

3.7.2 TCP Delay Modeling

We end this chapter with some simple models for calculating the time it takes TCP to send an object (such as an image, a text file, or an MP3). For a given object, we define the **latency** as the time from when the client initiates a TCP connection until the time at which the client receives the requested object in its entirety. The models presented here give important insight into the key components of latency, including initial TCP handshaking, TCP slow start, and the transmission time of the object.

This simple analysis supposes that the network is uncongested—that is, that the TCP connection transporting the object does not have to share link bandwidth with other TCP or UDP traffic. Also, in order not to obscure the central issues, we carry out the analysis in the context of the simple one-link network as shown in Figure 3.54. (This link might model a single bottleneck on an end-to-end path. See also the homework problems for an explicit extension to the case of multiple links.)

We also make the following simplifying assumptions:

- ◆ The amount of data that the sender can transmit is limited solely by the sender’s congestion window. (Thus, the TCP receive buffers are large.)
- ◆ Packets are neither lost nor corrupted, so that there are no retransmissions.
- ◆ All protocol header overheads—including TCP, IP, and link-layer headers—are negligible and ignored.
- ◆ The object (that is, file) to be transferred consists of an integer number of segments of size MSS (maximum segment size).
- ◆ The only packets that have non-negligible transmission times are packets that carry maximum-sized TCP segments. Request messages, acknowledgments, and TCP connection-establishment segments are small and have negligible transmission times.
- ◆ The initial threshold in the TCP congestion-control mechanism is a large value that is never attained by the congestion window.

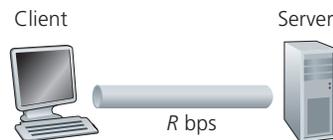


Figure 3.54 ◆ A simple one-link network connecting a client and a server

We also introduce the following notation:

- ◆ The size of the object to be transferred is O bits.
- ◆ The MSS is S bits (for example, 536 bytes).
- ◆ The transmission rate of the link from the server to the client is R bps.

Before beginning the formal analysis, let us try to gain some intuition. What would be the latency if there were no congestion-window constraint, that is, if the server were permitted to send segments back to back until the entire object was sent. To answer this question, first note that one RTT is required to initiate the TCP connection. After one RTT, the client sends a request for the object (which is piggybacked onto the third segment in the three-way TCP handshake). After a total of two RTTs, the client begins to receive data from the server. The client receives data from the server for a period of time O/R , the time for the server to transmit the entire object. Thus, in the case of no congestion-window constraint, the total latency is $2RTT + O/R$. This represents a lower bound; the slow-start procedure, with its dynamic congestion window, will of course increase this latency.

Static Congestion Window

Although TCP uses a dynamic congestion window, it is instructive to analyze first the case of a static congestion window. Let W , a positive integer, denote a fixed-size static congestion window. For the static congestion window, the server is not permitted to have more than W unacknowledged outstanding segments. When the server receives the request from the client, the server immediately sends W segments back to back to the client. The server then sends one segment into the network for each acknowledgment it receives from the client. The server continues to send one segment for each acknowledgment until all of the segments of the object have been sent. There are two cases to consider:

1. $WS/R > RTT + S/R$. In this case, the server receives an acknowledgment for the first segment in the first window before the server completes the transmission of the first window.
2. $WS/R < RTT + S/R$. In this case, the server transmits the first window's worth of segments before the server receives an acknowledgment for the first segment in the window.

Let us first consider the first case, which is illustrated in Figure 3.55. In this figure the window size is $W = 4$ segments. One RTT is required to initiate the TCP connection. After one RTT, the client sends a request for the object (which is piggybacked onto the third segment in the three-way TCP handshake). After a total of two RTTs, the client begins to receive data from the server. Segments arrive

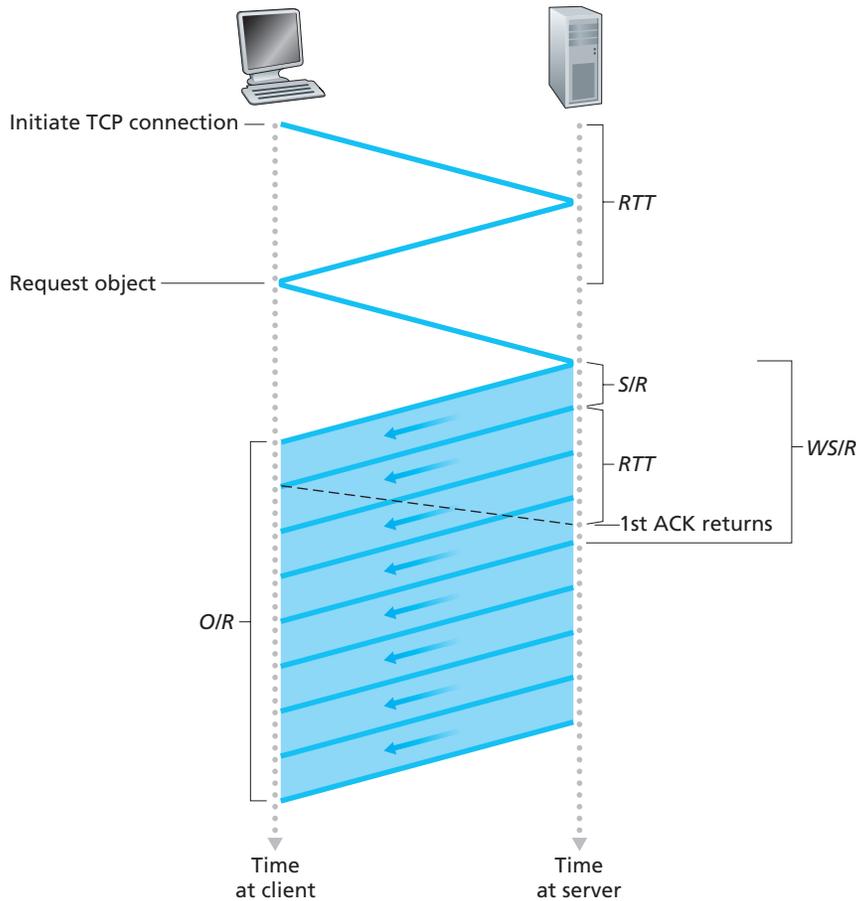


Figure 3.55 ♦ The case $WS/R > RTT + S/R$

periodically from the server every S/R seconds, and the client acknowledges every segment it receives from the server. Because the server receives the first acknowledgment before it completes sending a window's worth of segments, the server continues to transmit segments after having transmitted the first window's worth of segments. And because the acknowledgments arrive periodically at the server every S/R seconds from the time when the first acknowledgment arrives, the server transmits segments continuously until it has transmitted the entire object. Thus, once the server starts to transmit the object at rate R , it continues to transmit the object at rate R until the entire object is transmitted. The latency therefore is $2 RTT + O/R$.

Now let us consider the second case, which is illustrated in Figure 3.56. In this figure, the window size is $W = 2$ segments. Once again, after a total of two RTTs, the client begins to receive segments from the server. These segments arrive periodically every S/R seconds, and the client acknowledges every segment it receives from the server. But now the server completes the transmission of the first window before the first acknowledgment arrives from the client. Therefore, after sending a window, the server must stall and wait for an acknowledgment before resuming transmission. When an acknowledgment finally arrives, the server sends a new segment to the client. With the first acknowledgment, a window's worth of acknowledgments arrives, and each successive acknowledgment is spaced by S/R seconds. For each of these acknowledgments, the server sends exactly one segment. Thus, the server alternates between two states: a transmitting state, during which it transmits

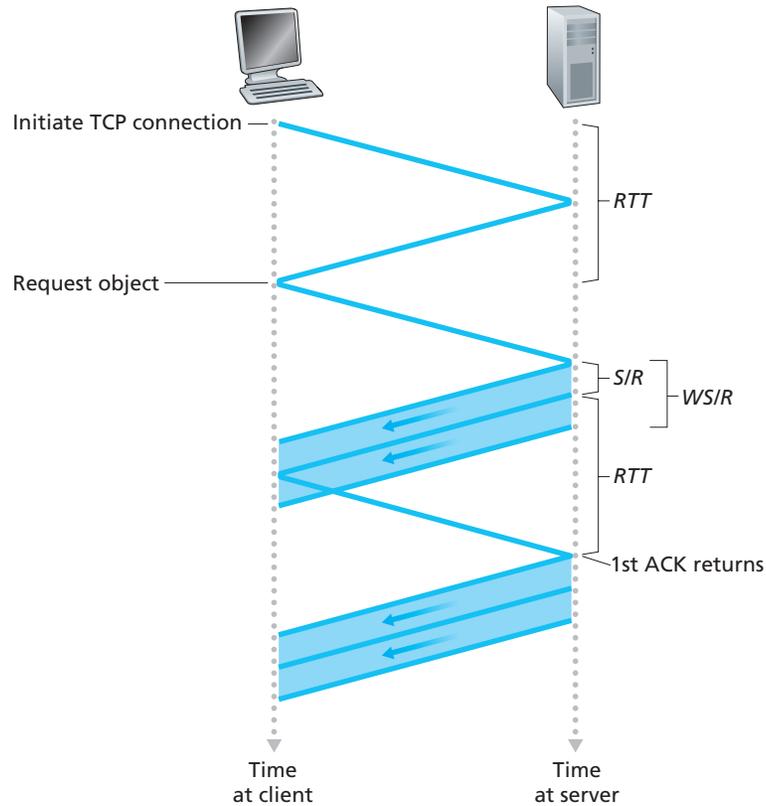


Figure 3.56 ♦ The case $WS/R < RTT + S/R$

W segments, and a stalled state, during which it transmits nothing and waits for an acknowledgment. The latency is equal to 2 RTT plus the time required for the server to transmit the object, O/R , plus the amount of time that the server is in the stalled state. To determine the amount of time the server is in the stalled state, let K be the number of windows of data that cover the object; that is, $K = O/WS$ (if O/WS is not an integer, then round K up to the nearest integer). The server is in the stalled state between the transmission of each of the windows, that is, for $K - 1$ periods of time, with each period lasting $\text{RTT} - (W - 1)S/R$ (see Figure 3.56). Thus, for case 2,

$$\text{latency} = 2 \text{ RTT} + O/R + (K - 1) [S/R + \text{RTT} - WS/R]$$

Combining the two cases, we obtain

$$\text{latency} = 2 \text{ RTT} + O/R + (K - 1) [S/R + \text{RTT} - WS/R]^+$$

where $[x]^+ = \max(x, 0)$. Notice that the delay has three components: 2 RTT to set up the TCP connection and to request and begin to receive the object; O/R , the time for the server to transmit the object; and a final term $(K - 1) [S/R + \text{RTT} - WS/R]^+$ for the amount of time the server stalls.

This completes our analysis of static windows. The following analysis for dynamic windows is more complicated but parallels that for static windows.

Dynamic Congestion Window

We now take TCP's dynamic congestion window into account in the latency model. Recall that the server starts with a congestion window of one segment and sends one segment to the client. When it receives an acknowledgment for the segment, it increases its congestion window to two segments and sends two segments to the client (spaced apart by S/R seconds). As it receives the acknowledgments for the two segments, it increases the congestion window to four segments and sends four segments to the client (again spaced apart by S/R seconds). The process continues, with the congestion window doubling every RTT. A timing diagram for TCP is illustrated in Figure 3.57.

Note that O/S is the number of segments in the object; in Figure 3.57, $O/S = 15$. Consider the number of segments that are in each of the windows. The first window contains one segment, the second window contains two segments, and the third window contains four segments. More generally, the k th window contains 2^{k-1} segments. Let K be the number of windows that cover the object; in the preceding diagram, $K = 4$. In general, we can express K in terms of O/S as follows:

$$K = \min \left\{ k : 2^0 + 2^1 + \dots + 2^{k-1} \geq \frac{O}{S} \right\}$$

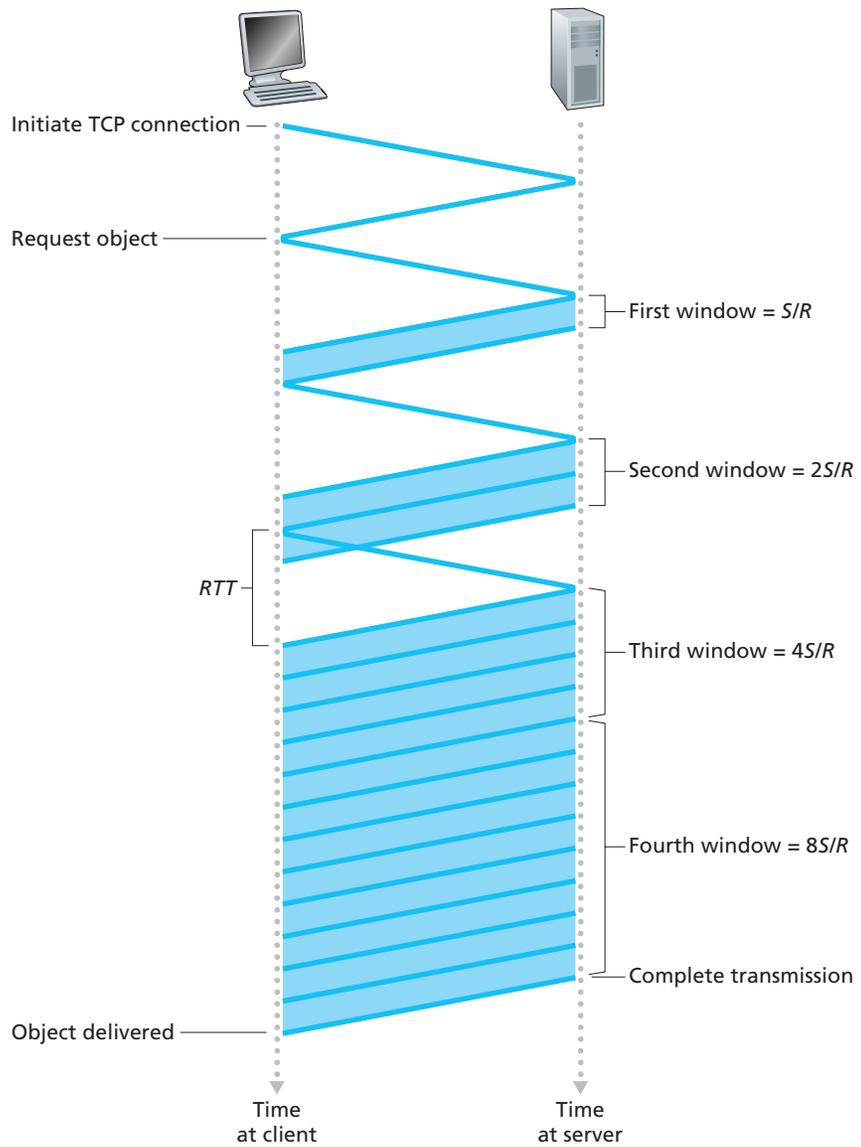


Figure 3.57 ♦ TCP timing during slow start

$$\begin{aligned}
&= \min \left\{ k : 2^k - 1 \geq \frac{O}{S} \right\} \\
&= \min \left\{ k : k \geq \log_2 \left(\frac{O}{S} + 1 \right) \right\} \\
&= \left\lceil \log_2 \left(\frac{O}{S} + 1 \right) \right\rceil
\end{aligned}$$

After transmitting a window's worth of data, the server may stall (that is, stop transmitting) while it waits for an acknowledgment. In Figure 3.55, the server stalls after transmitting the first and second windows but not after transmitting the third. Let us now calculate the amount of stall time after transmitting the k th window. From the time the server begins to transmit the k th window until the time when the server receives an acknowledgment for the first segment in the window is $S/R + \text{RTT}$. The transmission time of the k th window is $(S/R)2^{k-1}$. The stall time is the difference of these two quantities, that is,

$$[S/R + \text{RTT} - 2^{k-1} (S/R)]^+$$

The server can potentially stall after the transmission of each of the first $K - 1$ windows. (The server is done after the transmission of the K th window.) We can now calculate the latency for transferring the file. The latency has three components: 2RTT for setting up the TCP connection and requesting the file; O/R , the transmission time of the object; and the sum of all the stalled times. Thus,

$$\text{Latency} = 2\text{RTT} + \frac{O}{R} + \sum_{k=1}^{K-1} \left[\frac{S}{R} + \text{RTT} - 2^{k-1} \frac{S}{R} \right]^+$$

Compare this equation with the latency equation for static congestion windows; all the terms are exactly the same except that the term WS/R for static windows has been replaced by $2^{k-1}(S/R)$ for dynamic windows. To obtain a more compact expression for the latency, let Q be the number of times the server would stall if the object contained an infinite number of segments. Paralleling a derivation similar to that for K (see homework problems), we obtain:

$$Q = \left\lceil \log_2 \left(1 + \frac{\text{RTT}}{S/R} \right) \right\rceil + 1$$

The actual number of times that the server stalls is $P = \min \{ Q, K - 1 \}$. In Figure 3.55, $P = Q = 2$. Combining the equations (see homework problems) gives the following closed-form expression for the latency:

$$\text{Latency} = 2RTT + \frac{O}{R} + P \left[RTT + \frac{S}{R} \right] - (2^P - 1) \frac{S}{R}$$

Thus, to calculate the latency, we simply must calculate K and Q , set $P = \min \{Q, K-1\}$ and plug P into the formula above.

It is interesting to compare the TCP latency with the latency that would occur if there were no congestion control (that is, no congestion-window constraint). Without congestion control, the latency is $2 RTT + O/R$, which we define to be the *minimum latency*. It is a simple exercise to show that

$$\frac{\text{Latency}}{\text{MinimumLatency}} \leq 1 + \frac{P}{[(O/R)/RTT] + 2}$$

We see from the formula above that TCP slow start will not significantly increase latency if $RTT \ll O/R$, that is, if the round-trip time is much less than the transmission time of the object.

Let us now take a look at some example scenarios. In all the scenarios we set $S = 536$ bytes, a common default value for TCP. We use an RTT of 100 msec, which is a typical value for a continental or intercontinental delay over moderately congested links. First consider sending a rather large object of size $O = 100$ kbytes. The number of windows that cover this object is $K = 8$. For a number of transmission rates, the following table displays the effect of the slow-start mechanism on the latency.

R	O/R	P	Minimum Latency: $O/R + 2 RTT$	Latency with Slow Start
28 kbps	28.6 sec	1	28.8 sec	28.9 sec
100 kbps	8 sec	2	8.2 sec	8.4 sec
1 Mbps	800 msec	5	1 sec	1.5 sec
10 Mbps	80 msec	7	0.28 sec	0.98 sec

We see from the table that for a large object, slow start adds appreciable delay only when the transmission rate is high. If the transmission rate is low, then acknowledgments come back relatively quickly, and TCP quickly ramps up to its maximum rate. For example, when $R = 100$ kbps, the number of stall periods is $P = 2$, whereas the number of windows to transmit is $K = 8$; thus the server stalls only after the first two of eight windows. On the other hand, when $R = 10$ Mbps, the server stalls after each window, which causes a significant increase in the delay.

Now consider sending a small object of size $O = 5$ kbytes. The number of windows that cover this object is $K = 4$. For a number of transmission rates, the following table examines the effect of the slow-start mechanism.

R	O/R	P	Minimum Latency: $O/R + 2 RTT$	Latency with Slow Start
28 kbps	1.43 sec	1	1.63 sec	1.73 sec
100 kbps	0.4 sec	2	0.6 sec	0.76 sec
1 Mbps	40 msec	3	0.24 sec	0.52 sec
10 Mbps	4 msec	3	0.20 sec	0.50 sec

Once again, slow start adds an appreciable delay when the transmission rate is high. For example, when $R = 1$ Mbps, the server stalls after each window, which causes the latency to be more than twice that of the minimum latency.

For a larger RTT, the effect of slow start becomes significant for small objects for smaller transmission rates. The following table examines the effect of slow start for $RTT = 1$ second and $O = 5$ kbytes ($K = 4$).

R	O/R	P	Minimum Latency: $O/R + 2 RTT$	Latency with Slow Start
28 kbps	1.43 sec	3	3.4 sec	5.8 sec
100 kbps	0.4 sec	3	2.4 sec	5.2 sec
1 Mbps	40 msec	3	2.0 sec	5.0 sec
10 Mbps	4 msec	3	2.0 sec	5.0 sec

In summary, slow start can significantly increase latency when the object size is relatively small and the RTT is relatively large. Unfortunately, this is often the case with the Web.

An Example: HTTP

As an application of the latency analysis, let's now calculate the response time for a Web page sent over nonpersistent HTTP. Suppose that the page consists of one base HTML page and M referenced images. To keep things simple, let us assume that each of the $M + 1$ objects contains exactly O bits.

With nonpersistent HTTP, each object is transferred independently, one after the other. The response time of the Web page is therefore the sum of the latencies for the individual objects. Thus

$$\text{Response time} = (M + 1) \left\{ 2RTT + \frac{O}{R} + P \left[RTT + \frac{S}{R} \right] - (2^P - 1) \frac{S}{R} \right\}$$

Note that the response time for nonpersistent HTTP takes the form

$$\text{Response time} = (M + 1)O/R + 2(M + 1)RTT + \text{latency due to TCP slow start for each of the } M + 1 \text{ objects.}$$

Clearly, if there are many objects in the Web page and if RTT is large, then nonpersistent HTTP will have poor response-time performance. In the homework problems, we will investigate the response time for other HTTP transport schemes. The reader is also encouraged to see [Heidemann 1997; Cardwell 2000] for a related analysis.

3.8 Summary

We began this chapter by studying the services that a transport-layer protocol can provide to network applications. At one extreme, the transport-layer protocol can be very simple and offer a no-frills service to applications, providing only a multiplexing/demultiplexing function for communicating processes. The Internet's UDP protocol is an example of such a no-frills transport-layer protocol. At the other extreme, a transport-layer protocol can provide a variety of guarantees to applications, such as reliable delivery of data, delay guarantees, and bandwidth guarantees. Nevertheless, the services that a transport protocol can provide are often constrained by the service model of the underlying network-layer protocol. If the network-layer protocol cannot provide delay or bandwidth guarantees to transport-layer segments, then the transport-layer protocol cannot provide delay or bandwidth guarantees for the messages sent between processes.

We learned in Section 3.4 that a transport-layer protocol can provide reliable data transfer even if the underlying network layer is unreliable. We saw that providing reliable data transfer has many subtle points, but that the task can be accomplished by carefully combining acknowledgments, timers, retransmissions, and sequence numbers.

Although we covered reliable data transfer in this chapter, we should keep in mind that reliable data transfer can be provided by link-, network-, transport-, or application-layer protocols. Any of the upper four layers of the protocol stack can implement acknowledgments, timers, retransmissions, and sequence numbers and provide reliable data transfer to the layer above. In fact, over the years, engineers and computer scientists have independently designed and implemented link-, network-, transport-, and application-layer protocols that provide reliable data transfer (although many of these protocols have quietly disappeared).

In Section 3.5 we took a close look at TCP, the Internet's connection-oriented and reliable transport-layer protocol. We learned that TCP is complex, involving connection management, flow control, and round-trip time estimation, as well as reliable data