

Using this highly idealized model for the steady-state dynamics of TCP, we can also derive an interesting expression that relates a connection’s loss rate to its available bandwidth [Mahdavi 1997]. This derivation is outlined in the homework problems. A more sophisticated model that has been found empirically to agree with measured data is [Padhye 2000].

TCP Over High-Bandwidth Paths

It is important to realize that TCP congestion control has evolved over the years and indeed continues to evolve. For a summary of current TCP variants and discussion of TCP evolution, see [Floyd 2001, RFC 5681, Afanasyev 2010]. What was good for the Internet when the bulk of the TCP connections carried SMTP, FTP, and Telnet traffic is not necessarily good for today’s HTTP-dominated Internet or for a future Internet with services that are still undreamed of.

The need for continued evolution of TCP can be illustrated by considering the high-speed TCP connections that are needed for grid- and cloud-computing applications. For example, consider a TCP connection with 1,500-byte segments and a 100 ms *RTT*, and suppose we want to send data through this connection at 10 Gbps. Following [RFC 3649], we note that using the TCP throughput formula above, in order to achieve a 10 Gbps throughput, the average congestion window size would need to be 83,333 segments. That’s a *lot* of segments, leading us to be rather concerned that one of these 83,333 in-flight segments might be lost. What would happen in the case of a loss? Or, put another way, what fraction of the transmitted segments could be lost that would allow the TCP congestion-control algorithm specified in Figure 3.52 still to achieve the desired 10 Gbps rate? In the homework questions for this chapter, you are led through the derivation of a formula relating the throughput of a TCP connection as a function of the loss rate (L), the round-trip time (RTT), and the maximum segment size (MSS):

$$\text{average throughput of a connection} = \frac{1.22 \cdot MSS}{RTT \sqrt{L}}$$

Using this formula, we can see that in order to achieve a throughput of 10 Gbps, today’s TCP congestion-control algorithm can only tolerate a segment loss probability of $2 \cdot 10^{-10}$ (or equivalently, one loss event for every 5,000,000,000 segments)—a very low rate. This observation has led a number of researchers to investigate new versions of TCP that are specifically designed for such high-speed environments; see [Jin 2004; RFC 3649; Kelly 2003; Ha 2008] for discussions of these efforts.

3.7.1 Fairness

Consider K TCP connections, each with a different end-to-end path, but all passing through a bottleneck link with transmission rate R bps. (By *bottleneck link*, we mean

that for each connection, all the other links along the connection's path are not congested and have abundant transmission capacity as compared with the transmission capacity of the bottleneck link.) Suppose each connection is transferring a large file and there is no UDP traffic passing through the bottleneck link. A congestion-control mechanism is said to be *fair* if the average transmission rate of each connection is approximately R/K ; that is, each connection gets an equal share of the link bandwidth.

Is TCP's AIMD algorithm fair, particularly given that different TCP connections may start at different times and thus may have different window sizes at a given point in time? [Chiu 1989] provides an elegant and intuitive explanation of why TCP congestion control converges to provide an equal share of a bottleneck link's bandwidth among competing TCP connections.

Let's consider the simple case of two TCP connections sharing a single link with transmission rate R , as shown in Figure 3.55. Assume that the two connections have the same MSS and RTT (so that if they have the same congestion window size, then they have the same throughput), that they have a large amount of data to send, and that no other TCP connections or UDP datagrams traverse this shared link. Also, ignore the slow-start phase of TCP and assume the TCP connections are operating in CA mode (AIMD) at all times.

Figure 3.56 plots the throughput realized by the two TCP connections. If TCP is to share the link bandwidth equally between the two connections, then the realized throughput should fall along the 45-degree arrow (equal bandwidth share) emanating from the origin. Ideally, the sum of the two throughputs should equal R . (Certainly, each connection receiving an equal, but zero, share of the link capacity is not a desirable situation!) So the goal should be to have the achieved throughputs fall somewhere near the intersection of the equal bandwidth share line and the full bandwidth utilization line in Figure 3.56.

Suppose that the TCP window sizes are such that at a given point in time, connections 1 and 2 realize throughputs indicated by point A in Figure 3.56. Because the amount of link bandwidth jointly consumed by the two connections is less than

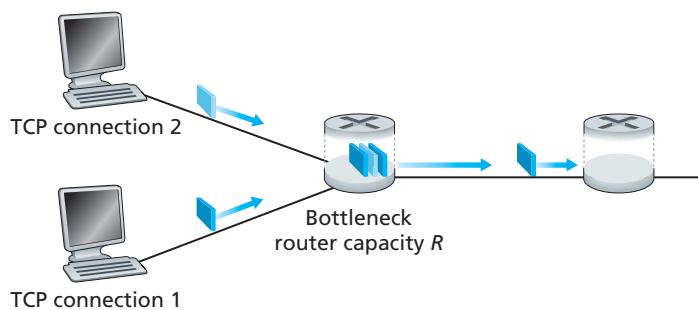


Figure 3.55 ♦ Two TCP connections sharing a single bottleneck link

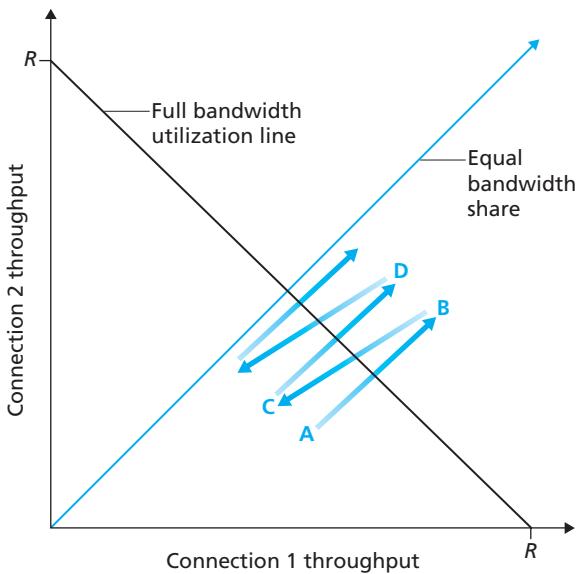


Figure 3.56 ♦ Throughput realized by TCP connections 1 and 2

R , no loss will occur, and both connections will increase their window by 1 MSS per RTT as a result of TCP's congestion-avoidance algorithm. Thus, the joint throughput of the two connections proceeds along a 45-degree line (equal increase for both connections) starting from point A. Eventually, the link bandwidth jointly consumed by the two connections will be greater than R , and eventually packet loss will occur. Suppose that connections 1 and 2 experience packet loss when they realize throughputs indicated by point B. Connections 1 and 2 then decrease their windows by a factor of two. The resulting throughputs realized are thus at point C, halfway along a vector starting at B and ending at the origin. Because the joint bandwidth use is less than R at point C, the two connections again increase their throughputs along a 45-degree line starting from C. Eventually, loss will again occur, for example, at point D, and the two connections again decrease their window sizes by a factor of two, and so on. You should convince yourself that the bandwidth realized by the two connections eventually fluctuates along the equal bandwidth share line. You should also convince yourself that the two connections will converge to this behavior regardless of where they are in the two-dimensional space! Although a number of idealized assumptions lie behind this scenario, it still provides an intuitive feel for why TCP results in an equal sharing of bandwidth among connections.

In our idealized scenario, we assumed that only TCP connections traverse the bottleneck link, that the connections have the same RTT value, and that only a

single TCP connection is associated with a host-destination pair. In practice, these conditions are typically not met, and client-server applications can thus obtain very unequal portions of link bandwidth. In particular, it has been shown that when multiple connections share a common bottleneck, those sessions with a smaller RTT are able to grab the available bandwidth at that link more quickly as it becomes free (that is, open their congestion windows faster) and thus will enjoy higher throughput than those connections with larger RTTs [Lakshman 1997].

Fairness and UDP

We have just seen how TCP congestion control regulates an application’s transmission rate via the congestion window mechanism. Many multimedia applications, such as Internet phone and video conferencing, often do not run over TCP for this very reason—they do not want their transmission rate throttled, even if the network is very congested. Instead, these applications prefer to run over UDP, which does not have built-in congestion control. When running over UDP, applications can pump their audio and video into the network at a constant rate and occasionally lose packets, rather than reduce their rates to “fair” levels at times of congestion and not lose any packets. From the perspective of TCP, the multimedia applications running over UDP are not being fair—they do not cooperate with the other connections nor adjust their transmission rates appropriately. Because TCP congestion control will decrease its transmission rate in the face of increasing congestion (loss), while UDP sources need not, it is possible for UDP sources to crowd out TCP traffic. An area of research today is thus the development of congestion-control mechanisms for the Internet that prevent UDP traffic from bringing the Internet’s throughput to a grinding halt [Floyd 1999; Floyd 2000; Kohler 2006].

Fairness and Parallel TCP Connections

But even if we could force UDP traffic to behave fairly, the fairness problem would still not be completely solved. This is because there is nothing to stop a TCP-based application from using multiple parallel connections. For example, Web browsers often use multiple parallel TCP connections to transfer the multiple objects within a Web page. (The exact number of multiple connections is configurable in most browsers.) When an application uses multiple parallel connections, it gets a larger fraction of the bandwidth in a congested link. As an example, consider a link of rate R supporting nine ongoing client-server applications, with each of the applications using one TCP connection. If a new application comes along and also uses one TCP connection, then each application gets approximately the same transmission rate of $R/10$. But if this new application instead uses 11 parallel TCP connections, then the new application gets an unfair allocation of more than $R/2$. Because Web traffic is so pervasive in the Internet, multiple parallel connections are not uncommon.