 573392

nsamaraweera@ndsuk.com

ABSTRACT

Satellite based Digital Video Broadcasting (DVB-S) allows the same low cost satellite dish to receive both television programs and Internet traffic. The satellite system is used to construct a high-speed simplex distribution system, while the return path, needed for the Internet service will be provided using a low speed terrestrial network. The bandwidth asymmetry between the return and forward paths results in a problem, which we have termed "ACK congestion". A number of techniques that may alleviate ACK congestion over a DVB satellite link are analysed through simulation. The paper also presents a new ACK Compaction technique to eliminate ACK congestion, and an ACK spacing technique to preserve the *self-clocking* principle of TCP.

1.0 INTRODUCTION

The growing use of multimedia-capable personal computers to access the Internet and in particular, the use of World Wide Web, has resulted in a growing demand for Internet bandwidth. The emphasis has moved from basic Internet access to the expectation that connectivity may be provided whatever the location. This presents fresh challenges to the networking community, particularly as users become familiar with the benefits of high-speed connectivity.

Along with an increased use of the Internet, there has also been a revolution in television transmission, with the emergence of Digital Video Broadcasting (DVB) [1]. Recent developments will allow a DVB satellite hub station to provide high-speed transmission of packet data to the same small satellite dish as used for TV reception. In many cases, the available low speed terrestrial infrastructure (e.g., dial-up modem, ISDN connection) or low speed satellite return links may be used for the return link. This results in a network connection in which the capacity *to* a remote server is lower than that *from* the server. Such an asymmetry in the network paths may be suited to user needs, since most Internet clients receive much more data than they need to send. However, if the bandwidth asymmetry in the forward and return paths is greater than the asymmetry in the volume of the forward and return data, the return link may become congested and limit the throughput of the forward path available for an individual client.

TCP/IP is the most commonly used protocol in the Internet. IP provides best effort datagram delivery service, while TCP provides other services (e.g., reliability and congestion control) required by Internet applications. A TCP receiver generates an Acknowledgment (ACK) for every other received packet (or every packet when the ACK delay [2] is zero). The transmitter provides reliability by retransmitting any unacknowledged packet. In addition, received ACKs are used to implement TCP congestion control and avoidance algorithms. These algorithms use the principle of *self-clocking*, that is, packets are introduced into the network roughly at the rate of receiving ACKs for previously transmitted packets. Therefore, TCP throughput over an asymmetric network may become constrained by the rate at which the sender receives ACKs through the low speed return path, a phenomenon known as *ACK congestion*.

The commonly used compression and suppression techniques [3, 4] do not significantly improve TCP performance over such a network [5]. New techniques are therefore required to allow an individual user to utilize a highly asymmetric path. A new *ACK Compaction* technique will be discussed which eliminates the effects of ACK congestion and achieves highest performance by preserving the TCP *self-clocking* principle. However, an implementation of the technique may result in transmission bursts, since the low speed return link may not preserve receiver generated spacing between ACKs. An ACK spacing technique is, therefore, presented which reinforces TCP *self-clocking*.

2.0 DVB NETWORKING

DVB provides a standard way of transmitting Internet traffic (as well as video and audio traffic) over most of the transmission media, without relying on proprietary systems. The DVB standard has adopted the MPEG-2 standard [6] for compression of video and audio signals. In addition, MPEG-2 provides the multiplexing and media synchronization functions needed for transmission of digital data.

Figure 1 shows a simple configuration of a satellite-based DVB network. A low cost receive only satellite terminal may be used to construct a cost-effective high speed simplex distribution system when used in a star configuration with a powerful (expensive, but shared) hub station at the center of the network. Individual users and a remote Local Area Network (LAN) access a data server (located in the Internet or at a DVB hub site) using the DVB link. A client sends requests for data transfer (and later acknowledgments as the session progresses) through the terrestrial network, while the server transfers the data to the client through the higher speed satellite link. A client networking device (the client computer itself or the router) has two network interfaces; one connected to the receive DVB link and the other connected by an Internet Service Provider (ISP) to the terrestrial network [7].

[1] The author was with Electronics Research Group, Department of Engineering, University of Aberdeen, Aberdeen, UK

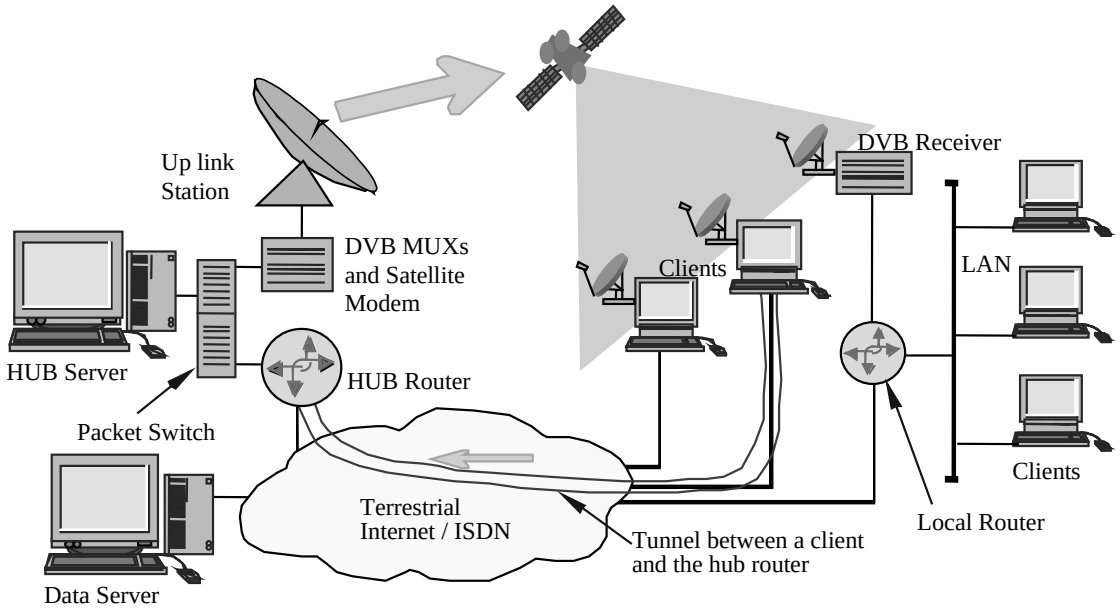

Figure 1: Configuration showing connection via a DVB satellite network

The hub router forwards data through the satellite network while the return data may be sent through the terrestrial link (as in this discussion) or in the future using a low power satellite return link. Tunneling may be desirable for a network with path asymmetry [8] (i.e., when the return packets of a connection are routed through a different network path than used to send the forward packets). IP tunneling (or encapsulating an IP packet with another IP header) is a commonly implemented technique in the Internet (e.g., MBONE [9] and DirectPC [10]). In the DVB case, a packet from the user (with the IP source of the user's satellite interface) is encapsulated with an IP header (with the IP source address of the terrestrial interface and an IP destination address of the remote extractor, which is usually the terrestrial interface of the hub router). The remote end of the tunnel extracts the encapsulated packet and forwards it through the Internet as if it were routed through the satellite link. The expander may also validate the received IP addresses to prevent spoofing. A detailed discussion of tunneling can be found in [8].

3.0 TCP EXTENSIONS FOR ASYMMETRY

The TCP protocol adopts a conservative approach to transmit data without causing network congestion. The TCP congestion control and avoidance algorithms use the principle of *self clocking*, that is, packets are introduced into the congested link on the forward path roughly at the rate of receiving ACKs (for previously transmitted packets) through the return path.

TCP implements this using a set of procedures called slow start and multiplicative decrease. These procedures allow a TCP transmitter to transmit a limited number of bytes determined by the smallest value of the receiver advertised window or the congestion window (*cwnd*). The algorithm also uses a variable to keep the threshold value of the send window (*ssthresh*) which is set to one half of the current *cwnd* (multiplicative decrease) when a packet is lost. The *cwnd* is set to one segment (Maximum Segment Size) when the slow start phase begins (when either the connection first starts or the retransmission timer expires), and increased by one segment (MSS) whenever an ACK is received (exponential increase), until it reaches the *ssthresh*. (However, this exponential growth of *cwnd* may cause congestion in the network as explained in section 7.) At the end of the exponential increase (slow start) phase, the algorithm switches to the congestion avoidance (linear increase) phase. On reception of an ACK, the sender is allowed to increase the *cwnd* by a fraction of the MSS (equivalent to one MSS per round trip delay) during the linear increase phase.

Therefore, the TCP throughput of a DVB link may become constrained by the rate at which the sender receives ACKs through the low speed return path. When the bandwidth asymmetry (in the forward and return paths) is greater than the data asymmetry (in the forward data and return ACKs), the rate of ACKs returned through the return link will be limited resulting in ACK congestion. When the ACK delay is zero, the data asymmetry is the ratio between the size of a TCP ACK (40 bytes, or 60 bytes with IP tunnel, plus the size of the data link protocol header) and the size of a TCP data packet (MSS was assumed as 1024 bytes for this discussion). MSS is normally a pre-negotiated value during the connection setup or an appropriate value derived using the MTU discovery [11]. Data asymmetry can be improved either by modifying the TCP protocol or by modifying the return ACK stream.

3.1 TCP Modifications

TCP modifications may be performed in the TCP/IP protocol stacks of the server and end user client machines [3]. Data asymmetry can be improved by reducing the number of ACKs generated by the receiver. Since TCP ACKs are cumulative [11], the TCP protocol is allowed to reduce the number of ACKs generated by the receiver by delaying the transmission of each (cumulative) ACK until a number of packets arrive at the receiver.

This is called ACK_delay [2, 11]. The receiver may generate only one ACK per *N* number of received packets, when using ACK_delay. This improves the data asymmetry by a factor of *N*. The TCP standard, however, recommends that at least every other packet is ACKed, so the maximum value for *N* is 2 [2].

TCP may still be modified to improve data asymmetry by delaying ACKs for more than 2 packets. However, this reduces the growth of *cwnd* during the slow start phase, resulting in a low throughput over a long delay network [12] (especially for small file transfers, which last only during the slow start period). The problem of slow growth of *cwnd* over a satellite network may be overcome using a larger initial value for *cwnd* [12, 13]. Alternatively, *cwnd* may be increased using the byte count of the ACKed sequence numbers (rather than number of ACKs) which increases *cwnd* by a few segments equivalent to *N* [12, 14]. Finally, the transmitter may use a large TCP MSS to increase the step size of the increment of *cwnd* (*cwnd* is increased by one segment or by the TCP MSS whenever an ACK is received by the sender). Such modifications are still being studied by the Internet community, but are not recommended to use in a shared Internet, since they may impact the stability of TCP over a large network [12, 15]. Therefore, alternative solutions are needed to reduce ACK Congestion which suit the characteristics of a DVB Network.

3.2 Transport Layer Gateway

One other possible way is to use a modified TCP or even an alternate (based on UDP) protocol over the DVB network while letting all the other hosts in the network use standard TCP [16]. In this case, the end-to-end connection has been divided into at least two sections, one with standard TCP and the other with a modified protocol. For example, the hub gateway will store and acknowledge the reception of any data from a standard TCP sender, while the gateway forwards data to the DVB receiver using the modified protocols. Therefore, the hub may act as a transport layer gateway and implement any necessary modifications (the gateway needs to implement both standard TCP and the modified protocols). The role of transport layer gateways is currently an area of active research within the Internet community. There are number of potential problems with this scheme [17] as follows:

(1) A substantial processing overhead may be required at the gateways (e.g., sending ACKs for the standard TCP sender, and processing each packet over two protocol stacks up to the transport layer).

(2) The gateway needs to take the full responsibility for the delivery of the data already ACKed and stored (i.e., the gateway is responsible for reliably transmitting stored data to the receiver; the gateway cannot delete these packets (like it can IP datagrams) until the receiver ACKs them, and therefore needs proper resource management and potentially substantial non-volatile storage).

(3) The scheme is vulnerable to unexpected failures (e.g., data may be lost due to gateway failure, even after the sender has finished the data transfer). These are difficult to correct after the original connection to the sender has been removed.

(4) The scheme does not work, if IP security is used over the forward path or the return path, since the gateway is unable to act in place of the receivers (i.e., it does not know its secret keys).

3.3 Transparent Modifications

The alternative approach is to modify the return link ACK stream. First, data over the return link can be compressed. CCITT V.42 bis data compression is commonly employed by modems. V.42 bis uses the Lempel and Ziv data compression algorithm [18]. This is particularly effective for ASCII text data, but less effective for pure protocol data (e.g. ACKs). Neither is compression suitable for random data (such as encrypted protocol headers). It reduces the TCP header from 40 bytes to an average of 16 bytes [19]. The TCP/IP header compression algorithm is an alternative technique specifically designed for compression of the TCP/IP header [19] and achieves higher compression for ACKs (reducing the TCP header from 40 bytes to an average of 6-7 bytes [19]). Both these compression techniques are designed for point-to-point modem links. Even though compression improves the modem link performance, the Internet Service Provider (ISP) may still need to guarantee a high bandwidth over the remainder of the Internet path to ensure uncompressed packets are transmitted after decompression at the end of the modem link. Failure to provide the required bandwidth (later in the network path) will also lead to ACK congestion.

Secondly, the return link ACK stream can be modified by using *ACK suppression* (also known as ACK filtering) [3-5] to reduce the volume of data transmitted through the return link. This technique uses the same concept as *ACK_delay* which exploits the cumulative nature of the TCP ACKs. The main advantage of this approach is that it can be implemented between the IP layer and link layer drivers (in a host or gateway) without modifying the TCP protocol.

3.4 Simulated Network Configuration

A DVB link has been simulated to understand the impact of ACK congestion and the performance of the three techniques described in the preceding sections. The techniques have been implemented in a TCP/IP simulator, which contains a full implementation of TCP [20]. The simulator was configured to use Reno with the window scaling extension [5]. The simulation configuration is shown in figure 2. A DVB-S transponder typically provides a bandwidth of 30-40 Mbps. It was assumed that up to 10Mbps of the satellite bandwidth was available for data transfer, while the rest of the capacity was allocated to video traffic. Most modern telephone networks support 28.8 kbps modems, although some areas still only offer 9.6 kbps capacity. Since we should not be optimistic about the bandwidth available through the Internet (often subjected to congestion), this study first considers a 9.6 kbps return link to analyse the worst case (a network with asymmetric ratio of ~1000:1). The performance using a range of return path speeds will be analysed later in the paper. TCP was configured to use a MSS of 1024 bytes, and had transmit and receive buffer sizes of 720Kbytes, sufficient for the DVB link [5].

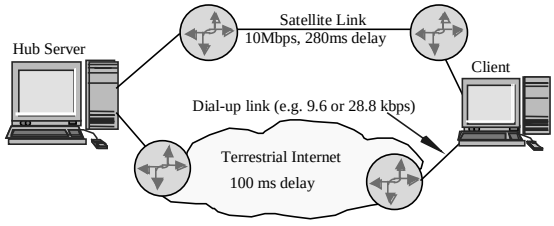

Figure 2: Network configuration.

4.0 TCP/IP HEADER COMPRESSION

TCP header compression [19] is widely used on modem links and may compress the TCP/IP header. This allows the size of an ACK to be reduced by ~70% over the return link. TCP performance over a DVB network was studied using the above described network configuration (section 3.4), by implementing TCP/IP header compression over the return link. For simplicity, a single session was considered limited to a maximum of 10 Mbps over the DVB channel.

Figure 3 shows the variation of *cwnd* with time for a data transfer from the hub server to a client when using TCP/IP header compression. Figure 4 shows the time required to transfer a number of bytes over the DVB link (the gradient of a curve represents the TCP throughput in Mbps). The results are also compared with unmodified TCP (i.e., without any transparent modifications, and assuming that the ACK delay at the TCP receiver is zero). The other graphs labeled "Suppression" and "Compactions" will be discussed in subsequent sections.

ACK Congestion arises when unmodified TCP is used, resulting in a slow increase in *cwnd* (figure 3) and a low TCP throughput (i.e. large transmission time, figure 4). In addition, the large queuing delay at the return link interface ultimately causes the TCP timers at the sender to expire indicating potential data loss and triggering unnecessary retransmission of data. The TCP congestion control algorithms is also triggered, reducing *cwnd* to a single segment and *ssthresh* to one half of the current value of *cwnd* [20, 21]. TCP throughput has therefore been degraded over the DVB network.

Since ACKs arrive at the sender at a higher rate when using header compression, the value of *cwnd* may grow more rapidly (figure 3). However, the compression ratio is not sufficient to eliminate ACK congestion for a DVB network, which still results in expiry of the TCP retransmission timer, and subsequent slow start. This results in only a small improvement in overall TCP throughput when using TCP/IP header compression (figure 4).

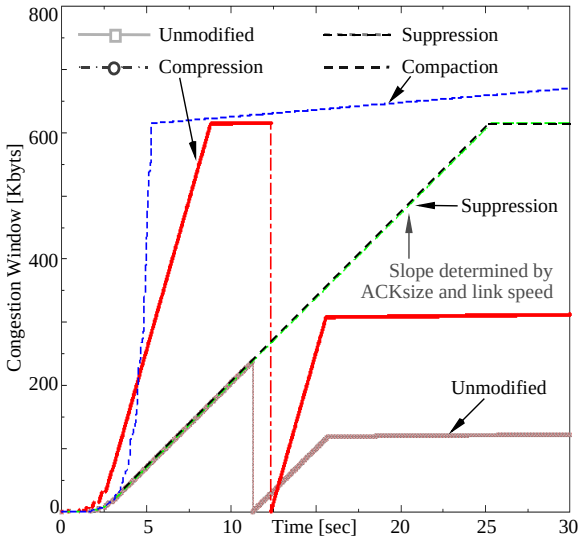

Figure 3: The variation of the *cwnd* with time.

The TCP/IP header compression method is designed for a point-to-point link and the performance of TCP may be degraded if a compressed packet is lost during the transmission, since the header compression uses differential encoding [22]. The packets tunneled through the return path may be lost during transmission. TCP/IP compression may therefore provide poor performance through such a tunnel. Improving the robustness of IP header compression algorithms is a current area of research [12, 22].

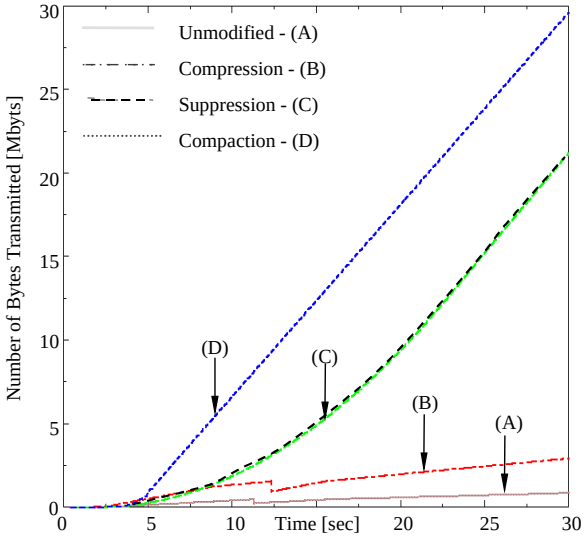

Figure 4: First sequence number of the transmitted packet against time.

5.0 ACK SUPPRESSION

Another way to avoid ACK congestion is to delete ACKs from the queue that builds up at the return interface. This technique is called *ACK Suppression* (or ACK filtering) [3-5], which exploits the cumulative property of TCP ACKs similar to the concept used in TCP *ACK_delay* (section 3.1). Such a scheme may be implemented below the transport layer. Avoiding modifications to standard TCP may compensate for the extra processing overhead (per ACK) performing ACK suppression. Implementation at the user machine may also reduce any interaction with security and encryption procedures.

To perform ACK suppression, each incoming ACK forwarded to the link interface queue is classified by a flow (TCP packets are identified by the protocol type of the IP header, TCP port numbers and IP addresses then uniquely identify a flow). The suppresser tags each ACK with a flow identifier before storing it in the transmission queue. Although a separate queue may be implemented for each flow, our implementation employs a single drop tail queue to store ACKs from all flows (as usually found in the Internet). However, the suppressor keeps a flow count for each flow, which is initialized to zero on the initialization of the interface driver. The corresponding flow counter is incremented by one whenever an ACK is received by the interface, and decremented by one whenever an ACK is transmitted or suppressed.

A flow counter (the number of ACKs belonging to a single flow) is allowed to be increased up to a low water mark value (e.g. a similar value used in Random Early Detection (RED) algorithm

[23]) before suppression. A suitable limit (set according to the burstiness of generating TCP ACKs) improves TCP performance, allowing the first ACKs of a flow to be received without suppression which, in turn, permits the sender to open *cwnd* quickly.

When a flow counter exceeds the low water mark (e.g., 4 ACKs), ACKs belonging to this session will be deleted from the front of the queue (oldest ACKs) leaving a single (cumulative) ACK. The maximum number of ACKs deleted from the flow is restricted to a high-water mark value to permit the sender to periodically receive ACKs (at least one ACK in a high water mark of ACKs). However, the high water mark should be large enough to achieve a high throughput (improving data asymmetry).

Some ACKs also have other functions in the TCP protocol and must not be deleted to ensure normal operation of TCP. The suppresser must therefore not delete an ACK that carries any data, or has any other TCP flags set (sync, reset, urgent, and final) [19]. In addition, the suppresser should avoid deleting a series of 3 duplicate ACKs which indicates a lost packet [21]. The suppressor caches the highest transmitted acknowledgement number (copied from the header of the latest transmitted ACK) for each flow. It looks at the header of the next ACK and the cached acknowledgement number to detect whether this is a duplicate ACK, before deleting an ACK from the front of the queue. This ACK will be transmitted without suppression if it is a duplicate ACK. This procedure may be optimized to limit the number of unsuppressed ACKs only up to 3, but our implementation prevents the risk of not detecting a lost packet by the sender due to the loss of a duplicate ACK.

The previous simulations have been repeated, this time implementing ACK suppression at the link layer. The low and high water marks were set to 4 and 60 ACKs respectively. Figure 3 shows the variation of *cwnd* with time for a data transfer from the hub server to a client. Figure 4 shows the time required to transfer a number of bytes over the DVB link (the gradient of a curve represents the TCP throughput in Mbps). The results are also compared with unmodified TCP and the TCP performance with header compression.

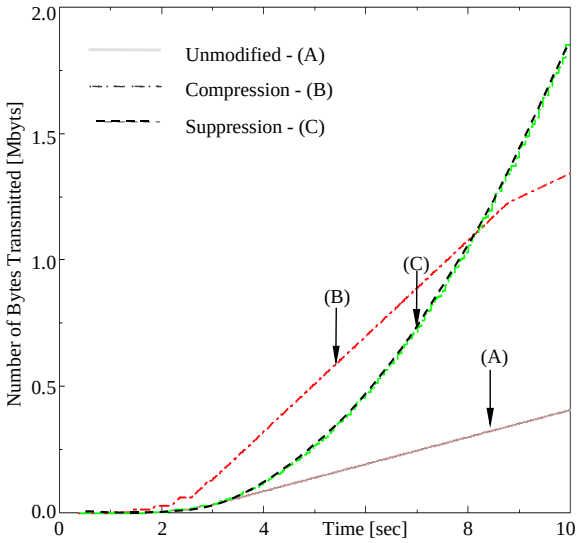

Figure 5: Number of bytes transmitted against time.

ACK Suppression achieves higher TCP performance (for transfer > 1.2 Mbps) compared to header compression by significantly reducing the queuing delay. ACK Suppression eliminates ACK congestion (the *cwnd* has been continuously increased, figure 3). However, the throughput may still remain low during the initial part of the file transfer. This is because *cwnd* still grows slowly, due to the small number of ACKs received - even though all data has been cumulatively acknowledged. (The sender congestion avoidance algorithm only increases *cwnd* on reception of an ACK and does not consider the cumulative nature).

A TCP sender uses each received ACK for two purposes. First, it indicates a previously transmitted packet has left the network, so the sender may increase *cwnd* allowing it to transmit more data to the network. Second, it indicates the receiver will accept more data. During the slow start phase, the transmission rate is constrained by the first factor, and when the *cwnd* is sufficiently open, the transmission rate is constrained by the second factor.

The return path acts as a bottleneck and determines the rate at which ACKs are received at the sender so that the congestion window increases slowly during the initial phase of a data transfer (figure 3). Figure 5 zooms in on the initial part of the data transfer during the previous simulation. TCP performance with ACK suppression is poor for short file transfers (e.g. when the file size is less than 1.2 Mbytes with this simulation configuration, as shown in figure 5). This is particularly significant for Internet traffic as most connections are short-lived [24] and therefore are always constrained by the value of *cwnd*. Therefore we have developed a new technique which almost reproduces the original ACK stream at the sender without increasing the return link bandwidth.

6.0 ACK COMPACTION

A new *ACK Compaction* technique is proposed which eliminates the effect of ACK congestion, but maintains an acceptable arrival rate of ACKs at the sender. A data link layer compactor first establishes an IP tunnel to a remote expander that could be at the DVB transmitter site (i.e., collocated with the tunnel router at the hub site).

ACK Compaction uses similar techniques as ACK Suppression, but with several modifications. When a queue of ACKs builds up at the return interface, older ACKs will be deleted from the front of the queue, leaving the latest ACK for transmission (as explained in section 5). In performing ACK Compaction, the transmitter appends information to this remaining ACK that is later used by an expander for extraction (regeneration) of deleted ACKs. The information contains the number of deleted ACKs (one byte) and the total number of bytes acknowledged by the deleted ACKs (two bytes). (Please note that this modified ACK may not be removed from the transmission queue, but marked to indicate that it has already been compacted.)

Since these modified ACKs are no longer IP datagrams, they need to be tunneled to the expander that recognizes the new header. Tunneling packets over the asymmetric return link is already desirable as explained in section 2.

The expander is stateless (i.e. processes each received packet independently of previously received packets). It extracts the header (3 bytes of prefixed information), and uses this information together with the acknowledgement field in the TCP header of the received ACK (the highest acknowledgment number from all

compacted packets) to produce a burst of ACKs equivalent to the number of ACKs previously deleted by the compactor. In performing extraction, the expander generates the required number of copies of this received ACK, and overwrites the acknowledgement number of each generated ACK by a calculated value (starting with the calculated lowest value and then incrementing it by equal steps up to the highest acknowledgement number). The expander also needs to recalculate the TCP/IP checksum of each extracted ACK. These ACKs are then forwarded to the original destination (i.e. the TCP sender). This process is simple to implement and requires little modification to the tunneling software at the remote site.

The previous simulations have been repeated, this time implementing ACK Compaction at the link layer. The low and high water marks were set to 4 and 60 ACKs respectively. Figure 3 shows the variation of *cwnd* with time for a data transfer from the hub server to a client. Figure 4 shows the time required for transfer of a number of bytes over the DVB link.

TCP throughput has been significantly improved using the ACK Compaction technique (figure 4). Our simulated implementation only compacts the ACKs when a threshold value of 4 ACKs is exceeded at the return interface queue. When compared with header compression, there is still an initial slow growth of *cwnd* (for the first 4 secs) for ACK compaction (figure 3 and 4). This arises since compressed ACKs are received more rapidly than the initially non-compacted ACKs. Once ACK compaction starts to take place, the performance rapidly exceeds header compression and completely eliminates ACK congestion. The simulation results suggest that this approach has significant advantages over the other techniques. Section 8 will also compare the performance of this technique with other return link optimization techniques, using a range of network topologies.

7.0 ACK SPACING

ACK Compaction and subsequent extraction generate a series of back to back ACKs and forward them to the TCP sender. This allows the sender to open *cwnd* within a very short period of time resulting in a transmission of a burst of packets to the network. This bursty nature of transmission could be alleviated by either modifying the current method of opening *cwnd* [3] or by spacing the generated ACK stream at the expander.

The former approach modifies the sender congestion control algorithm [25] to adjust *cwnd*, preventing the transmission of a burst of packets. On reception of a burst of ACKs, the sender will not open *cwnd* to its maximum possible value, instead it will smoothly increase within the next round trip time (the average round trip time (*srtt*) is usually cached by the sender). The sender will implement a timer for this purpose. The timer interval will be calculated using the maximum possible value of *cwnd* (according to the number of received ACKs and the unmodified congestion control algorithm), the current value of *cwnd* and *srtt*.

We have implemented an alternative ACK spacing technique (which does not need modifications to the TCP sender). The expander implements an algorithm similar to a leaky bucket traffic shaping algorithm [26]. The expander does not forward extracted ACKs immediately to the sender and instead they are stored in a queue. It also keeps a *token counter* that will be periodically incremented by a timer. Our implementation does not classify packets from different flows (sessions). If a classifier is also implemented, the expander may maintain a separate queue and a *token counter* for each flow. A received normal packet (not compacted) will also be stored in the same queue, but the expander increments the *token counter* without waiting for the next timer event. The number of ACKs forwarded to the sender is constrained by the available tokens. Every time an ACK is stored in the queue or when the timer expires (at the end of the period of the timer), and if tokens are available, ACKs are forwarded to the sender from the front of the queue (and an equivalent number of tokens are removed from the *token counter*).

The *token counter* is periodically incremented by a timer (ACK spacing timer). The period of the timer is calculated dynamically by measuring the inter-arrival time between two compacted ACKs. If all the extracted ACKs arrived well spaced, the inter-arrival time between each ACK (*ack_iat*) would be the measured inter-arrival time (between two compacted ACKs) divided by the number of ACKs that have been compacted into the received compacted ACK. To reduce the number of timer interrupts, *ack_iat* is multiplied by N for the calculation of the period of the ACK spacing timer (*ack_tov*). N is the number of tokens added to the *token counter* on expiry of the timer. Every time ACKs are forwarded to the sender, the expander (re) starts the timer, if it is not already running or the remaining time before the current timer due to be expired is greater than *ack_tov*. In our implementation, N is set to 4 or a lesser value equivalent to the number of ACKs remaining in the queue.

It is important to accurately estimate the initial inter-arrival time to space the first compacted ACK, since an underestimation may cause the sender to transmit a large burst of packets to the network or an overestimation may result in a slow growth of *cwnd*. The expander, therefore, continuously measures the average inter-arrival time between initial normal ACKs (not compacted). The average value is estimated using a recent window of N received ACKs. Prior to the compaction of ACKs (from the queue that is built at the return interface), the interface transmits packets at the line speed. The most recent estimated average value, therefore, represents the inter-arrival time for the first compacted packet.

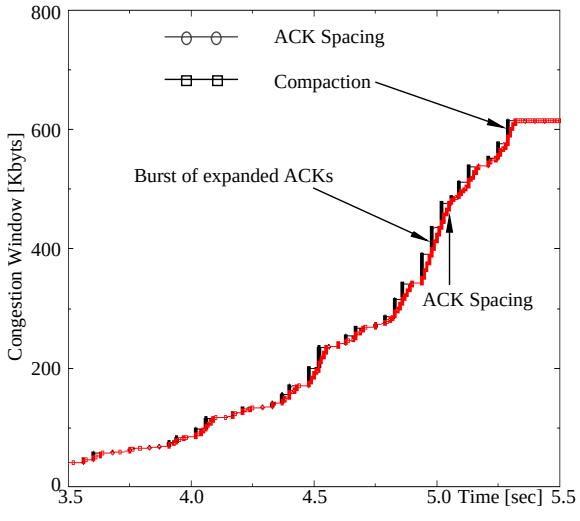

Figure 6: The variation of the *cwnd* with time when using Compaction and Compaction with ACK spacing

The previous simulations (with the same configuration) have been repeated, also implementing the ACK spacing algorithm at the expander. Figure 6 shows the variation of *cwnd* with time for a data transfer from the hub server to a client. As shown in the figure 6, ACK Compaction causes transmission bursts (abrupt increments of the *cwnd*, each represented by a horizontal line followed by a time gap). ACK spacing significantly reduces this by properly spacing extracted ACKs.

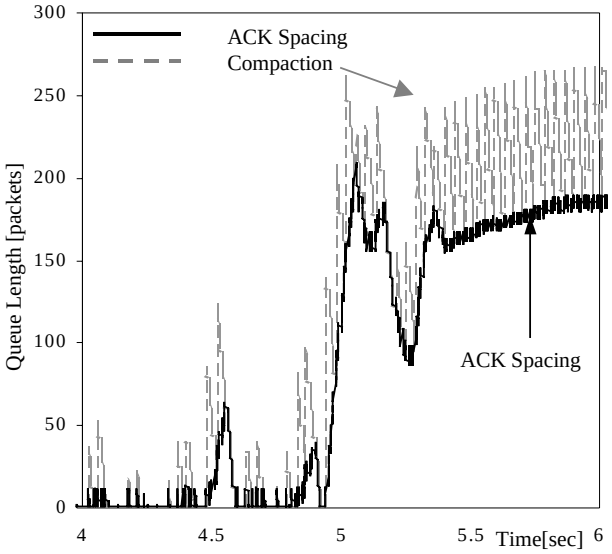

Figure 7: The variation of the queue length at the DVB satellite interface (when a single session is active).

Figure 7 shows the variation of the queue size at the satellite interface of the hub router during the previously simulated data transfer. ACK spacing has reduced the sharp peaks in the queue size (figure 7) resulting from transmission bursts caused by ACK Compaction.

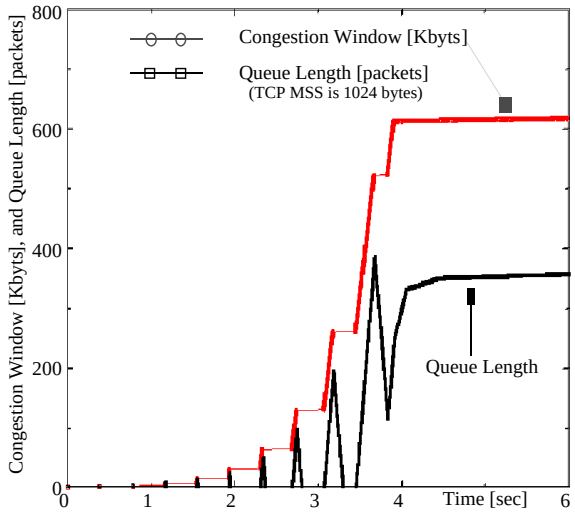

Figure 8: The variation of the queue length at the DVB satellite interface and the variation of *cwnd* with time (the return link speed is 430kbps).

However, a persistent queue of packets has been formed in the satellite interface (even after ACK Spacing). This is due to the transmission bursts caused by the slow start algorithm [27]. The previous simulations have been repeated using a high speed return link (sufficient to prevent ACK congestion) to explain this problem. The return link speed was 430kbps, but other configurations were remained without any change. Figure 8 shows the variation of the queue size at the satellite interface of the hub router and the variation of *cwnd* with time for a data transfer from the hub server to a client.

As shown in figure 8, every burst of ACKs received during the slow start period results in a queue of packets formed at the router. This is because the sender transmits two new segments for each received ACK, although the spacing between received ACKs roughly represents the available bandwidth of the bottleneck link. This results in transmission of packets at twice the speed of the bottleneck link, forming a queue of packets equal to half the bandwidth-delay product of the network at the end of the slow start phase [27]. A solution to this problem of overwhelming the bottleneck router due to the slow start procedure has not been addressed in this paper, but identified as future research.

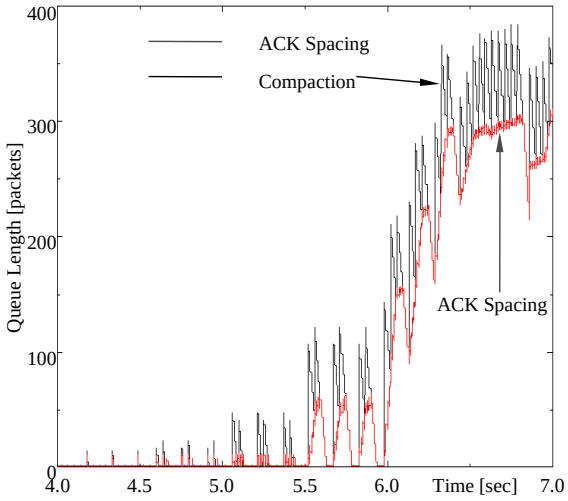

Figure 9: The variation of the queue length at the DVB satellite interface (when three sessions are sharing the satellite bandwidth).

The previous simulations (with the same configuration) have been repeated with three data transfer sessions sharing the DVB satellite link capacity (the return link bandwidth was again 9.6 kbps). Figure 9 shows the variation of the queue size at the satellite interface of the hub router for three simultaneous data transfers from the hub server to a client. The simulation results demonstrate the accuracy of the ACK Spacing algorithm even when a number of sessions are sharing the DVB network. (Suppression of sharp peaks due to ACK Compaction can be clearly noticed.)

Simulations were extended to examine the performance of the ACK Spacing algorithm in the presence of network congestion. This time, each session started with a time gap of 10ms (after the previous session has reached the bandwidth delay product of the network). Figure 10 shows the time required to transfer a number of bytes over the DVB link (the gradient of a curve represents the TCP throughput in Mbps). The network has been congested

resulting in some packet loss from the first session, when the second session started bursting packets to the network (figure 10). This resulted in a low throughput for the first session (the gradient of the curve has been reduced) and a fair sharing of the bandwidth between two sessions. When the third session started bursting packets to the network, the throughput of the first session has again been reduced, and the other sessions have gained the bandwidth.

This unfairness for the first session is due to the drop-tail queuing policy of the router. When a network queue is nearly full, subsequent packets from any session sharing the queue may be dropped (irrespective of the transmission rate of the session). This results in a low throughput for the session that experiences the packet loss due to the involvement of TCP congestion control and avoidance algorithms [25]. An implementation of a fair queuing algorithm [26] may alleviate such unfairness by (a) isolating different sessions sharing the resources of the router and (b) allocating a fair share of the total bandwidth to each session. Alternatively, the router may implement a fair packet discarding policy (e.g., ERD [23]), which increases the probability of losing packets from the sessions that overuse the bandwidth. Such a queuing technique is easier to implement in a video broadcasting network where the broadcaster has a better control over the transmission of data to the network and may also limit the number of simultaneous sessions sharing the network.

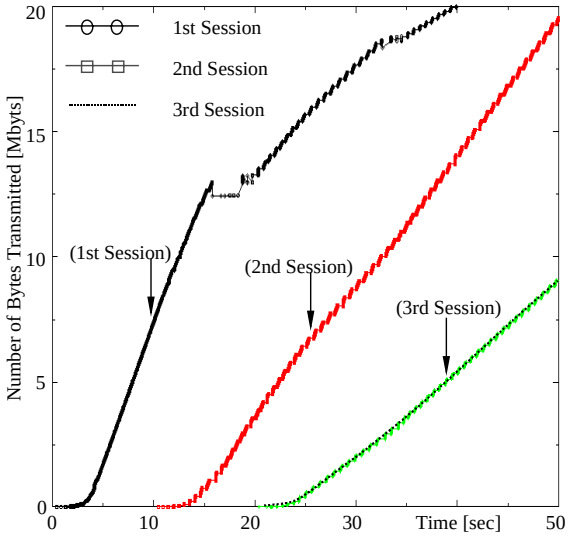

Figure 10: First sequence number of the transmitted packets against time, when three sessions are sharing the satellite bandwidth (the gap between starting time of each session is 10ms).

8.0 END TO END PERFORMANCE

The end to end performance of a range of techniques for optimization of the return path was studied using the same simulation configuration shown in figure 2. Three TCP sessions shared the DVB network, and it was assumed that 10 Mbps of the satellite bandwidth was allocated for data transfer. The set of simulations was repeated, varying the bandwidth of the return link, to study the performance over a range of network topologies. The satellite delay was 280ms, while the return path delay was 200ms. TCP MSS was 1024 bytes. The window scaling extension was enabled and had transmit and receive buffers sufficient for the bandwidth delay product of the DVB network.

Figure 11 shows the aggregated throughput for 3 TCP sessions over the DVB network. As explained in the previous sections, the TCP performance over a DVB network significantly suffers due to the bandwidth asymmetry, if the return link is not optimized (the graph labeled "Unmodified"). TCP/IP header compression improves the throughput by improving the data asymmetry, but the return link may still suffer from ACK congestion, since the bandwidth asymmetry is still higher than the data asymmetry.

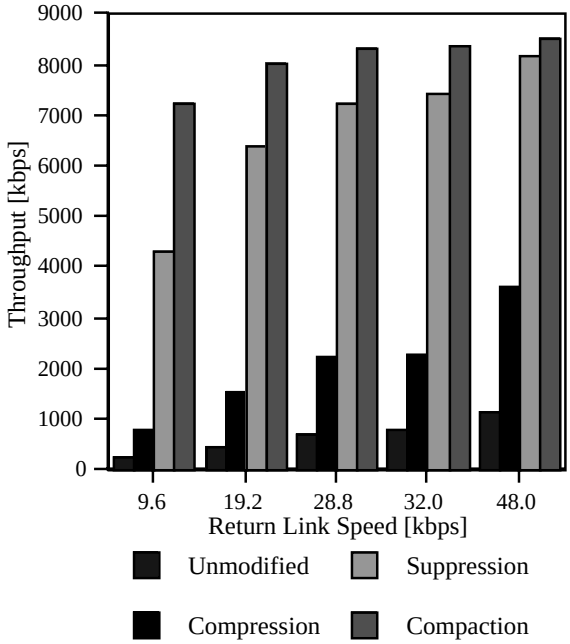

Figure 11: TCP throughput over a DVB network for a range of return path speeds (when using different return link optimization techniques).

Figure 12 shows the variation of the queue length at the return interface (only the case of a 32 kbps return link was considered for simplicity). Sharp glitches in the graph labeled "Unmodified" are due to the transmitter time outs which result from high variation of the queuing delay. TCP header compression prevents these timeouts by increasing the arrival rate of (compressed small) ACKs at the transmitter. However, the return link still suffers from ACK congestion (a persistent long queue of ACKs formed at the return interface), resulting in a low TCP throughput.

Both suppression and compaction eliminate queuing delay at the return interface (figure 12). This result was observed for most of the other network topologies. However suppression achieves lower throughput over low speed return links, since the growth rate of *cwnd* limits performance (figure 11). Even though a high-speed modem link can be used, the ISP may still need to provide a bandwidth (equivalent to the speed of the modem link) over the Internet. Compaction overcomes this limitation and achieves highest performance irrespective of the bandwidth of the return path (figure 11).

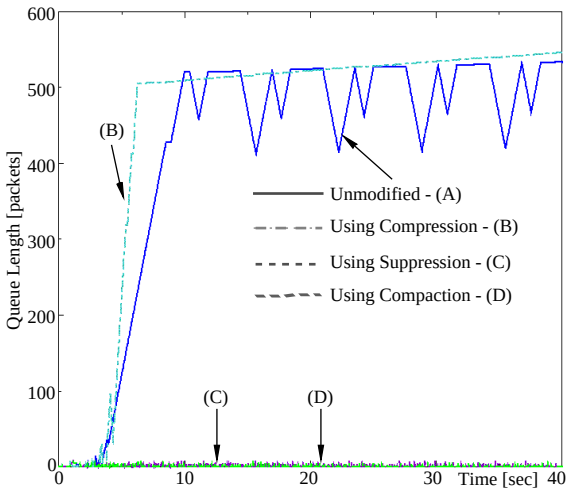

Figure 12: The variation of the queue length at the return interface of a client.

9.0 CONCLUSIONS AND FUTURE WORK

This study has investigated a range of different techniques to enable utilizing the full bandwidth of a satellite link when using a low speed return link. The bandwidth asymmetry in the forward and return paths may be higher than the asymmetry in the volume of the forward and return data leading to ACK congestion in the return link.

Link and TCP/IP header compression techniques reduce the volume of data over the return path, but they are typically designed for point-to-point modem links. Furthermore, since the data is uncompressed at the Internet Service Provider (ISP), the ISP may still need to guarantee a high bandwidth over the Internet. Therefore, even if these techniques could be used for this type of link, it would not provide sufficient gain to allow bulk transfers to take full advantage of the available satellite bandwidth.

ACK Suppression achieves higher TCP performance by significantly reducing queuing delay. However, since the actual number of ACKs received by the sender is significantly reduced due to suppression, the growth rate of *cwnd* limits performance, especially when the return bandwidth is very low (e.g., a congested Internet path).

The new ACK Compaction technique with the ACK Spacing achieve the highest performance by maintaining an acceptable arrival rate of ACKs at the sender and preserving TCP *self-clocking*. An implementation would require only modifications to network driver software at both ends of the return tunnel (i.e., at the client and the hub station).

The techniques may not interact with any security service implemented over the DVB link. We are currently investigating an appropriate implementation of security functions at the return interface of the DVB receiver. Alternatively, the security functions may be implemented at the transport layer (e.g., [28]).

10.0 ACKNOWLEDGMENTS

The Author wishes to thank the European Space Agency (ESA), and R Donadio (ESA), P. Glover (ESA), K. Hodson, R Heron (Delta Communication), R. Eley C. Yildez (Globecast Northern Europe) for their contribution to this project. This work greatly benefited from valuable comments from the CCR editor, the CCR reviewer and Gorry Fairhurst.

11.0 REFERENCES

[1] European Telecommunication Standards Institute, 'Digital Video Broadcasting (DVB) specification for data broadcasting', *ETSI*, Draft EN 301 192 V1.1.1 (1997-08), 1997.

[2] R. Braden, 'Requirements for Internet Hosts - Communication Layers', *IETF*, RFC 1122, October 1989.

[3] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, 'The Effects of Asymmetry on TCP Performance', *Mobile Computing and Networking (MobiCom)*, ACM/IEEE, Hungary, 13 (1997).

[4] R. C. Durst, G. J. Miller, and E. J. Travis, 'TCP Extensions for Space Communications', *MobiCom*, ACM/IEEE, USA, 15-26 (1996).

[5] N. Samaraweera and G. Fairhurst, 'High Speed Internet Access Using Satellite-Based DVB Networks', *International Network Conference (INC'98)*, IEE, Plymouth, UK, 23-28 (1998).

[6] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*, Vol:**1**, Chapman & Hall, USA, 1997.

[7] DVB, 'Digital Video Broadcasting - www.deb.org/dvb_articles/dev_datacasting.htm',

[8] E. Duros, W. Dabbous, Sophia-Antipolis, H. I. Noboru, and Y. Zhang, 'A Link Layer Tunneling Mechanism for Unidirectional Links', *IETF*, draft-ietf-udlr-lltunnel-00.txt, Work in progress, March 1998.

[9] S. Casner and S. E. Deering, 'First IETF Internet Audiocast', *ACM Computer Communication Review*, **22**(3), 92-97 (1992).

[10] A. D. Falk, V. Arora, N. Suphasindhu, and D. Dillon, 'Hybrid Internet Access', *NASA Centers for Commercial Development Of Space*, American Institute of Physics, New York, 69-74 (1995).

[11] W. R. Stevens, *TCP/IP Illustrated: The protocols*, Vol:**1**, Addison Wesley, New York, 1994.

[12] M. Allman, *et al.*, 'Ongoing TCP Research Related to Satellites', *IETF*, draft-partridge-tcpsat-res-issues-05.txt, Work in progress, 1998.

[13] S. Floyd, M. Allman, and C. Partridge, 'Increasing TCP's Initial Window', *IETF*, Draft-floyd-incr-init-win-00.txt, Work in progress, July 1997.

[14] M. Allman, 'On the Generation and Use of TCP Acknowledgments', *Submitted to ACM Computer Communication Review*, SIGCOMM, July 1998.

[15] T. Shepard and C. Partridge, 'When TCP Starts Up With Four Packets Into Only Three Buffers', *IETF*, Draft-shepard-tcp-4-paclet-3-buff-00.txt, Work in progress, July 1997.

[16] A. V. Bakre and B. R. Badrinath, 'Implementation and Performance Evaluation of Indirect TCP', *IEEE Transactions on Computers*, **64**(3), 260-278 (1997).

[17] C. Partridge and T. Shepard, 'TCP/IP Performance over Satellite Links', *IEEE Network*, **11**(5), 44-49 (1997).

[18] G. Held and T. R. Marshall, *Data Compression*, 3 ed, John Wiley & Sons Ltd, New York, 1991.

[19] V. Jacobson, 'Compressing TCP/IP headers for low-speed serial links', *IETF*, RFC1144, February 1990.

[20] N. Samaraweera and G. Fairhurst, 'Reinforcement of TCP Error Recovery for Wireless Communication', *ACM Computer Communication Review*, **28**(2), 30-38 (1998).

[21] W. R. Stevens, 'TCP Slow Start, Congestion Avoidance, Fast Retransmission, and Fast Recovery Algorithms', *IETF*, RFC 2001, January 1997.

[22] M. Degermark, B. Nordgren, and S. Pink, 'IP Header Compression', *IETF*, draft-degermark-ipv6-hc-06.txt, Work in progress, June 1998.

[23] S. Floyd and V. Jacobson, 'Random Early Detection Gateways for Congestion Avoidance', *IEEE/ACM Transactions on Networking*, **1**(4), 397-413 (1993).

[24] V. Paxson, 'End-to-End Internet Packet Dynamics', *SIGCOMM 97*, ACM, France, 139-152 (1997).

[25] V. Jacobson, 'Congestion Avoidance and Control', *SIGCOMM '88*, ACM, Stanford, USA, 314-329 (1988).

[26] C. Partridge, *Gigabit Networking*, Addison-Wesley Publishing Company, 1993.

[27] C. Partridge, 'ACK Spacing for High Delay-Bandwidth Paths with Insufficient Buffering', *IETF*, draft-partridge-e2e-ackspacing-00.txt, Work in progress, July 1997.

[28] T. Dierks and C. Allen, 'The TLS Protocol Version 1.0', *IETF*, drat-ietf-tls-protocol-05.txt, Work in progress, November 1997.