INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2003; **16**:513–534 (DOI: 10.1002/dac.593)

Evaluation of performance enhancing proxies in internet over satellite

Navid Ehsan[1,†], Mingyan Liu[1,*,‡] and Roderick J. Ragland[2,§]

[1] *Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI 48109-2122, U.S.A.*
[2] *Hughes Network Systems, Inc., Germantown, MD, U.S.A.*

SUMMARY

Performance enhancing proxies (PEPs) are widely used to improve the performance of TCP over high delay-bandwidth product links and links with high error probability. In this paper we analyse the performance of using TCP connection splitting in combination with web caching via traces obtained from a commercial satellite system. We examine the resulting performance gain under different scenarios, including the effect of caching, congestion, random loss and file sizes. We show, via analysing our measurements, that the performance gain from using splitting is highly sensitive to random losses and the number of simultaneous connections, and that such sensitivity is alleviated by caching. On the other hand, the use of a splitting proxy enhances the value of web caching in that cache hits result in much more significant performance improvement over cache misses when TCP splitting is used. We also compare the performance of using different versions of HTTP in such a system. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: performance enhancing proxy; TCP connection splitting; caching; persistent connection; satellite network

1. INTRODUCTION

The performance of TCP over heterogeneous connections such as those including satellite and wireless links has been extensively studied for the past few years. Proposed performance enhancing techniques can roughly be categorized into link layer solutions (see for example, References [1,2]), end-to-end solutions where the end-to-end semantic of TCP is maintained (see for example, References [3–10]) and non-end-to-end solutions where the end-to-end semantic is violated (see for example, References [11–13]). Various link layer and end-to-end approaches can be quite effective for connections over wireless links through better error correction schemes, local retransmission schemes and schemes that distinguish congestion losses from link failure losses for TCP. In a connection that incorporates a satellite link on the other hand, the

*Correspondence to: M. Liu, Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI 48109-2122, U.S.A.
† E-mail: nehsan@eecs.umich.edu
‡ E-mail: mingyan@eecs.umich.edu
§ E-mail: rragland@hns.com

Received 12 September 2002
Revised 7 January 2003
Accepted 3 February 2003
Published online 9 May 2003

main bottleneck in TCP performance is due to the large delay-bandwidth product of the satellite link (for an overview of research on TCP over satellite see for example, Reference [14]). Over such a link the normal TCP window dynamics result in significantly increased latency before the channel is fully utilized. This problem cannot be effectively solved simply by improving the satellite channel quality, or by using large initial window size. This is because a connection using the satellite link typically also has a terrestrial part, thus using large window end to end could affect the performance and fairness of the terrestrial part of the connection.

One typical non-end-to-end solution that has been adopted by many satellite data communication service providers is the TCP connection splitting technique. The idea behind this technique is to segregate the end-to-end connection into segments so that each can be optimized separately, and in particular so that the TCP window over the satellite segment can be opened up faster. This involves placing at the Network Operating Center (NOC) a splitting proxy that acknowledges end user packets on behalf of the remote server and acknowledges the remote server on behalf of the end user (assuming the end user is directly connected to the satellite down link). This proxy operates at the TCP level and therefore breaks the end-to-end semantic of TCP. It is also not compatible with IPSec [15]. The benefit, however, is that (1) splitting a connection results in shortened TCP feedback control loop, so that the remote server receives ACKs much sooner from the NOC (proxy) than from the end user, and therefore its window can quickly ramp up; and (2) by segregating an end-to-end connection into segments, each segment is enhanced separately. The proxy can use large initial window over the satellite link, which can either be the last hop or somewhere in the middle of the end-to-end connection.

The performance gain of using a splitting proxy has been reported in Reference [16] as a result of simulation study and in Reference [11] from experimental study. In References [17,18] some analysis was presented to quantitatively study the benefit of such a proxy. We observe that in a typical satellite system TCP connection splitting is not the only performance enhancing technique that is commonly used. Web caching is also widely implemented to reduce latency by pushing contents closer to the end users. Moreover, whenever there is a cache miss, the cache in effect 'breaks' the server–client transfer into two separate connections. This is because the cache opens up a connection to the remote server and starts downloading the file to the cache (for cacheable objects) while forwarding packets to the client at the same time. Although this happens at the application layer, it results in a server–cache connection and a cache–client connection, and thus has a very similar effect on the end-to-end performance as that of a TCP splitting proxy, as will be shown more clearly in the next section. Our focus in this study is the combined performance implication of using both types of enhancements—TCP splitting and web caching. By taking measurements from a live Internet over satellite system, we hope to gain more insights into the use of proxy as a general solution to such systems, and more importantly to apply such understanding to system level design issues. Our measurements are obtained via repeated downloads of selected files (both pure text and with embedded objects).

Our main observations are as follows:

(1) Connection splitting enhances the value of caching. When splitting is not used, whether there is a cache hit or cache miss generates almost identical performance (in terms of delay and throughput). When splitting is used, a hit at the cache results in much higher throughput.

(2) The throughput of a split connection is more sensitive to increased congestion delay than an end-to-end connection. Having simultaneous connections will cause the performance of splitting to decrease much faster than the end-to-end case.
(3) The throughput of a split connection is highly sensitive to packet losses for small file transfers. Split connection provides much better *guarantee of performance improvement* for a large file than for a small file. Guarantee of performance improvement refers to the probability that the split connection results in higher throughput than the end-to-end connection. This probability is defined as the relative frequency of such events in our measurements. This guarantee is improved when there is a cache hit.
(4) The performance gain of using connection splitting is reduced as the number of embedded objects in a file increases. In addition, connection splitting is no substitute for persistent connection. If a splitting proxy is used, it is important that persistent connection is also used between the client and the proxy. Non-persistent connection even when connection splitting is used can result in much worse performance than an end-to-end persistent connection.

It's worth pointing out that TCP connection splitting is also often called TCP spoofing. Strictly speaking splitting refers to breaking up a connection and spoofing refers to faking an address. They are often related because in splitting a connection a transparent proxy typically spoofs the end points' addresses. Since our interest is in the end-to-end performance as a result of split connections (either at the transport layer or at the application layer), we will limit ourselves to the term connection splitting in this paper.

The rest of the paper is organized as follows. We describe the system configuration and our experiment methodology in the next section. We then present the measurements and our observation/explanation in Section 3. These results are organized into six subsections, where we examine the effect of file size, caching, number of simultaneous connections, congestion and random losses, number of embedded objects and persistent connection, respectively. Conclusions are given in Section 4.

2. SYSTEM DESCRIPTION

Our measurements are taken from a commercial satellite system that uses a geo-stationary (GEO) satellite (Ku band) for forward data transfer (from the NOC to the client/end host) and a regular phone line as the return channel (from the client/end host to the NOC via an ISP), as shown in Figure 1. Available bandwidth on the forward channel is up to 24 Mbps and the one way propagation delay of the satellite link is roughly 250 ms (however, due to NOC configuration the maximum throughput we were able to achieve per client was 300–400 kbits/s). The return channel has a speed of up to 56 kbps.

The TCP connection splitting proxy (which we will simply call proxy in the following) is implemented on a Hybrid Gateway (HGW) located in the NOC. The end host can choose to either enable or disable the proxy. When the proxy is disabled, a packet from the server to the end host passes through the HGW as if passing through a normal router, and goes directly to a Satellite Gateway (SGW) connected to the satellite uplink. When the proxy is enabled, it breaks up the server–client end-to-end connection in two, and pre-acknowledges one on behalf of the other in the following way, as illustrated in Figure 2(a).

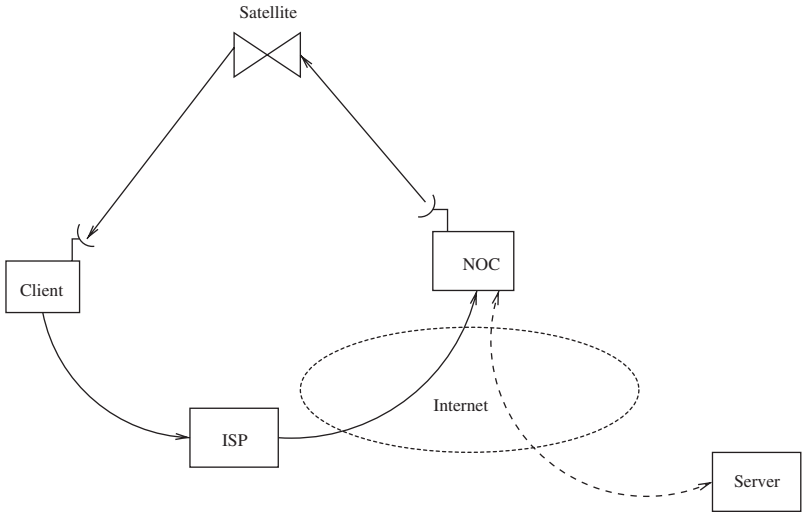

Figure 1. Main components of the satellite system.

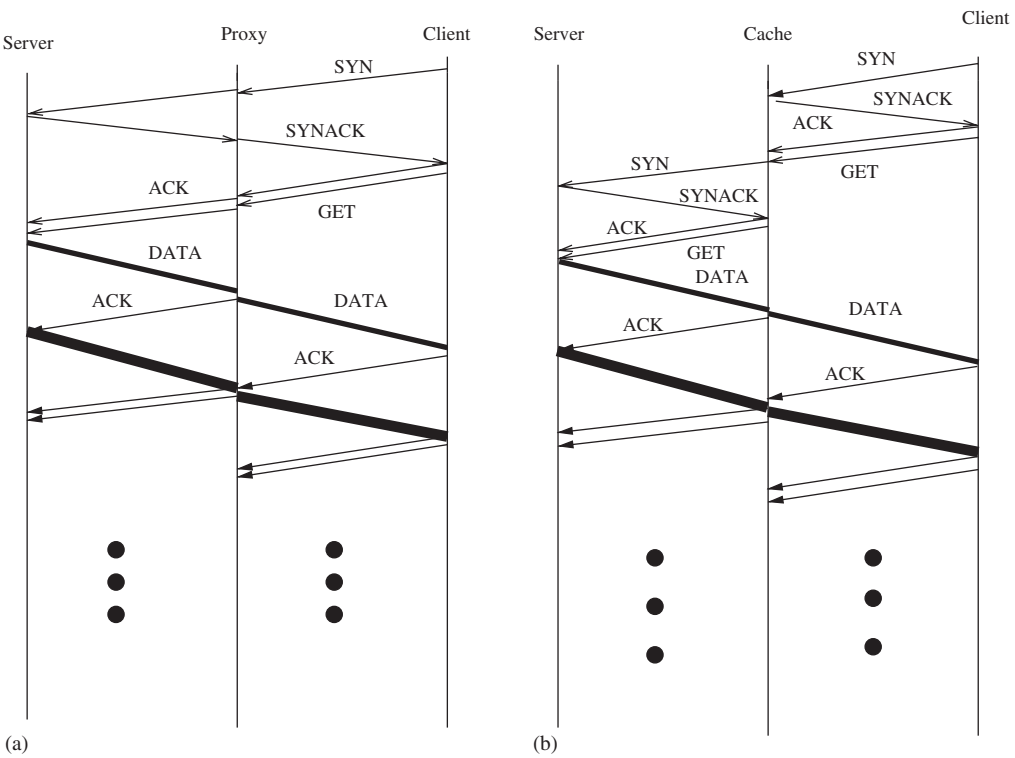

Figure 2. Splitting a connection at (a) a proxy and (b) a cache (miss).

During the connection establishment, the proxy simply forwards the SYN and SYNACK packets. Then the server starts sending data to the client upon receipt of the file request. Upon receiving a data packet from the server, the proxy sends an ACK back to the server. Since the proxy is transparent by spoofing the client's address, the server takes the ACK as an indication that the data being ACKed has been received successfully by the client, and therefore moves on to the next window and so on. Since the proxy is located closer to the server (than the client), because of the large satellite delay, this results in a much shorter round-trip time (RTT) seen by the server and thus enables the server to reach a much higher sending rate. At the same time the proxy is maintaining a separate connection with the client by forwarding data packets to the client, waiting for ACKs to come back from the client, and then releasing more data packets to the client. All packets received from the server are stored at the proxy. When an ACK is received from the client, data being ACKed is purged from the buffer. Otherwise the proxy retransmits upon duplicate ACKs or timeouts (local retransmission) in contrast with the end-to-end case where each retransmission comes from the server.

The web caches are also located in the NOC. Regardless of whether the splitting proxy is enabled or disabled, when an HTTP request is received at the NOC, it first goes through the cache before being sent to the server. For cacheable content, if a fresh copy of the requested file is located in the cache (a *hit*), the file is delivered to the client directly from the cache without going to the remote server. If the requested file is not found in the cache (a *miss*), the cache will open up a connection to the remote server to fetch the file, as shown in Figure 2(b). This server–cache connection is concurrent with the cache–client connection in that as soon as the cache starts receiving data from the server (via the server–cache connection), it will transfer it to the client (via the cache–client connection). Thus, in the case of a miss, the cache effectively handles two connections that constitute the end-to-end connection between the server and the client. In terms of data transfer, this is very similar to a splitting proxy. (However, with a cache this takes place at the application layer so the server sees a connection with the cache rather than being pre-acknowledged by the cache.) Figure 2 compares the packet flow in the case of a splitting proxy and in the case of a cache miss. Except for the connection establishment process, the data transfer essentially proceeds in an identical manner (note that this figure does not show processing delay). Consequently the splitting proxy together with the cache results in an end-to-end connection split twice upon a cache miss and once upon a cache hit, as shown in Figures 3(a) and 3(b),

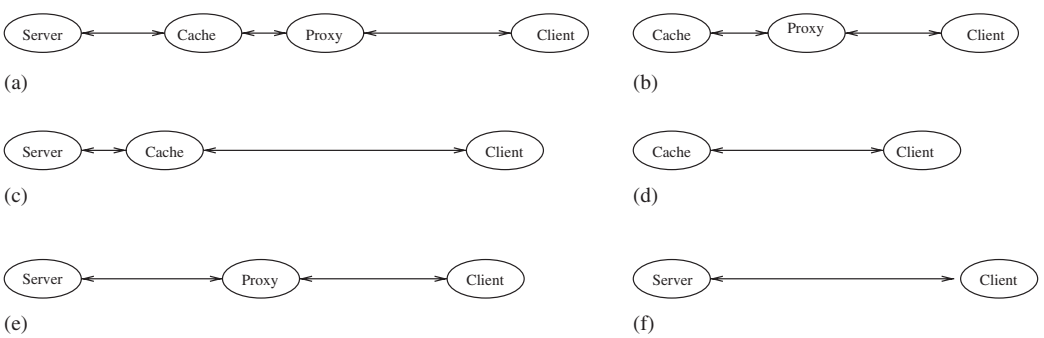

Figure 3. Experiment scenarios: (a) Cache miss, proxy enabled; (b) Cache hit, proxy enabled; (c) Cache miss, proxy disabled; (d) Cache hit, proxy disabled; (e) No cache, proxy enabled; (f) No cache, proxy disabled.

respectively. Figures 3(c) and 3(d) illustrates cache hit and cache miss, respectively, when the proxy is disabled. Figures 3(e) and 3(f) illustrates the cases where connections bypass the cache with the splitting proxy enabled and disabled, respectively.

Important parameters of our system are as follows. The client is running Windows ME that uses TCP SACK [19] with delayed acknowledgments (one ACK for every two received packets). Our web server is running Linux Redhat 7.1. Because of the high asymmetry in link speeds between the forward and return paths, we also use ACK filtering [20] at the client and send one out of every four ACKs. Thus each ACK received at the server (assuming no loss) in general represents eight packets. Byte counting instead of ACK counting is used at the server as well as the proxy, so that the lower rate of ACKs does not reduce the data sending rate. The splitting proxy uses an initial window size of 64 kbytes for the proxy–client connection over the satellite whenever enabled.

Our study consists of six main scenarios: splitting enabled or disabled with cache hit or cache miss and the option of whether to bypass the cache or not, as shown in Figure 3. Whether to use the file in the cache or not is controlled by a *no-cache pragma* [21,22] set in the request header. When set, this parameter tells the cache not to use the cached copy even if there is one and to get the most updated version from the server. Whether the connection splitting proxy is used or not is controlled by the end hosts. We have two end hosts, one of which has the proxy option enabled and the other one has the option disabled. This option is then encapsulated in the header of packets sent to the NOC. For comparison purposes, we always run experiments on these two hosts simultaneously. We download files from a dedicated web server onto both hosts repeatedly for durations of 1–2 h per data/point (measurements over this period are averaged into one data point).

All measurements are taken from a live commercial system with varying amounts of customer traffic. Our connections and experiment traffic go through a dedicated HGW and therefore a dedicated splitting proxy that is not shared by other traffic. However, our connections does share the cache access, the satellite gateway and the satellite transponder with other through traffic. Such a setup results in both controllable and uncontrollable factors as we will point out when we discuss our results in the next section.

The performance metrics we use in this study are the file transfer latency (using HTTP) and throughput. We define latency seen by the client as the time between when the SYN request is sent and the time when FINACK is sent by the client. For files with multiple embedded objects, this is defined as the time between when the first SYN request is sent and the time when the last FINACK is sent. Throughput is defined as file size divided by latency. Files used in this study contain a set of text files of varying sizes and a set of HTML files with varying number of embedded objects. Often for comparison purposes we keep the total file transfer size (base page plus embedded objects) the same while changing the number of embedded objects and thus the size of each object. The list of files with file types and file sizes are provided in Table A1 in the appendix.

3. RESULTS AND ANALYSIS

For comparison purposes and better illustration of our results, we define the *Gain of Splitting* (GoS) as

$$\mathrm{GoS} = \frac{\mathrm{Throughput}_{\mathrm{splitting}} - \mathrm{Throughput}_{\mathrm{end\ to\ end}}}{\mathrm{Throughput}_{\mathrm{end\ to\ end}}}$$

Thus larger GoS means higher throughput gain from using the splitting proxy. A negative GoS means that the end-to-end connection has a higher throughput (or smaller latency) than the split connection. We will frequently use this metric in subsequent discussions.

3.1. Effect of varying file sizes

We first compare the file transfer throughput with the splitting proxy enabled and disabled under scenarios described in Figures 3(e) and 3(f). We download files onto both the splitting enabled and the splitting disabled hosts repeatedly over a 1 hour period and average the measured throughput over this period. The corresponding GoS for files ranging from 10 kbytes to 1 Mbytes are shown in Figure 4.

This result essentially confirms some earlier reports, see for example, References [16,18], that splitting provides better performance gain for larger files. This is due to the fact that the time spent in TCP connection establishment has a bigger portion in the overall latency for smaller files and this time is not reducible by using splitting. At the same time larger files benefit more from the large initial window size (for example, a 1 kbyte file and a 64 kbyte file can both be transmitted in the first window and experience almost the same latency, if file transfer from the source to the proxy is fast enough). However, this gain saturates around 80 K. This is because when the satellite link is fully utilized (as in the case of a very large file), the difference in throughput between the two scenarios becomes smaller and smaller. The main benefit of using a split connection comes from sending faster over the satellite link. When the capacity of the satellite link is fully utilized this benefit starts decreasing. The larger the file size, the more diluted this benefit gets. Therefore, we see the gain decreases as the file becomes bigger.

3.2. Effect of caching

In this section, we compare the performance under scenarios described in Figures 3(a)–3(d). Again files are downloaded onto both the splitting enabled and the splitting disabled hosts

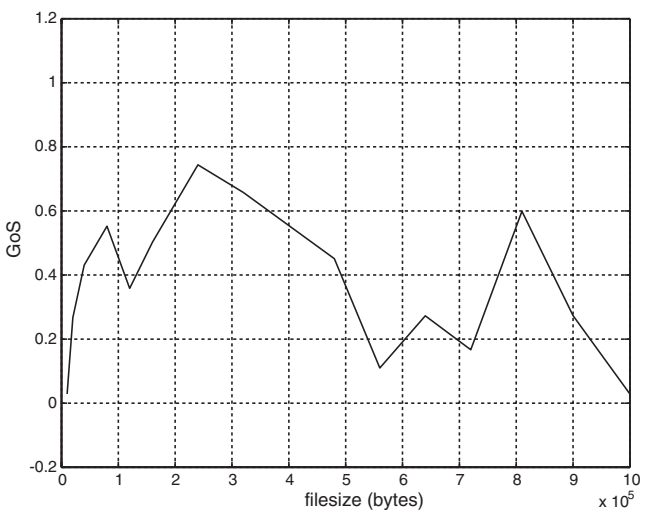

Figure 4. GoS for varying file sizes.

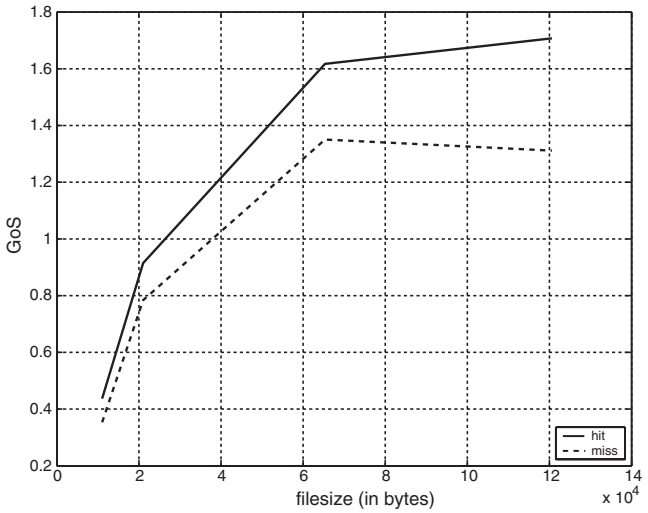

Figure 5. Comparing GoS in the case of cache hit and cache miss.

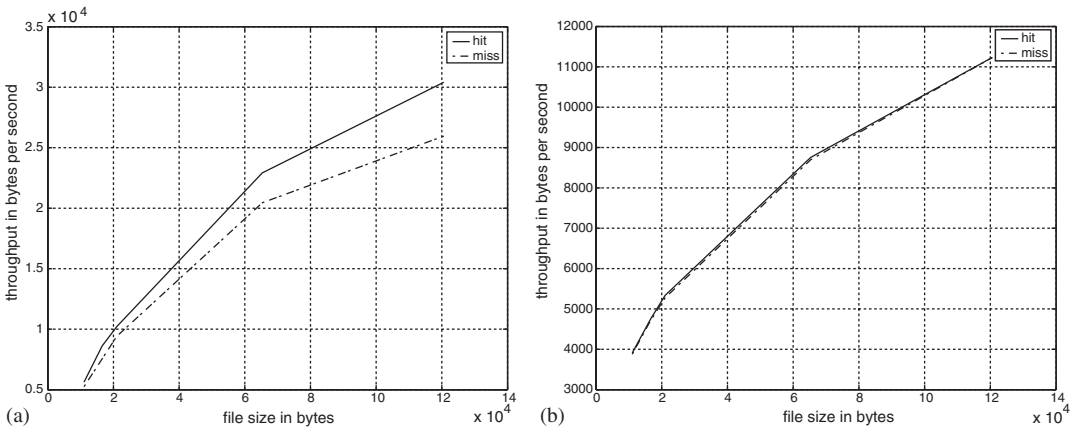

Figure 6. Comparing throughput for cache hit and cache miss with (a) splitting enabled and (b) splitting disabled.

repeatedly over a 1 h period. Figure 5 shows the GoS of five files ranging from 10 to 120 kbytes in the cache hit and the cache miss cases. Each point is an average over the 1 h download period.

An interesting observation from this comparison is that when a file is in the cache, the splitting gain is higher. This can be more clearly seen in Figures 6(a) and 6(b), where we compare the file transfer throughput separately for the cache hit and cache miss cases.

It becomes obvious, by comparing the two, that the use of splitting enhances the value of caching, i.e. when splitting is used, having the file in the cache provides significant increase in throughput over the case where the file has to be retrieved remotely from the server. In addition, this improvement increases as the file size increases. On the other hand, when splitting is disabled, whether the file is in the cache or not makes very little difference.

The reason lies in the following. Consider the case where the connection is not split by the proxy. Assuming there is a cache miss (Figure 3(c)), since the cache–client connection is much slower (as a result of higher propagation delay) than the server–cache connection, by the time the cache–client connection gets to the first few windows, the entire file could be available in the cache (i.e. the server–cache connection is completed). As an example consider 1 kbyte packet size and a 7 kbyte file. Assume that the RTT of the server–cache connection is 50 ms and the RTT of the cache–client connection to be 500 ms. As the first packet arrives at the cache it is immediately sent to the client. It takes two more RTTs for the whole file to be available in the cache (100 ms), but by this time the first packet has not even reached the client yet. By the time the first acknowledgement arrives at the cache from the client, the file will be completely available in the cache. So a cache hit and a cache miss have about the same latency and throughput.

Therefore, having the file locally in the cache provides very limited benefit. Intuitively when an end-to-end connection is split in two, the slower segment (as a result of higher propagation delay, smaller initial window size, or higher losses) will dominate the end-to-end performance. A more precise analytical explanation is out of the scope of this paper, but can be found in Reference [18]. In this case the cache–client segment is much slower than the server–cache segment, and clearly dominates the end-to-end performance. Having the file locally in the cache has the effect of 'speeding up' the server–cache connection, i.e. this connection is completely eliminated. However, since the overall performance is governed by the cache–client connection, whether the server–cache connection is a bit faster or not does not matter much, as shown in Figure 6(b).

Now consider the case where the connection splitting proxy is enabled. Splitting the connection at the gateway results in either three or two segments of an end-to-end connection (Figures 3(a) and 3(b), respectively). As we have just discussed, if the proxy only splits the connection, then the server–cache connection and the cache–proxy connection would still be much faster than the satellite link and therefore the proxy–client connection would again dominate the overall performance. However, in addition to splitting the connection, the proxy also opens up the window size over the satellite link much faster by using an initial window size of 64 kbytes and thus bypassing the slow-start stage of normal TCP window evolution. This means that the satellite link is now comparable to or even faster than the server–cache and cache–proxy connections in terms of throughput. For instance, for a file smaller than 64 kbytes, the entire file fits into the very first window. Therefore, the transfer of the file is constrained by how fast the proxy receives rather than how fast the proxy can send since the window size would be perceived as 'unlimited' for such a file. Thus having the file in the cache (much closer to the proxy) would enable the proxy to receive much faster than having to fetch the file remotely, and results in higher throughput and lower latency. This result highlights the importance of optimizing different segments of a split connection. More importantly, such optimization has to be done in a way to reduce asymmetry between the segments, e.g. to bring the slower link faster, which in this case corresponds to using a large initial window size over the satellite link.

3.3. Effect of simultaneous connections

Results from the previous section were obtained by having 10 simultaneous connections to the HGW from each client. We showed that receiving faster at the proxy results in higher throughput. However, the proxy's sending rate can be constrained by the number of

connections it is handling. That is, although each connection has a large initial window size, this large window does not get fully utilized fast enough due to the number of simultaneous connections. In this section, we compare the performance as a result of different number of simultaneous connections from the same host.

Our measurements are taken as follows. First we simultaneously download six files, Files 1, 3, 6, 10, 13 and 16, repeatedly over a period of 1 h from each host. Then we repeat this process with nine simultaneous downloads, for Files 1, 3, 4, 6, 8, 10, 13, 15 and 16, and 14 simultaneous downloads, for Files 1, 3–4, 6–8, 10–13, 15–17 and 19. In this experiment, connections do not go through the cache and files are directly originated from the remote server. The same experiment is run on both the proxy-enabled host and the proxy-disabled host (corresponding to Figures 3(e) and 3(f)). Figure 7 shows the throughput of split connection and end-to-end connection under these three scenarios and Figure 8 compares their GoS.

As can be seen from these results, connection splitting suffers more from higher number of simultaneous connections than end-to-end connections (although the gain is still positive). This is mostly due to the fact that with more simultaneous connections, the large initial window size of the split connections becomes less effective.

As mentioned earlier, the maximum overall throughput achieved per client was observed to be between 300–400 kbytes/s. Although each of the split connections has an initial window size of 64 kbytes, when there are multiple simultaneous connections, this 64 kbytes cannot be fully utilized all at once. The effective share of window size each connection actually realizes decreases with the increase in the number of simultaneous connections. For connections with the proxy disabled, the initial window size is much smaller than 64 kbytes, and therefore the impact of increased number of simultaneous connections is also much smaller.

Another reason for this could be due to the higher round trip time in end to end. Both end to end and split-connection packets experience the same queuing delay at the Satellite Gateway. It is well known that the steady-state throughput of TCP is inversely proportional to RTT (see for example, References [23,24]). For the split-connection case the RTT (R) is obviously smaller than the end-to-end case (R'). Now we are adding a constant q (queuing delay) to the RTT, in

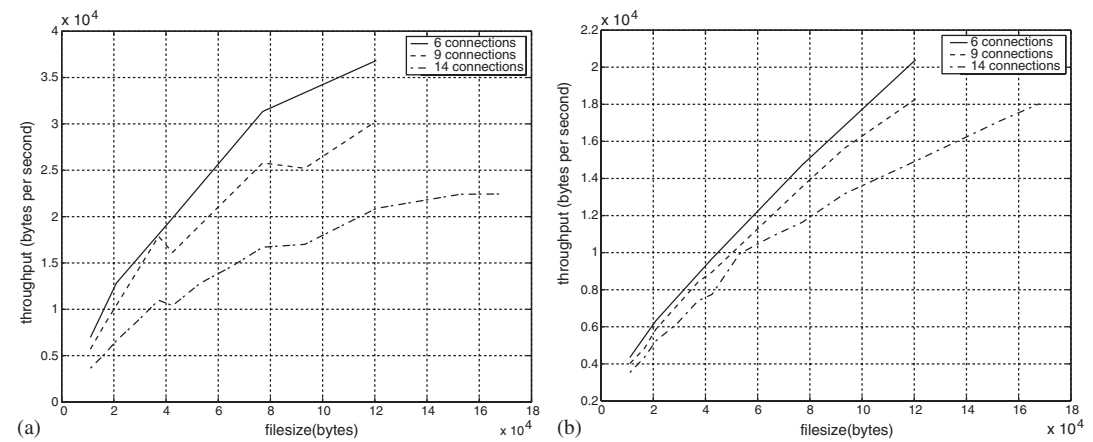

Figure 7. Throughput for different number of simultaneous connections:
(a) Splitting enabled; (b) splitting disabled.

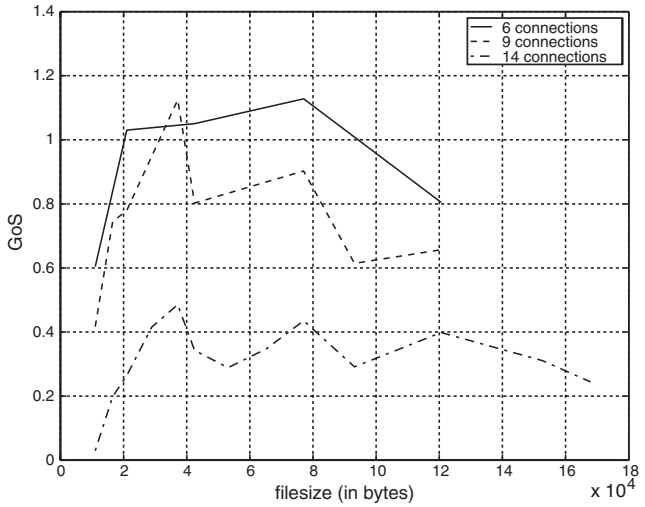

Figure 8. GoS with different number of simultaneous connections.

both cases. Since $R < R'$ we expect this constant to have a larger impact on the steady-state throughput of split connections.

We note that the results shown in Figures 7 and 8 are obtained over different time periods, and therefore could reflect different traffic load and congestion levels in the network. However, the same experiments were repeated several times and each time the results show the same trend with similar measurements. Therefore, although random fluctuations in traffic load do exist in the network, the results we show here is typical and representative of the performance and change in performance under the given scenarios.

3.4. Effect of congestion and packet losses

By examining the traces of each file download, we can determine the exact number of losses and retransmissions occurred per connection. However, such losses could involve both random and congestion losses, the distinction of which not directly available to us by only taking measurements at the end point. On the other hand, congestion and losses are highly correlated with increased end-to-end delay, which is observable. In this section, we illustrate the relationship between increased file transfer delay and the gain from using connection splitting. In doing so we attempt to understand the relationship between the splitting gain and congestion/losses.

First we repeatedly download a file directly from the server for 2 h so that the resulting trace may reflect a reasonable range of work load and congestion conditions in the network. We then sort the latency trace of the proxy-enabled connection in descending order, and reorder the proxy-disabled trace accordingly. These two cases corresponds to that illustrated in Figures 3(a) and 3(c). Figure 9 shows the reordered traces for Files 3 (11 kbytes) and 16 (120 kbytes). Figure 10 shows the GoS for these two files. It can be seen that the gain decreases as the latency of the proxy-enabled connection increases. This decrease is much steeper for the 11 kbyte file and there is a sizable portion of samples showing the proxy-enabled connections under-performing the proxy-disabled connections. This, however, is not observed in the case of the 120 kbyte file. We

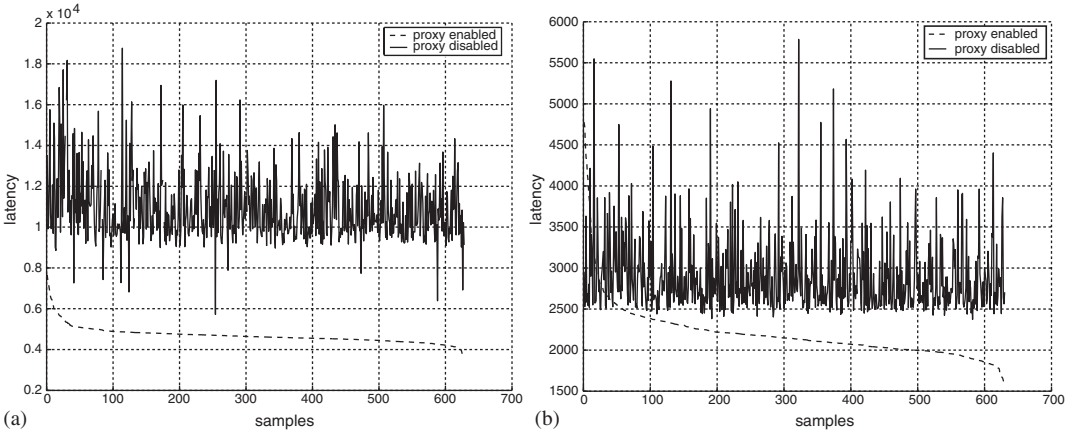

Figure 9. Sorted latency traces in case of a cache miss for (a) File 12 (120 K) and (b) File 2 (11 K).

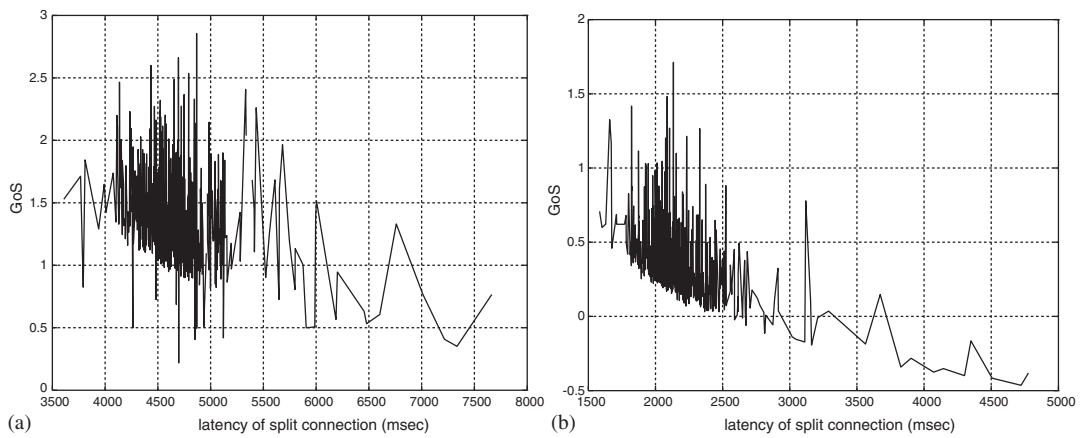

Figure 10. GoS in case of a cache miss for: (a) file 12 (120 K) and (b) file 2 (11 K).

define the *guarantee of performance improvement* as number of samples with GoS > 0 divided by total number of samples.

We then repeat the same experiment with the same files, but this time files are directly from the cache (corresponding to Figures 3(b) and 3(d)). Figures 11 and 12 show the latency and GoS in this case. Note that the correlation between the slowdowns of proxy-enabled and proxy disabled connections is not very obvious. This is due to the fact that the slowdowns are mostly caused by random packet losses rather than persistent congestion.

There are two main observations from these results. (1) The splitting gain decreases as the file transfer latency increases due to higher loss and/or congestion; (2) Whether the file is directly from the cache or from the remote server, the proxy-enabled connections experience worse performance (higher latency) than the proxy-disabled connections for small file transfers (e.g. 11 kbyte), for a small portion of the samples. This portion is reduced in the case of a cache hit. The same phenomenon was not observed for large file transfers (e.g. 120 kbyte). The

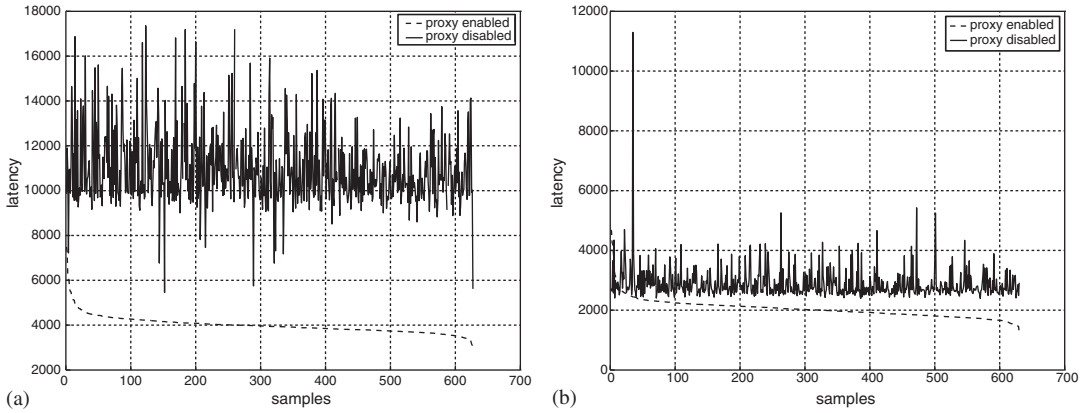

Figure 11. Sorted latency traces in case of a cache hit for: (a) file 12 (120 K) and (b) file 2 (11 K).

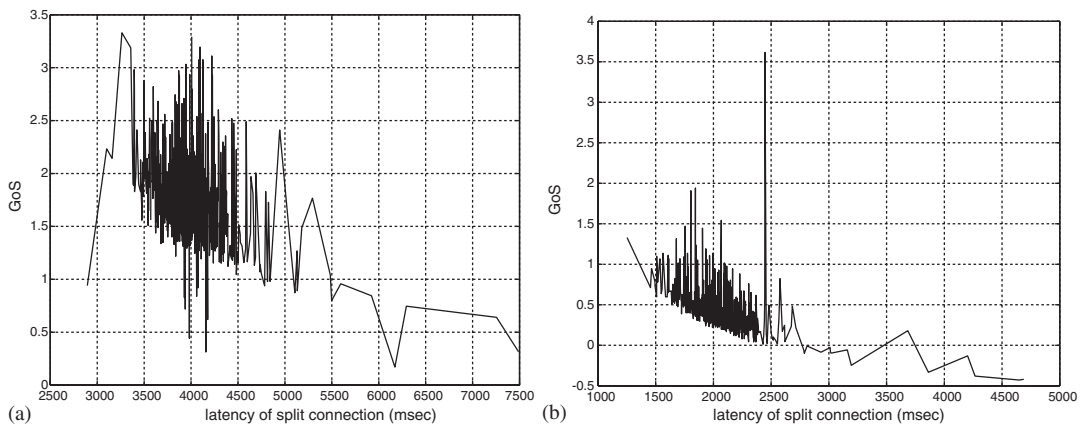

Figure 12. GoS in case of a cache hit for: (a) file 12 (120 K) and (b) file 2 (11 K).

percentages of negative GoS for different files are listed in Table I (guarantee of performance improvement for a given file size is one minus the corresponding term in the table). Note that the average performance of using the splitting proxy is still above that of disabling the proxy.

Since our connections go through a dedicated proxy, the fluctuation in file transfer latency as seen from these traces is mainly due to the fluctuation in work load, congestion and loss situations elsewhere along the path of the connection, i.e. from the server to the cache (in the case of a cache miss), from the cache to the proxy, and from the proxy to the end host. (Note that in general, connection splitting leading to worse performance can be caused by excessive congestion and delay at the proxy, which is only experienced by split traffic, but not by end-to-end traffic. This can happen if the proxy handles many split connections at the TCP layer, while the end-to-end traffic simply goes through the IP queue and is unaffected. However, since we use a dedicated proxy the increase in delay and loss incurred by splitting is minimal.) The reason that connection splitting can result in higher latency for small files lies within the relationship between the reduction in latency due to splitting and the increase in latency due to losses in

Table I. Percentage of samples where disabling the proxy outperforms enabling the proxy.

File no.	Size (bytes)	Percentage of GoS < 0 (cache hit)	Percentage of GoS < 0 (cache miss)
3	11033	2.2	3.49
4	16565	0.79	1.26
6	21027	0.0	0.48
12	65331	0.16	0.0
16	120579	0.0	0.0

general, both as a function of file size. The reduction in latency by using the proxy is largely due to the fact that the end-to-end TCP is broken down into shorter feedback loops and that a large initial window size is used over the satellite. When the file is small, one more or one less lost packet during the transfer, and whether the loss occurs sooner or later can result in significant difference in latency. This difference may not be compensated by the benefit from using the splitting proxy when the file is relatively small since the transfer completes soon afterwards. However, as the file size increases, the benefit of splitting becomes more prominent. In other words, it would take more packet losses for a split connection to perform at a similar level as a non-split connection, which can have very small probability considering the fact that a long connection tends to reflect more average loss rate. When the file is located in the cache, one segment (server–cache) is eliminated from the path, thus reducing the probability of packet losses and consequently reducing the probability that a particular split connection experiences longer latency than a non-split connection due to different loss occurrences.

In summary, when the file size is large split connections can sustain more losses than non-split connections and still yield shorter latency, thus provide higher probability of performance improvement. When the file is small, the split connection is more affected by packet losses and therefore the probability of performance improvement is lower.

3.5. Effect of embedded objects

So far all our measurements are taken from full text files transferred using HTTP. In this and the next section we examine the effect of embedded objects in a file/page, and the effect of using persistent connection.

We first compare the latency for files with same size but different numbers of embedded objects. We repeatedly download Files 12, 39 and 43 over a 2 h period. File 12 is a text file, and Files 39 and 43 contains 7 and 19 equal-sized embedded objects, respectively. The total size of each of these three files is about 65 kbytes. In downloading these files, HTTP/1.0 [21] is used between the end hosts and the proxy or the cache. No parallel connections are used. The throughput of proxy-enabled and proxy-disabled transfers is shown for both the cache miss (corresponds to Figures 3(a) and 3(c)) and cache hit (corresponds to Figures 3(b) and 3(d)) cases, in Tables II and III, respectively.

We see that when a file contains a larger number of embedded objects the GoS decreases. This result is expected considering our observations in Sections 3.1 that the gain from using the splitting proxy decreases as the file size decreases (before the satellite channel is fully utilized). This is because the time spent in handshake has a bigger portion in the overall latency for smaller files. Thus, if we break a large file into many small objects and open a new connection for each of these objects, we expect to see a lower performance gain.

Table II. Throughput of files with different number of embedded objects (in the case of a cache miss).

No. of embedded objects	Proxy-enabled (bytes/s)	Proxy-disabled (bytes/s)	GoS
0	32124	12544	1.56
7	6027	3663	0.64
19	2701	2026	0.33

Table III. Throughput of files with different number of embedded objects (in the case of a cache hit).

No. of embedded objects	Proxy-enabled (bytes/s)	Proxy-disabled (bytes/s)	GoS
0	31451	15899	0.98
7	6006	3845	0.56
19	2772	2110	0.31

It should be noted that the measurements in Tables II and III are taken over different time periods and thus numbers from different tables are not directly comparable.

3.6. Effect of persistent connection

In this section, we explore the performance comparison between using HTTP/1.0 and HTTP/1.1 [22]. Web browsers are typically configured to use HTTP/1.0 in the proxy mode where a separate connection is established for each embedded object on a page. HTTP/1.0 generally results in large delay due to the time spent in handshaking and low throughput in the slow-start phase of TCP. HTTP/1.1 on the other hand opens up a *persistent* connection which is used to deliver both the base page and subsequent embedded objects. Latency is thus reduced since with a single connection, there is only one handshake procedure and one slow-start stage of TCP. There has been extensive studies on the performance of different versions of HTTP, see for example, References [25,26].

Here we compare HTTP/1.0 vs HTTP/1.1 with the option of enabling and disabling the splitting proxy. The connection set-up of this part of our measurements corresponds to Figures 3(e) and 3(f), i.e. the connections do not go through the cache and that the connection is either end to end (proxy disabled) or split into two (proxy enabled).

For comparison purposes we summarize the results obtained from different combination of splitting enabled/disabled and persistent connection enabled/disabled in Tables IV and V. These results are obtained with a total transfer size of 65 kbytes.

The graphical representation of GoS for three transfer sizes (base page plus embedded objects) vs different number of embedded objects are shown in Figure 13. This figure corresponds to 16, 65 and 180 kbyte file transfer sizes.

From these results we see that overall the gain from using connection splitting decreases with increased number of embedded objects in a file (of the same size). This is consistent with results from the previous section. However, when persistent connection is used, GoS decreases much faster than the case for non-persistent connections as shown in Tables IV, V and Figure 13 (by ignoring the first point with 0 embedded object since that is a text file). As the number of embedded objects increases, the splitting gain under persistent connection and non-persistent

Table IV. Comparison between splitting enabled and disabled with non-persistent connection.

No. of embedded objects	0	2	4	9	11	19	23
Throughput (kbytes/s) for non-persistent, splitting disabled	7.30	5.00	3.54	3.00	2.36	2.12	1.91
Throughput (kbytes/s) for non-persistent, splitting enabled	21.84	10.63	6.14	3.77	2.97	1.83	1.67
GoS	1.99	1.12	0.73	0.25	0.25	−0.13	−0.12

Table V. Comparison between splitting enabled and disabled with persistent connection.

No. of embedded objects	0	2	4	9	11	19	23
Throughput (kbytes/s) for persistent, splitting disabled	6.99	5.95	4.23	3.95	4.09	3.95	3.44
Throughput (kbytes/s) for persistent, splitting enabled	21.38	15.99	10.29	6.68	5.56	3.62	3.19
GoS	2.05	1.68	1.43	0.69	0.35	−0.08	−0.07

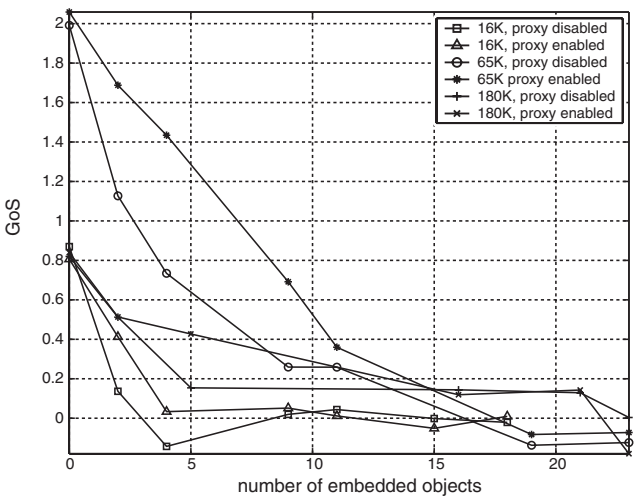

Figure 13. GoS with varying number of embedded objects.

connection starts to converge. The decreasing gain is due to the fact that the client requests each object individually (although within the same connection) after receiving the base page. Depending on the number of objects and the size of these objects, using end-to-end connection could achieve very similar performance to that of a split connection. Here pipelining is not used. Note that although we did not provide any measurement for the case of persistent connection with pipelining, its effect can be predicted using Table V. When pipelining is used, all the requests for embedded objects are sent at the same time after receiving the base page. This is similar to having one large embedded object instead of many small ones. Using Table V as well as Figure 13 (moving toward the left) we would expect pipelining to result in higher GoS.

To better examine the effect of persistent connection we further define the *Gain of Persistent Connection* (GoP) as follows:

$$\mathrm{GoP} = \frac{\mathrm{Throughput}_{\text{persistent}} - \mathrm{Throughput}_{\text{non-persistent}}}{\mathrm{Throughput}_{\text{non-persistent}}}$$

Figures 14–16 show the GoP for three transfer sizes.

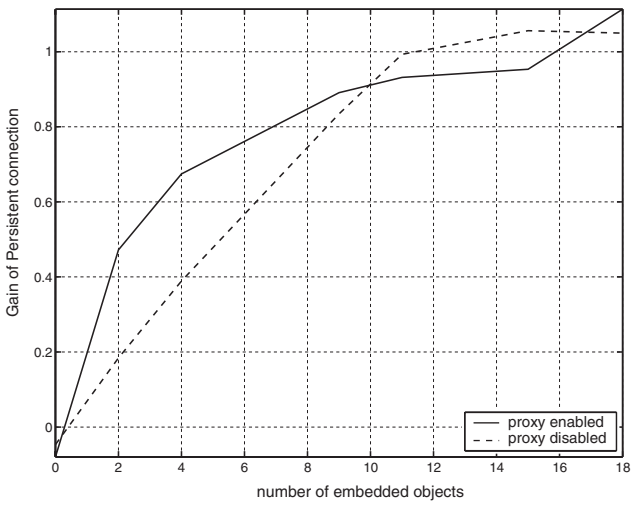

Figure 14. GoP with total transfer size of 16 kbytes.

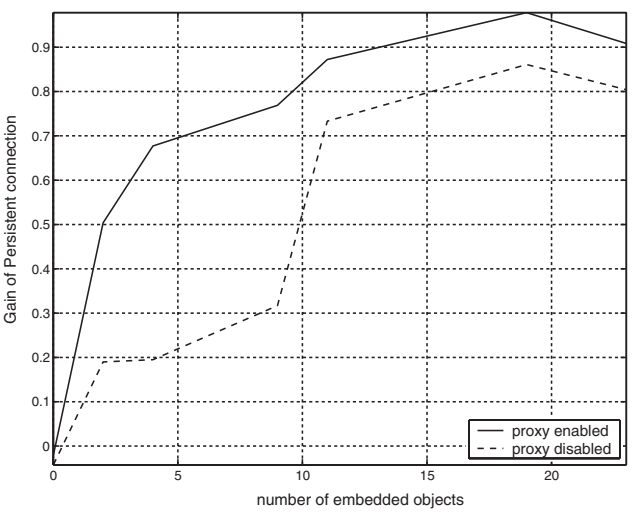

Figure 15. GoS with total transfer size of 65 kbytes.

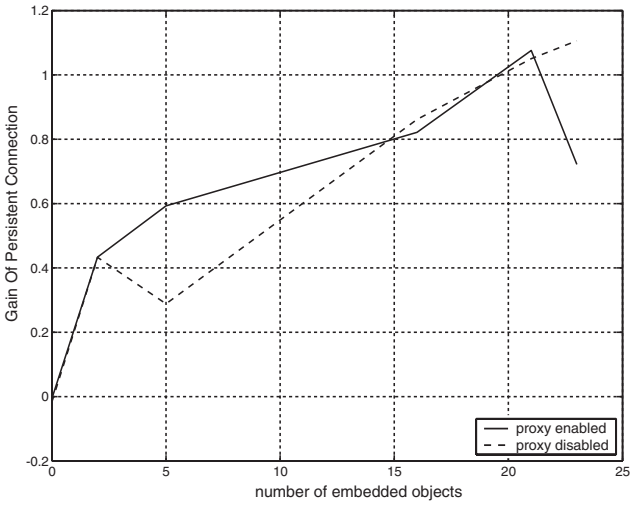

Figure 16. GoS with total transfer size of 180 kbytes.

The general observation from this set of figures is that using persistent connection provides higher performance gain over non-persistent connection when the proxy is enabled. This can be explained as follows. If we denote by d_p and d_n the total transfer delay when persistent connection is used and non-persistent connection is used with the proxy enabled, then approximately

$$d_p = d_n - (k-1)\mathrm{RTT}$$

where k is the number of objects, and RTT is the round trip time between the proxy and the client, denoted by R_2. This is because by using persistent connection we save one RTT per object on connection establishment (without pipelining). This approximation also ignores the connection between the server and the proxy. Therefore, we have

$$\mathrm{GoP} = \frac{1/d_p - 1/d_n}{1/d_n} = \frac{(k-1)R_2}{d_p}$$

Similarly for the proxy disabled case, we have

$$\mathrm{GoP}' = \frac{(k-1)(R_1 + R_2)}{d'_p}$$

where R_1 is the RTT between the server and the proxy and d'_p is the total transfer latency when using persistent connection with the proxy disabled. Since d'_p is much larger than d_p for k not too big, we have GoP > GoP′. As k increases, however, this relationship can reverse. Note that the above analysis is much simplified for the purpose of illustrating the relationship only and not intended to be rigorous.

Finally we compare non-persistent connection with proxy vs persistent connection without proxy for a total file transfer size of 65 kbytes, as shown in Table VI (we did not compare HTTP/1.0 when proxy is disabled vs HTTP/1.1 when proxy is enabled, as we do not see particular interest in this case).

Table VI. Comparison between splitting enabled with persistent connection and disabled with non-persistent connection.

No. of embedded objects	0	2	4	9	11	19	23
Throughput (kbyte/s) for persistent, splitting disabled	6.99	5.95	4.23	3.95	4.09	3.95	3.44
Throughput (kbyte/s) for non-persistent, splitting enabled	21.84	10.63	6.14	3.77	2.97	1.83	1.67
GoS	2.03	0.94	0.54	−0.06	−0.47	−0.99	−0.93

We see that the gain from using connection splitting quickly becomes negative, i.e. split connection results in smaller throughput than end-to-end connection. This shows that using a persistent connection is far more important than connection splitting for a file that contains multiple embedded objects (for even a relatively small number) since the time spent in handshake in this case is significant over the satellite link. In other words, the gain from using connection splitting cannot substitute for the performance improvement achieved by having a persistent connection. Therefore, in an Internet over satellite system, configuring the browser to run the default HTTP/1.0 with a connection splitting proxy would defeat the purpose of having a proxy, unless all the files are pure text. It is thus crucial that a persistent connection be established between the client and the proxy in such a system.

4. DISCUSSIONS AND CONCLUSIONS

In this paper, we examined the gain from using a TCP connection splitting proxy along with web caching under various scenarios by taking measurements from a live commercial system. In these experiments, not all parameters are controllable, e.g. the bit error rate over the satellite link, the amounts of traffic and congestion in the network, etc. However, by properly isolating elements that contribute to the result, not only does our measurement confirm earlier studies on connection splitting, but it also revealed additional insights into the common performance enhancing techniques used for Internet over satellite.

To summarize, connection splitting is a valid approach to improve TCP throughput and reduce file transfer latency over the satellite. The performance gain increases as the file size increases, but decreases as the number of embedded objects in the file increases. This gain is also very sensitive to the number of simultaneous connections and to congestion and packet losses, especially for small files. On the one hand having a cache hit will alleviate such sensitivity, on the other hand using connection splitting proxy enhances the benefit of caching. We also showed that although connection splitting improves throughput, it is no substitute for persistent connection. The best performance is achieved by using both the splitting proxy and persistent connection (HTTP/1.1) between the proxy and the client.

By segregating an end-to-end connection into separate connection segments we have shortened TCP feedback control loops. Thus for each of these connection segments we have a smaller RTT and lower loss probability p comparing to the original end-to-end connection. Since the TCP throughput is inversely proportional to RTT and $\sqrt{p}$ (see for example, References [23,24]), having smaller connection segments naturally results in

higher throughput. However, these segments are not completely un-coupled, and the slowest segment will constrain the other segments and dominate the end-to-end performance. The proxy cannot forward client data it has not received from the server. Therefore, if the proxy is constrained by slow receiving then increasing the rate at which proxy receives will improve the overall performance. This is why cache hits improve the throughput when the splitting proxy is enabled. On the other hand if we only use the cache, we will have a cache–client connection highly constrained by slow sending due to the large propagation delay. This is why having a large initial window size over the satellite link is so important. Overall this illustrates the importance of properly provisioning the proxy and the NOC, and the importance of optimizing each split segment to reduce the asymmetry among them so that one is not constrained by the other.

Our measurement and subsequent results are obtained from a live system, over which we have limited control. These results therefore may reflect the combined effect of a range of factors. We have considered the most prominent factor(s) and cause(s) for each of these results, but we strongly believe that deriving a detailed quantitative analysis will help explain the exact role of each factor and serve as good supplement to our measurement-based study. This is part of our current research.

APPENDIX

Table A1. Files used for measurements.

File no.	Type	Size (kbytes)	File no.	Type	Size (kbytes)
1	Full text	2.5	26	Full text	720
2	Full text	10	27	Full text	810
3	Full text	11	28	Full text	900
4	Full text	16	29	Full text	1000
5	Full text	20	30	2 Embedded objects	16
6	Full text	21	31	4 Embedded objects	16
7	Full text	28	32	9 Embedded objects	16
8	Full text	37	33	11 Embedded objects	16
9	Full text	40	34	15 Embedded objects	16
10	Full text	42	35	18 Embedded objects	16
11	Full text	53	36	2 Embedded objects	65
12	Full text	65	37	3 Embedded objects	65
13	Full text	77	38	4 Embedded objects	65
14	Full text	80	39	7 Embedded objects	65
15	Full text	93	40	9 Embedded objects	65
16	Full text	120	41	11 Embedded objects	65
17	Full text	152	42	18 Embedded objects	65
18	Full text	160	43	19 Embedded objects	65
19	Full text	168	44	23 Embedded objects	65
20	Full text	180	45	2 Embedded objects	180
21	Full text	240	46	5 Embedded objects	180
22	Full text	320	47	16 Embedded objects	180
23	Full text	480	48	21 Embedded objects	180
24	Full text	560	49	23 Embedded objects	180
25	Full text	640			

REFERENCES

1. Parsa C, Garcia-Luna-Aceves JJ. Improving TCP performance over wireless network at the link layer. *Mobile Networks and Applications* 2000; **5**(1):57–71.
2. Parsa C, Garcia-Luna-Aceves JJ. TULIP: a link-level protocol for improving TCP over wireless links. *Proceedings of IEEE WCNC'99*, 1999; 1253–1257.
3. Ratnam K, Matta I. WTCP: an efficient mechanism for improving TCP performance over wireless links. *Proceedings of IEEE ISCC*, 1998; 74–78.
4. Ratnam K, Matta I. Effect of local retransmission at wireless access points on the round trip time estimation of TCP. *Proceedings of 31st Annual Simulation Symposium*, 1998; 150–156.
5. Cáceres R, Iftode L. Improving the performance of reliable transport protocol in mobile computing environments. *IEEE Journal on Selected Areas in Communications* 1995; **13**(5):850–857.
6. Vaidya NH, Mehta M, Perkins C, Montenegro G. Delayed duplicated acknowledgments: a TCP-unware approach to improve performance of TCP over wireless. *Technical Report* 99-003, *TAMU*, 1999.
7. Biaz S, Vaidya NH. Discriminating congestion losses from wireless losses using inter-arrival times at the receiver. *Proceedings of IEEE ASSET*, 1999; 10–17.
8. Balakrishnan H, Seshan S, Amir E, Katz RH. Improving TCP/IP performance over wireless networks. *ACM/IEEE International Conference on Mobile Computing and Networking* (*MobiCom'95*), vol. 2(11), 1995.
9. Balakrishnan H, Padmanabhan VN, Seshan S, Katz RH. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking* 1997; **5**(6):756–769.
10. Badrinath BR, Sudame P. To send or not to send: implementing deferred transmissions in a mobile host. *Proceedings of IEEE ICDCS*, 1996; 327–333.
11. Bharadwaj VG. Improving TCP performance over high-bandwidth geostationary satellite links. Technical Report, MS 99-10, Institute for Systems Research, University of Maryland, College Park, 1999, http://www.isr.umd.edu/TechReports/ISR/1999/.
12. Bakre A, Badrinath BR. I-TCP: indirect TCP for mobile hosts. *Proceedings of IEEE ICDCS*, 1995; 136–143.
13. Bakre AV, Badrinath BR. Implementation and performance evaluation of indirect TCP. *IEEE Transactions on Computers* 1997; **46**(3):260–278.
14. Allman M, Dawkins S, Glover D, Tran D, Henderson T, Heidemann J, Touch J, Kruse H, Ostermann S, Scott K, Semke J. Ongoing TCP research related to satellites. *IETF RFC* 2760, 2000.
15. Karir M. IPSEC and the internet. *Technical Report MS* 99-14, Institute for Systems Research, University of Maryland, College Park, 1999, http://www.isr.umd.edu/TechReports/ISR/1999/.
16. Ishac J, Allman M. On the performance of TCP spoofing in satellite networks. *IEEE Milcom*, 2001.
17. Rodriguez P, Sibal S, Spatscheck O. TPOT: translucent proxying of TCP. *Technical Report*, *AT & T labs-Research and EURECOM Technical Report*, 2000.
18. Liu M, Ehsan N. Modeling TCP performance with proxies. *International Workshop on Wired/Wireless Internet Communications* (*WWIC*), *in conjunction with International Conference on Internet Computing* (*IC'02*), June 2002.
19. Mathis M, Mahdavi J, Floyd S, Romanow A. TCP selective acknowledgement options. *IETF RFC 2018*, 1996.
20. Balakrishnan H, Padmanabhan VN, Katz RH. The effects of asymmetry on TCP performance. *ACM/IEEE International Conference on Mobile Computing and Networking* (*MobiCom'97*), 1997; 77–89.
21. Berners-Lee T, Fielding R, Frystyk H. Hypertext transfer protocol – HTTP/1.0. *IETF RFC 1945*, 1995.
22. Fielding R, Gettys J, Mogul J, Frystyc H, Berners-Lee T. Hypertext transfer protocol – HTTP/1.1. *IETF RFC 2068*, 1997.
23. Padhye W, Firoiu V, Towsley DF, Kurose JF. Modeling TCP reno performance: a simple model and its empirical validation. *IEEE Transactions on Networking* 2000; **8**(2):133–145.
24. Lakshman TV, Madhow U, Suter B. TCP performance with random loss and bidirectional congestion. *IEEE Transactions on Networking* 2000; **8**(5):541–555.
25. Heidemann J, Obraczka K, Touch J. Modeling the performance of HTTP over several transport protocols. *IEEE/ACM Transactions on Networking* 1997; **5**(5):616–630.
26. Nielsen HF *et al*. Network performance effects of HTTP/1.1, Css1, and PNG. *Technical Report* 1997, http://www.w3.org/TR/NOTE-pipelining.

AUTHORS' BIOGRAPHIES

Navid Ehsan was born in Tehran, Iran on 23 September 1976. He received his BS in Electronics Engineering in 1998 and MS in Communications in 2002 from Sharif University of Technology and University of Michigan, respectively. He is currently a PhD student working on bandwidth allocation in satellite networks at university of Michigan.

Mingyan Liu (S'96-M'00) received the BS degree in electrical engineering from the Nanjing University of Aero. and Astro., Nanjing, China, in 1995, MS degree in systems engineering and PhD degree in electrical engineering from the University of Maryland, College Park, in 1997 and 2000, respectively.

She joined the University of Michigan, Ann Arbor, in September 2000, and is currently an Assistant Professor with the Department of Electrical Engineering and Computer Science. Her research interests are in performance modelling, analysis, energy efficiency and resource allocation issues in wireless mobile *ad hoc* networks, wireless sensor networks, and terrestrial/satellite hybrid networks.

Roderick J. Ragland (S'78-M'78) received the BS degree in computer engineering from the University of Michigan, Ann Arbor, in 1978 and the MS degree in Electrical Engineering from the Georgia Institute of Technology, Atlanta, in 1979.

From 1979 to 1980 he was with AT&T Bell Laboratories in Naperville, IL. From 1980 to 1985 he was with Tellabs Incorporated in Lisle, IL. From 1985 to 1990 he was with the Gould Research Centre, which was acquired by the Martin Marietta Aero and Naval Division in Glenn Burnie, MD, in 1986. From 1990 to 1997 he was with COMSAT Laboratories. While at COMSAT Laboratories, he was a scientist in the Voice-Band Processing Department, developing real-time hardware and software applications of digital signal processing, speech and bilevel image transmission over satellite networks, secure communications, and digital filtering. During his final years at COMSAT Laboratories, he worked in the Network Technology Group developing real-time Internet applications for the company's next generation satellite Very Small Aperture Terminal (VSAT) product. Presently, he is a Technical Director with Hughes Network Systems in Germantown, MD developing software applications for the company's DIRECWAY 2-way satellite network products. His research and development interests include computer network traffic management; congestion controls and traffic engineering for satellite related data networks.