Transport Layer Part IV

ram2ur and knt4xx

March 5th 2020

1 Overview

The Transmission Control Protocol (TCP) is the Internet's Transport-level reliable transmission protocol, as discussed in earlier lectures. This section focuses on how TCP answers the following questions:

1. If the flow of packets in is faster than I can process them, how do I prevent loss?
2. How do I open and close a connection?
3. How can I avoid sending data too fast for the network to handle?

The solutions are built into TCP via flow control, connection management, and congestion control strategies.

2 Flow Control

Packets are sent over a TCP connection at some rate (R_S), captured by a buffer in the receiver, and then removed and processed by the receiver's application layer at a different rate (R_R).

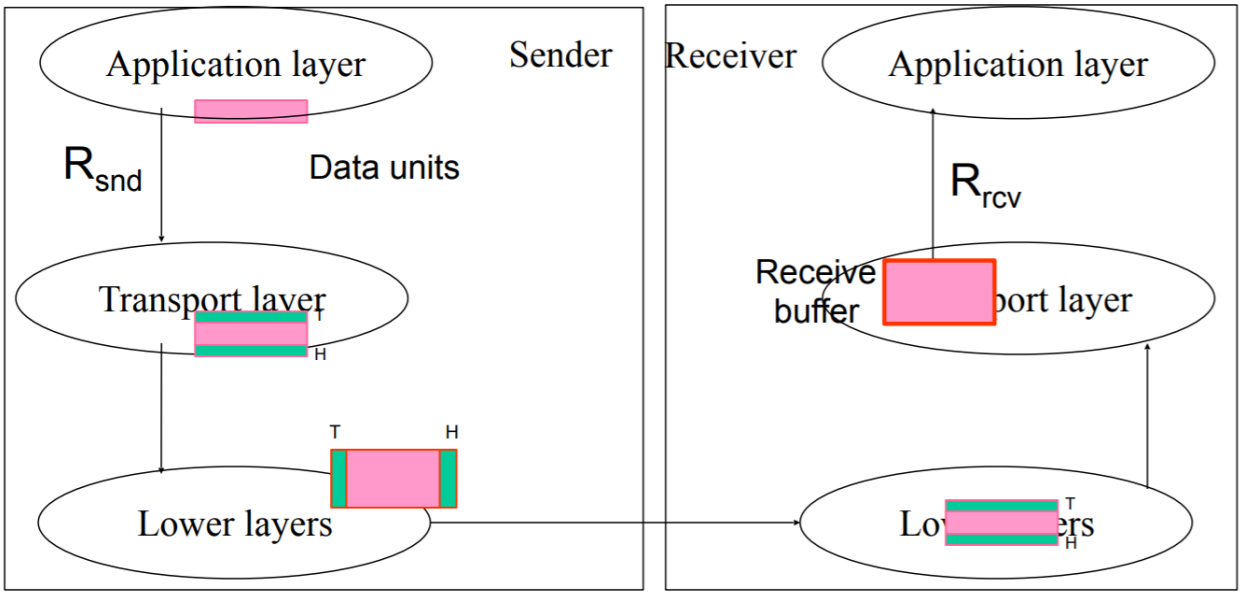

Figure 1: Flow of Packets over TCP Connection

If packets are sent faster than they are received, any receiver buffers will eventually overflow and packets

will be dropped. This is the flow control problem.

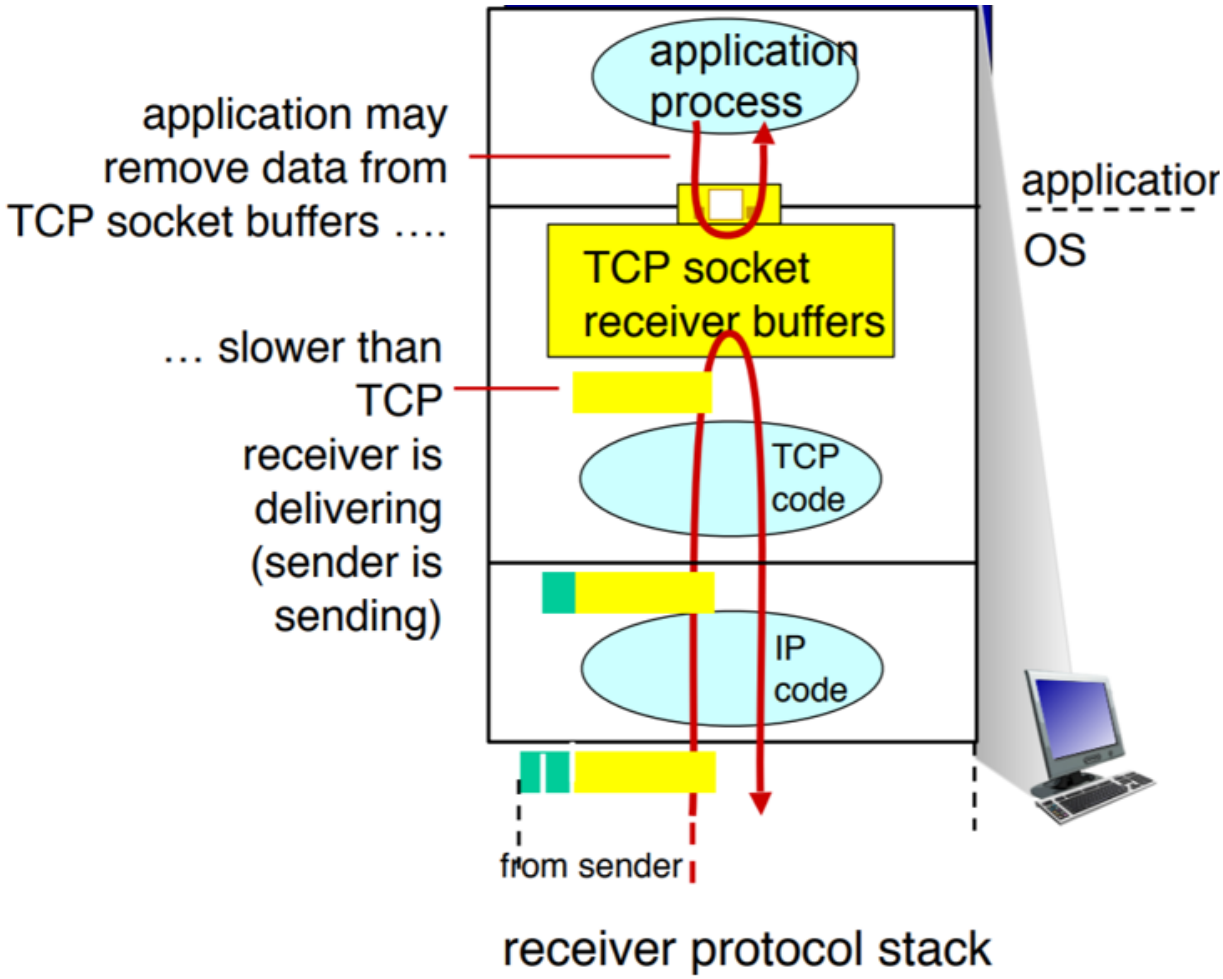

Figure 2: Flow Control Problem ($R_S > R_R$)

This problem could arise due to inherent limitations in the receiver, or due to more frequent context switches on it. The problem of context switches affecting R_R means that our solution must be adaptable to many different speeds.

The general idea of the solution is to implement some **flow control**, or method for the receiver to control the sender so that the sender will not overflow the buffer by sending too much, too fast. There are several different ways to achieve this.

2.1 Stop-and-Wait Flow Control

The simplest form of flow control is for the sender to wait after every sent package for an ACK. The receiver only sends the ACK after its application layer has retrieved the package. This can be optimized based on the buffer size, so that packets are only sent in groups less than the buffer size.

The advantage of this method is that it is **simple**, and the buffer can **never overflow** since the sender only sends packages when the buffer can hold them. However, this is a very **slow** way to send information, and can be improved upon.

2.2 ON-OFF Flow Control

Rather than waiting for an ACK after every message, the buffer can be allowed to come close to filling before an OFF message is sent. This OFF message will make the sender wait until an ON message is sent (when the buffer has enough space).

An important question in this method is when to send the OFF message. If it were to be sent right when the buffer filled, whatever packets were already in transit would be dropped. Given that the sender transmits at rate R_{snd} and the receiver's application layer depletes the buffer at rate R_{rcv}, the difference gives the rate at which data is sent that will contribute to filling the buffer. Multiplying by the Round Trip Time (RTT) accounts for time to transmit a package and receive a subsequent STOP message. This means the decision of when to send a STOP message can be based on:

$$2d_{\text{prop}} * (R_{\text{snd}} - R_{\text{rcv}}) \tag{1}$$

However, this presents some difficulties, such as how to know R_{snd} and R_{rcv} accurately, especially since both might be variable. To improve this design, we can take the idea of transmitting feedback about the remaining buffer to the sender and give it more flexibility than a simple ON/OFF message.

2.3 Sliding Window Flow Control

Sliding Window Flow Control transmits the remaining buffer space to the receiver, called the **advertised window** or **flow window**. The sender uses this to determine how many unacknowledged frames it can have.

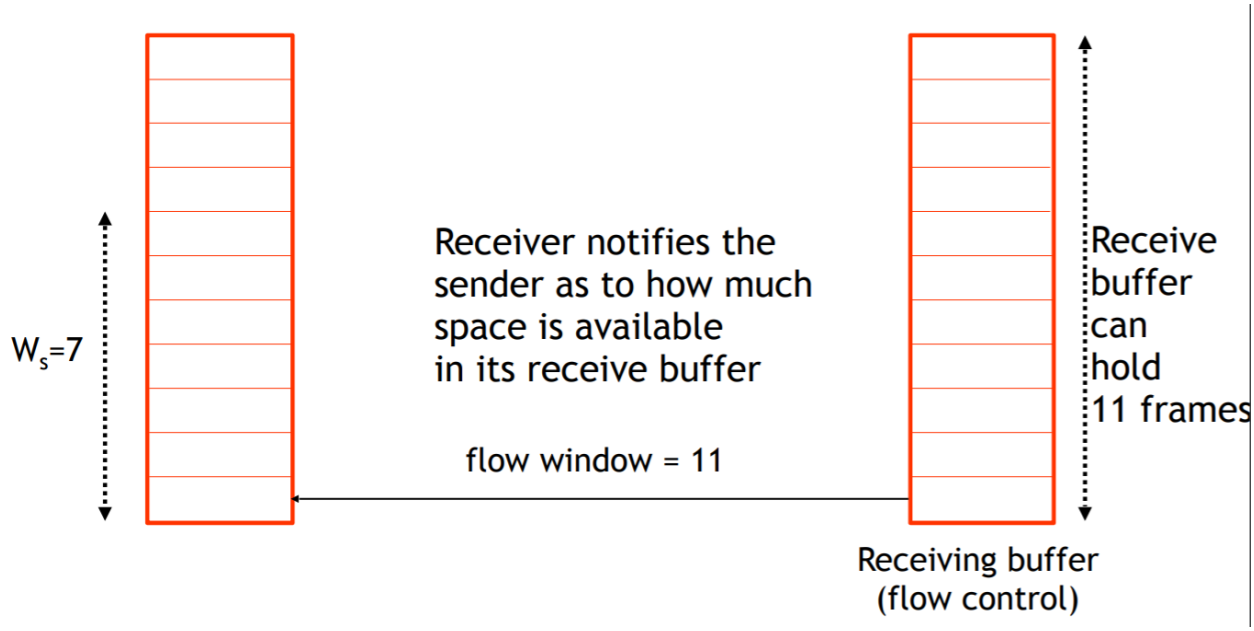

Figure 3: Sliding Window Flow Control

The sender itself has a Window Size (W_S), which can be used to wrap around the indices of sent packets (0, 1, ..., W_S, 0, 1...). It uses the following process to determine how many packets to send:

1. Find X = min(window size (W_S), flow window)
2. Find Y = number of outstanding frames (send but no ACK)
3. Max number of frames the sender is permitted is X - Y

If the max number of frames is sent, the number of outstanding frames becomes X, or the limit of allowed outstanding frames.

The receiver can use the **rwnd** value in the TCP header to advertise how much buffer space it has available.

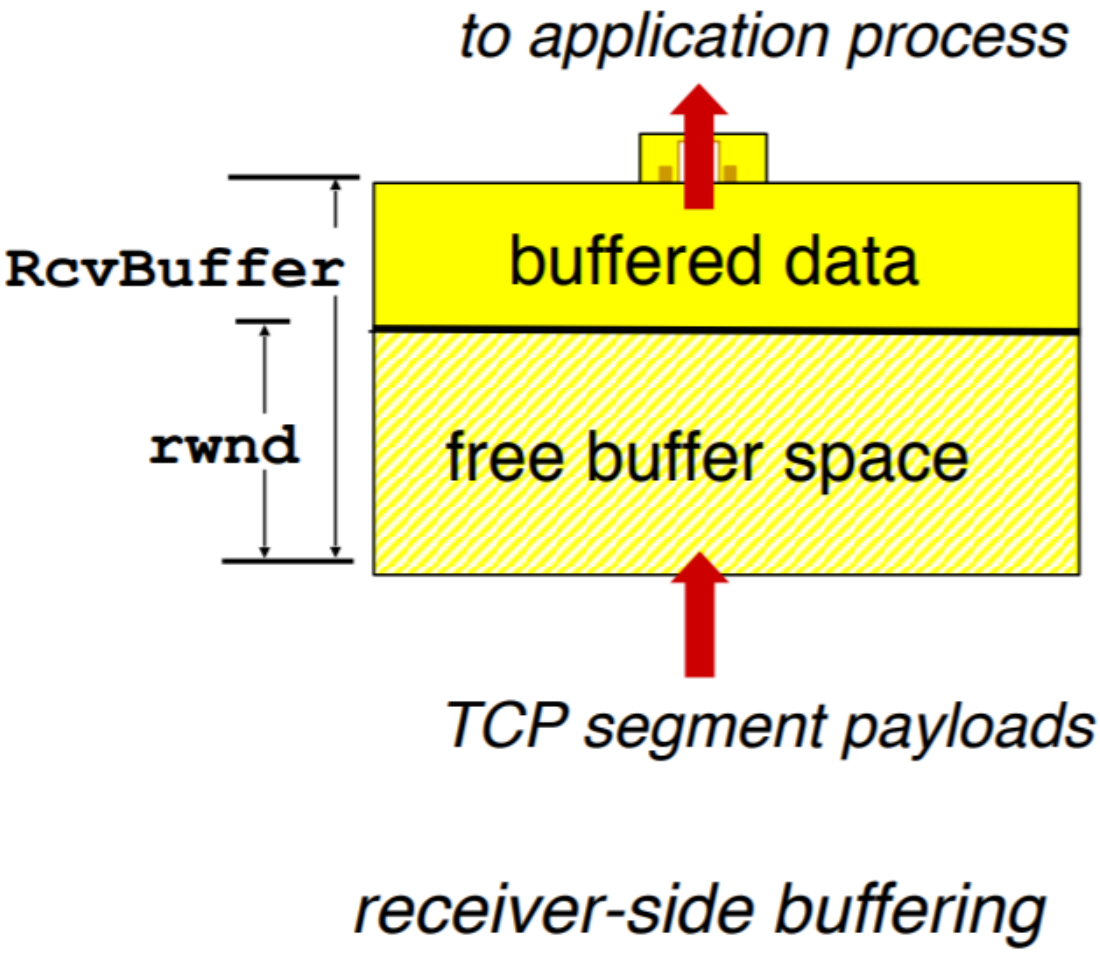

Figure 4: Advertising Free Buffer Space

The sender can then use this info to make sure that there are never enough ”in flight” unACKed messages to overflow this free buffer space.

3 Connection Management

In the earlier discussion of flow control, an initial buffer size needs to be agreed upon, and a connnection established. How is this done? Establishing a connection must be done via some **handshake**, or agreement between the two communicating parties that sets up the connection parameters.

3.1 The Three-Way Handshake

A two-way handshake would be the simplest way to establish a connection, in which the sender requests to transmit and the receiver sends an ACK.

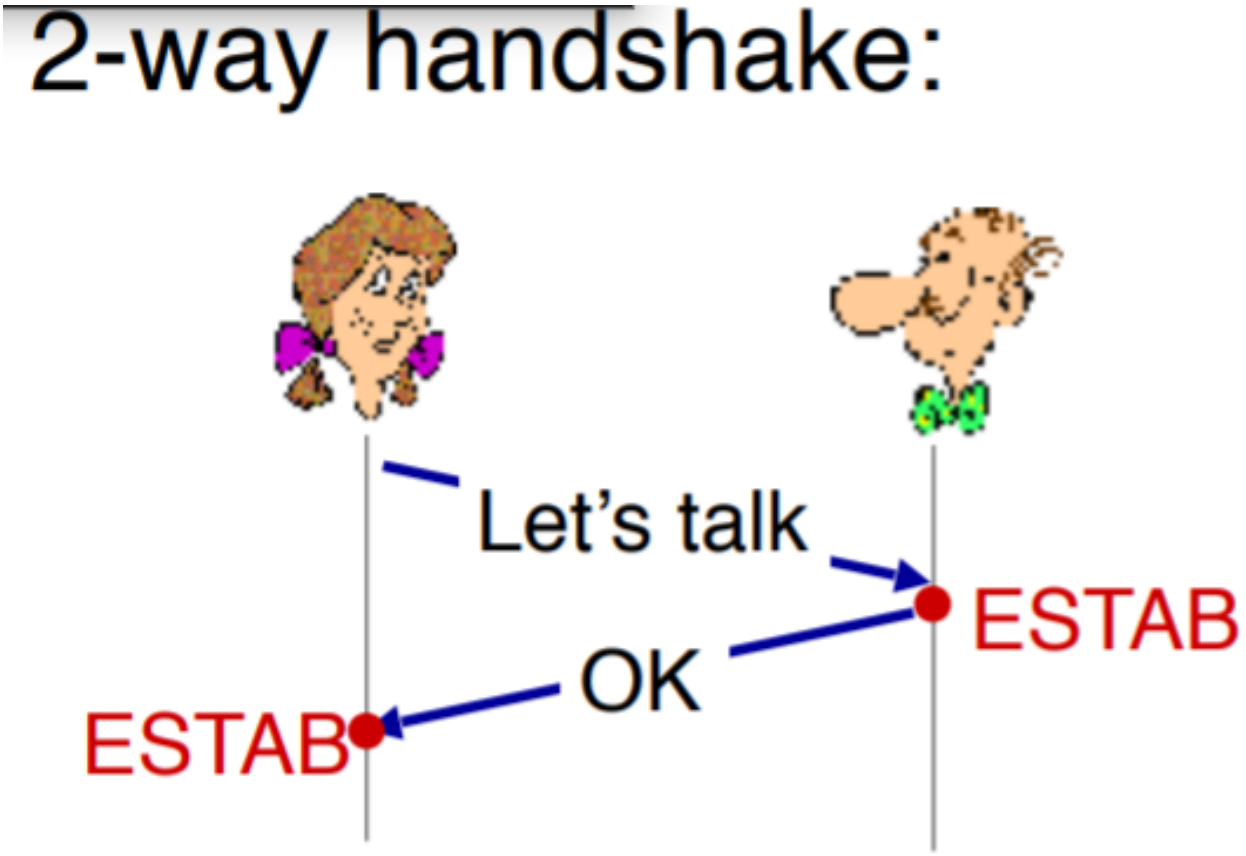

Figure 5: Two-Way Handshake

However, there are many problems with this approach on an unreliable network:

1. variable delays
2. retransmitted messages due to message loss
3. message reordering
4. can't "see" other side

These issues can result in several failure scenarios, such as when the sender re-transmits the connection request (the issue of variable delays) which arrives at the server after the server has forgotten the client (leading to a half-open connection). A second failure scenario, shown on the right, has a similar root issue, except that some re-transmitted data arrives at the latent established connection.

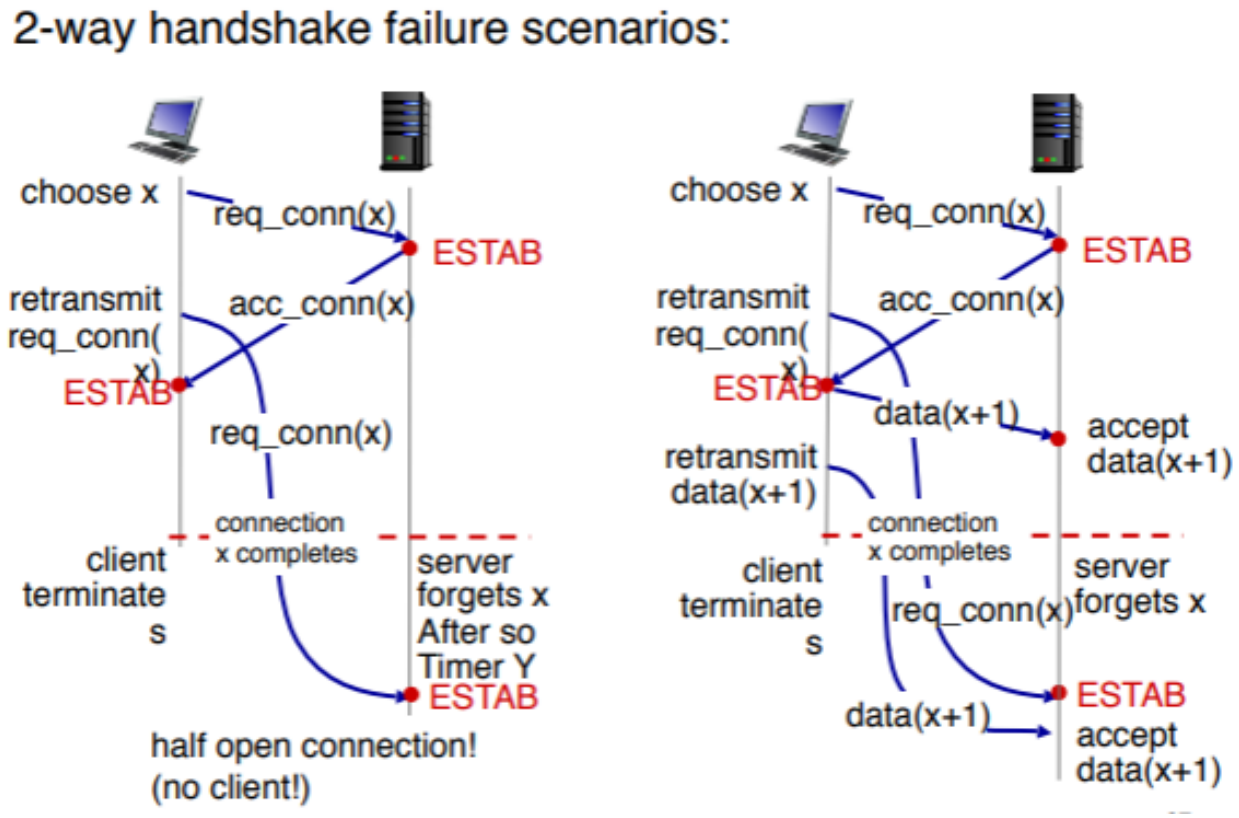

Figure 6: Two-Way Handshake Failures

These issues are rectified by the TCP 3-way handshake design. Instead of just a request, the sender transmits a SYN bit, which the server replies to with a SYN and ACK together. Finally, the sender transmits an ACK, which concludes the handshake. This way, if the failure scenarios shown in Figure 6 were to occur, a hanging connection would not be made, since the server would still require an ACK to finish the handshake.

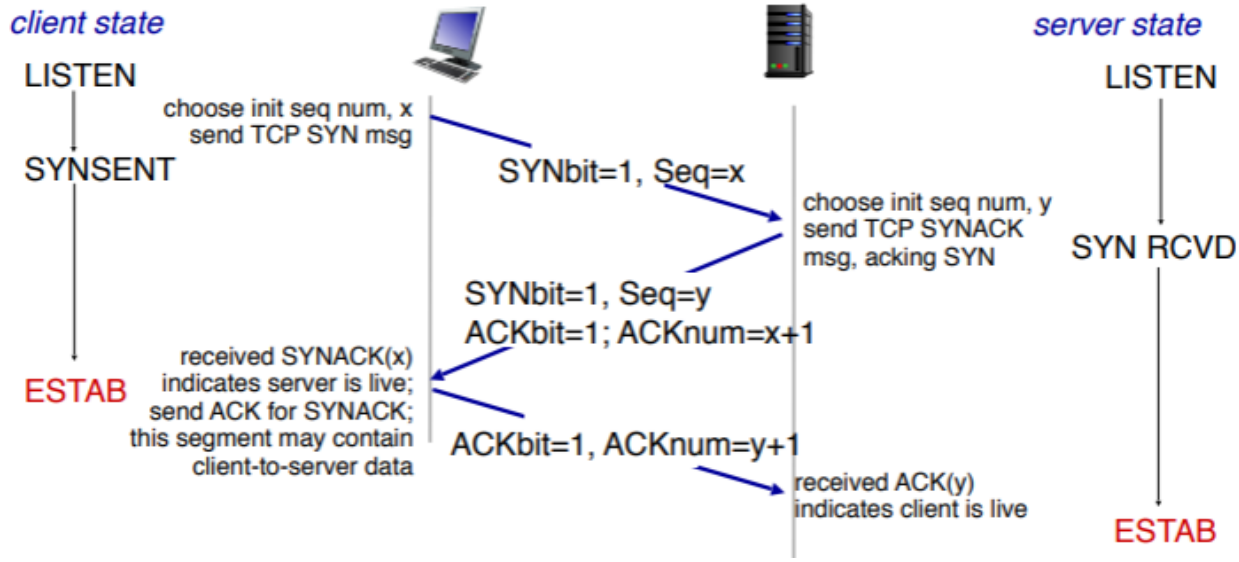

Figure 7: Three-Way Handshake

To close a connection without losing data, the client and server both close their side of the connection by sending a TCP segment with FIN bit = 1, then send an ACK upon receiving a FIN message.

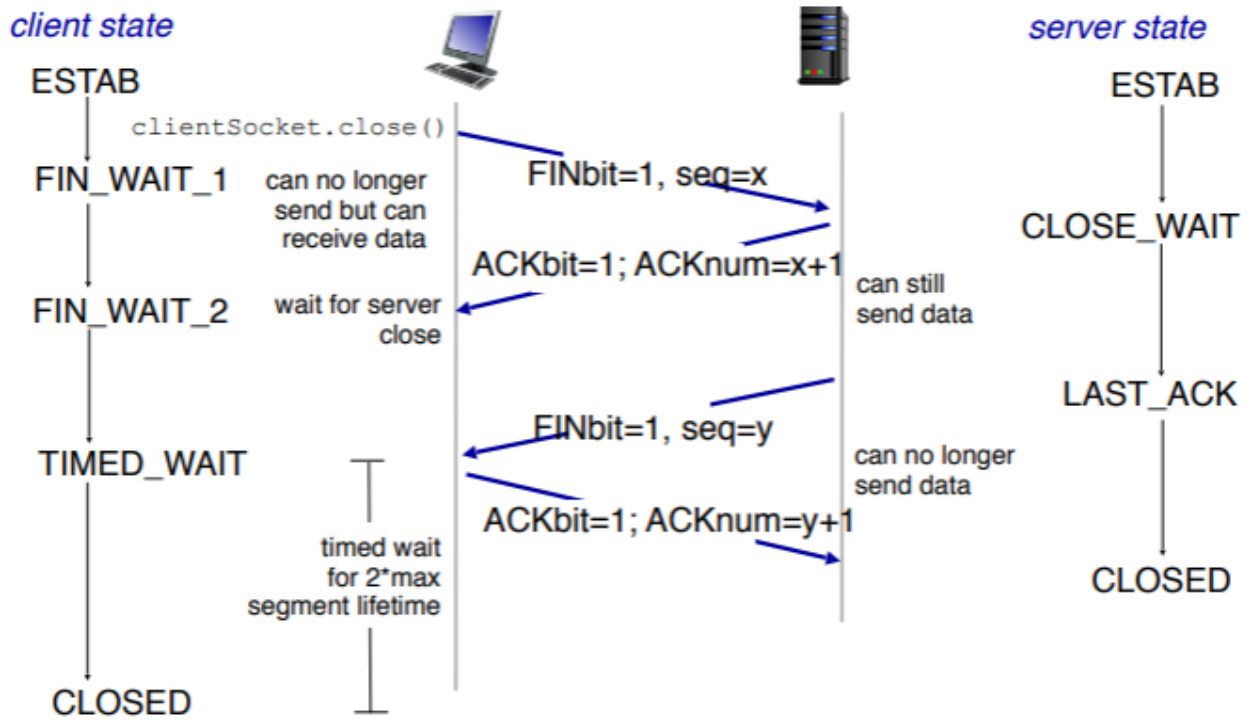

Figure 8: Closing Connection

Once one side sends their FIN, they wait for an ACK, then a FIN from the other party, then send an ACK and close.

4 TCP Congestion Control

Congestion control is the issue of too many sources sending too much data too fast for the *network* to handle, rather than the receiver. It can lead to:

1. Lost packets (buffer overflow at routers)
2. Long delays (queuing in router buffers)

This is a problem which occurs when the router buffers rate limit the transmission of data on the network. One possible solution is to automatically halve the window sizes (W_S) of the senders when many losses not due to window size are detected.