INTERNATIONAL JOURNAL OF SATELLITE COMMUNICATIONS
Int. J. Satell. Commun. 2001; 19:29–50

An architecture for Internet service via broadband satellite networks

Vijay G. Bharadwaj[1,*,†], John S. Baras[1] and Norman P. Butts[2]

[1]*Institute for Systems Research, University of Maryland, College Park, MD, U.S.A.*
[2]*Lockheed Martin Global Telecommunications, U.S.A.*

SUMMARY

High bandwidth satellites offer the promise of a rapidly deployable communications infrastructure with a natural support for mobility. However, many widely used versions of the Transmission Control Protocol perform poorly over satellite links, and this presents an obstacle to the deployment of such systems. We present an architecture that overcomes these problems and allows easy integration of heterogeneous networks into the larger Internet. We also present some results from our initial implementation, which uses TCP connection splitting to improve TCP performance over satellite links. Copyright © 2001 John Wiley & Sons, Ltd.

KEY WORDS: satellite; TCP/IP; performance enhancing proxy; Internet; latency

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 of conditions and links. An important member of this protocol suite is the Transmission Control Protocol (TCP) [1,2], 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.

*Correspondence to: Vijay G. Bharadwaj, Institute for Systems Research, A.V. Williams Building, University of Maryland, College Park, MD 20742, U.S.A.
† E-mail: vgb@isr.umd.edu

Contract/grant sponsor: Lockheed Martin Global Telecommunications and NASA; contract/grant number: NCC-3528

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. 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 KB. Some implementations further limit the maximum window size to 32 KB, and many popular applications default to a window of 8 KB. 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.

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. This is a major problem in environments with bursty error characteristics, such as links prone to intermittent fading. Solutions have been proposed for this problem as well, as described 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 acknowledgments. 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

[3]. Such problems are studied further in Reference [4], and it is shown that this bias of TCP also leads to poor performance of TCP connections passing through multiple congested gateways relative to connections that cross one or no congested gateway.

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 s 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. Thus the costs of even a single 'false alarm' due to a corruption-based packet loss are very high. For bulk transfers over uncongested paths, if such 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.

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 Reference [5], also falls into this category—it basically tries to implement link level ARQ using TCP acknowledgments as the triggering mechanism. However, this solves the problem of TCP assuming that all losses are due to congestion by assuming instead that all losses are due to corruption. Thus, if the loss is actually due to congestion, snoop produces undesirable effects, and so it cannot be deployed in the middle of a network. The other solutions mentioned above 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 [6] allow TCP to use window sizes up to 30-bit wide. The use of larger initial congestion windows [7] can mitigate problems due to slow start. The selective acknowledgment (SACK) [8] 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 so may not have been implemented, for example, on small embedded systems. Further, some of these options are very hard to configure correctly on any given system. For example, the window scaling factor can only be negotiated at connection setup, when neither host has an estimate of the connection round trip time; unless some additional mechanisms are used to determine the RTT, the hosts can only guess at an appropriate scale factor. These options

also 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 corruption-based 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 accommodate 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 hosts from such 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.

Connection splitting proxies [9] belong to this class of solutions. In what follows we explore the capabilities and limitations of this architecture.

3. CONNECTION SPLITTING

Figure 1 illustrates the connection splitting architecture as applied to networks with satellite links. 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

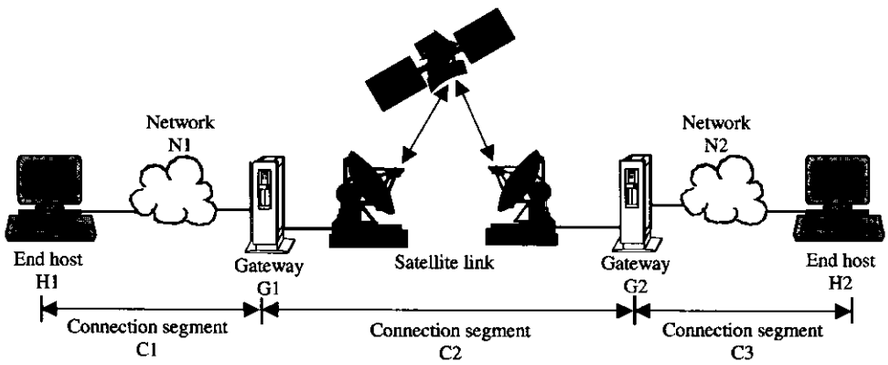

Figure 1. Overview of TCP connection splitting.

each of the gateways transparently act as a TCP proxy for the remote host, thus isolating the hosts from the characteristics of the satellite link. For instance, having each gateway acknowledge data on behalf of the remote host reduces the connection round trip time seen by the hosts. The use of such proxies allows the hosts to implement very simple versions of TCP, as they will only be communicating over a relatively simple network, with the proxy. It also allows the proxy to optimize the transfer taking into account the nature of the satellite link.

Various connection-splitting proxy designs have been proposed in the literature. One approach, specific to cellular networks, was proposed in Reference [10]. The authors implemented a version of TCP that used connection splitting to deal with host mobility. The major drawback of this solution was that it needed the new protocol to be deployed in all base stations and in the mobile hosts, and it required the applications on the mobile hosts to be modified to use the new API. Another more recent approach, specifically for satellite systems, is described in Reference [11]. Here the authors use an approach similar to ours, but define a new protocol for use over the satellite link.

Most connection-splitting implementations currently available either require changes to host software or fail to deal gracefully with exceptional circumstances. For example, they cause TCP connections to fail if network routing is asymmetric, that is if the routes in the two directions of a connection are different. Another common problem is that if one of the proxies responsible for connection splitting runs out of resources or fails, many undesirable effects are seen, especially if other routes still exist between the communicating hosts. We attempt to tackle these problems in our implementation.

Our approach was motivated by the observation that in homogeneous networks, TCP can nearly always be tuned to perform well, by including various enhancements and choosing parameters beforehand. Therefore, by choosing the locations of the connection-splitting proxies so that they partition the network into relatively homogeneous parts and using appropriately tuned TCP implementations within each part, it should be possible to achieve good performance for all network users. This would allow us to build on the widely available and extensively tested base of TCP implementations.

3.1. Design considerations

We now look at the problem of designing proxies like those in Figure 1. We would like to improve the throughput for TCP connections using the satellite link while allowing the 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 the satellite link were replaced by an uncongested terrestrial 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 proxy acknowledge data as soon as it receives it, and to perform flow control by allowing a backlog to accumulate in its receive buffer, thus reducing the offered window, when the satellite link is congested. This 'back-pressure' mechanism is a relatively slow way of throttling the sender—the receiver's buffer must fill up before the sender will stop sending. However, it is the only way a TCP receiver can slow down the sender without dropping packets, as 'shrinking the window' [12] is deprecated in TCP.

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 proxies 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 proxy 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 proxies should at least be capable of simply forwarding all subsequent segments on that connection.

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

The proxy 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 proxy will have to abort the connection, thus causing the end user to see a difference in behavior when the proxy is used. It would also cause the two 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 proxy 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 proxy.

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 [13] congestion control algorithm, and an increased value for the initial congestion window during connection startup. We chose TCP for its flexibility, as well as to speed up the implementation process. It is envisaged that any protocol modifications for the satellite link will be implemented as TCP options [14].

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 proxy intercepts all data on that connection, returns an acknowledgment to the sender bearing the address of the destination, and buffers the data for downstream transmission. When a proxy receives a FIN segment, it immediately closes the

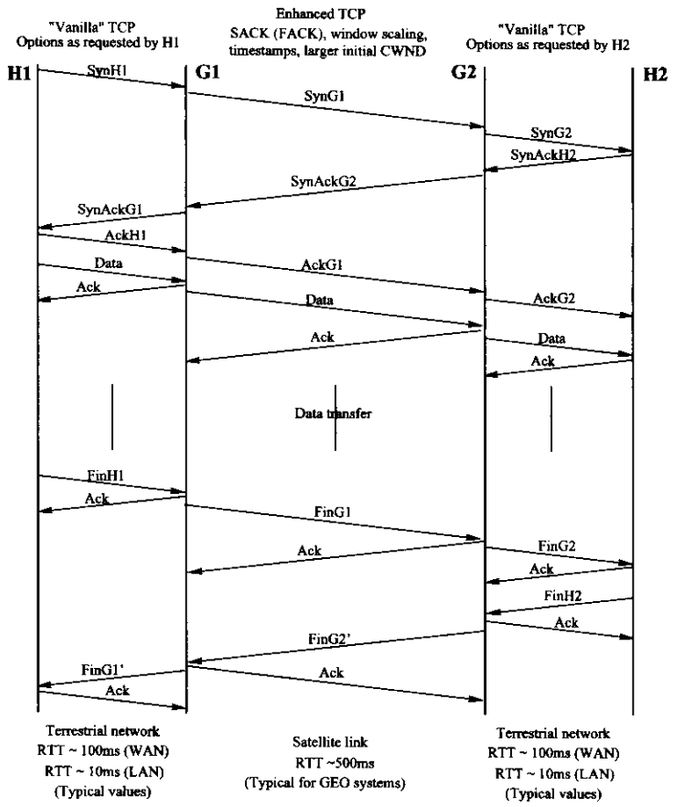

Figure 2. Timing diagram for simple unidirectional transfer from H1 to H2.

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; segments are removed from the receive buffer on the upstream connection only if they can be sent immediately 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 proxy. If a backlog begins to build in a receive buffer, 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. No packets are dropped if there are out-of-order segments in the receive queue; this avoids multiple packet drops in a single window, which would cause serious performance degradation.

More details on the implementation can be found in Reference [14].

3.3. Failure conditions

Our implementation is capable of handling a number of failure conditions gracefully. Due to the flexible implementation [14], the proxy can be configured to split only selected TCP connections, to keep resource utilization low. Also, due to the use of TCP on the satellite link, a proxy can simply forward connection setup (SYN) messages for new connections if it does not have resources available, without causing any side-effects except for a possible performance loss relative to connection splitting.

Our proxy implementation never retransmits a SYN segment on its own. This way, if a connection happens to follow different routes in the two directions, the proxy does not see the SYNACK and simply times out the SYN request from its cache. Meanwhile, because no connection has been set up, further data arriving on this connection is simply forwarded. If the route is symmetric and a SYN gets lost, the initiating host eventually retransmits its SYN and the proxy treats it as a new SYN.

However, many failure conditions remain. If a proxy fails, then any data that is buffered on the proxy but has not yet been transmitted is surely lost. Even data that has been transmitted and not yet acknowledged may be lost if the transmitted copy is lost in the network. The proxy is also as yet unable to deal with route changes during the lifetime of a connection. These weaknesses appear to be unavoidable consequences of the connection splitting architecture. Some possibilities for improving robustness under route changes are discussed later.

4. PERFORMANCE MEASUREMENTS

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

We tested various scenarios, involving single TCP connections as well as multiple simultaneous connections. Of these, the single connection tests are fairly simplistic and intended mainly as benchmarks. The numbers quoted were all obtained by performing each test thrice and averaging the measured times, since time was the quantity we directly measured. For instance, the throughput values were obtained by dividing the size of the transferred object by the average time, and not by averaging the throughput over three runs.

4.1. Single TCP connection

4.1.1. Test methodology. The test configuration is shown in Figure 3. The server host was a PC running Microsoft Windows NT Workstation 4.0 with the default TCP/IP parameters, and the client host was running Microsoft Windows 95 with the default parameters. We used the FTP

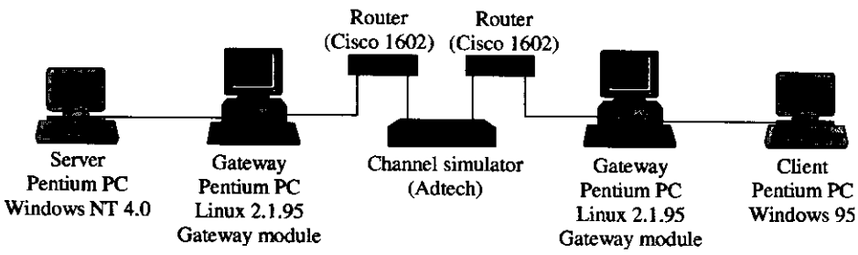

Figure 3. Test configuration. The proxy module was run on the two gateways.

server and the HTTP server from the NT Peer Web Services software. The clients used were the command-line FTP client supplied with Windows 95, and Netscape Communicator 4.05 for the HTTP tests. Tests were carried out using a data channel simulator to simulate the satellite channel, as well as over a commercial Ku band satellite link.

For the FTP tests, various combinations of the following parameter values were used, with no competing traffic on the link:

- File sizes: 10, 100, 1000, 10 000 and 100 000 KB.
- Link rates: 384 kb/s, 1.536 and 8 Mb/s.
- Link delay each way: 0, 250 ms.
- Bit-error rates: 0, 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}.

To provide a baseline for comparison, identical tests were carried out with connection splitting disabled on the proxy machines.

HTTP tests using different kinds of webpages were carried out over the same range of link rates and delays. Three webpages were used for testing:

- Test page 1: HTML document (356 B) + one image (391 KB).
- Test page 2: HTML document (1.7 KB) + 16 images (sizes 19–100 KB, total size 669 KB).
- Test page 3: HTML document (1.6 KB) + 16 images (nearly equal sizes, total size 262 KB).

4.1.2. Results—simulated satellite channel. Figure 4 shows the degradation of TCP performance when delay and errors are added to a link. 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 restricted to small values. Thus, link utilization decreases with increasing link bandwidth. The connection splitting approach, which uses much larger window sizes on the satellite link, yields better throughput. Both approaches suffer significantly from bit errors, especially at higher link speeds. This is due to TCP assuming that all losses are caused by congestion.

Figure 5 shows the dependence of throughput on the size of the file transferred. As expected, the end-to-end approach is limited to a constant rate due to the small size of the window. The proxy approach does considerably better. 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, and the utilization improves. This is shown clearly in Figure 5(b), in the plot of throughput against the ratio of file size to bandwidth-RTT product. Clearly, for good performance, the file size should be an order of magnitude larger than the bandwidth-RTT product.

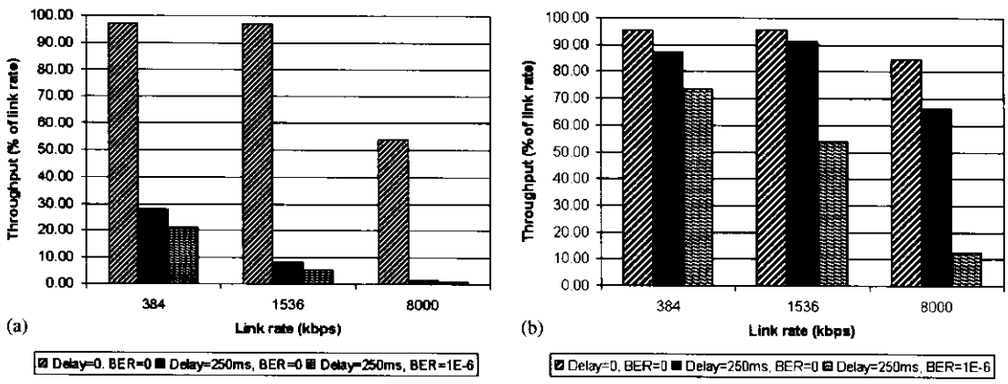

Figure 4. Effect of delay and errors on FTP performance. File size = 10 MB. (a) End-to-end TCP; (b) proxy, IW = 1.

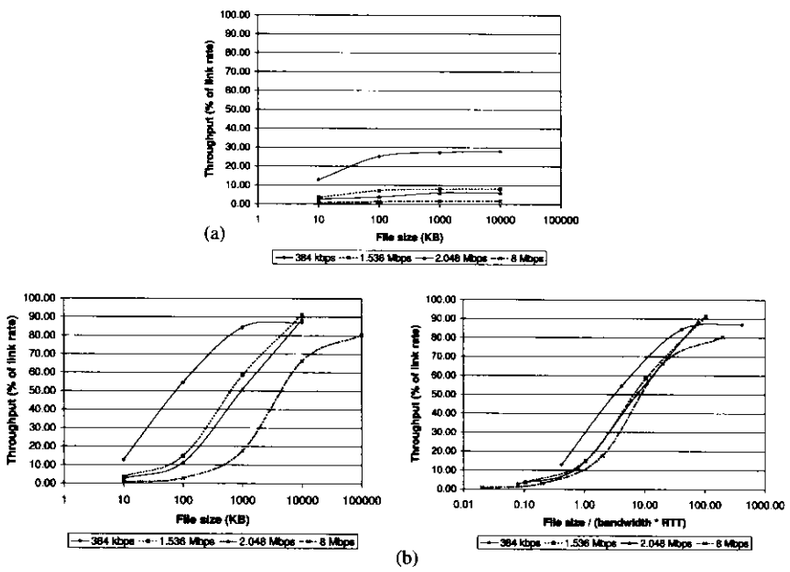

Figure 5. Effect of file size on TCP performance at different link speeds. Delay = 250 ms each way, BER = 0. (a) End-to-end TCP; (b) proxy, IW = 1.

The effect of slow start on throughput can be reduced by increasing the value of the initial congestion window. As Figure 6 shows, this improves throughput at all link rates, but the increase is appreciable only for medium-sized files at higher link speeds. This is because each time the initial window is doubled, the slow start phase is shortened by one round trip. For very large transfers, this is a small fraction of the total number of round trips required, and has little effect on

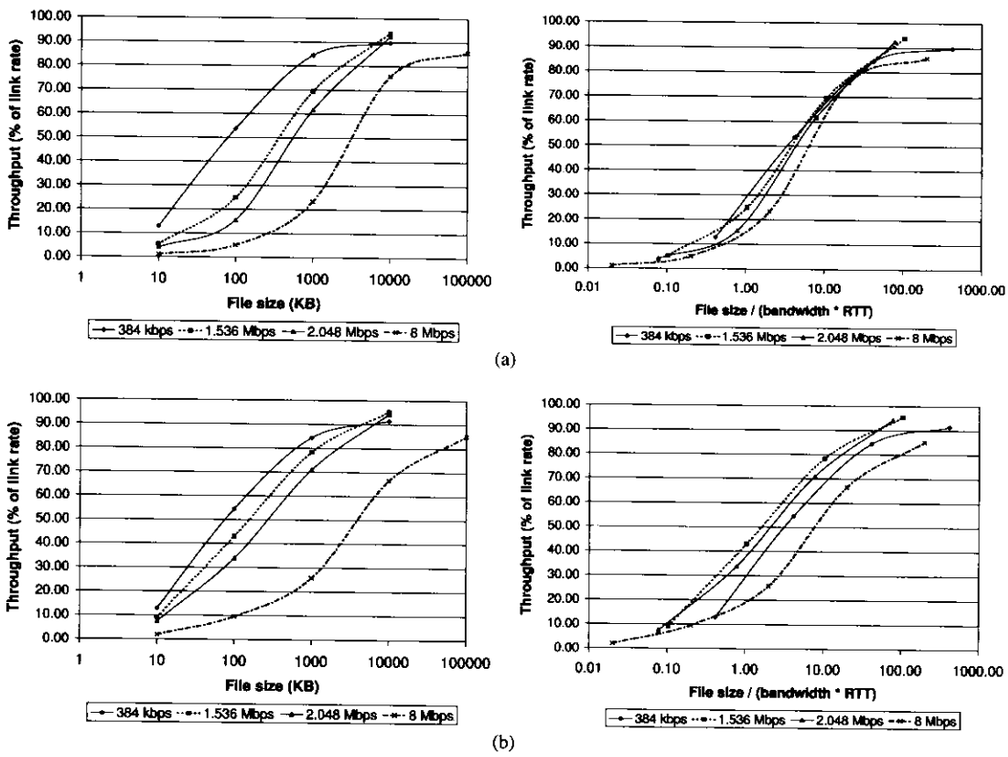

Figure 6. Effect of increasing IW on TCP performance. different link speeds. Delay = 250 ms each way, BER = 0, proxies enabled. (a) Proxy, IW = 4; (b) proxy, IW = 16.

average throughput. For small transfers, increasing the window fails to make any difference beyond a point, since the transfer time cannot be reduced to less than one round trip time. For medium-sized files (i.e. those with size approximately equal to the bandwidth-RTT product), the entire transfer is completed during slow start, so doubling the initial window has a significant effect.

In other words, increasing the initial congestion window, at least up to sizes that are small compared to the bandwidth-RTT product, does not change the fact that high link utilization can be achieved only when the transfer size is an order of magnitude larger than the bandwidth-RTT product. However, for small transfers, the reduction in user response time is often significant.

Figure 7 shows the link utilization achieved by the split-connection system when bit errors are present on the satellite link. Performance is fairly good even at relatively high error rates, mostly due to our use of SACK information and the FACK algorithm on the satellite link. Throughput is more affected by bit errors at higher link speeds, since the susceptibility of TCP to errors (in terms of reducing the congestion window) depends on the number of errors per round trip time and not on the absolute BER. Figure 7(b) plots the throughput as a function of the error rate per round

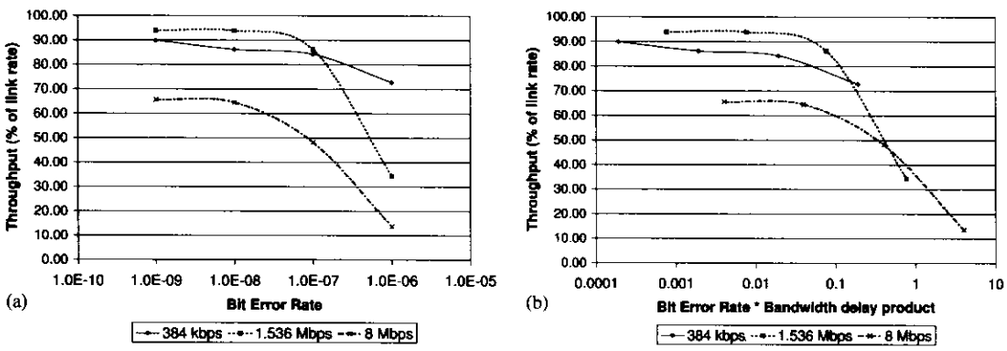

Figure 7. Effect of bit errors on FTP performance at different link speeds. File size = 1 MB, delay = 250 ms each way. Proxies enabled, IW = 1. (a) Throughput vs BER; (b) throughput vs normalized BER.

trip. We see that performance drops sharply when the error rate approaches one error per round trip, as is characteristic of TCP.

For the HTTP transfers, performance was measured by the total time required for each webpage to load, as measured by a stopwatch. Due to the imprecise nature of this measurement method, as well as due to the variance introduced by the HTTP client (which requires a significant amount of time and processing power for its image manipulation and rendering algorithms), these results are by no means exact. However, they serve very well to illustrate some general trends, and give an idea of perceived user delays.

HTTP, like FTP, uses a request-response mechanism, wherein the client requests one object at a time from the server. However, each FTP transfer consists of a single file, whereas a single webpage typically consists of several smaller objects, each of which must be requested separately by the client. Due to the request–response mechanism, 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. Therefore, the traffic generated by an HTTP session is intermittent in nature, with long pauses. Thus, throughput is not a good indicator of HTTP performance.

As Figure 8 shows, HTTP performance declines markedly with increasing link delay, and in this case the improvements due to using the proxy are not as remarkable as in the case of FTP. Further, as seen from Figure 9, increasing the initial congestion window does not have much effect on performance. It is worth noting that for test pages number 2 and 3, the request-response delays contribute 8 s (16 round trips of 0.5 s each), which is a major portion of the total loading delay. The only way to overcome this problem seems to be to add request aggregation capabilities to HTTP.

4.1.3. Results—Ku band satellite. Similar results were obtained when the above FTP tests were repeated over a commercial Ku band satellite link. The bandwidth of the satellite link was E1 (2.048 Mb/s). During the tests, the satellite modems on the two ends of the link reported bit error rates of 10^{-6} and 4.5×10^{-5}, respectively, on the raw channel. The Reed–Solomon coding used by the modems reduced the error rate to 10^{-12} or better in both directions, and so for all practical purposes the channel was error-free. Thus the results obtained were very similar to those obtained with the channel simulator at the same link speed with the BER set to zero.

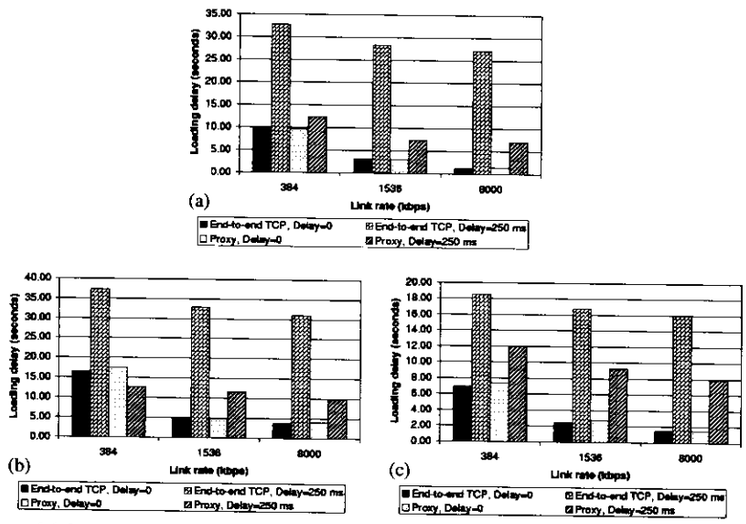

Figure 8. Effect of delay on HTTP performance for three sample webpages. For these tests, the proxies were set to IW = 1 and the channel simulator to BER = 0. (a) Test page 1; (b) test page 2; (c) test page 3.

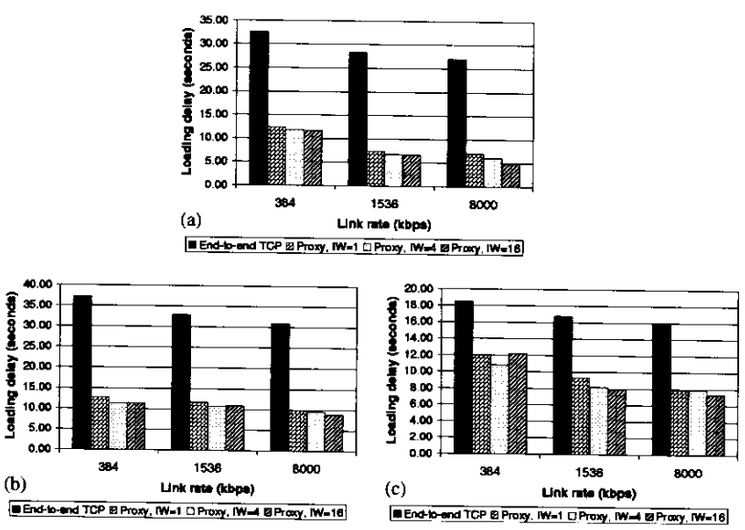

Figure 9. Effect of increasing IW on HTTP performance over simulated satellite link on three sample webpages. For these tests, the channel simulator was set to BER = 0. (a) Test page 1; (b) test page 2; (c) test page 3.

Figure 10(a) shows the variation of throughput with changing file size. As expected, performance with proxies enabled is much better than with end-to-end TCP. We see that performance picks up when file size is about an order of magnitude larger than the bandwidth-RTT product

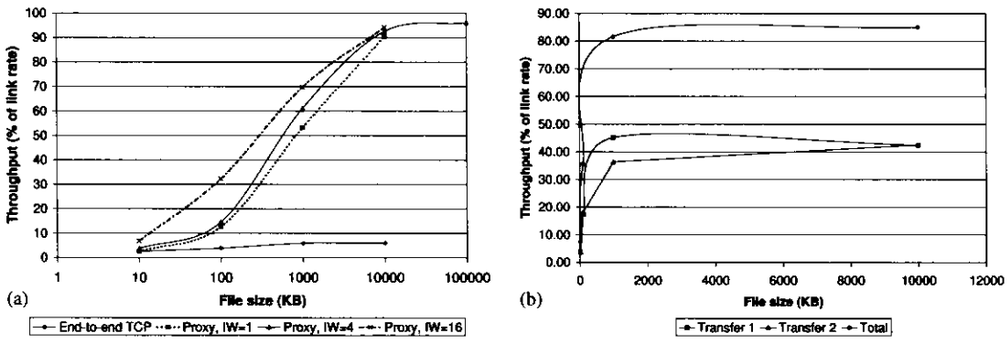

Figure 10. FTP Performance over Ku band satellite link. (a) Single FTP session; (b) two simultaneous FTP sessions.

(approximately 125 KB), and that only medium-sized file transfers are affected by increasing the initial congestion window.

Figure 10(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.

4.2. Multiple TCP connections

4.2.1. Test methodology. A setup similar to the single connection case was used. The server and client machines in this case were PCs running Linux. As before, all links were 10 Mbps Ethernet links, except for the simulated satellite link, which was an 8 Mbps serial link. The TCP benchmarking tool, distributed benchmarking system (DBS), [15] was used to create multiple connections between these two machines. In the tests for measuring fairness to flows with different RTTs, a third machine was used as an additional client. This machine was also a Pentium PC running Linux connected to the server through a Cisco 7000 series router.

We measured application-level throughput at the receiver for different numbers of parallel connections transferring data from the server to the client(s). Receiver throughput is measured instead of sender throughput because the latter merely measures how quickly the program could write its data to the TCP send buffer, and is in no way related to the actual rate of data transfer. All connections were made before the measurement was started, so that differing connection setup times did not affect throughput measurements. This was done using the BEFORE connection mode of DBS.

We measured throughput for transfers of size 10 and 100 KB. These sizes were chosen to approximate the typical size range of HTTP objects, so that the measurements would be somewhat realistic. Tests were carried out with the channel simulator set to 250 ms delay each way. In none of the tests were any bit errors introduced by the channel simulator. This was done for two reasons: firstly, the preceding tests show that bit errors do not have an appreciable effect on performance on typical Ku band satellite channels. Secondly, at the file sizes we tested, the probability of any single transfer encountering a bit error, even at relatively high error rates such as 10^{-6}, is so low that any results obtained from a small number of tests would not be statistically significant.

We performed tests with 10, 20, 30 and 40 simultaneous connections. The total bandwidth-RTT product of our link is 500 KB, or approximately 343 segments at our chosen MTU of 1500 bytes. Therefore, when 10 connections share the channel, the optimal window for each of them is about 50 KB, and we can expect to see some effects of limited receive window sizes, as the client and server were using the Linux default window sizes of 32 KB. On the other hand, when 40 connections share the channel, the optimal window size for each is around 12.5 KB, or about 9 segments. So at this point we do not expect to see any effects due to limited receive window sizes; instead we expect to see severe congestion when we increase the initial window on the proxies to 16 segments.

We note that some of the results shown here, especially for the 10 KB transfer size, may not be very accurate, due to limits imposed by timer granularity on the test systems. However, the general trends are quite reliable, and reveal some interesting insights.

4.2.2. Results—simulated channel. Figures 11 and 12 show the results when different numbers of parallel TCP connections transfer data simultaneously over the simulated link. As expected, throughput is low for small transfers, as the transfer time is dominated by delays due to slow start.

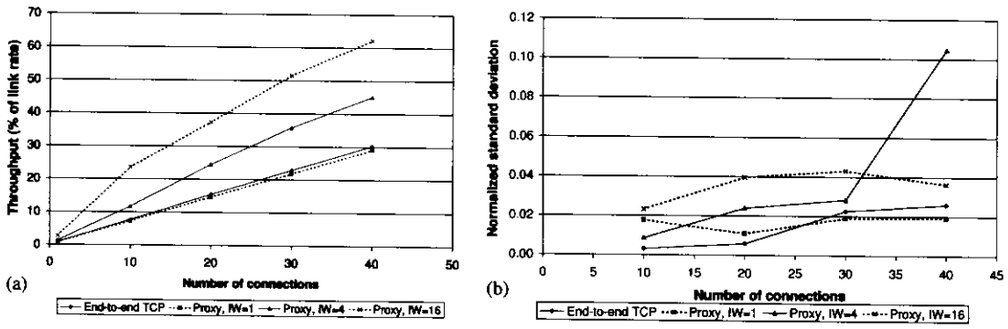

Figure 11. Throughput for simultaneous 10 KB transfers. Delay = 250 ms each way.
(a) Mean utilization; (b) normalized standard deviation.

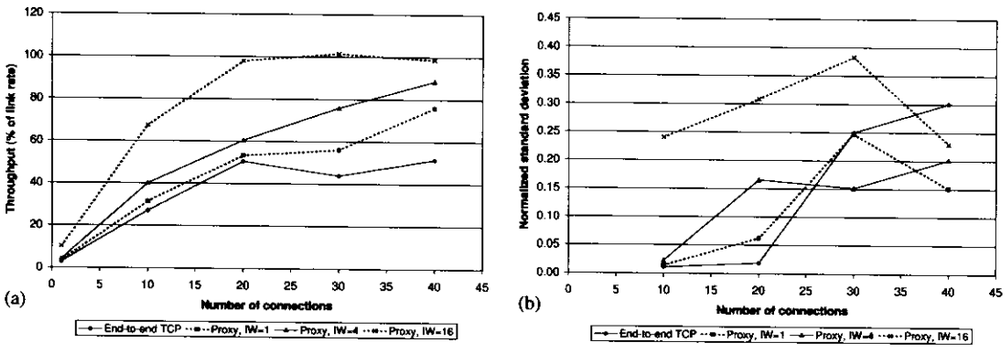

Figure 12. Throughput for simultaneous 100 KB transfers. Delay = 250 ms each way.
(a) Mean utilization; (b) normalized standard deviation.

Even 40 transfers of 10 KB each only represent a total transfer of 400 KB, which is less than the bandwidth-RTT product of our link, so it is not surprising that the use of proxies does not help much. Increasing the initial window makes a large difference to throughput—when IW is increased to 16 segments, the entire transfer can be accomplished in a single round-trip.

More interesting results are observed for the larger transfer size of 100 KB. Here each transfer involves approximately 70 segments, which means that with IW = 1, the transfer takes about seven round trips to complete, with the senders sending at most 32 segments in a round trip. Increasing the IW to 4 segments reduces the transfer time by two round trips, and so yields significant benefit. Increasing IW to 16 segments further reduces the transfer time by two round trips, improving throughput even more.

With 40 simultaneous transfers of 100 KB each, the total amount of data transferred is 4000 KB, which is about an order of magnitude larger than the bandwidth-RTT product of the link. Therefore, it is not surprising that the proxies perform well. However, it is remarkable that end-to-end TCP does not do as well, since for this case the optimal window size is only 12.5 KB. A possible explanation is provided by Figure 12(b). These plots show the standard deviation of the throughput of individual connections in each experiment, normalized by the mean throughput for connections in that experiment. We see that when the number of connections becomes large enough for the sum of the senders' windows to exceed the path bandwidth-RTT product, the standard deviation rises sharply. This occurs when the number of connections is about 20 (20 connections, each with window 32, make a total window of 640 segments, which is slightly less than twice the bandwidth-RTT product of the simulated satellite link). This indicates that, at least for the transfer sizes and link parameters involved, the bandwidth sharing mechanisms of TCP do not work very well when utilization is high. This observation, if true, would indicate that increasing the IW too much may have harmful effects on throughput.

The measurements plotted in Figures 13 and 14 show how proxies can improve performance by performing localized error recovery. During these tests, two simultaneous flood pings were carried out from the client host to its adjacent proxy, alongside the TCP transfers. A flood ping consists of ICMP Echo Request messages sent from one machine to the other 100 times a second or as fast as Echo Response messages are received by the sender, whichever is greater. We used Echo Request messages with 1472 B of data, so that the final IP packets containing these

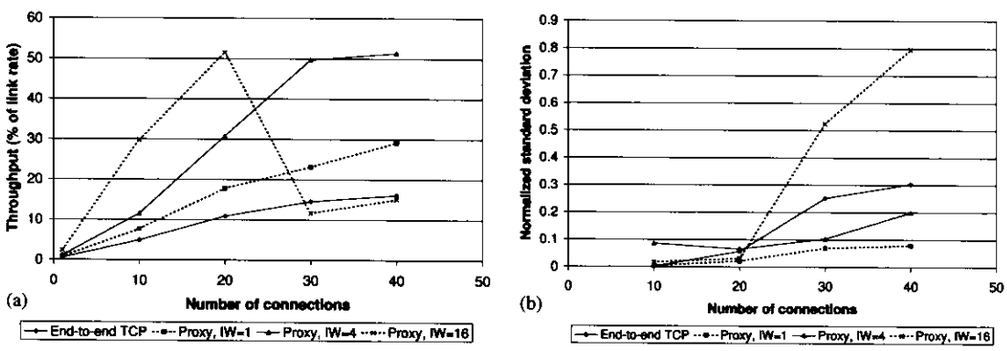

Figure 13. Throughput for 10 KB transfers with congested terrestrial link. Congested link is downstream of satellite link, which has delay = 250 ms in each direction.
(a) Mean utilization; (b) normalized standard deviation.

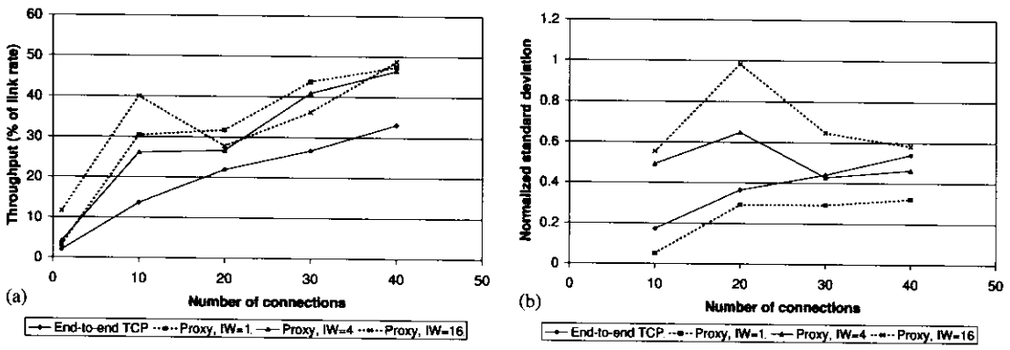

Figure 14. Throughput for 100 KB transfers with congested terrestrial link. Congested link is downstream of satellite link, which has delay = 250 ms in each direction.
(a) Mean utilization; (b) normalized standard deviation.

messages were exactly 1500 B long, i.e. sized equal to the Ethernet MTU. This arrangement simulates congestion on the terrestrial link, making it the bottleneck instead of the satellite link. As expected, the proxies perform better, since any packets dropped due to congestion can be retransmitted locally without traversing the satellite link.

In this case, increasing IW does not help much; it may even be harmful for throughput, though our data are not clear enough to say for certain. It is worth noting that with increased IW the standard deviation of throughput among simultaneous connections becomes very large, and of the order of the mean throughput.

For the experiments plotted in Figures 15 and 16, an additional client host was used. This machine was connected to the server by Ethernet through a Cisco 7000 router. In each case, half the total number of transfers (referred to as set L, for low delay) were to this additional client while the rest (set H, for high delay) were to the client on the other side of the simulated satellite link. Thus the only link common to set L and set H was the Ethernet link connected to the server.

Due to the much larger round-trip delay experienced by clients in set H, end-to-end TCP is seen to be extremely unfair to them. However, the proxies do a much better job of sharing bandwidth between the connections by enabling long-delay connections to compete on equal terms with low-delay wired links. As the initial window is increased, the sharing behavior improves. The remaining unfairness is due to the fact that the proxies use TCP over the satellite link, where the window evolves much slower than over the terrestrial low-delay link. This points to the need for a more efficient protocol over the satellite link. One obvious solution to this problem could be to use TCP with different window increments during slow start and congestion avoidance depending on the round-trip time.

Figure 16 shows an interesting trend—the normalized standard deviation for set L is much higher than that for set H. This seems to strengthen the argument that TCP does not share bandwidth equally between identical connections when congestion is present.

4.3. Some comments

From the above, we see that the value of IW plays a major role in determining throughput. However, it is not possible for a host to know such a value when it sets up the connection.

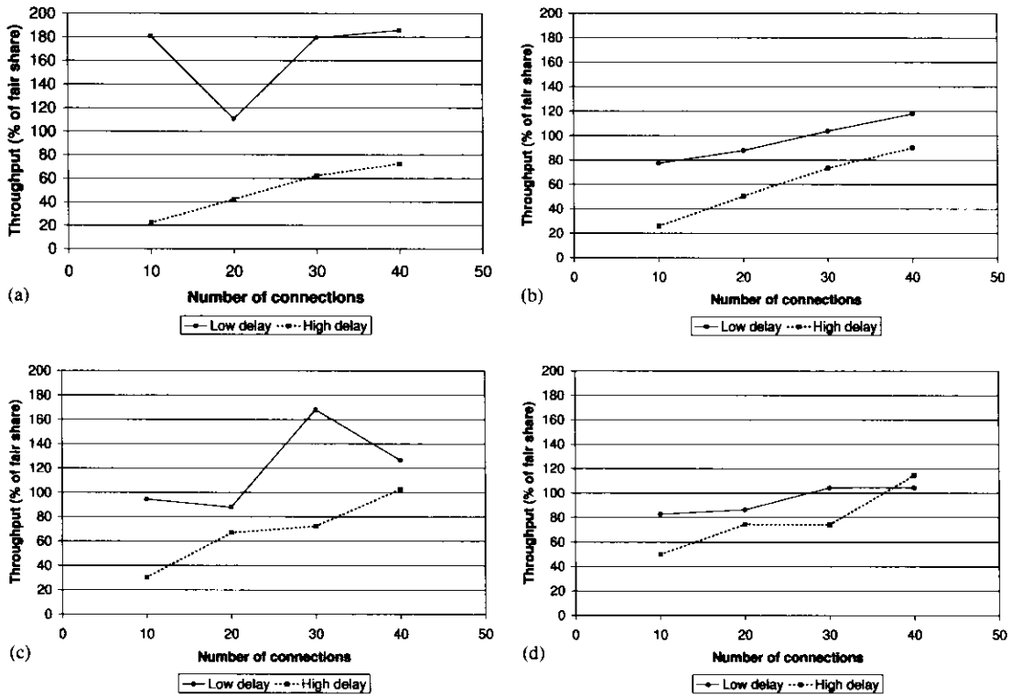

Figure 15. Bandwidth sharing among flows with different RTT. (a) End-to-end TCP; (b) proxy, IW = 1; (c) proxy, IW = 4; (d) proxy, IW = 16.

This suggests that it might be useful to have the network participate in flow control by determining a good value for the initial window at connection setup, and informing the host about this value. In any event, it is clear that at least for the transfer sizes tested, the transfer does not last long enough for the window to stabilize or for equilibrium to be reached.

Looking closer at Figure 15, a disturbing fact becomes apparent. The throughput achieved by set H in the proxy case with IW = 1 is only slightly higher than that achieved under end-to-end TCP. However, the throughput for set L is much lower. This suggests that with the proxies enabled, high-RTT connections become unnecessarily aggressive. Further investigation shows that the use of proxies for set H causes the window for these connections to grow as rapidly as for set L, and so all the data is transferred very quickly to the first proxy on the path, where it must wait for transmission when the congestion window on the satellite link grows sufficiently large. This behaviour unnecessarily reduces the throughput of set L, and uses up large amounts of memory on the proxy.

The above observation motivates us to consider methods for flow control at the proxies, so that they are never unnecessarily aggressive, and can keep their memory requirements low without sacrificing throughput. Such issues are discussed in the next section.

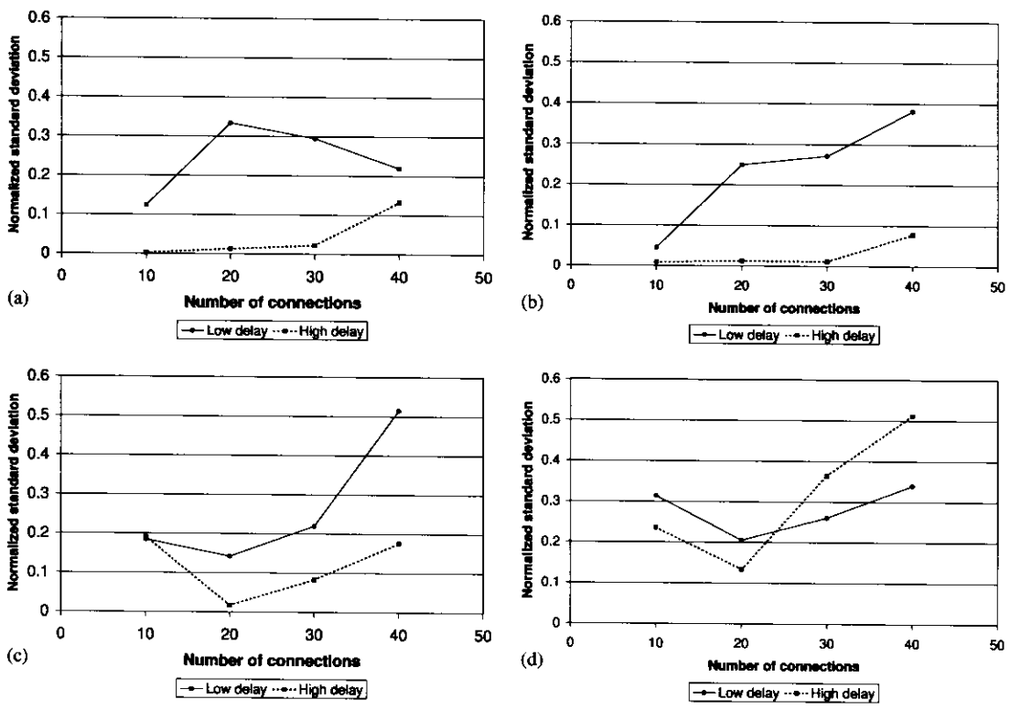

Figure 16. Bandwidth sharing among flows with different RTT. (a) End-to-end TCP; (b) proxy, IW = 1; (c) proxy, IW = 4; (d) proxy, IW = 16.

5. DISCUSSIONS AND FUTURE WORK

During our experience with implementing and testing the proxy, we found a few areas that need improvement and further research. The biggest problem was that since the proxy has to buffer any data transmitted on a split connection until it is acknowledged, the memory required for a proxy is quite high. In fact, a little thought shows that we cannot guarantee a bound on the memory requirement, since downstream congestion or a host failure can cause data to be held in transmit buffers at a proxy for long and indeterminate periods of time.

An obvious way to ameliorate this problem is to split TCP connections selectively, since only large transfers can benefit much from connection splitting. This could be done by dynamically varying the set of split connections in response to activity. Connections that have been idle for a long time could be dropped from the set by discarding their associated state information, so that future segments would simply get forwarded. Connections which are not currently being split, but have witnessed high activity recently, could be added to the set of split connections by dynamically acquiring the required state information. Such a 'soft-state' approach would also improve the proxy's tolerance to some error conditions. This is a challenging problem, and one that requires more work. A major difficulty is in formulating an algorithm for deciding when to add a flow to the set of split connections and how to acquire the requisite state information.

Another useful addition would be a method to detect when routes change during the lifetime of a split connection. Especially when routes change in only one direction, the proxy can deduce this fact by examining the flow in the other direction. However, there does not seem to be a good way to tackle the case when routes change in both directions and the proxy has unacknowledged data in its send buffer. This is an area for future work.

A problem specific to our architecture 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 on the slower connections, but it takes a long time, and end-to-end throughput is reduced. This method also requires the proxy to buffer a lot of untransmitted data—a better algorithm would keep proxy buffering low by ensuring that data arrived just as a window became available for sending it. We have explored some simple schemes for better flow control [14], but as yet there does not seem to be a good way to carry out such control in all situations.

Scalability is an important consideration for all connection implementations. It seems inevitable that connection splitting proxies will have higher memory and processor requirements than IP routers. However, whether these requirements necessarily increase linearly with the number of active connections or whether they can be restricted to a constant multiple of the requirements of a router seems an open question.

The experiments reported here do not represent very high data rates. Also, traffic patterns for high-bandwidth satellite networks are not very well understood, and much work needs to be done before we can extend our results to such networks. However, given the flexibility of the implementation and its ability to deal gracefully with situations when resources are short, we believe that such proxies can provide noticeable performance improvements to users of such systems. Implementing mechanisms to minimize unnecessary resource usage, as outlined above, would further improve robustness and performance.

ACKNOWLEDGEMENTS

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. Jacobson V. Congestion avoidance and control, *Proceedings of ACM SIGCOMM '88*, August 1998; 314–329.
2. Postel J (ed.). Transmission control protocol—protocol specification. *RFC 793*, September 1981.
3. Lakshman TV, Madhow U. The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, June 1997.
4. Floyd S. Connections with multiple congested gateways in packet-switched networks, Part 1: one-way traffic. *ACM Computer Communications Review* 1991; **21**(5):30–47.
5. Balakrishnan H, Seshan S, Katz R. Improving reliable transport and handoff performance in cellular wireless networks. *ACM Wireless Networks* 1995; **1**(4).
6. Jacobson V, Braden R, Borman D. TCP extensions for high performance. *RFC 1323*, May 1992.
7. Allman M, Floyd S, Partridge C. Increasing TCP's initial window. *RFC 2414*, September, 1998.

8. Mathis M, Mahdavi J, Floyd S, Romanow A. TCP selective acknowledgment options. *RFC 2018*, October 1996.
9. Butts NP, Bharadwaj VG, Baras JS. Internet service via broadband satellite networks. *Multimedia Systems and Applications: Proceedings of SPIE* 1999; **3528**:169–180.
10. Bakre A, Badrinath BR. I-TCP: indirect TCP for mobile hosts. *Technical Report DOS-TR-314*, Department of Computer Science, Rutgers University, October, 1994.
11. Henderson TR, Katz RH. Transport protocols for internet-compatible satellite networks. *IEEE Journal on Selected Areas in Communications* 1999; **17**(2):345–359.
12. Braden B. (ed.) Requirements for Internet hosts—communication layers. *RFC* 1122, October 1989.
13. Mathis M, Mahdavi J. Forward acknowledgment: refining TCP congestion control. *Proceedings of SIGCOMM '96*, Stanford, CA, August 1996.
14. Bharadwaj VG. Improving TCP performance over high-bandwidth geostationary satellite links. *Master's Thesis*, Department of Electrical Engineering, University of Maryland, 1999.
15. DBS: A TCP benchmark tool. http://shika.aist-nara.ac.jp/member/yukio-m/dbs/.

AUTHORS' BIOGRAPHIES

Vijay Bharadwaj received his BTech in Electrical Engineering from the Indian Institute of Technology, Mumbai, in 1996, and his MS in Electrical Engineering from the University of Maryland, College Park, in 1999. He is currently a research assistant at the Center for Satellite and Hybrid Communication Networks at the University of Maryland, where he is working toward his PhD. His research interest is the design and analysis of large hybrid networks, with special emphasis on protocol efficiency and security.

John S. Baras was born in Piraeus, Greece, on 1 March 1948. He received the BS in Electrical Engineering with highest distinction from the National Technical University of Athens, Greece, in 1970. He received the MS and PhD degrees in Applied Mathematics from Harvard University, Cambridge, MA, in 1971 and 1973 respectively. Since 1973 he has been with the Department of Electrical Engineering, University of Maryland at College Park, where he is currently Professor and member of the Applied Mathematics Faculty. From 1985 to 1991 he was the Founding Director of the Systems Research Center, now the Institute for Systems Research. On February 1990 he was appointed to the Lockheed Martin Chair in Systems Engineering. Since 1991 he has been the Director of the Center for Satellite and Hybrid Communication Networks, a NASA Center for the Commercial Development of Space, which he co-founded. Dr Baras has held visiting research scholar positions with Stanford, MIT, Harvard, University, the Institute National de Reserche en Informatique et en Automatique, and the University of California, Berkeley. He has numerous publications in control and communication systems, and is the co-editor of *Recent Progress in Stochastic Calculus*, Springer-Verlag, 1990. His current research interests include stochastic systems, signal processing and understanding with emphasis on speech and image signals, real-time architectures, symbolic computation, intelligent control systems, robust nonlinear control, distributed parameter systems, hybrid communication network modeling, performance evaluation and management. Among his awards are: a 1978 Naval Research Laboratory Research Publication Award, the 1980 Outstanding Paper Award of the IEEE Control Systems Society, 1983 and 1993 Alan Berman Research Publication Award from the Naval Research Laboratory, the first Mancur Olson Research Achievement Award (1998) for sustained outstanding research accomplishments from the University of Maryland College Park. Dr Baras has been awarded one patent and has three patents pending. He has served in: the IEEE Engineering R & D Committee, the Aerospace Industries Association advisory committee on advanced sensors, the IEEE Fellow evaluation committee, the IEEE Control Systems Society Board of Governors, and the Editorial Board of the IEEE Transactions on Automatic Control. He is currently serving on the editorial boards of *Mathematics of Control, Signals, and Systems*, of *Systems and Control: Foundations and Applications*, of the *IMA Journal of Mathematical Control and Information*. He is a member of Sigma Xi, the American Mathematical Society, and the Society for Industrial and Applied Mathematics. Dr Baras is a Fellow of the IEEE.

Norm Butts has a 19-year history of diverse and challenging systems engineering positions with multiple Lockheed and Lockheed Martin Companies. Assignments spanning commercial and defense projects, as well as two foreign assignments, have resulted in a broad understanding of systems engineering principles and underlying technologies. Norm has 5 years prior systems engineering experience including design of local government telecommunications systems and commercial video systems. Since 1997 he has acted as

a network-engineering consultant to numerous Lockheed Martin terrestrial and satellite based programs, overseeing internal research and development projects including University partnerships with emphasis on hybrid satellite networks. He has published research papers on transparent TCP/IP satellite gateways and experimental results of Voice over IP applications transmitted over ATM based satellite networks and developed and implemented two successful live satellite network technology demonstrations commissioned by Lockheed Martin's Astrolink Broadband ATM Satellite Program.