TECHNICAL RESEARCH REPORT

An Architecture for Internet Service via Broadband Satellite Networks

by Vijay G. Bharadwaj, John S. Baras, Norman P. Butts

CSHCN T.R. 99-12
(ISR T.R. 99-22)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

An Architecture for Internet Service via Broadband Satellite Networks *

Vijay G. Bharadwaj and John S. Baras, University of Maryland †
Norman P. Butts, Lockheed Martin Global Telecommunications ‡

Abstract

High bandwidth satellites hold out the promise of a rapidly deployable communications infrastructure with a natural support for mobility. However, the Transmission Control Protocol, widely used in the Internet, performs poorly over satellite links, and this presents an obstacle to the deployment of such systems. We present an architecture that overcomes these problems and provides an approach to building complex heterogeneous networks from simple units. We also present some results from our initial implementation, which uses TCP connection splitting to improve TCP performance over satellite links.

1 Introduction

With the rapid advancement of computer technology, portable and handheld computing devices are becoming increasingly common. Along with this trend comes the demand for these devices to be connected to the Internet. At the same time, there is also a growing demand for Internet connectivity in regions of the world that do not possess a good pre-existing communication infrastructure. Satellites offer an attractive solution to both these problems. Mobility can be supported easily, and coverage can be extended relatively quickly, even to remote areas.

Such universal connectivity requires the widespread deployment of a single family of standard protocols to ensure seamless interoperability between various systems. The Internet protocol suite is a natural choice, having been widely deployed and shown to work well over a variety

*This work is part of a collaborative research effort between the University of Maryland and Lockheed Martin Telecommunications.

†These authors can be contacted at: Center for Satellite and Hybrid Communication Networks, University of Maryland, College Park MD 20742. Phone: (301) 405 7900. Fax: (301) 314 8586. Email: {vgb,baras}@isr.umd.edu

‡Address: Lockheed Martin Telecommunications, Interactive Technologies Center, O/GR-42, B/551, 1272 Borregas Avenue, Sunnyvale, CA 94089. Phone: (408) 543 3021. Fax: (408) 543 3370. Email: norm.butts@lmco.com

of conditions and links. An important member of this protocol suite is the Transmission Control Protocol [1], which is an end-to-end transport protocol for reliable sequenced data delivery. Most Internet traffic uses TCP; examples are web traffic, email and file transfers. However, there are a number of problems with using TCP over satellite links, many of which arise from the high propagation delays inherent in satellite links.

In this paper we look at some solutions that have been proposed for these problems and describe an implementation of one such solution. We argue that building heterogeneous networks requires that we take into account the special characteristics of the underlying links, and that such knowledge can be used to improve the performance of protocols such as TCP. Section 2 outlines the problems with TCP and some of the proposed solutions. In Section 3 we describe our implementation of one of these solutions, and set out some of the design considerations that went into our implementation. Section 4 details some results obtained and Section 5 discusses some of the lessons learned and discusses some directions for future work.

2 TCP: Limitations and proposed solutions

TCP was designed as an end-to-end transport protocol that would handle wide network variation and run over many different kinds of networks, without having knowledge of the underlying network characteristics. Thus the design of TCP, especially in its flow control mechanisms, is very conservative. In the years since TCP was first proposed, many new link technologies have been developed, and added to the Internet. Many of these links have distinctive characteristics, and so the network as a whole has grown more and more heterogeneous. In the process, many cases have been found where the conservative approach of TCP leads to poor performance over certain types of links. At the same time, TCP has needed to stay conservative in order to handle the larger network variation, and so the gap between optimal performance and that obtained with TCP has widened in many cases.

2.1 TCP over satellite

The best-known shortcoming of TCP is that the offered window size field in the TCP header is only 16 bits long, which restricts its value to 64 kilobytes. Some implementations further limit the maximum window size to 32 kilobytes, and many popular implementations default to a window of 8 kilobytes. Since TCP cannot send more than one window of data per round-trip time, the maximum throughput attainable by a connection over a geostationary satellite link, which has a delay of about 250 ms in each direction, may be restricted to as low as 128 kbps. However, simple solutions exist for this problem, and are discussed in the next subsection.

Harder problems are raised by the flow control and congestion control mechanisms in TCP. TCP is a self-clocked protocol - the sender uses the stream of acknowledgments from the receiver to time its transmissions. This "ACK clocking" reduces the complexity of the sender at the expense of using up more bandwidth on the return channel to send frequent acknowl-

edgments. It also leads to unfairness between connections that traverse widely differing paths in the network - connections with smaller round-trip times can increase their rate of sending more rapidly, and so end up capturing most of the network bandwidth, at the expense of long-delay connections [2].

Related problems arise due to the slow start and congestion avoidance algorithms. Due to the small initial window in slow start, a significant number of round trips may be required for the congestion window to grow large enough to effectively utilize the link bandwidth. This is a problem in the satellite environment, where the round trip delays are long. A small web transfer, consisting of four or five packets, takes three round trips (1.5 seconds over a GEO satellite link) to complete, regardless of the link bandwidth available. Most data transfers over a satellite link can complete without ever having attained a window large enough for optimal link utilization.

In the congestion avoidance phase window growth is much slower than in slow start. However, even a single loss results in halving the window. This particularly affects bulk transfers. For these applications, when links are idle, a large window is desirable as it speeds up the transfer without harming other users of the link. However, if a loss occurs after the window has grown quite large, the window is halved and the satellite link is under-utilized for a prolonged period while TCP recovers and grows its window back to the former size.

In error-prone environments, especially those with bursty error characteristics, packet losses due to bit errors cause the sender's congestion window to be halved even though no congestion is present, thus under-utilizing the link. In addition, TCP's cumulative acknowledgment scheme can discover only one segment loss every round trip, so if multiple segments are lost in one window of data, throughput is reduced sharply.

2.2 Proposed solutions

In light of the above problems, a number of solutions have been proposed. These fall into three categories - link level solutions, end-to-end solutions, and proxy-based solutions. These categories of solutions are not mutually exclusive - all three kinds of solutions may be used together in a network.

Link level solutions include the use of link layer techniques like strong Forward Error Correction (FEC) and link-level Automatic Repeat Request (ARQ) mechanisms to mitigate the problem of corruption loss. In many situations, deploying these mechanisms can ensure that most losses seen by TCP are in fact due to congestion. However, these solutions do not address problems due to the TCP window size and congestion control mechanisms. The snoop protocol, as described in [3] also falls into this category - it basically tries to implement link level ARQ using TCP acknowledgments as the triggering mechanism. However, this protocol produces undesirable effects if it is deployed in the middle of a network, as it destroys the information that TCP uses for congestion control. The other link-layer solutions do not have this problem.

Many end-to-end solutions have been proposed, mostly as extensions to TCP, and a number of them have been adopted by the IETF as TCP options or enhancements. The window

scaling and timestamp options [4] allow TCP to use window sizes up to 30 bits wide. The use of larger initial congestion windows can mitigate problems due to slow start [5]. The Selective Acknowledgment (SACK, [6]) allows the receiver to return more information in its acknowledgments, and so addresses the problem of multiple losses in a window. The use of these options is good and should be encouraged. However, some of the options require additional complexity and state information at the TCP layer, and this makes them impractical to implement on small embedded systems. Further, these options do not address some of the problems pointed out in the previous section, such as the high penalties imposed by the congestion control algorithms for packet losses on connections using satellite links.

Even these remaining problems can doubtless be solved by implementing further enhancements to TCP. However, there is another, more philosophical, objection to adopting this approach. One of the major motivations behind the design of TCP was that by using an end-to-end approach that ignored link characteristics, a much simpler protocol design could be obtained. The fact that such an approach was suboptimal was considered an acceptable tradeoff, especially as the performance penalty was relatively low for more homogeneous networks. Today, as TCP becomes more and more complex in order to accomodate network heterogeneity while at the same time staying independent of link characteristics, it becomes important to ask if an approach that used a knowledge of link characteristics might not actually provide a simpler and more optimal solution.

Proxy-based architectures seem to provide such a solution. In this approach proxies are deployed in the network to separate links or groups of links with highly dissimilar characteristics. These proxies can then take advantage of their knowledge of link characteristics, while isolating the end hosts from these details. This allows simplification of the protocols used in the end-user terminal, at the expense of additional complexity in the network. Since the proxies are designed to take advantage of local network characteristics, we can obtain closer-to-optimal performance than with the end-to-end approach.

This approach also greatly simplifies the architecture of heterogeneous networks. Complexity is confined to those regions of the network where it is needed without affecting the rest of the network or the end user. It is no longer necessary to design protocols as conservative as TCP; new types of links can be added by just implementing appropriate proxies around them, without changing the rest of the network. Hence this architecture allows networks to be extended in an incremental fashion without affecting interoperability.

Connection splitting proxies [7] belong to this class of solutions. In the case of a satellite link, a connection splitting proxy would provide a TCP proxy for the remote host and use a completely different protocol to send the data reliably over the satellite link. Since this protocol can be optimized for the satellite link, all the problems in the previous subsection can be solved. In the following sections we shall describe the design of such a system and the results obtained from it, and present some suggestions for future work.

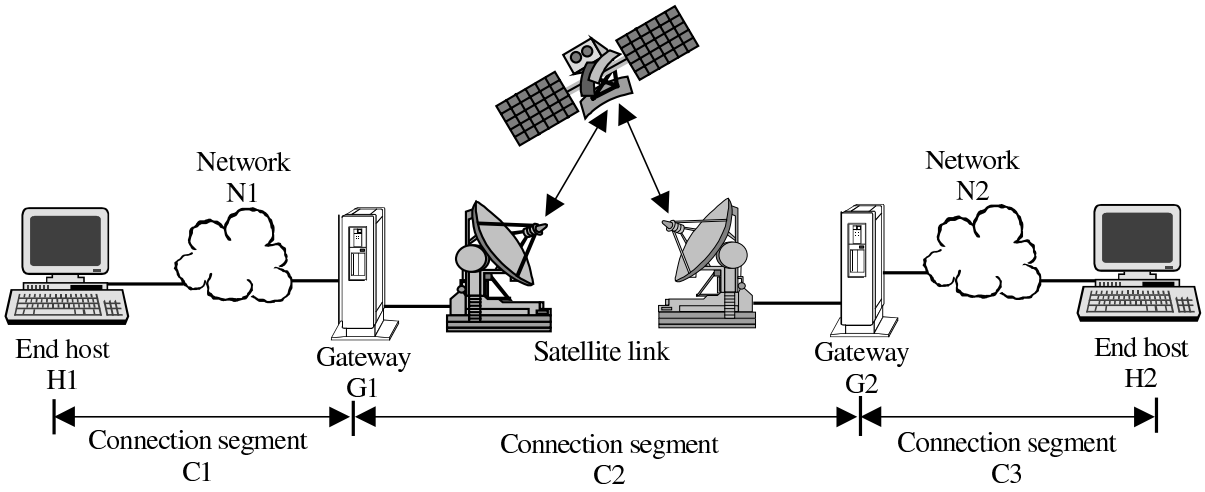


Figure 1: Overview of TCP connection splitting.

3 Connection splitting

An example of the connection splitting architecture applied to satellite links is shown in Figure 1. The end-to-end TCP connection between H1 and H2 is broken into three separate connections, namely C1, C2 and C3, by the gateways G1 and G2. C1 and C3 use TCP, while C2 may use a different protocol, optimized for the satellite link. The splitting is achieved by having each of the gateways transparently act as a TCP proxy for the remote host, thus isolating the end hosts from the characteristics of the satellite link. For instance, having the gateways acknowledge data on behalf of the remote host reduces the connection round trip time seen by the end hosts. The use of such proxies allows the end hosts to implement very simple versions of TCP, as they will only be communicating over a relatively simple network, with the gateway. It also allows the gateway to optimize the transfer taking into account the nature of the satellite link.

3.1 Design considerations

We now look at the problem of designing gateways like those in Figure 1. We would like to improve the throughput for TCP connections using the satellite link while allowing the end hosts to continue using TCP as before. Since TCP requires ACKs for its clock, we must ensure that when the satellite link is uncongested, the sender TCP receives a stream of ACKs that is similar to what it would receive if it was communicating with a terrestrially-connected host over an uncongested link of similar bandwidth. In other words, we want to decouple the ACK clock, which is supposed to provide flow control by representing the state of congestion in the network, from the link delay, which is a characteristic of the link. One way to do this is to have the gateway acknowledge data as soon as it receives it, and to perform flow control by reducing the offered window when the satellite link is congested, thus slowing down the sender.

A TCP host uses the offered window to compute the largest sequence number of data that it

can send without further acknowledgment. This value is computed by adding the sequence number acknowledged and the window offered in a TCP ACK segment, and is known as the "right edge" of the window. The receiver may exert flow control by not emptying its receive buffer fast enough, thus allowing the offered window to grow smaller with each segment received. "Shrinking the window", which involves reducing the offered window by more than the amount of data received, so that the right edge of the window is also reduced, is deprecated in TCP [8]. Therefore TCP does not offer a way for the receiver to rapidly throttle the flow of data from the sender. An alternative method of throttling the sender is to shrink the congestion window on the sender by causing a segment loss when the sender is sending at too high a rate.

TCP is designed to run on top of IP, which is a connectionless network protocol. It is robust to routing changes and reordering of segments in the network, and our proxy implementation should have these properties as well. Use of the proxy must not cause failure or data loss when network routing changes or when routes in the two directions are different. The TCP standard specifies that if a host receives a segment with an invalid sequence number it must acknowledge the segment and drop it. Therefore to avoid data loss in case of routing changes our gateways should ensure that TCP sequence numbers used on C1 and C3 are identical. The initial exchange of SYN segments by two hosts at connection setup time establishes the synchronization in sequence numbers. Therefore the gateway must use the information in the SYN exchange to synchronize itself with the sequence numbers on a connection. If due to routing changes or other reasons (such as IP layer encryption) this information cannot be acquired, the gateways should at least be capable of simply forwarding all subsequent segments on that connection.

Similarly, port numbers must also be preserved by the gateways, as many services use them as an authentication mechanism.

The gateway must not return a SYNACK to the host before the remote host has responded. Such a response would imply that the remote host is functional and is sure to accept the connection. If this does not happen, then the gateway will have to abort the connection, thus causing the user on the end host to see a difference in behavior when the proxy is used. It would also cause the two end hosts to be in a combination of states that is not valid as per the TCP specification, opening up the possibility of failure if routing changes should occur or asymmetric routes be found. Therefore the gateway must only return a SYNACK after the remote host has accepted the connection.

A similar statement might be made about the FIN sent to indicate a half-duplex close on a connection. However, the standard API for TCP does not provide a way for an application to find out whether a FIN has been successfully acknowledged by the remote end. Therefore for simplicity of implementation it might be desirable to acknowledge a FIN as soon as it is received by the gateway.

The earliest reported implementations of connection splitting did not include any flow control for the satellite link. In fact, no flow control is necessary if the satellite is a "bent pipe" system, since the necessary rate limitation can be imposed by the link layer itself. However, in case the satellite has multiple spot beams and onboard switching capabilities (as is the case with many of the proposed Ka band satellites) some flow control method is necessary

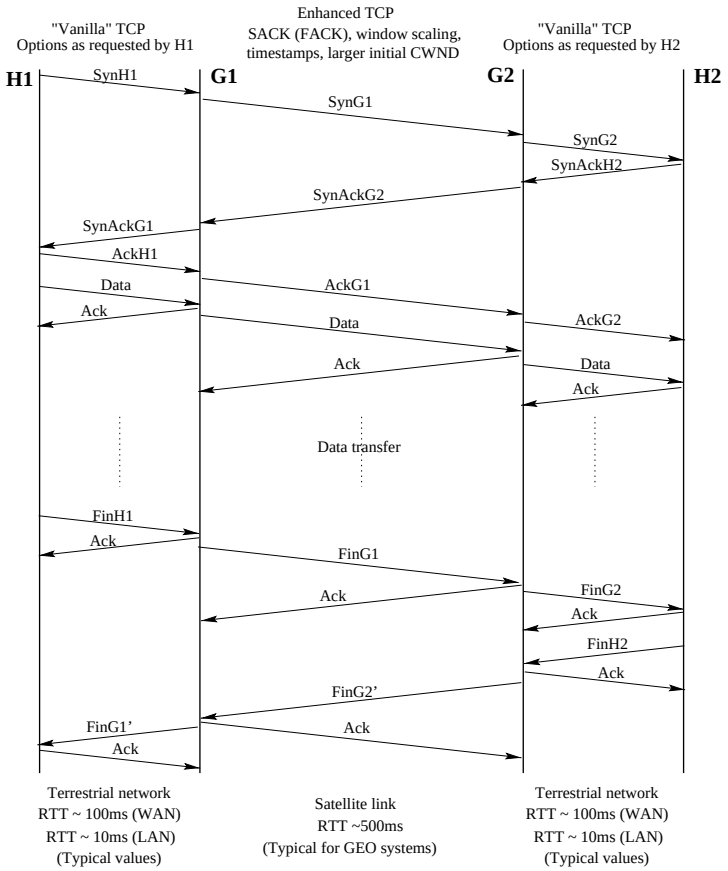


Figure 2: Timing diagram for simple unidirectional transfer from H1 to H2.

to prevent congestion on the satellite switch.

3.2 Implementation description

Our implementation of a connection splitting proxy for satellite links was designed keeping the above concepts in mind. We used TCP, enhanced with timestamp, window scaling and SACK options, on the satellite link. This TCP implementation also uses the FACK [9] congestion control algorithm, and an increased value for the initial congestion window during connection startup. We chose TCP mainly to speed up the implementation process; better alternatives for the satellite link are under investigation, as discussed in the following section.

The procedure used to perform connection splitting is as shown in Figure 2. Whenever a gateway sees a connection request (i.e. a SYN segment), it intercepts the request and originates a similar connection request with an enhanced option set. When all downstream connections are completed, an acknowledgment (i.e. a SYN ACK) is returned to the host that originated the original request. Both the gateways always negotiate and accept all the TCP options listed above during connection setup, so that the connection between G1 and G2 will always use all these options.

Once the connection has been set up, the gateway intercepts all data on that connection,

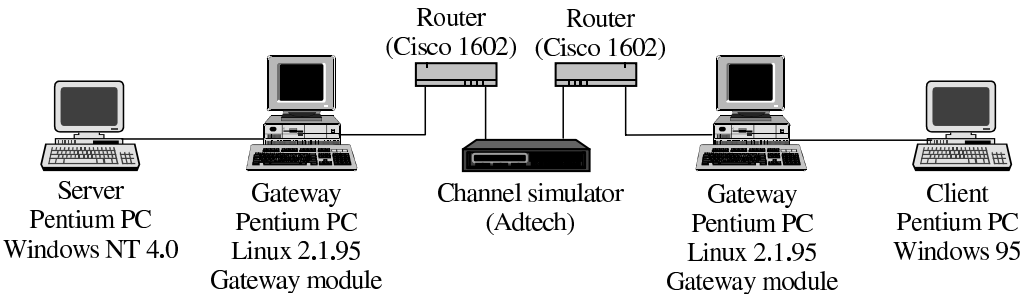


Figure 3: Test configuration.

returns an acknowledgment to the sender bearing the address of the destination, and buffers the data for downstream transmission. When a gateway receives a FIN segment, it immediately closes the corresponding half-duplex connections. When a FIN has been received for both directions of a TCP connection, all the resources for the corresponding connection segments are freed to minimize resource usage.

As described in Section 3.1, all sequence numbers and port numbers are preserved, and packets received on unknown connections are simply forwarded.

Note that there is a variable delay due to buffering at G1 as well as G2, which is not shown in Figure 2. Also, a host may choose to piggyback ACKs on other kinds of packets. In the figure, AckH1 may be piggybacked on the data following it, and FinH2 may be combined with the ACK immediately preceding it.

During the lifetime of a connection, a "back-pressure" algorithm is used for flow control; incoming segments on the upstream connection are served at a rate that matches the rate of transmission on the downstream connection. Thus when the downstream path gets congested, the offered window on the upstream connection reduces correspondingly, and congestion information is propagated back to the sender.

An additional mechanism is used to limit the size of the buffers at the gateways. If data is arriving on a connection at a much higher rate than it can be sent out, and if a large enough amount of data is already buffered for that connection, then an arriving packet is discarded without acknowledgment. This causes the sender to retransmit the segment and reduce its congestion window, and so keeps the buffers small without appreciably affecting end-to-end throughput. The implementation is careful not to drop more than one packet per window, to avoid causing serious performance degradation.

4 Performance measurements and analysis

4.1 Test methodology

Our current implementation of the transparent TCP gateway functionality runs on the Linux operating system, kernel version 2.1.95. Though the gateway module is processor-independent, all the tests have been carried out on a pair of Pentium PCs running at 166

MHz. The test configuration is shown in Figure 3. The server machine was running Microsoft Windows NT Workstation 4.0 with the default TCP/IP parameters, while the client machine was running Microsoft Windows 95 with the default parameters. We used the FTP server and the HTTP server from the NT Peer Web Services software. The client for the FTP testing was the standard FTP client supplied with Windows 95, while the HTTP client was Netscape Communicator 4.05. A data channel simulator was used to simulate the satellite channel.

We measured throughput for single FTP connections using different file sizes, with different data rates, delays and error rates on the simulated satellite link. There was no other traffic on the link. Files of sizes 10 KB, 100 KB, 1000 KB, 10000 KB and 100000 KB were tested. The link rates used were 384 kbps, 1.536 Mbps and 8 Mbps. Tests were carried out for zero delay and for a delay of 250 ms each way on the link. These delays were chosen to approximate a terrestrial leased line and a typical geostationary satellite link respectively. Throughput was measured for bit error rates of 0, 10^{-9}, 10^{-8}, 10^{-7} and 10^{-6}. To provide a baseline for comparison, identical tests were carried out with the gateway machines functioning as IP routers without any connection splitting.

HTTP tests using various kinds of webpages were carried out over the same range of conditions. In this case we measured the total time required to load a page. Throughput is not meaningful in this situation, since the request-response mechanism of HTTP causes unavoidable periods of almost zero link utilization.

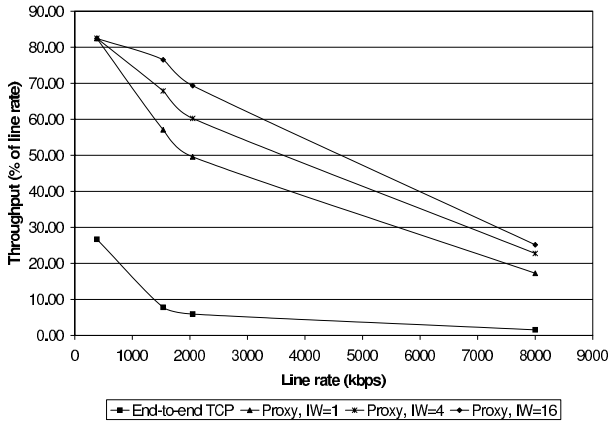
We repeated the above tests using a commercial Ku band satellite. The results were similar to those obtained using the channel simulator with no injected errors. Overall, our results indicate that bit errors are not a problem on typical commercial Ku band satellite systems.

4.2 Results obtained

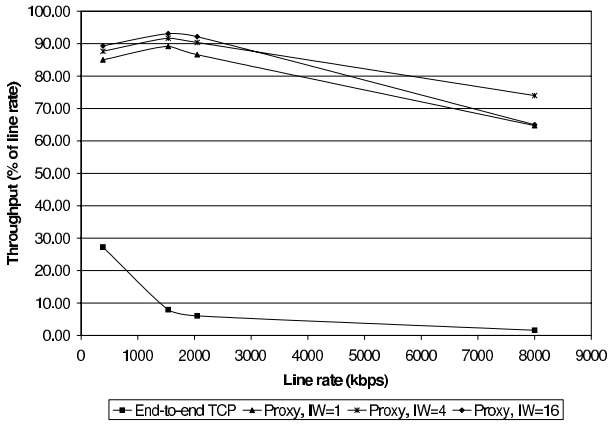
Figure 4 shows the throughput obtained when a single FTP session runs over an error-free link with a 250 ms delay in each direction. The end-to-end TCP transfer is limited to a constant maximum transfer rate, independent of link bandwidth, by the fact that its offered window is no more than 16 bits long. Thus its percentage utilization decreases with increasing link bandwidth. The connection splitting approach, which uses much larger window sizes on the satellite link, yields better throughput, and always uses a large fraction of link bandwidth.

The decrease in utilization at higher link rates for the 1 Megabyte transfer in the connection splitting case is mainly due to slow start. With smaller file sizes, the TCP on the satellite link does not attain a large enough congestion window to achieve a rate of transmission sufficiently close to the link rate, and lower utilization is observed. As the file size is increased, the cost of low utilization during slow start is amortized over a longer interval of near-line-rate transfer, so the utilization improves. This phenomenon motivates the use of a different protocol from TCP on the satellite link.

Figure 5(a) illustrates this effect. The curve has a distinct knee at approximately the delay-bandwidth product of the satellite link. The throughput with connection splitting improves sharply when the file size is an order of magnitude greater than the delay-bandwidth product

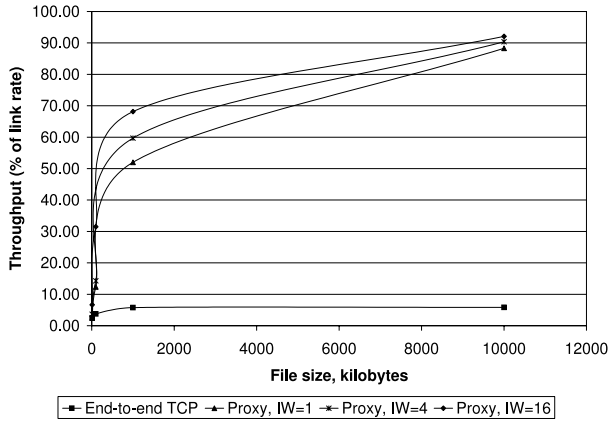


(a) File size = 1 Megabyte

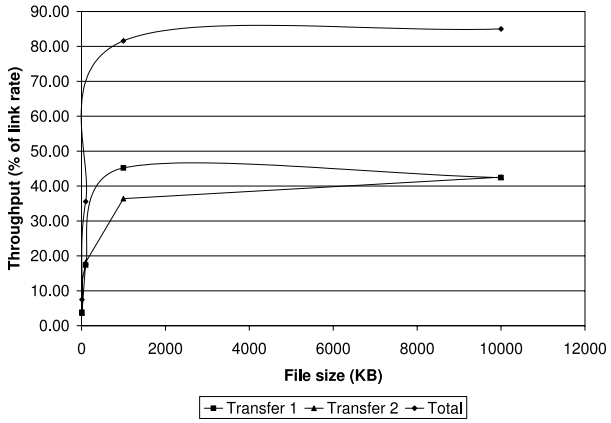


(b) File size = 10 Megabytes

Figure 4: Effect of delay on throughput during FTP transfers. Delay=250 ms each way, from channel simulator. No errors injected.



(a) Single FTP session.



(b) Two simultaneous FTP sessions.

Figure 5: FTP Performance over Ku band satellite link. Link rate on the satellite was E1 (2.048 Mbps). During this test, the satellite modems reported raw bit error rates of 10^{-6} to 10^{-5}, which was reduced to 10^{-12} by Forward Error Correction.

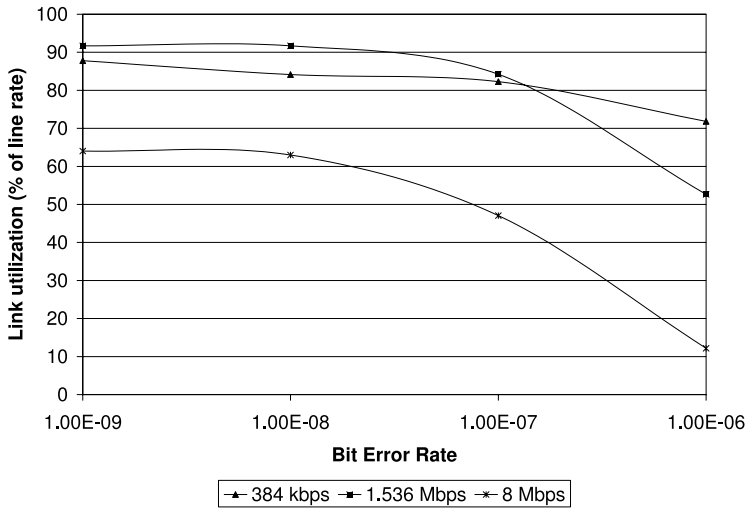


Figure 6: Split connection performance under different BER. Delay=250 ms each way, from channel simulator. File size=10 Megabytes.

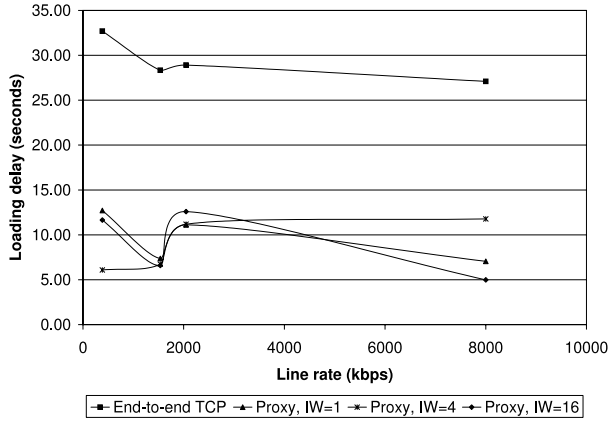
of the link. Thus retaining the TCP congestion control mechanisms on the satellite link carries a penalty for small transfers. This problem is discussed further in the next section.

We also see from the figure that increasing the initial congestion window for slow start improves throughput for small sizes by mitigating the effect of slow start, but has little effect for larger file sizes.

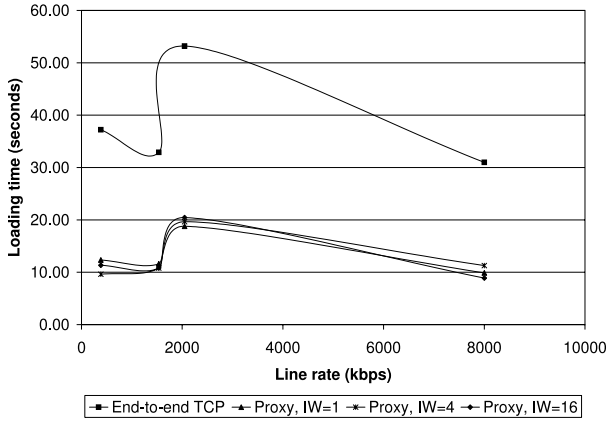
Figure 5(b) shows the performance achieved by two identical simultaneous FTP connections sharing the satellite link. We see that for large enough file sizes, link utilization is still high, with each of the two connections getting a roughly equal share of bandwidth.

Figure 6 shows the link utilization achieved by the split-connection system when bit errors are present on the satellite link. It is observed that even at high bit error rates like 10^{-6}, the split-connection system performs better than end-to-end TCP does in the absence of bit errors. This is mostly due to our use of SACK information and the FACK algorithm on the satellite link. Since the susceptibility of TCP to errors (in terms of reducing the congestion window) depends on the number of errors per round trip time, throughput is more affected at higher link rates. Throughput is relatively unaffected for small files, which are less likely to encounter any bit errors in transmission, whereas large files are affected considerably. We see from the figure that performance drops sharply when the error rate is close to one error per round trip.

HTTP uses a request-response mechanism, wherein the client requests one object at a time from the server. Thus there is an interval of one round trip between the time that the client finishes receiving an object and the time that it begins to receive the next object. During this time, there is no traffic on the link except for the request by the client. Therefore it is meaningless to compute the efficiency for such transfers, as the traffic generated is intermittent in nature, with long pauses. We measured the total time required for each webpage to load, as measured by a stopwatch. The results for two sample webpages are shown in Figure 7. As seen in the graphs, there is a considerable improvement when connection splitting



(a) Webpage composition: 356 bytes HTML document + single 391 KB image.



(b) Webpage composition: 1.7 KB HTML document + 16 images (total 669 KB). Image sizes range from 19 KB to 100 KB.

Figure 7: HTTP Performance over simulated satellite link for two sample webpages. No errors were introduced by the simulator.

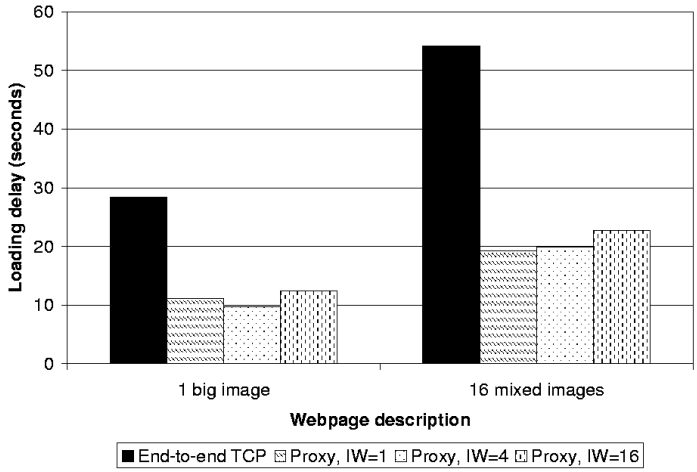


Figure 8: HTTP Performance over Ku band satellite link. Link rate on the satellite was E1 (2.048 Mbps). During this test, the satellite modems reported raw bit error rates of 10^{-6} to 10^{-5}, which was reduced to 10^{-12} by Forward Error Correction.

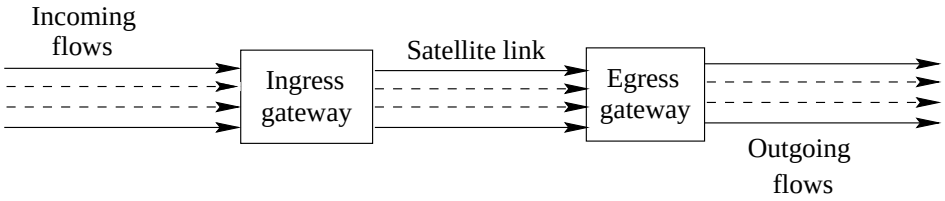


Figure 9: Sketch of flow control problem

is used. However, the pauses between the HTTP requests are the major limiting factor, so increasing the link rate or the initial congestion window does not improve performance.

Figure 8 shows the results obtained from HTTP tests over a Ku band satellite link. Once again, using connection splitting yields an improvement, while increasing the initial congestion window does not help much.

5 Discussion: The need for intelligent flow control

During our experience with implementing the gateways and testing them, we found a few areas that need improvement and further research. We observed that the throughput is highly dependent on the TCP buffer sizes defined on the gateways and the number of connections. TCP flow control over satellite links works much better, even in the absence of connection splitting, if the offered window is set to the bandwidth-delay product available to the connection. If the offered window is smaller than this, throughput is unnecessarily low. If the offered window is larger, the sender's congestion window will eventually grow larger than the network can support, and a packet loss will occur. At this point the sending TCP halves its window, and enters congestion avoidance. Since the window was fairly large when the loss occurred, it takes a long time for the window to grow back to its original size. Thus TCP follows an inefficient cycle of window growth, congestion and backoff, which reduces throughput and causes unnecessary retransmissions.

A related problem specific to connection splitting is that for bulk transfers, the TCPs on the satellite link try to share the link equally between them. However this is not necessarily desirable. For example, some connections have faster downstream links than others, and can be given a larger share of the bandwidth on the satellite link without adversely affecting the end-to-end transfer rate of other connections. This kind of proportional sharing does eventually come about due to back-pressure (see Section 3.2) on the slower connections, but it takes a long time, and end-to-end throughput is reduced. Another danger is that if a gateway buffers too much it may be prone to stalls like those reported with the naive implementation of connection splitting in [10].

Both the above problems can be solved by implementing some kind of adaptive flow control between connection segments. Providing intelligent flow control mechanisms at the gateways to match their input rates to their output rates is thus an interesting problem. By solving this problem, we hope to gain some insight into improving flow control in end-to-end TCP as well.

A sketch of the flow control problem is shown in Figure 9. It is important to recognize that the performance of the satellite link as perceived by the user depends on the total output rate over all the outgoing flows. Therefore we need to share the bandwidth on the satellite link among the flows in such a way that the total output rate of the egress gateway is maximized. This will ensure low queueing in the egress gateway and optimum utilization of the satellite link. It will also enable the ingress gateway to provide feedback to the sender to effectively control the sending rate, and hence queueing at the ingress gateway.

Now, in the simplest case, the egress gateway is using TCP on its outgoing link. Since the growth of the TCP window in a given time is bounded, we can use the congestion window and round trip time estimate on the downstream connection to estimate the maximum sending rate that will be available at a given time in the future. So a simple scheme to try and achieve optimal flow control in our setup is for the egress gateway to advertise two kinds of windows in its acknowledgments to the ingress gateway. One would be a soft limit based on the current window on the downstream connection. The other would be a hard limit, similar to the usual TCP offered window, based on the amount of buffer space available at the egress gateway. Then, if the ingress gateway sends data that fits in the "soft window" in preference to data outside it, a bandwidth allocation is achieved that is optimal in the sense described above.

If the receiver can identify corruption losses, this scheme would allow us to completely eliminate the slow start and congestion avoidance algorithms. The egress gateway would simply adjust its soft window when it detected losses that were not due to corruption, and would lower its hard window as well if losses persisted.

In addition, given some support at the sender, we can greatly reduce the frequency of acknowledgments. Acknowledgments only need to be sent when loss is detected, and at some regular intervals otherwise. Thus this scheme reduces the dependence of the flow control algorithm on the round trip time, by replacing the implicit feedback obtained from TCP acknowledgments with explicit feedback provided by the receiver.

We are currently studying this and other schemes for adaptive flow control over satellite links. Such a mechanism is expected to be an important part of any practical connection splitting system.

6 Conclusions and future work

In this paper we demonstrated an architecture based on connection splitting which uses transparent proxies to enhance TCP performance over satellite links. Such proxies are easy to incorporate into existing networks, and improve end user performance by a large factor in the presence of satellite links. The proposed architecture also allows for end-to-end congestion control and fair allocation of bandwidth among competing flows.

Throughput on satellite links is affected substantially by the TCP congestion control mechanisms. We are currently investigating more efficient protocols for these links. We are also studying strategies for carrying out end-to-end flow control in split connection systems.

Acknowledgments

The authors would like to thank Bob Randall, without whose expertise the satellite testing could not have been carried out, and Rohit Tripathi and Koroush Saraf for their assistance in carrying out the performance testing with the delay simulator. We are also indebted to Mingyan Liu and Manish Karir for their helpful comments and advice, which greatly improved this paper.

References

[1] J. Postel, ed. Transmission Control Protocol – protocol specification. *RFC 793*, September 1981.

[2] T.V. Lakshman and U. Madhow. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, June 1997.

[3] H. Balakrishnan, S. Seshan, and R. Katz. Improving reliable transport and handoff performance in cellular wireless networks‘. *ACM Wireless Networks*, 1(4), December 1995.

[4] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. *RFC 1323*, May 1992.

[5] Mark Allman, Sally Floyd, and Craig Partridge. Increasing TCP's initial window. RFC 2414, September 1998.

[6] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. *RFC 2018*, October 1996.

[7] N.P. Butts, V.G. Bharadwaj, and J.S. Baras. Internet service via broadband satellite networks. *Multimedia Systems and Applications: Proceedings of SPIE*, 3528:169–180, February 1999.

[8] B. Braden, ed. Requirements for Internet hosts – communication layers. *RFC 1122*, October 1989.

[9] M. Mathis and J. Mahdavi. Forward Acknowledgment: Refining TCP congestion control. In *Proceedings of SIGCOMM '96*, Stanford, CA, August 1996.

[10] Hari Balakrishnan, Venkat Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking*, December 1997.