Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT

Computer Communications 27 (2004) 961–975

computer communications

www.elsevier.com/locate/comcom

Modeling TCP performance with proxies

Navid Ehsan*, Mingyan Liu

Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122, USA

Abstract

This paper investigates TCP dynamics and performance over proxies that shorten the TCP feedback loop by segmenting the end-to-end connection. Such proxies are often used to improve the TCP performance, e.g. a splitting/spoofing proxy in satellite communication, and more commonly, a web cache. Through analysis, we attempt to develop a basic understanding of the properties of TCP dynamics when such proxies are used, and further obtain design principles of systems involving such proxies. We present simple models capturing some features of the proxy performance in both the lossless and lossy scenarios. Due to the complexity involved, detailed analysis is only available in the lossless scenario, and our discussion in the lossy scenario is largely limited to steady-state behavior. Nevertheless we are able to obtain useful insight. We identify conditions under which using a proxy provides significant or marginal performance gain by investigating factors including initial window size, congestion level of the proxy, and the level of asymmetry between the segments segregated by the proxy. We also discuss how these conditions affect the deployment and provisioning of such systems.

Keywords: Performance enhancing proxy; TCP connection splitting; TCP connection spoofing; Cache; TCP performance

1. Introduction

This paper investigates certain type of proxies that cause changes in the TCP dynamics and the resulting performance implications. In particular, we focus on proxies that shorten the TCP feedback loop either by design or as a by-product. Such proxies are normally used to reduce the connection response time and achieve higher link utilization.

One typical example of such is a TCP connection splitting and spoofing proxy that pre-acknowledges the sender on behalf of the receiver (by spoofing the receiver's address), and forwards packets to the receiver on behalf of the sender (by spoofing the sender's address). Such a scheme is usually called split TCP, TCP spoofing [1] or indirect TCP (I-TCP) [2,3]. It is commonly used in satellite communication to improve TCP performance over the satellite link (which has a large bandwidth-delay product) since it can speed up the window growth and achieve higher capacity utilization, especially for short connections. It has also been proposed for terrestrial wireless networks (e.g. I-TCP) [3–5] as a means of separating the wired and the wireless part of the connection, and separating congestion losses from link failure losses. The motivation behind this approach is TCP's performance degradation in a heterogeneous environment. The idea is that if a communication path consists of physical media that have very different characteristics, the end-to-end performance is optimized by isolating one type of physical medium from another and optimizing each separately. It has to be mentioned that this approach generally violates the TCP end-to-end semantics, and will not work if the IP packet payload is encrypted [6].

Another example of such a proxy, which may seem less obvious, is a common web cache (e.g. with the web browser set to 'proxy' mode). When there is a 'hit' at the cache, a file is directly sent to the client from the cache. When there is a 'miss', 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. Thus the cache automatically 'breaks' the server–client transfer into two separate connections [5]. In terms of TCP performance during the file transfer, this has exactly the same effect as split TCP (although the connection establishment is different). However, in this case the TCP semantics is preserved because the cache does not spoof the client's address, and so it acknowledges the server on behalf of itself rather than 'pre-ack' on behalf of the client. Caching not only reduces latency by pushing the content closer to end users but also results in redirection of traffic that is meant for web servers, and can achieve better load balancing.

* Corresponding author.
E-mail addresses: nehsan@eecs.umich.edu (N. Ehsan); mingyan@eecs.umich.edu (M. Liu).

0140-3664/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2004.01.005

There has been implementation and experimental study of TCP performance improvement using such proxies, especially split TCP in satellite and terrestrial wireless communications (e.g. Refs. [1–3]). In this paper, we develop simple mathematical models to derive the TCP performance (mainly latency) when such a proxy is used, and analyze the level of performance improvement under different scenarios. Our motivation is three-fold: to have an analytical and quantitative study to gain insights into the dynamics of a shortened TCP loop in addition to simulation and experimental studies; to investigate the use of proxy as a general solution to problems involving heterogeneous links and large amounts of traffic; and more importantly, to apply such understanding to system level design issues.

In general, shortening the TCP feedback loop does not have to come at the price of breaking the end-to-end semantics, e.g. feedback from the proxy can be a signal for the server to increase its window, rather than to purge the retransmission buffer. This of course may imply that the proxy is not transparent to the end users anymore and involve modifications to the existing systems, which is out of scope of this paper. In subsequent sections, we will ignore whether the proxy spoofs addresses or not since it does not affect our analysis, and instead focus on a general model of server–proxy–client communication. The server and the client can either have an end-to-end TCP connection, where the proxy simply acts as a router, or they can have a connection in which the proxy breaks up the connection in two.

Due to the complexity involved, detailed analysis is only available in the lossless scenario, and our discussion in the lossy scenario is largely limited to steady-state behavior. Nevertheless, we are able to obtain useful insight through such analysis. In summary, we found that overall, using the proxy results in higher utilization of the link capacity and lower latency. However, when the proxy becomes congested this performance gain is limited. In addition, when a connection is broken in two, the slower one always dominates the overall performance, and as this dominance increases, the gain from using the proxy is again reduced. These results imply that while optimization of separate parts of a connection (segregated by the proxy) is important, it is equally important to minimize the 'asymmetry' between these parts, especially in a heterogeneous environment.

The organization of the paper is as follows. In Section 2 we present the network model and describe how a proxy functions. In Sections 3 and 4 we analyze the latency in file transfer with or without using the proxy under the lossless and lossy scenario, respectively. The accuracy of our model is also evaluated and discussed. Section 5 discusses the effects of initial window size, the congestion level of the proxy, and the asymmetry between the two segments segregated by the proxy. Section 6 summarizes our results and concludes the paper.

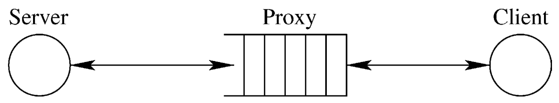

Fig. 1. Network model.

2. Split connection and the network model

2.1. Network model

Our analysis is based on a two-link model with one end host on each side and a proxy in the middle, as shown in Fig. 1. This is a simple abstraction of certain connectivity. In reality, each of the two links may contain multiple intermediate routers and physical links, which are abstracted into a single link with a single round-trip time (RTT) parameter and a single loss rate parameter. Note that under such abstraction the transmission rate of the server essentially represents the slowest physical link on the path from the server to the client. In practice, a spoofing proxy is usually placed between a wired part and a wireless (or satellite) part, and the client is usually located at the end of the wireless link. The location of a cache proxy is more arbitrary. File transfer is our main application of interest, and without loss of generality is considered to be from the server to the client.

When an end-to-end connection is established, the proxy functions as a normal router that forwards packets from the server to the client and vice versa. When connection splitting is used, the proxy acknowledges to the server, the client acknowledges to the proxy, and the proxy relays packets from the server to the client. Same procedure is used for the other direction of the connection. The two connections are inevitably coupled, but they keep separate sequence numbers and queues, and the proxy does not relay out-of-order packets from one to the other thus acting as a virtual source of the file. In general, with a spoofing proxy the initial connection establishment (three-way handshake) and the final closing are done in an end-to-end fashion. The connection is only split in two during the data transfer period, as shown in Fig. 2(a). With a cache proxy there are two separate connection from the very beginning, i.e. three-way handshake is first conducted between the client and the proxy. If there is a cache miss, another three-way handshake is conducted between the proxy and the server, as shown in Fig. 2(b). Both situations result in approximately the same delay in connection establishment for a single connection, about 1.5 times the RTT between the server and the client. We therefore do no include this initial delay in our analysis and instead focus solely on the delay of data transfer, which is the duration between when the server sends the first data packet of a file and the time when the client receives the last data packet of the file. For a cache proxy if there is a hit on the file request, the content is retrieved directly from the cache. In this case, the connection model is simply end-to-end from the client to the proxy, with a fraction of the entire

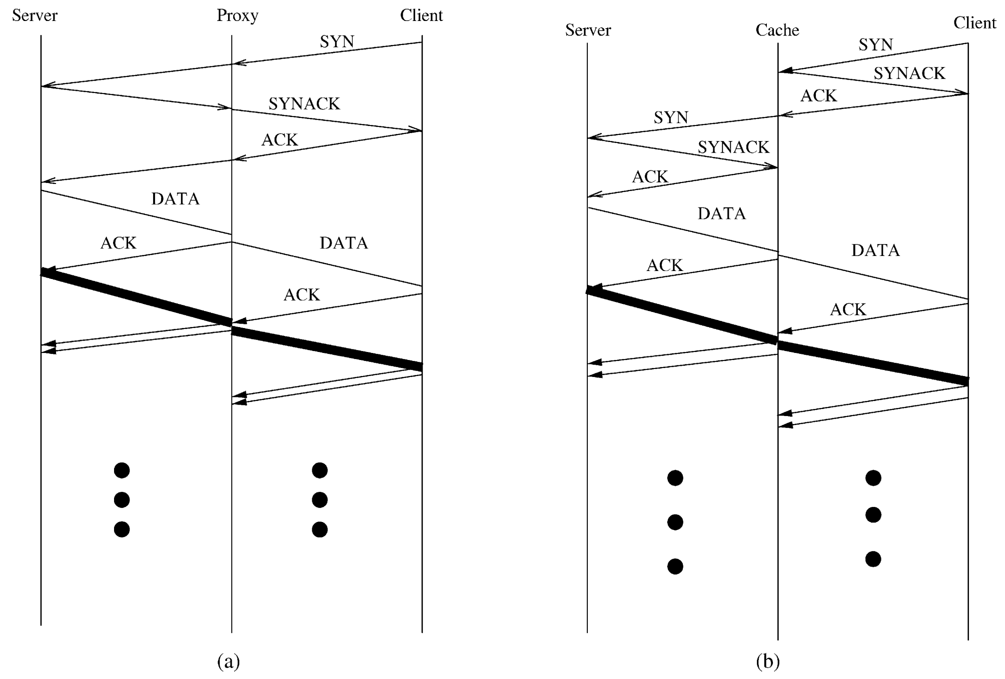

Fig. 2. File transfer using (a) a splitting proxy and (b) a cache upon miss.

server–client RTT. Our analysis, therefore only applies to situations where there is a cache miss.

2.2. Assumptions and parameters

We assume that a file contains exactly M segments of the maximum segment size (MSS). This is an approximation to an arbitrary file size whose last segment may be a fraction of MSS. However, this does not affect our method of analysis, and also does not affect the comparison between with or without using the proxy. Slow-start threshold (*ssthresh*), W_{ss}, and the maximum window size W_{max} are assumed to be in number of segments rather than number of bytes to simplify our analysis.

The server, the proxy, and the client have transmission rates of C_1, C_p, and C_2, respectively. Assuming packet length of $D = \text{MSS} + h$ (where h includes both TCP and IP headers and is typically equal to 40 bytes), the time it takes for the server to transmit a packet is $\mu_1 = D/C_1$, and $\mu_p = D/C_p$, $\mu_2 = D/C_2$ for the proxy and the client, respectively. When there are two separate connections, we assume a per-packet processing delay of t_p at the proxy. All other processing delays are ignored. We assume that each link has the same propagation delay in both directions. The one-way propagation delay on the server–proxy link (also referred to as the first link or the first connection) and the proxy–client link (also referred to as the second link or the second connection) are denoted by I_1 and I_2, respectively. Throughout our analysis, we assume that the transmission time of an ACK is negligible.

We further assume that the TCP sender is only constrained by the congestion window and not the advertised receive window size. Most work in TCP analysis assumes an infinite source, e.g. Refs. [7–9]. However, when we have two connections, the window of the second connection (proxy–client) evolves not only according to the window dynamics of TCP, but also according to the availability of packets (from the server–proxy connection), i.e. the first connection may not ‘catch up’ with the second connection due to factors like initial window size, transmission rate, etc. Therefore, the window of the second connection will be forced to grow at a slower rate. We will discuss both in the subsequent sections.

3. Delay analysis of a connection over lossless links

Assuming that the window grows in the slow-start phase and congestion avoidance stages until the maximum window size is achieved, the number of windows that is needed to cover a file of M segments can be calculated by extending the method presented in Ref. [10]. We also assume that delayed ACK is implemented. As shown in Ref. [11], since one ACK is generated for every b packets received before the timer expires, the rate of exponential growth of the congestion window is $r = 1 + (1/b)$, which equals 2 when no delayed ACK is used. Let w_o denote the initial window size. Let S be such that

$$w_o r^{S-1} < W_{ss} \le w_o r^S \tag{1}$$

if $M > \sum_{i=1}^{S} w_o r^{i-1}$, i.e. the slow-start threshold *ssthresh* is reached during the $(S+1)$th window if the file is big enough. Therefore the $(S+1)$th window size is W_{ss} and the $(S+2)$th window size is $W_{ss}+1$, and so on. Similarly, let M_x be such that

$$W_{\mathrm{ss}} + \frac{M_x - S - 1}{b} < W_{\max} \le W_{\mathrm{ss}} + \frac{M_x - S}{b} \tag{2}$$

i.e. the maximum window size is achieved during the $(M_x + 1)$th window if the file is big enough. All subsequent windows have the same window size $W_{\max}$. The number of windows needed to transfer a file is then given by the following:

$$K = \begin{cases} \min\left\{k : \sum_{i=1}^{k} w_{\mathrm{o}} r^{i-1} \ge M\right\}, & \text{if } k \le S \\ \min\left\{k : \sum_{i=1}^{S} w_{\mathrm{o}} r^{i-1} + \sum_{i=S+1}^{k} \left(W_{\mathrm{ss}} + \frac{i - S - 1}{b}\right) \ge M\right\}, & \text{if } S < k \le M_x \\ \min\left\{k : \sum_{i=1}^{S} w_{\mathrm{o}} r^{i-1} + \sum_{i=S+1}^{M_x} \left(W_{\mathrm{ss}} + \frac{i - S - 1}{b}\right) + \sum_{i=M_x+1}^{k} W_{\max} \ge M\right\}, & \text{if } M_x < k \end{cases} \tag{3}$$

3.1. Delay of an end-to-end connection

We first consider an end-to-end connection between the server and the client. Assuming that the links are lossless and that connections are only constrained by congestion window size, after the server sends a window's packets it waits for the first ACK to come back, if it takes longer for the ACK to arrive than it takes to transmit the window's worth of data. The time it takes to transmit the kth window is a function of the packet transmission time at the sender given by

$$t_k(\mu_1) = \begin{cases} w_{\mathrm{o}} r^{k-1} \mu_1, & \text{if } k \le S \\ \left(W_{\mathrm{ss}} + \frac{k - S - 1}{b}\right)\mu_1, & \text{if } S < k \le M_x \\ W_{\max}\mu_1, & \text{if } M_x < k \end{cases} \tag{4}$$

Therefore if $\mu_1 \ge \mu_{\mathrm{p}}$, which indicates that the proxy transmits at least as fast as the server and thus packets will not experience queueing delay at the proxy, the RTT of the end-to-end connection is $2(I_1 + I_2)$. The time it takes for the first ACK to arrive after the first packet was sent is equal to $b\mu_1 + \mu_{\mathrm{p}} + 2(I_1 + I_2)$.

Note that this expression assumes that there are at least b packets in a window so that the receiver can immediately return an ACK upon receipt of the bth packet. If for example $w_{\mathrm{o}} = 1$ and $b = 2$, then the receiver may have to wait for the delayed ACK timer to expired to return an ACK. For simplicity in the rest of our analysis, we will ignore this difference, which can be taken into account by straightforward modifications to our model. The total time it takes to transfer the file is then

$$T_{\mathrm{e}}(M) = M\mu_1 + I_1 + I_2 + \mu_{\mathrm{p}} + \sum_{k=1}^{K-1} [b\mu_1 + \mu_{\mathrm{p}} + 2(I_1 + I_2) - t_k(\mu_1)]^+ \tag{5}$$

where $[a]^+ = a$ for a positive and 0 otherwise. K is calculated by replacing the server's initial window size, *ssthresh* and maximum window size in Eq. (3). This latency reflects the total transmission time, the time that the server spends waiting for ACKs, and the time for the last window to reach the client.

When $\mu_1 < \mu_{\mathrm{p}}$, packets could build up at the proxy waiting to be transmitted into the slower link and experience additional queueing delay at the proxy. In this case the ACKs of the same window arrive at the server approximately μ_{p} apart instead of μ_1, thus the server may need to wait for every ACK of the same window instead of stalling after sending out the entire window. We derive the latency by examining from the client's side. Since $\mu_{\mathrm{p}} > \mu_1$, the client receives packets of the same window continuously at rate $1/\mu_{\mathrm{p}}$. The time that the client is idle is therefore $[\mu_1 + \mu_{\mathrm{p}} + 2(I_1 + I_2) + (b-1)\mu_{\mathrm{p}} - t_k(\mu_{\mathrm{p}})]^+$, where $t_k(\mu_{\mathrm{p}})$ is the time it takes the client to receive the kth window, and $t_k(\cdot)$ is given by Eq. (4). The latency is then

$$T_{\mathrm{e}}(M) = M\mu_{\mathrm{p}} + I_1 + I_2 + \mu_1 + \sum_{k=1}^{K-1} [b\mu_{\mathrm{p}} + \mu_1 + 2(I_1 + I_2) - t_k(\mu_{\mathrm{p}})]^+ \tag{6}$$

which reflects the time the client spends receiving the file, waiting for the next window, and the time for the first window to reach the client. Defining R_{e} as

$$R_{\mathrm{e}} = \min(\mu_1, \mu_{\mathrm{p}}) + b\max(\mu_1, \mu_{\mathrm{p}}) + 2(I_1 + I_2) \tag{7}$$

Eqs. (5) and (6) can be combined to obtain the following proposition

Proposition 1. *The latency of transmitting a file of M packets using an end-to-end connection is*

$$T_{\mathrm{e}}(M) = M\max(\mu_1, \mu_{\mathrm{p}}) + I_1 + I_2 + \min(\mu_{\mathrm{p}}, \mu_1) + \sum_{k=1}^{K-1} [R_{\mathrm{e}} - t_k(\max(\mu_1, \mu_{\mathrm{p}}))]^+ \tag{8}$$

3.2. Delay of a split connection

When the proxy is used, we have two serial connections. Note that these two connections are not independent but coupled by data. This is because the second connection (proxy–client) cannot send any data packets it has not received from the first connection (server–proxy) and therefore is constrained. This can be caused by a much larger initial window size and/or a much shorter RTT on the second connection. In this scenario, the second connection has a limited source depending on the sending of the first connection. In what follows we will first study the case where the server is fast enough so that the proxy is never constrained by unavailability of data, then study the case where the proxy–client connection may be constrained by the first connection.

3.2.1. Unconstrained proxy–client connection

The fact that the second connection is never constrained by the first connection could imply $w_o \geq w'_o, \mu_1 \leq \mu_p$ and/or $I_1 \leq I_2$, where w'_o is the proxy's initial window size. In the following we assume that $I_1 < I_2$, and that the server's initial window size is greater or equal to that of the proxy. Since a larger initial window size of the server–proxy connection does not change the end-to-end latency (for additional packets will simply be queued at the proxy), we consider cases where both connections use the same initial window size. We also assume that $\mu_p \geq \mu_1$. For an initial window size of w_o, assuming both connections use the same W_{ss} and W_{max}, we get S, M_x, and K as shown in Eqs. (1)–(3), for both connections. The proxy receives the first packet from the server at time $\mu_1 + I_1$. Assuming there is t_p delay for TCP processing, the proxy starts sending this packet to the client at time $\mu_1 + I_1 + t_p$. From this point on, we only need to focus on the second connection since the latency is only determined by this connection. We therefore have the following result on the latency of a split connection.

Proposition 2. *The latency of transmitting a file of M packets using a split connection where the proxy is never constrained by unavailability of data is*

$$T_p(M) = \mu_1 + I_1 + t_p + M\mu_p + \sum_{k=1}^{K'-1} [R_2 - t_k(\mu_p)]^+ + I_2 \tag{9}$$

where $t_k(\cdot)$ is given in Eq. (4), K' is the total number of windows needed for the transfer (calculated by replacing proxy's initial window size, ssthresh and maximum window in Eq. (3)), and $R_2 = b\mu_p + 2I_2$ is the time it takes for the ACK to come back to the proxy.

This latency reflects the initial delay for the first packet to arrive at the proxy, the total transmission time at the proxy, stall time and the time for the last packet to reach the client.

Fig. 3 compares the numerical results from our model with NS-2 simulation, for both the end-to-end and split connections. In this case, $C_1 = C_p = C_2 = 1$ Mbps. The initial window size is set to 1 and 4, respectively. Unless pointed out explicitly, our numerical results and simulation throughout this paper are based on the following parameters: MSS = 512 bytes, $W_{ss} = 128$ segments. Each graph contains four curves, two from NS-2 simulation (sim) and two from our model (mod). We see that each pair (sim and mod) overlaps almost completely. This shows that our analysis in the unconstrained proxy case is very accurate.

3.2.2. Constrained proxy–client connection

The conditions under which the proxy–client connection can be modeled as an infinite source, i.e. $w_o \geq w'_o, \mu_1 \leq \mu_p$ or $I_1 \leq I_2$ in a lossless situation, may be

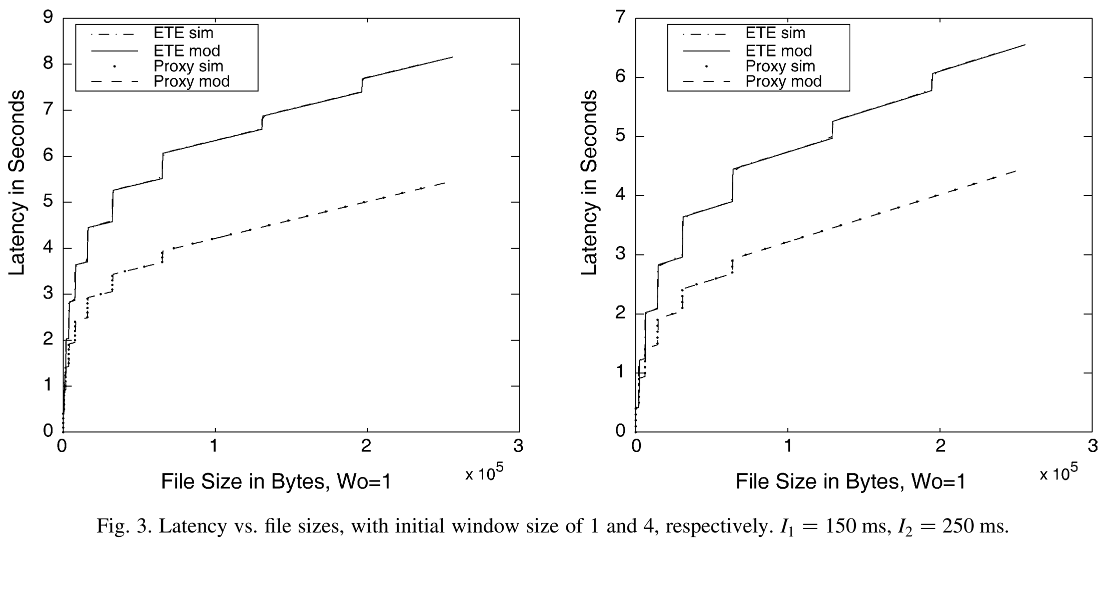

Fig. 3. Latency vs. file sizes, with initial window size of 1 and 4, respectively. $I_1 = 150$ ms, $I_2 = 250$ ms.

reasonably satisfied when the proxy–client link is a satellite link (for GEO satellite, the most commonly used for data communications, the one-way propagation delay is 250 ms), but is not in general true for a terrestrial wireless link. When these conditions do not hold, it may take certain period of time for the server–proxy connection to catch up with the proxy–client connection when the proxy starts sending packets at full window size, or it may never catch up with the proxy–client connection. During this period, the proxy–client connection will be constrained by the availability of packets to send, in addition to its window size. The dynamics of the connection with such limited source model is much more difficult to analyze, and it is not clear whether the file transfer latency can be found in a closed form as before. Below we develop a fluid flow based model to describe the coupling between the two connections, by defining the sending rates of the server and proxy as functions of time. This model can then be used to compute the file transfer latency of a split connection.

Assuming lossless links, the speed of a connection is mainly determined by the initial window size and the RTT (assuming fixed W_{ss}). We will define $R_2 = 2I_2 + b\mu_p$ for the proxy–client connection as before, and similarly define $R_1 = 2I_1 + b\mu_1$ for the server–proxy connection. Both values represent the time it takes for an ACK to reach the data sender over its respective connection.

There are four possible cases: (1) $w_o > w_o'$ and $R_1 \le R_2$, the server–proxy connection is always going to be faster than the proxy–client connection, so the analysis in Section 3.2.1 can be applied; (2) $w_o < w_o'$ and $R_1 > R_2$, the server–proxy connection can never catch up with the proxy–client connection; (3) $w_o < w_o'$ and $R_1 \le R_2$, the server–proxy connection may eventually catch up depending on the file size; (4) $w_o > w_o'$ and $R_1 > R_2$, which is exactly the opposite of (3). In this case, the proxy–client connection may be able to send at full window size for some time, but eventually will be constrained by the server–proxy connection depending on the file size. Here we mainly focus on cases (1)–(3) by noting that similar analysis can be used for case (4) as well.

Let $s(t)$, $p(t)$, $W_s(t)$, $W_p(t)$ be the sending rate (in packets per second) and the congestion window size of the server and the proxy at time t, respectively. Let $N_s(t)$ and $N_p(t)$ denote the total number of packets sent by the sever and proxy up to time t, respectively. W_{ss} and W'_{ss} denote the slow-start threshold of the two connections, respectively. To simplify our analysis, we will further assume that the pipe/link capacity is lower than, $W_{\max}$, i.e. $(R_1/\mu_1) \le W_{\max}$, $(R_2/\mu_p) \le W_{\max}$, and therefore the transmission is ultimately limited by the capacity of the link rather than the maximum window size. This assumption is adopted purely for simplicity of presentation and does not affect the applicability of the analysis and the result it provides.

Adopting the above notation, the window dynamics of the server can be expressed as follows:

$$W_s(t) = \begin{cases} w_o r^{\frac{t}{R_1}}, & W_s(t) < W_{ss} \\ W_{ss} + \dfrac{t - t_{ss}}{bR_1}, & W_{ss} \le W_s(t) < \dfrac{R_1}{\mu_1} \\ \min\left\{ W_{\max}, \dfrac{R_1}{\mu_1} + \displaystyle\int_{t_f}^{t} \frac{1}{W_s(t) b\mu_1} \mathrm{d}t \right\}, & \text{otherwise} \end{cases} \tag{10}$$

where t_{ss} and t_f are such that $W_s(t_{ss}) = W_{ss}, W_s(t_f) = (R_1/\mu_1)$. The last equation can be shown to be

$$\min\left\{ W_{\max}, \sqrt{\frac{2t}{b\mu_1} + C} \right\}$$

where

$$C = \left(\frac{R_1}{\mu_1}\right)^2 - \left(\frac{2t_f}{b\mu_1}\right)$$

The sending rate of the server is therefore

$$s(t) = \min\left\{ \frac{W_s(t)}{R_1}, \frac{1}{\mu_1} \right\}$$

$$= \begin{cases} \dfrac{w_o}{R_1} r^{\frac{t}{R_1}}, & t \le t_{ss} \\ \dfrac{W_{ss}}{R_1} + \dfrac{t - t_{ss}}{bR_1^2}, & t_{ss} < t \le t_f \\ \dfrac{1}{\mu_1}, & t_f < t \end{cases} \tag{11}$$

The number of packets sent by the server up to time t is

$$N_s(t) = \begin{cases} (W_s(t + R_1) - w_o)b, & t < t_{ss} \\ N_{ss} + \displaystyle\int_{t_{ss}}^{t} \frac{W_s(t)}{R_1} \mathrm{d}t, & t_{ss} \le t < t_f \\ N_f + \dfrac{t - t_f}{\mu_1}, & t_f < t \end{cases}$$

$$= \begin{cases} \dfrac{w_o}{r - 1}\left(r^{\frac{t + R_1}{R_1}} - 1 \right), & t < t_{ss} \\ N_{ss} + \dfrac{W_{ss}(t - t_{ss})}{R_1} + \dfrac{(t - t_{ss})^2}{2bR_1^2}, & t_{ss} \le t < t_f \\ N_f + \dfrac{t - t_f}{\mu_1}, & t_f < t \end{cases} \tag{12}$$

where $N_{ss} = N_s(t_{ss})$, $N_f = N_s(t_f)$. The proxy's transmission depends on the amount of packets received from the server in addition to its congestion window size, and thus

$$p(t) = \min\left\{ s(t - I_1 - \mu_1), \frac{W_p(t)}{R_2}, \frac{1}{\mu_p} \right\} \tag{13}$$

that is, the proxy sending rate is governed by the receiving rate from the server, its congestion window size and its transmission capability. We want to find t^*, the time at which the proxy–client connection is no longer constrained by the server–proxy connection, i.e. t^* is the smallest of all t such that

$$p(t) = \min\left\{\frac{W_p(t)}{R_2}, \frac{1}{\mu_p}\right\}$$

For any $t \le t^*, p(t) = s(t - I_1 - \mu_1)$, $N_p(t) = N_s(t - I_1 - \mu_1)$. For a file of size M, if t^* exists and $N_p(t^*) \le M$, then the server–proxy connection catches up with the proxy–client connection sometime before the transfer is completed ($t^* = 0$ corresponds to case (1) discussed earlier). The time it takes to transfer the remaining $M - N\mathrm{p}(t*)$ packets, denoted by $T_e(M - N_p(t^*))$ can be derived using methods described in Section 3.1 for the end-to-end connection, with initial window size $W_p(t^*)$ and end-to-end round-trip delay R_2. The total latency of the file transfer is then $T_p(M) = t^* + T_e(M - N_p(t^*))$.

If $N_p(t^*) > M$ or t^* does not exist, then the proxy–client connection is constrained by the server–proxy connection throughout the entire file transfer period. In this case, the total latency in transferring the file can be approximated by $T_p(M) = t' + I_2$, where t' is such that $N_p(t') = M$.

What remain to be calculated are t^* and $W_p(t^*)$. For $t \le t^*$,

$$W_p(t) = \begin{cases} w_o' + \dfrac{N_p(t - R_2)}{b}, & W_p(t) < W_{ss}' \\ W_{ss}' + \displaystyle\int_{t_{ss}'}^{t} \frac{p(t - I_2 - \mu_p)}{bW_p(t)}\mathrm{d}t, & W_{ss}' \le W_p(t) < \dfrac{R_2}{\mu_p} \\ \dfrac{R_2}{\mu_p} + \displaystyle\int_{t_{f'}}^{t} \frac{1}{b\mu_p W_p(t)}\mathrm{d}t, & \dfrac{R_2}{\mu_p} < W_p(t) \le W_{\max} \end{cases}$$

$$= \begin{cases} w_o' + \dfrac{N_s(t - R_2 - I_1 - \mu_1)}{b}, & W_p(t) < W_{ss}' \\ \left(\dfrac{2}{b}N_s(t - I_1 - I_2 - \mu_1 - \mu_p) + C_1\right)^{1/2}, & W_{ss}' \le W_p(t) < \dfrac{R_2}{\mu_p} \\ \left(\dfrac{2t}{b\mu_p} + C_2\right)^{1/2}, & \dfrac{R_2}{\mu_p} < W_p(t) \end{cases} \tag{14}$$

where $W_p(t_{ss}') = W_{ss}'$, $W_p(t_f') = R_2/\mu_p$, $C_1 = W_{ss}'2 - (2/b)N_s(t_{ss}' - I_1 - \mu_p)$,

$$C_2 = \left(\frac{R_2}{\mu_p}\right)^2 - \left(\frac{2t_f'}{b\mu_p}\right)$$

Combining Eqs. (13) and (14), we can compute t^*. Fig. 4 compares the results of the above model with simulation. In the first case the proxy–client connection has a larger initial window size and the server–proxy connection catches up at roughly $t^* = 0.54$ s. In the second case, the proxy–client connection is not constrained. We see that since this is a continuous fluid-based model, there is discrepancy between the prediction and the actual evolution of the number of transmitted packets (which is discrete). But overall it provides quite accurate estimates.

We now show how the asymmetry in the RTT of the two links affects the proxy performance by using the above model. Suppose $w_o \le w_o'$ and $R_1 > R_2$, i.e. the proxy–client connection is constrained by the server–proxy connection throughout the entire file transfer. Further assume that the file transfer is only limited to the slow-start phase. Using Eq. (12),

$$N_p(t') = N_s(t' - I_1 - \mu_1) = \frac{w_o}{r - 1}\left(r^{\frac{t' + R_1 - I_1 - \mu_1}{R_1}} - 1\right) = M \tag{15}$$

Therefore

$$t' = R_1 \log_r\left(\frac{M(r-1)}{w_o} + 1\right) - R_1 + I_1 + \mu_1,$$

and

$$T_p(M) = t' + I_2 = R_1 \log_r\left(\frac{M(r-1)}{w_o} + 1\right) - I_1 + I_2$$

$$\approx R_1 \log_r\left(\frac{M(r-1)}{w_o} + 1\right)$$

with an error within half of R_1.

Suppose we now let $R_1 < R_2$, but keep $R_1 + R_2$ unchanged, and let $w_o > w_o'$, then using the analysis in Section 3.2 we get

$$T_p(M) = R_2 \log_r\left(\frac{M(r-1)}{w_o'} + 1\right)$$

while the latency of end-to-end connection remains the same

$$T_e(M) = (R_1 + R_2)\log_r\left(\frac{M(r-1)}{w_o} + 1\right)$$

We see that when using the proxy, the longer connection of the two ($\max\{R_1, R_2\}$) determines the total latency. As the difference between the two RTTs increases, the gain from using the proxy reduces. In this scenario the performance of the proxy is maximized when the two connections are 'similar', i.e. $R_1 \approx R_2, w_o \approx w_o'$. Similar argument can also be used for file transfers that enter congestion avoidance phase. This is an interesting observation considering the fact that many such proxies are used or proposed to be used in a heterogeneous environment where links have very different properties. This indicates that while it is very important to optimize each link separately, it is equally important to minimize the asymmetry between links since if separate optimization only increases the difference, e.g. making the fast link even faster, the resulting performance might not be improved.

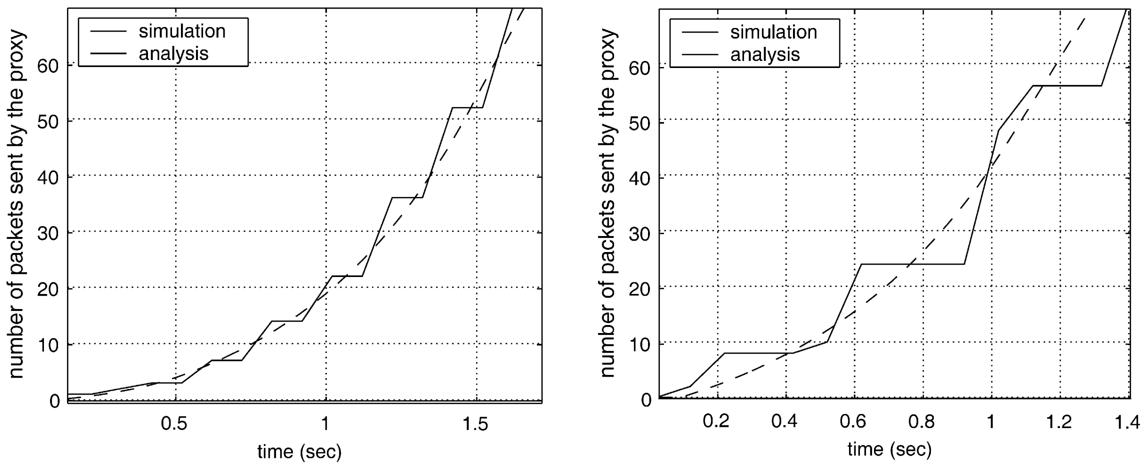

Fig. 4. Graph shows latency vs. file sizes using the proxy. $I_1 = 100$ ms, $I_2 = 200$ ms. One on the left: $w_o = 1, w'_o = 8$ so the proxy–client connection is constrained by the server–proxy connection; the one on the right: $w_o = 8, w'_o = 8$ so the proxy–client connection is never constrained.

4. Delay analysis of a connection over lossy links

When losses (either due to congestion or link failure) are present the analysis becomes more complicated. Moreover the analysis is largely limited to the steady-state study of TCP connections which is applicable in the case of an unlimited file transfer, less accurate in the case of a finite TCP connection, and much less in the case of a proxy as shown in this section.

4.1. Server–proxy link lossless

If we assume that the server–proxy link is lossless, then methods introduced in Refs. [7–9] can be applied to determine the throughput and delay of the proxy–client connection. In particular, the TCP bulk data transfer throughput is shown to be well approximated by

$$\lambda(\mathrm{RTT}, p) = \sqrt{\frac{3}{2bp}}\frac{1}{\mathrm{RTT}}$$

where p is the probability of loss for a single packet at low loss rate [9], and is more accurately approximated in Ref. [8] by considering timeouts. These results were developed for bulk TCP transfers and were based only on analysis of the TCP congestion avoidance phase. In Ref. [7], it was shown that they can be equally effective when applied to short TCP connections if combined with slow-start phase analysis.

The performance implication of using a proxy when losses are present immediately follows: these results show that the throughput of a TCP transfer is inversely proportional to the connection RTT and the square root of the loss rate. If losses are concentrated on the proxy–client link, then using the proxy effectively isolates the part of the connection that involves loss, and reduces the RTT required to recover the losses, thus achieving higher throughput and lower latency. The same key concept can be seen in schemes such as Snoop TCP [12,13], WTCP [14,15], and [16,17] that use local retransmission (some at the link layer, some at the transport layer).

Specifically, denoting the loss rate on the proxy–client link by p_2, and the throughput by $\lambda(\mathrm{RTT}, p_2)$, the transfer latency of a file of size M using end-to-end connection is given by

$$T_e = \sum_{n=0}^{M} p(n)\left(T_e(n) + \frac{M-n}{\lambda(\mathrm{RTT}, p_2)}\right)$$
$$= \sum_{n=1}^{M} p(n)T_e(n) + \frac{M - m_{\mathrm{loss}}}{\lambda(2(I_1+I_2), p_2)} \qquad (16)$$

where $p(n) = (1-p)^n p$ for $n < M$ and $p(n) = (1-p)^M$ for $n = M$ are the probability that n packets are successfully sent before the first loss occurs.

$$m_{\mathrm{loss}} = \frac{(1-(1-p_2)^M)(1-p_2)}{p_2} + 1$$

is the expected number of packets sent before the first loss occurs. $T_e(\cdot)$ is the latency function of an end-to-end connection shown in Section 3.1.

When using the proxy, assuming that the proxy–client connection is not constrained by the server–proxy connection (e.g. $I_1 < I_2, \mu_1 < \mu_2$),

$$T_p = \sum_{n=1}^{M} p(n)T_p(n) + \frac{M - m_{\mathrm{loss}}}{\lambda(2I_2, p_2)}. \qquad (17)$$

The difference of the two (using results from Section 3) is

$$T_e - T_p = \sum_{n=1}^{M} p(n)(T_e(n) - T_p(n)) + \frac{(M - m_{\text{loss}})\sqrt{2bp_2}}{\sqrt{3}}(2I_1)$$
$$\approx \left(\sum_{n=1}^{M} p(n)(k_n - 1) + \frac{(M - m_{\text{loss}})\sqrt{2bp_2}}{\sqrt{3}}\right)(2I_1)$$

where k_n is the number of windows needed to cover a file of n segments. The approximation in the last equation is based on the assumption that the link capacity is not filled during the transfer of n packets. For a given file size and loss rate on the proxy–client link, the first term of the above equation is a constant, and the amount of gain in using the proxy depends on the RTT of the server–proxy connection.

When the proxy–client connection is constrained by the server–proxy connection, the effective throughput of the second connection becomes $\min\{\lambda(R_2, p_2), s(t)\}$. If the loss is sufficiently low and the server–proxy link has a much larger round-trip than the proxy–client link, then again the delay could be dominated by the server–proxy connection.

Fig. 5 compares the latency obtained using this analysis with the result from simulation, averaged over 50 independent runs. There is an obvious discrepancy between the two curves. This is mainly due to the fact that the delay model assumes that the connection goes into steady state right after the first loss. The two curves eventually approach each other as the file size increases (number of packets sent in this figure). This is because for a large file transfer the effect of the above assumption is diluted (since the effect of steady state will dominate).

4.2. Random losses on both links

When losses are present on both links, using the previous analysis for both connections provides only gross approximation. Suppose the loss rates on the server–proxy link and the proxy–client link are p_1 and p_2, respectively. Assuming losses are independent, the overall loss rate experienced by an end-to-end connection is $p = p_1 + p_2 - p_1 p_2$. For an infinite file transfer, in the long run the server–proxy connection has an average throughput

$$\lambda_1 = \sqrt{\frac{3}{2bp_1}}\frac{1}{R_1}$$

and the proxy–client connection has an average throughput

$$\lambda_2 = \sqrt{\frac{3}{2bp_2}}\frac{1}{R_2}$$

if unconstrained by the server–proxy connection. The slower one of the two is going to dominate the combined throughput and delay. However, both values are greater than the throughput of the end-to-end connection

$$\lambda = \sqrt{\frac{3}{2bp}}\frac{1}{R_1 + R_2}$$

since it has a larger loss rate and a larger RTT. Therefore by segregating the server–client connection into parts that each has a smaller loss rate and RTT, using the proxy achieves higher throughput and thus lower latency.

For a finite file transfer, the latency of the end-to-end connection is given by Eq. (16) with loss rate p. When using the proxy, we consider m_1, m_2, the expected number of packets sent successfully before the first packet loss occurs to the two connections, respectively. The two connections have bulk transfer throughput $\lambda_1 = \lambda(2(I_1 + I_2) + \mu_p, p_1), \lambda_2 = \lambda(2I_2, p_2)$, respectively. Since we have complete characterization $(s(t), W_s(t), N_s(t))$ of the server for the first m_1 packets from Section 3.2.2, and using λ_1 as an approximation to the remaining of the transfer, the characterization of the proxy can be obtained by listing all possible cases comparing m_1 and m_2, $T_e(m_1)$ and $T_p(m_1)$, λ_1 and λ_2. Fig. 6 compares the results obtained using this approach to that of simulation. It can be seen that the two do not match for a wide range of file sizes. Indeed the discrepancy seems to diverge. Our conclusion is that the bulk TCP (infinite source) throughput/latency model is insufficient to accurately predict the latency of a short file (finite source) transfer using a proxy with losses on both links, due to the complicated dynamics caused by random losses and the coupling between two connections. In Section 4.3, we investigate an alternative model.

4.3. An alternative latency model for short file transfers

The reason that the previous analysis is not accurate enough is because the throughput model for bulk TCP transfer only gives the steady-state throughput. The latency model built on this throughput model considers the time

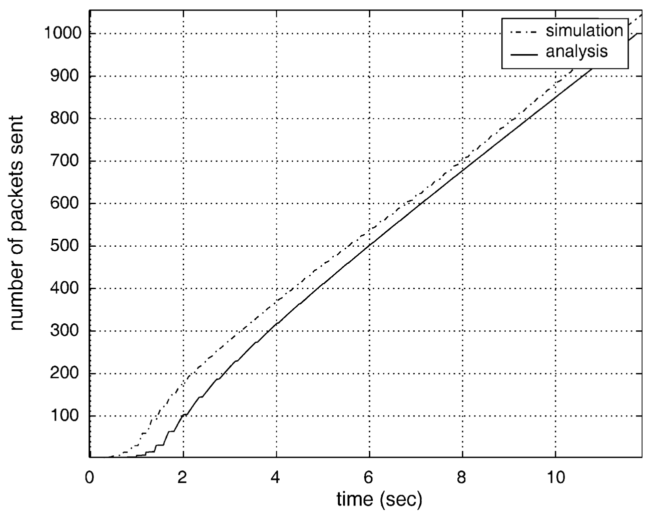

Fig. 5. Latency when splitting is used, where the first link lossless, $I_1 = 50$ ms, $I_2 = 100$ ms, $p_2 = 0.005$.

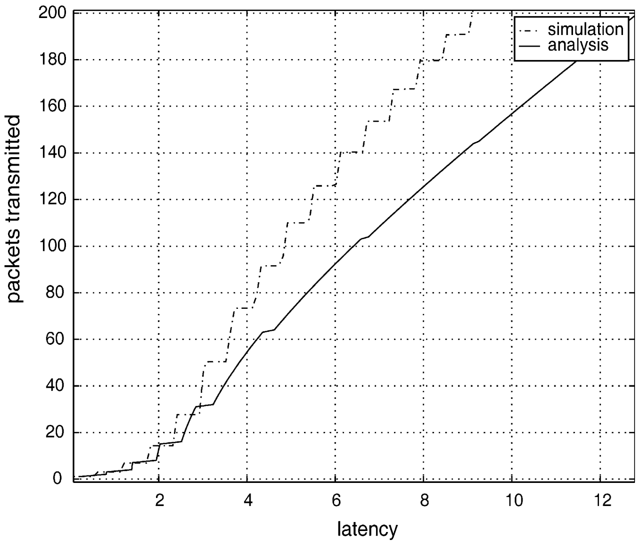

Fig. 6. Latency when splitting is used with losses on both links, $l_1 = 100$ ms, $l_2 = 200$ ms, $p_1 = 0.01$, $p_2 = 0.005$.

spent in the slow-start phase, but assumes that the connection reaches steady state right after the initial slow-start is ended (when the congestion control phase is first entered). This is in general not true. For example, Fig. 7 illustrates the window size evolution of TCP (averaged over 50 independent connections) for the same scenario shown in Fig. 6. We see that after the initial slow-start ended (i.e. after the first loss) there is a certain 'transient period' before the connection reaches the steady state (when the window size stabilizes). In this case, it takes a relatively long time for TCP to achieve this steady state after the first loss. Therefore assuming that the steady state is immediately reached can result in significant error in estimating the connection throughput and latency when the connection is short.

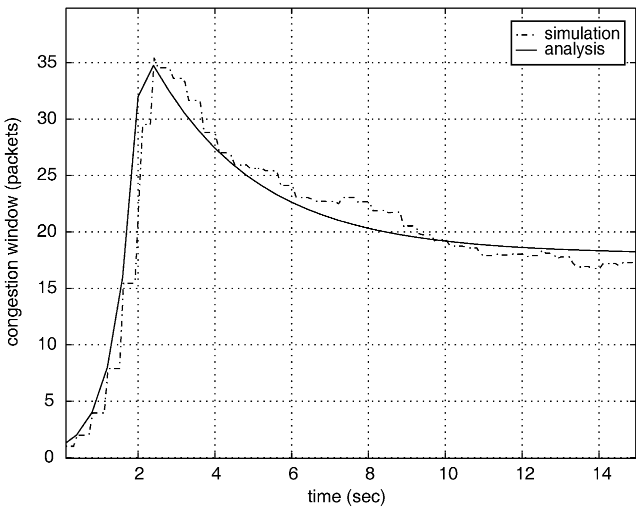

Fig. 7. Congestion window size for the case where $l_1 = 100$ ms, $l_2 = 200$ ms, $p_1 = 0.01$, $p_2 = 0.005$.

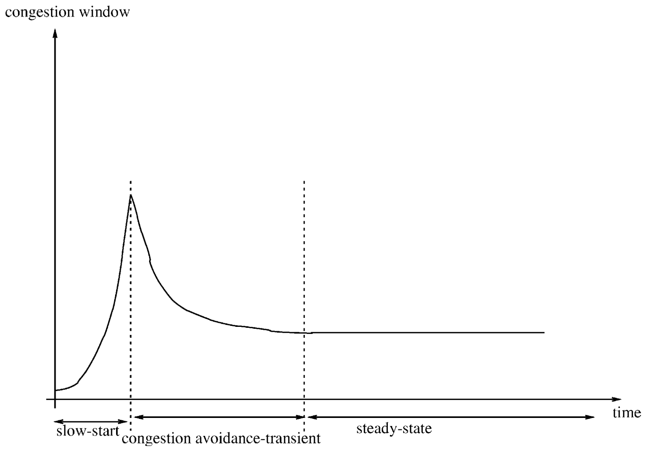

Fig. 8. Summary of the congestion window evolution of TCP.

In Ref. [18], we developed the following TCP window evolution model to take into account this transient period and it results in very accurate estimate of short file transfer latency over an end-to-end connection, also shown in Fig. 7. We divide the TCP window evolution into three distinct regions: the slow-start phase where the window grows exponentially, the congestion control transient phase (which approximately decays exponentially), and the steady state, as illustrated in Fig. 8.

The detailed derivation of this model can be found in Ref. [18]. In Ref. [18] this model has been shown to give very accurate latency estimate for an end-to-end connection under a wide range of file sizes. However, this alternative model again fails to provide satisfying quantitative estimate when a proxy is used. This is mainly due to the complex coupling relationship between the two split connections. Nevertheless, in Section 5 we will use this alternative model as shown in Fig. 8 to show qualitatively some interesting effects connection asymmetry has on proxy performance.

5. Analysis and discussions

In this section we use the models developed in previous sections to further examine a number of factors that affect the performance comparison of an end-to-end connection and a split connection. Specifically we will first use the lossless latency model to analyze the implication of the initial window size, a slow or congested proxy, and file sizes. This is not a completely realistic scenario, nevertheless insights are obtained via certain simplification especially for short file transfers. We will then use the alternative model presented in Section 4.3 to study a few special cases where the server–proxy connection and the proxy–client connection are highly asymmetric in terms of their steady-state sending rate.

5.1. Initial window size

From the definition of S and M_x,

$$S = \left\lceil \log_r\left(\frac{W_{\rm ss}}{w_{\rm o}}\right)\right\rceil,\ M_x = b(W_{\max} - W_{\rm ss}) + S$$

Consider a file that finishes transferring within the slow-start phase, the total number of windows needed to cover the file, K, would be such that time $K \le S$. Since

$$M \le \sum_{i=1}^{K} w_{\rm o} r^{i-1} = w_{\rm o}\frac{r^{K-1}}{r-1}$$

this means

$$M \le \frac{W_{\rm ss} - w_{\rm o}}{(r-1)}$$

K is the smallest integer that satisfies

$$M \le w_{\rm o}\frac{r^K - 1}{r-1}$$

therefore,

$$K = \left\lceil \log_r\left(\frac{M}{w_{\rm o}}(r-1)+1\right)\right\rceil \approx \log_r\left(\frac{M}{w_{\rm o}}(r-1)+1\right)+1$$

Assuming $\mu_1 W_{\rm ss} \le R_{\rm e}$, i.e. the link (or pipe) capacity is not filled during slow-start, and $\mu_1 < \mu_{\rm p}$ that the proxy is slower than the server, we have

$$T_{\rm e} = M\mu_{\rm p} + (K-1)R_{\rm e} - \sum_{k=1}^{K-1} w_{\rm o} r^{k-1}\mu_{\rm p} + I_1 + I_2 + \mu_1 \approx R_{\rm e}\log_r\left(\frac{M}{w_{\rm o}}(r-1)+1\right) + C_1 \quad (18)$$

where $C_1 = I_1 + I_2 + \mu_1$. This last equation is the same as presented in Ref. [7], but derived in a different way.

Similarly, for an initial window size $w'_{\rm o}$ used by the proxy, assuming $\mu_{\rm p} W_{\rm ss} \le R_2$, we have

$$K' = \left\lceil \log_r\left(\frac{M}{w'_{\rm o}}(r-1)+1\right)\right\rceil \approx \log_r\left(\frac{M}{w'_{\rm o}}(r-1)+1\right)+1$$

and

$$T_{\rm p} \approx R_2\log_r\left(\frac{M}{w'_{\rm o}}(r-1)+1\right) + C_2 \quad (19)$$

where $C_2 = I_1 + I_2 + \mu_1 + t_{\rm p}$. Note that $C_1 \approx C_2$ and both are close to one half of $R_{\rm e}$. Since $R_{\rm e} > R_2, w_{\rm o}$ has to be greater than $w'_{\rm o}$ in order to achieve the same delay. More specifically,

$$\left(\frac{M}{w_{\rm o}}(r-1)+1\right)^{R_{\rm e}} \approx \left(\frac{M}{w_{\rm o}}(r-1)+1\right)^{R_2}$$

leads to

$$w_{\rm o} = \frac{M(r-1)}{\left(\frac{M}{w'_{\rm o}}(r-1)+1\right)^{R_2/R_{\rm e}} - 1}$$

Table 1
Initial window size of the end-to-end connection

File size (kbytes)	10	20	25	30	35
$w_{\rm o}$	5	7	8	9	9

Table 1 shows some values of M and $w_{\rm o}$ based on this approximation for $r = 2$, $w'_{\rm o} = 1$, $R_2/R_{\rm e} = 0.5$ and $W_{\rm ss} = 128$ segments. We see that in order to achieve similar latency even for relatively small files we need significantly larger initial window size for the end-to-end connection.

5.2. Slow or congested proxy

When the same initial window size $w_{\rm o}$ is used, the difference in delay between the two is

$$T_{\rm e} - T_{\rm p} = (R_{\rm e} - R_2)\log_r\left(\frac{M}{w_{\rm o}}(r-1)-1\right) = (2I_1 + \mu_1)\log_r\left(\frac{M}{w_{\rm o}}(r-1)-1\right) \quad (20)$$

This difference increase as M and I_1 increase, but seems invariant to changes in $\mu_{\rm p}$. As $\mu_{\rm p}$ increases, which corresponds to a slower proxy, the difference in delay remains constant so long as $\mu_{\rm p} w_{\rm o} r^{k-1} \le R_2$ for any $k \le K$. However, as $\mu_{\rm p}$ keeps increasing to the point where the pipe is filled before the file transfer completes, the difference quickly reduces. In particular, if this is achieved during the $k_{\rm e}$th window for the end-to-end connection, i.e. $\mu_{\rm p} w_{\rm o} r^{k_{\rm e}-1} \ge R_{\rm e}$, then

$$k_{\rm e} = \left\lceil \log_r\left(\frac{R_{\rm e}}{\mu_{\rm p} w_{\rm o}}\right) + 1\right\rceil$$

and

$$T_{\rm e} \approx M\mu_{\rm p} + R_{\rm e}\log_r\left(\frac{R_{\rm e}}{\mu_{\rm p} w_{\rm o}}\right) - \frac{R_{\rm e} - \mu_{\rm p} w_{\rm o}}{r-1} + I_1 + I_2 + \mu_1 \quad (21)$$

We can get a similar expression for $T_{\rm p}$, and thus

$$T_{\rm e} - T_{\rm p} \approx R_{\rm e}\log_r\left(\frac{R_{\rm e}}{\mu_{\rm p} w_{\rm o}}\right) - R_2\log_r\left(\frac{R_2}{\mu_{\rm p} w'_{\rm o}}\right) + \frac{\mu_{\rm p}(w_{\rm o} - w'_{\rm o}) - 2I_1 - \mu_1}{r-1} \quad (22)$$

This expression decreases as $\mu_{\rm p}$ increases. This result can be clearly observed in Fig. 9. A slower proxy (increased $\mu_{\rm p}$ and decreased transmission rate) can be viewed as an approximation to a busier or more congested proxy, because under such situation each TCP connection only gets a fraction of the total proxy capacity (assuming the proxy has sufficient buffer), and queuing is increased. This result shows that as the proxy becomes busy, the gain from using separate connections reduces because the bottleneck dominates the overall performance no matter which scheme we use.

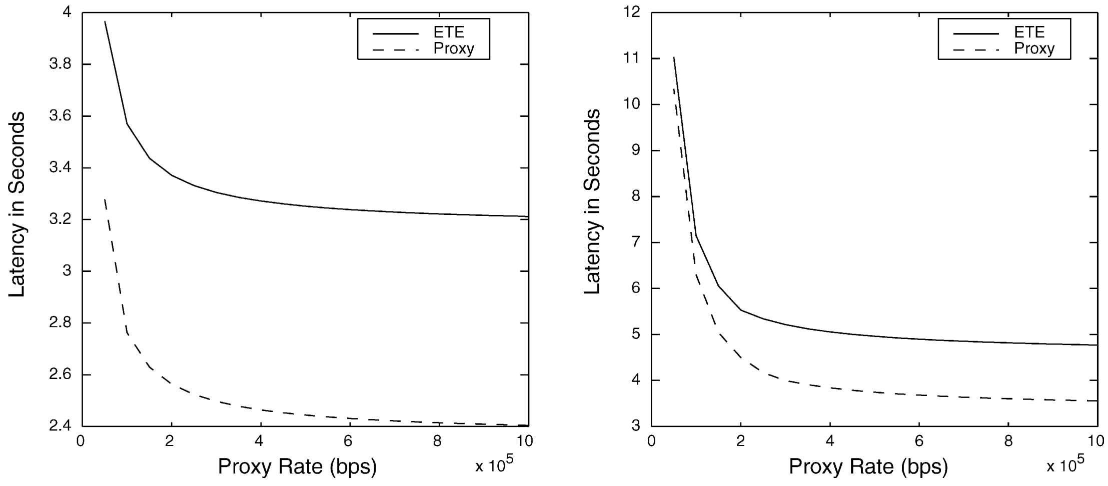

Fig. 9. Latency vs. C_p, the transmission rate of the proxy. File size is 11 and 51 kbytes, for the graph on the left and right, respectively. $C_1 = C_2 = 1$ Mbps in both cases. These graphs are derived from our model.

In a system where a proxy is placed at the aggregation point of incoming traffic, adequate provisioning of such a proxy becomes very important since otherwise very little is gained from using a proxy.

5.3. File size

In the case where the file transfer enters the congestion avoidance stage, i.e.

$$M > \frac{W_{ss} - w_o}{r - 1}$$

similar analysis apply and we have

$$T_e = M\mu_p + R_e(k_e - 1) - \sum_{k=1}^{k_e-1} t_k(\mu_p) + I_1 + I_2 + \mu_1 \quad (23)$$

$$T_p = M\mu_p + R_2(k'_e - 1) - \sum_{k=1}^{k_e-1} t_k(\mu_p) + I_1 + I_2 + \mu_1 + t_p \quad (24)$$

where k_e and k'_e are the total number of windows sent before the pipe becomes full for the end-to-end and the split connection, respectively. In case when the file finishes transfer before the pipe is full, $k_e = k'_e = K$, and the difference between the two is mainly $(R_e - R_2)(K - 1)$, which increases as M increases (K increases with M). However, if M is large enough and the pipe is filled up before the transfer completes, then the difference between the two stays constant, and both increase with rate μ_p as the file size increases. This can be observed in Fig. 10.

5.4. Connection with asymmetric segments

In Section 3.2.2 we briefly compared the performance of an end-to-end connection and a split connection when the two segments of the connection are highly asymmetric in terms of their respective RTTs when losses are ignored. Here we use the alternative model presented in Section 4.3

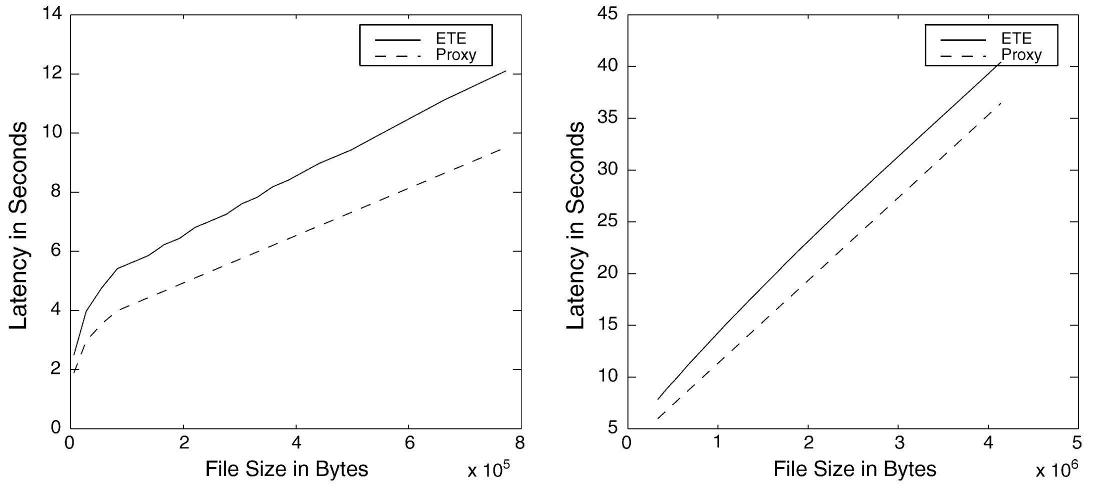

Fig. 10. Latency vs. file size. For small files the latency of an end-to-end connection increases faster than that of split connections. However as file size grow big enough to fill up the capacity, the two have same growth rate and the difference stays constant. These graphs are derived from our model.

to re-examine the effect of connection asymmetry when losses are taken into account.

Using the model in Fig. 8 and the fact that the sending rate is the ratio between the congestion window size and RTT, we have the characterization of the sending rate of both segments/links that looks like Fig. 8. We will call this the *unlimited sending rate* of a segment. To simplify our discussion, we will define a segment to *dominate* another segment, if the unlimited sending rate of the first segment is less than or equal to the unlimited sending rate of the second segment at all times. Intuitively one may expect the dominant link to determine the overall performance, i.e. the overall latency of the connection is primarily determined by the dominant link.

(1) Case I: server–proxy link dominant, $R_2 \ll R_1$, $p_1 \approx p_2$: The comparison between the server window evolution of an end-to-end connection and a split connection is summarized in Fig. 11. Fig. 11 can be explained as the following. When splitting is used the server merely sees the loss in the server–proxy link q_1, but when the connection is end-to-end, the loss seen by the server is equal to $p = p_1 + p_2 - p_1p_2$, which is larger than the loss seen in the splitting case. Thus, the average window size where the congestion avoidance starts is smaller than in the end-to-end case. Also because of the larger loss probability the steady-sate window size

$$\left(\approx \sqrt{\frac{3}{2bp}}\right)$$

is smaller in the end-to-end case, thus justifying Fig. 11. The comparison between the server sending rate in two cases are approximately the same as shown in Fig. 11 since the RTTs seen by the server in both cases are approximately R_1.

(2) Case II: server–proxy link dominant, $R_1 \ll R_1 + R_2, p_2 \ll p_1$: This is a case where the delay on the proxy–client link is significant, but due to much higher error

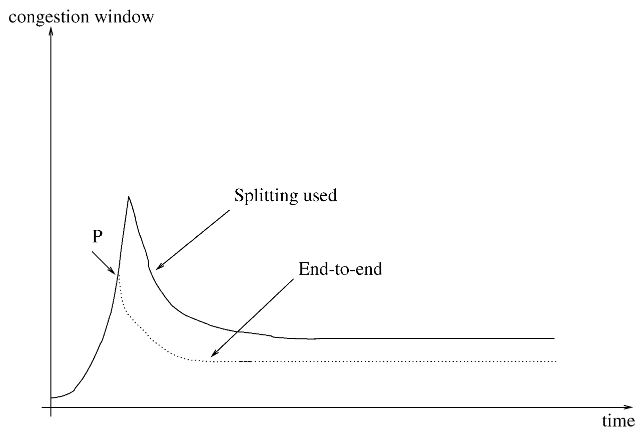

Fig. 11. Comparison between server window evolution of end-to-end and splitting cases (Case I).

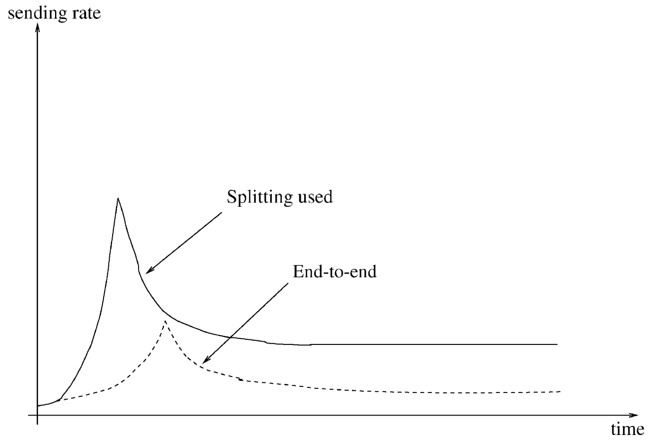

Fig. 12. Comparison between server sending rate of end-to-end and splitting cases (Case II).

probability on the server–proxy link, the latter dominates. The comparison between the server sending rate in two cases are qualitatively shown in Fig. 12. It should be obvious that the average server window size evolution is the same in both cases, since the server perceived error is approximately the case in both cases ($\approx p_1$). Thus the server sending rate when splitting is used is higher, since the server perceived RTT is much smaller in the splitting case (R_1 vs. $R_1 + R_2$). Therefore the ratio between the steady-state throughput in the end-to-end case and the splitting case is approximately

$$\frac{R_1 + R_2}{R_1}$$

There are some interesting observations. We see that in Case I, where the second link is very short and lossy, splitting does not give any gain for small file transfers (until point P in Fig. 11). As the file-size increases, splitting becomes more effective in decreasing the latency. In Case II, where the second link is much less lossy, we have splitting gain even for very small files and the difference between the end-to-end and splitting cases increases as the file-size increases.

Case III: proxy–client link dominant, $R_1 \ll R_2$, $p_1 \approx p_2$: When the proxy–client is dominant, in the end-to-end case the sending rate of the proxy is essentially a time shifted version of the server sending rate. Therefore, the comparison between the two cases reduces to the comparison between the proxy sending rate in the two cases. Since the proxy sees a much lower error probability in the splitting case, under the condition that $R_1 \ll R_2$, $p_1 \approx p_2$, the comparison is very similar to that shown in Fig. 11.

Case IV: proxy–client link dominant, $R_1 \ll R_2, p_1 \approx p_2$: this is a case where the proxy–client link is much worse than the server–proxy link. Since $p = p_1 + p_2 - p_1p_2 \approx p_2$ and $R_1 + R_2 \approx R_2$, the server perceived error probability and RTT in the end-to-end case are approximately the same as that perceived by the proxy in the splitting case. This results in similar proxy sending rate (a shifted version of the server sending rate) in the end-to-end case as in the splitting case. Therefore in this case, the two result in very similar performance. In other words, using connection splitting provides very little performance gain.

To summarize, the overall observation in the lossy scenario remains the same as in the lossless scenario. That is, when the performance in terms of file transfer latency over the two links is significantly different, connection splitting provides limited gain. This suggests that the placement of the proxy should be such that the two segments result in as similar performance as possible.

6. Conclusion

In this study we examined using proxy as a way of improving TCP performance in various situations. Such proxies typically break an end-to-end connection into two segments, such as a spoofing proxy. We developed models to investigate the TCP dynamics when proxies are used and compared its performance with end-to-end TCP connection. We summarize our observations and conclusions as follows.

In general using proxy (or separate TCP connections) results in lower latency from our analysis. For an end-to-end connection this can be compensated by increasing the initial window size. However, as we show in Table 1, it requires significantly larger initial window size even for reasonably small file sizes, which makes it less practical in real applications. When the proxy becomes the bottleneck, the gain from using the proxy quickly diminishes. In systems where such a proxy is positioned at a place that all connections have to go through, e.g. in a satellite system the proxy is co-located with the satellite gateway so that all connections go through the proxy, the performance gain from the proxy can be limited especially during busy hours. This may also cause buffer overflow at the proxy, or cause the proxy to advertise smaller receive window size, which we did not consider explicitly in this paper. When this is the case, queueing becomes severe and packets quickly build up at the proxy, especially if in addition I_1 is much smaller than I_2. It is therefore important to properly provision such systems and implement some form of dynamic flow control at the proxy. This may be prevented by using a much larger initial window size over the proxy–client link.

A proxy achieves the effect of localizing error/loss recovery and in general improves the throughput and reduces latency of a connection when losses are present. One of the common situations where proxies are used is a heterogeneous environment where parts segregated by the proxy have very different link characteristics, e.g. propagation delay, loss rate, etc. Interestingly, the performance gain in using a proxy is maximized when both parts have similar properties. Since the slower part always dominates the overall performance, as the level of asymmetry increases, the performance gap between using a proxy and using an end-to-end connection becomes smaller. This implies that while it is important to separately optimize these heterogeneous parts of the connection, it is also important that such optimization reduces the asymmetry between them.

It is worth pointing out that for a cache proxy when there is a hit the resulting connection is simply between the client and the cache proxy, so all our analysis on end-to-end connections applies with a smaller RTT. This essentially cuts down the size of the connection needed if not using a proxy and achieves better throughput and latency. Throughout our analysis we assumed that the time it takes to transmit an ACK is negligible and that the return channel is the same as the forward channel (ACK path same as the data path). As pointed out in Ref. [19], asymmetry increases TCP's sensitivity to random packet losses, and adjustments to our models are needed.

References

[1] V.G. Bharadwaj, Improving TCP performance over high-bandwidth geostationary satellite links, Tech. Rep. MS 99-12, Institute for Systems Research, University of Maryland, College Park, 1999, http://www.isr.umd.edu/TechReports/ISR/1999/.

[2] A. Bakre, B.R. Badrinath, I-TCP: indirect TCP for mobile hosts, Proceedings of the IEEE ICDCS, 1995, pp. 136–143.

[3] A.V. Bakre, B.R. Badrinath, Implementation and performance evaluation of indirect TCP, IEEE Trans. Comput. 46 (3) (1997) 260–278.

[4] K. Brown, S. Singh, A network architecture for mobile computing, IEEE INFOCOM (1996) 1388–1396.

[5] S. Sibal, P. Rodriguez, O. Spatscheck, TPOT: translucent proxying of TCP, Tech. Rep., AT & T labs—Research and EURECOM Technical Report, 2000.

[6] M. Karir, IPSEC and the Internet, Tech. Rep. MS 99-14, Institute for Systems Research, University of Maryland, College Park, 1999, http://www.isr.umd.edu/TechReports/ISR/1999/.

[7] N. Cardwell, S. Savage, T. Anderson, Modeling TCP latency, IEEE INFOCOM (2000).

[8] J. Padhye, V. Firoiu, D.F. Towsley, J.F. Kurose, Modeling TCP reno performance: a simple model and its empirical validation, IEEE Trans. Netw. 8 (2) (2000) 133–145.

[9] T.V. Lakshman, U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE Trans. Netw. 5 (3) (1997) 336–350.

[10] J. Kurose, K. Rose, Computer networking, a top-down approach featuring the Internet.

[11] M. Allman, V. Paxson, On estimating end-to-end network path properties, SIGCOMM (1999).

[12] H. Balakrishnan, S. Seshan, E. Amir, R.H. Katz, Improving TCP/IP performance over wireless networks, ACM/IEEE Int. Conf. Mobile Comput. Netw. (MobiCom'95) 2 (11) (1995).

[13] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, R.H. Katz, A comparison of mechanisms for improving TCP performance over wireless links, IEEE/ACM Trans. Netw. 5 (6) (1997) 756–769.

[14] K. Ratnam, I. Matta, WTCP: an efficient mechanism for improving TCP performance over wireless links, Proceedings of the IEEE ISCC, 1998, pp. 74–78.
[15] K. Ratnam, I. Matta, Effect of local retransmission at wireless access points on the round trip time estimation of TCP, Proceedings of the 31st Annual Simulation Symposium (1998) 150–156.
[16] C. Parsa, J.J. Garcia-Luna-Aceves, Improving TCP performance over wireless network at the link layer, ACM Mobile Netw. Appl. J. (1999).
[17] C. Parsa, J.J. Garcia-Luna-Aceves, TULIP: a link-level protocol for improving TCP over wireless links, Proceedings of the IEEE WCNC'99, 1999, pp. 1253–1257.
[18] N. Ehsan, M. Liu, Analysis of TCP Transient behavior and its effect on file transfer latency, May 2003, Anchorage, AK.
[19] T.V. Lakshman, U. Madhow, B. Suter, TCP performance with random loss and bidirectional congestion, IEEE Trans. Netw. 8 (5) (2000) 541–555.