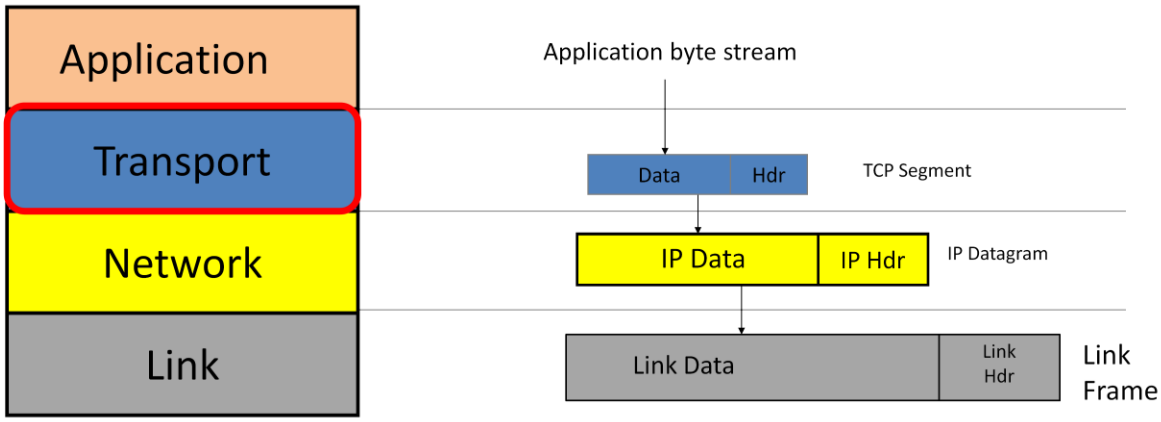
CS144
An Introduction to Computer Networks

What the Internet is
The TCP Service Model

Nick McKeown
Professor of Electrical Engineering
and Computer Science, Stanford University

In this video, you're going to learn about the service provided to applications by TCP – the Transmission Control Protocol - which is used by over 95% of Internet applications. TCP is almost universally used because it provides the reliable, end-to-end, bi directional byte-stream service that almost all applications want.

Transmission Control Protocol (TCP)



2

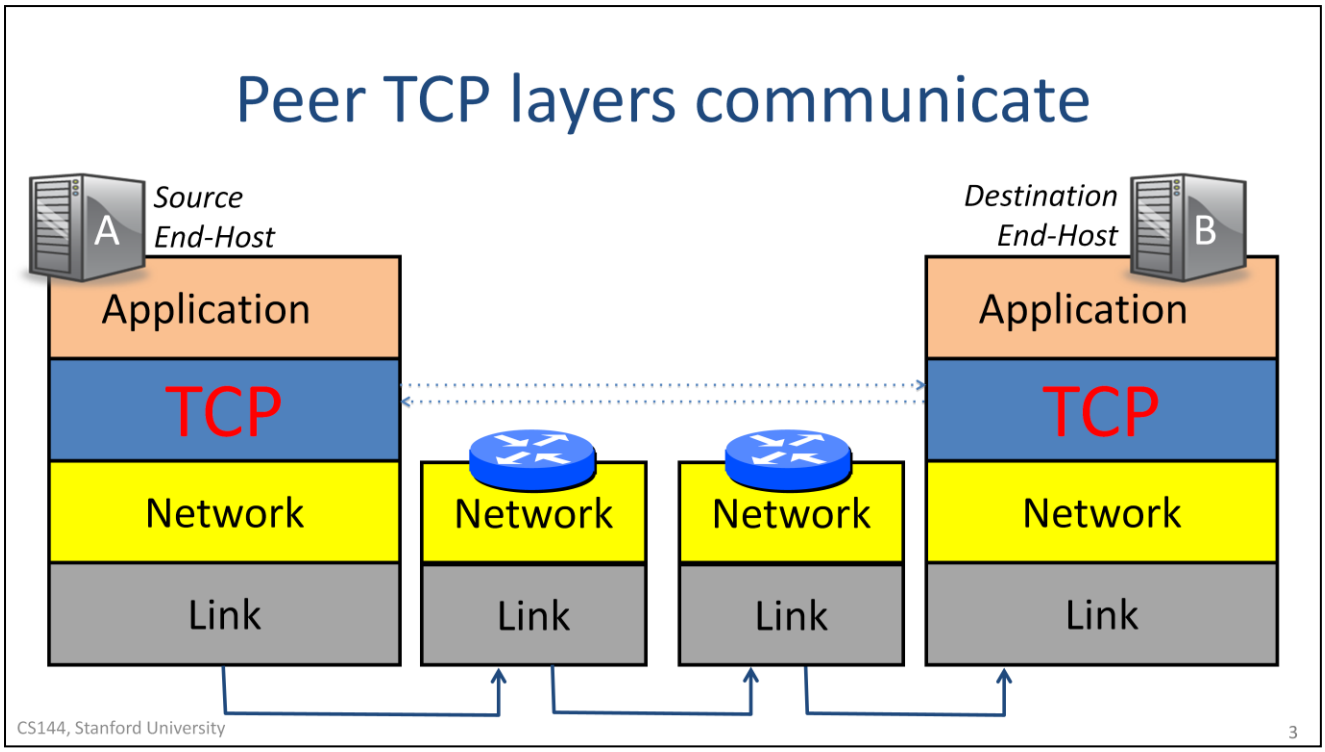
TCP is an example of the transport layer. <click>

When an application calls TCP, it hands it some bytes that it wants delivered to the other end.

TCP places these bytes into a TCP Segment, and then takes it from there.

<click> TCP hands the segment to the IP layer, which encapsulates it in an IP datagram. The IP addresses are added.

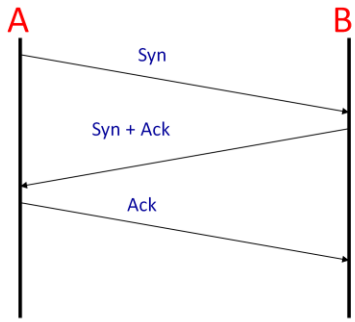
<click> The IP datagram is handed to the Link Layer, which builds the link frame, adds the Link address – for example, the Ethernet addresses – and then sends it onto the wire.



When two applications use TCP, they establish a two-way communication channel between the TCP peers at both ends. First TCP establishes a communication channel from A to B <click> Then it establishes a channel from B to A <click>.

We call the two way communication a “connection”. At both ends of the connection, TCP keeps a state machine to keep track of how the connection is doing. We’ll see how the state machine works in a separate video.

Connection setup
3-way handshake



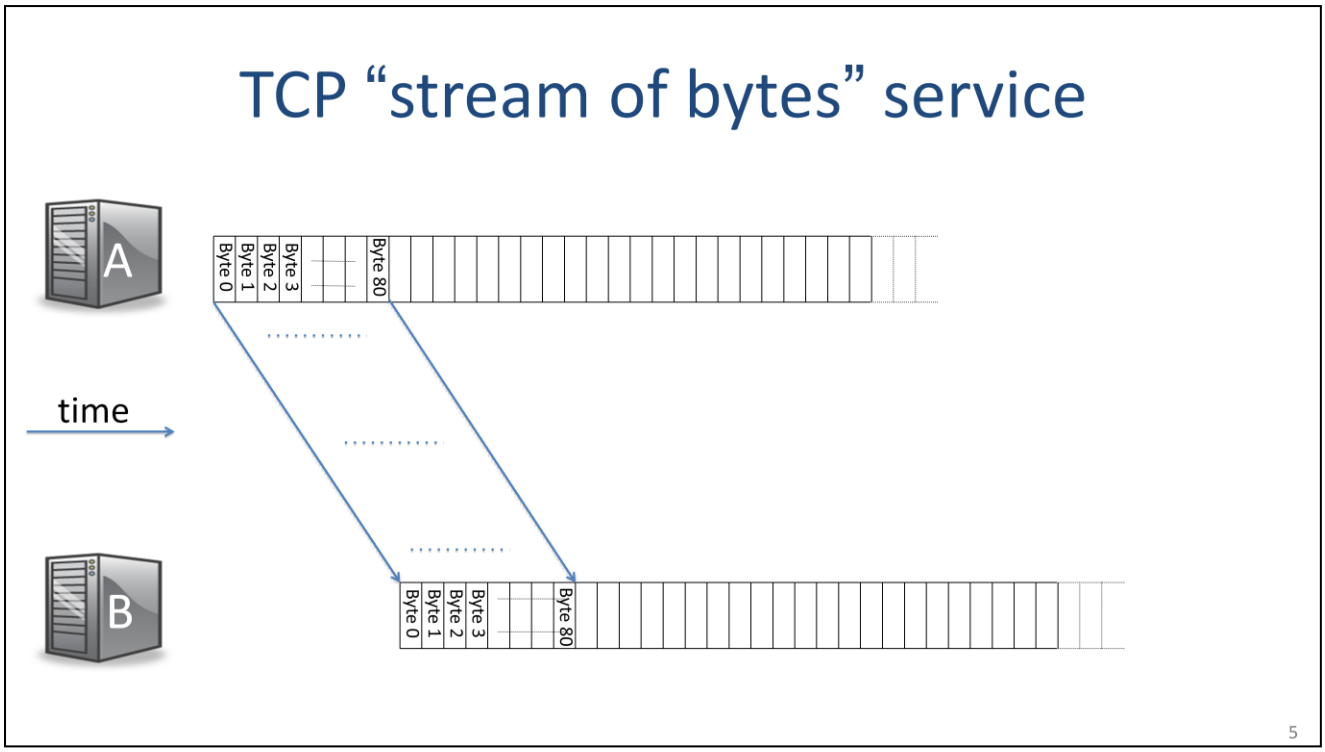
4

The TCP connection is established using a 3-way handshake between hosts A and B.

First of all <click>, Host A sends a message to B indicating that the TCP layer at A wants to establish a connection with the TCP layer at B. The message is called a SYN message, which is short for synchronize, because A also sends along the base number it will use to identify bytes in the byte stream. If it sends "0" then the numbers will start at zero. If it sends "1,000" then they will start at 1,000.

<click> B responds with what we call a SYN + ACK. B signals an ACK because B is acknowledging A's request and agreeing to establish the communication from A to B. The TCP layer at B also sends a SYN back to A to indicate that the TCP layer at B wants to establish a connection with the TCP layer at A. It sends a number too, indicating the starting number for the byte stream.

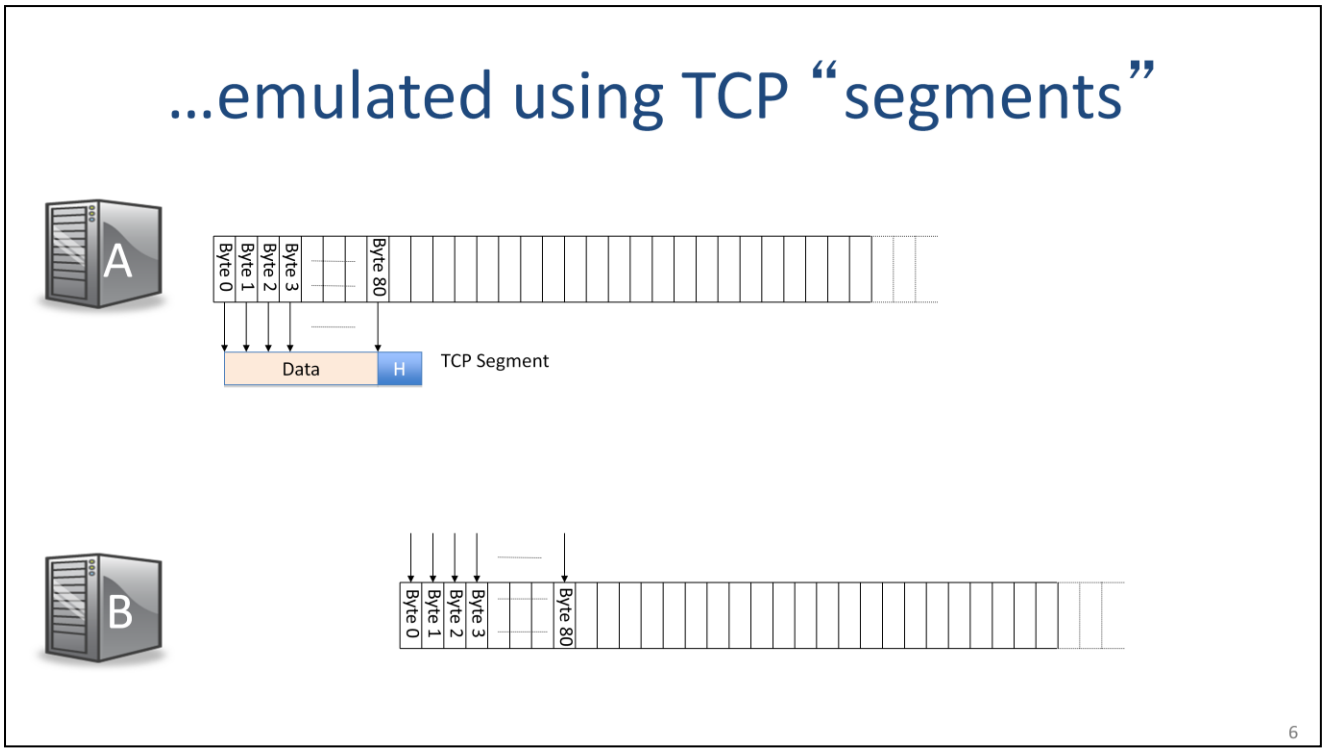
<click> Finally, A responds with an ACK to indicate that it is accepting the request for communication in the reverse direction. The connection is now setup in both directions. They are now ready to start sending data to each other.



The hosts send data to each other as if it is from a continuous stream of bytes.

<click to show time> Assume time is increasing from left to right and the stream of bytes next to A represents the bytes it wants to send to B. The stream of bytes might exist in advance – for example, they are read from an html file describing a static web page. Or it could be a stream being generated on the fly – for example from a video camera. Either way, TCP sees it as a stream of bytes.

<click to show arrows> Data from the application on A is delivered to the application at B. The TCP layers on A and B work together to make sure the stream of bytes is delivered correctly in order to the application at B.



The stream of bytes is delivered by TCP segments. <click twice>

A puts bytes from the stream into a TCP segment, hands it to the IP layer which delivers it to B. <click to move>

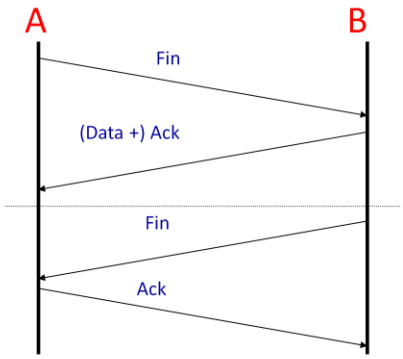
<click to extract> The TCP layer at B extracts the bytes to recreate the byte stream and delivers them to the application at B.

In practice, the TCP segment may need to be transmitted multiple times, in the case a segment is dropped along the way, or if A doesn’t receive an acknowledgment.

The TCP segment can be a small as 1 byte – for example, if you are typing characters in an ssh session, each character is sent one at a time, rather than waiting for the whole segment to fill up.

This isn’t very efficient when we have lots of data to send; so we can fill the TCP segment all the way up to the maximum IP datagram size.

Connection teardown



7

When A and B have finished sending data to each other, they need to close the connection. We say they “teardown” the connection, which means they tell each other they are closing the connection and both ends can clean up the state associated with the state machine.

<click> The TCP layer at Host A can close the connection by sending a FIN message, which is short for FINISH.

<click> Host B acknowledges that A no longer has data to send and stops looking for new data from A. This closes down the data stream from A to B. But B might still have new data to send to A and is not ready to close down the channel from B to A. So the message from B to A carrying the ACK can also carry new data from B to A. B can keep sending new data to A as long as it needs to.

<click> Sometime later B finishes sending data to A, and now sends its own FIN to tell A they can close the connection.

<click> Host A replies by sending an ACK to acknowledge that the connection is now closed. Because both directions have finished, the connection is now fully closed and the state can be safely removed.

The TCP Service Model

Property	Behavior
Stream of bytes	Reliable byte delivery service.
Reliable delivery	1. Acknowledgments indicate correct delivery. 2. Checksums detect corrupted data. 3. Sequence numbers detect missing data. 4. Flow-control prevents overrunning receiver.
In-sequence	Data delivered to application in sequence transmitted.
(Congestion Control	Controls network congestion.)

Here is a table summarizing the services provided by TCP.

The first three are services TCP provides to the application. As we just saw, <click> it provides a reliable stream of bytes between two applications.

<click> It uses four mechanisms to make the communication reliable – in other words, to make sure the data is correctly delivered.

1.When a TCP layer receives data, it sends an acknowledgment back to the sender to let it know the data arrived correctly.

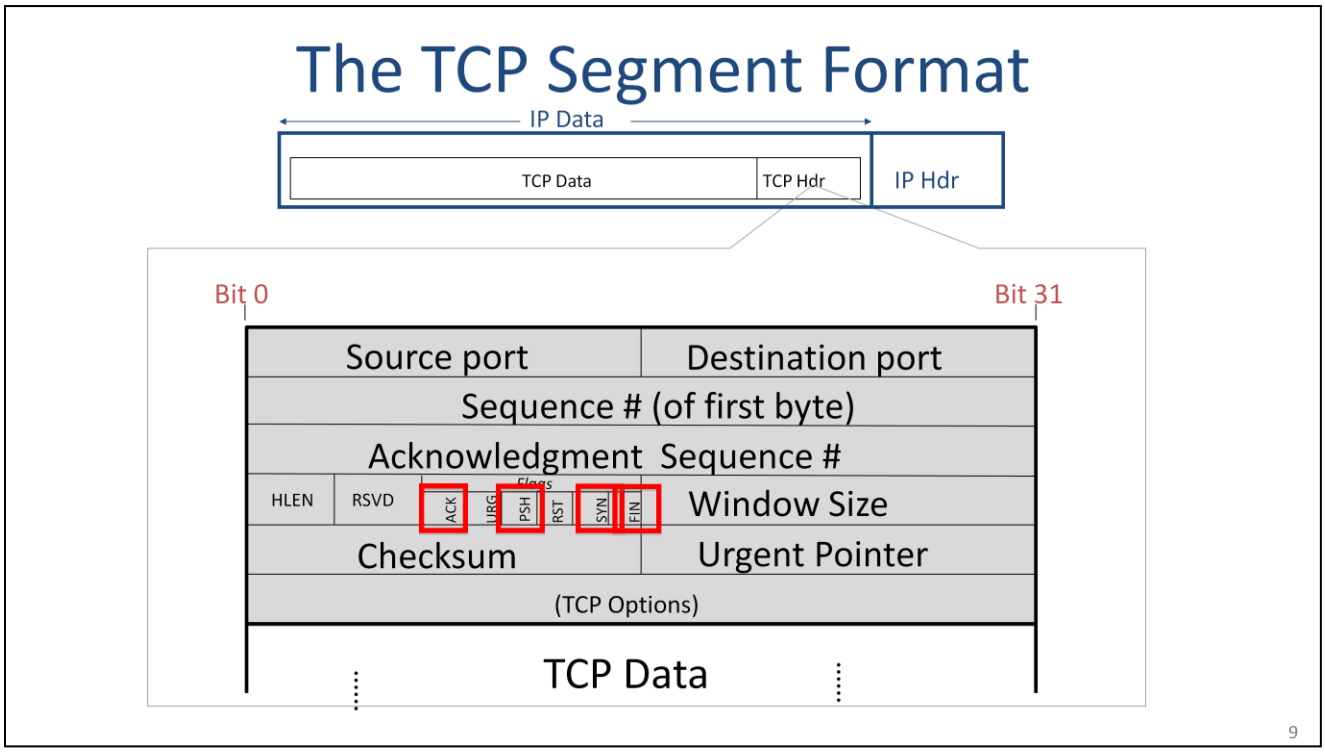
2.Checksums detect corrupted data. The TCP header carries a checksum covering the header and the data inside the segment. The checksum is there to detect if the segment is corrupted along the way, for example by a bit-error on the wire or by a memory fault inside a router.

3.Sequence numbers detect missing data. Every segment's header carries the sequence number - in the stream of bytes – of the first byte in the segment. For example, if the two sides agree that the sequence numbers start at 1,000 then the first segment will have a sequence number of 1,000. If the segment carries 500 bytes of data then the next segment will carry the sequence number 1,500. If a segment gets lost, then the sequence number will be incorrect and the TCP layer knows some data is missing. It is possible it will show up later – perhaps it took a longer path – or it might have gone missing, in which case the sender will need to resend the data.

4.Flow-control prevents overrunning the receiver. If Host A is much faster than Host B then it's possible for Host A to overwhelm Host B by sending data so fast that Host B can't keep up. TCP prevents this from happening using something we call flow-control. In TCP, the receiver keeps telling the sender if it can keep sending; specifically, it tells the sender how much room it has in its buffers to accept new data. If Host B is falling behind, the space drops – possibly all the way to zero. When it has more room, it tells A and it can send more data.

<click> TCP delivers data to the application in the right sequence; in other words, whatever sequence the data was delivered from the application to TCP at host A, this is the same order in which it is sent from TCP to the application at B. If segments arrive out of order, the TCP layer re-sequences them to the correct order, using the sequence number.

<click> Finally, TCP provides a service to the whole network by controlling congestion. TCP tries to divide up the network capacity equally among all the TCP connections using the network. The congestion control mechanisms in TCP are very complicated and we'll devote the whole of Unit 4 to studying congestion control.



The TCP Segment header is much longer and more complicated than, say the IP and Ethernet headers. That is because a TCP connection is reliable – In order to make the communication reliable, the two ends of the connection need to exchange more information so they know which bytes have arrived, which are missing, and the status of the connection.

Here is a quick summary of the most important fields in the TCP header. You don't need to remember the layout of the header, but you should learn what each field does. If you need a reference, I'd recommend Wikipedia or the Kurose and Ross textbook.

<click> The Destination port tells the TCP layer which application the bytes should be delivered to at the other end. When a new connection starts up, the application tells TCP which service to open a connection with. For example, if TCP is carrying web data, it uses port 80, which is the port number for TCP. You'll learn more about port numbers later, but if you are curious, you can look up the well known port numbers at the IANA website. Search for IANA port numbers. You'll find thousands of port numbers defined for different well known services. For example, when we open a connection to an ssh server, we use destination port 22. For smtp (the simple mail transfer protocol) we use port 23. Using a well known port number lets Host B identify the application it should establish the connection with.

<click> The Source port tells the TCP layer at the other end which port it should use to send data back again. In our example, when Host B replies to Host A, it should place Host A's source port number in the destination port field, so that Host A's TCP layer can deliver the data to the correct application. When a new connection starts, the initiator of the connection – in our case Host A – generates a unique source port number, so differentiate the connection from any other connections between Host A and B to the same service.

<click> The Sequence number indicates the position in the byte stream of the first byte in the TCP Data field. For example, if the Initial Sequence number is 1,000 and this is the first segment, then the Sequence number is 1,000. If the segment is 500 bytes long, then the sequence number in the next segment will be 1,500 and so on.

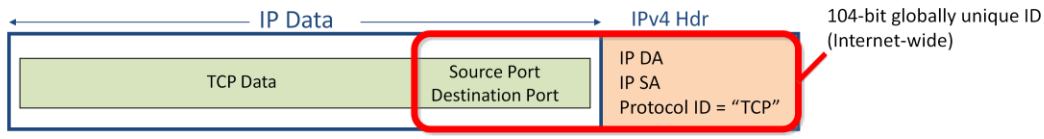
<click> The Acknowledgment sequence number tells the other end which byte we are expecting next. It also says that we have successfully received every byte up until the one before this byte number. For example, if the Acknowledgment Sequence number is 751, it means we have received every byte up to and including byte 750. Notice that there are sequence numbers for both directions in every segment. This way, TCP piggybacks acknowledgments on the data segments traveling in the other direction.

<click> The 16 bit checksum is calculated over the entire header and data, and helps the receiver detect corrupt data. For example, bit errors on the wire, or a faulty memory in a router. You'll learn more about error detection and checksums in a later video.

<click> The Header Length field tells us how long the TCP header is. <click> The TCP Options fields are, well, optional. They carry extra, new header fields that were thought of and added after the TCP standard was created. The Header Length field tells us how many option fields are present. Usually there are none.

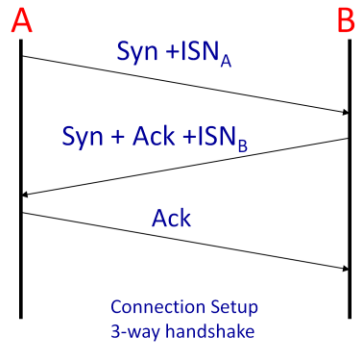
Finally, there are a bunch of Flags used to signal information from one end of the connection to the other. <click> The ACK flag tells us that the Acknowledgement sequence number is valid and we are acknowledging all of the data up until this point. <click> The SYN flag tells us that we are signalling a synchronize, which is part of the 3way handshake to set up the connection. <click> And the FIN flag signals the closing of one direction of the connection. <click> Finally, the PSH flag tells the TCP layer at the other end to deliver the data immediately upon arrival, rather than wait for more data. This is useful for short segments carrying time critical data, such as a key stroke. We don't want the TCP layer to wait to accumulate many keystrokes before delivering them to the application.

The Unique ID of a TCP connection



1. Host A increments source port for every new connection

2. TCP picks ISN to avoid overlap with previous connection with same ID.



10

A TCP connection is uniquely identified by five pieces of information in the TCP and IP headers.

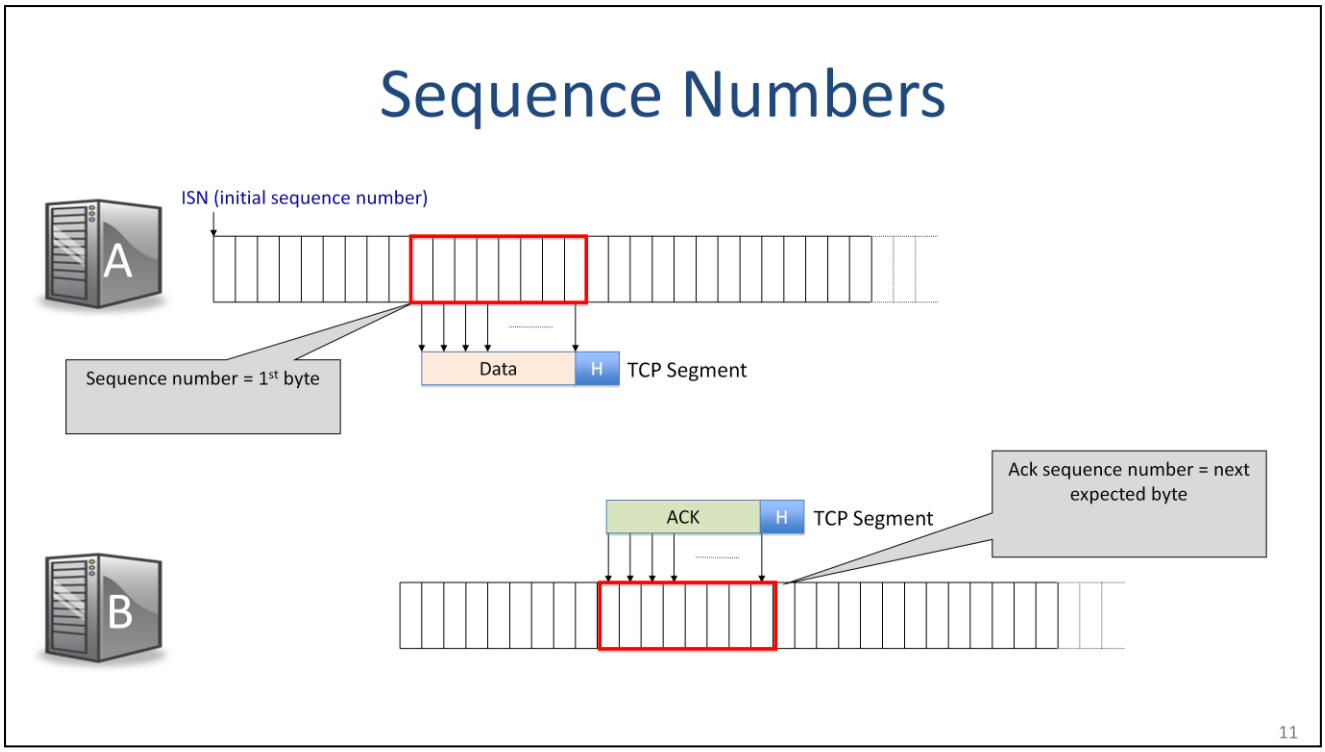
The IP source and destination addresses uniquely identify the end points, and the IP Protocol ID for TCP tells us the connection is TCP.

The TCP source and destination ports identify they application processes on the end hosts. Together, at any instant, all 5 fields uniquely identify the TCP connection Internet-wide.

Now, the unique ID only holds if a few things hold. First, we need to make sure Host A – the initiator of the connection - picks a unique source port ID. We need to make sure it doesn't accidentally pick the same source port number it is already using with another connection to the same service on Host B. Host A uses a simple method to minimize the chances: <click> It increments the source port number for every new connection. The field is 16bits, so it takes 64k new connections before the field wraps round.

There is also a very slight danger that if Host A suddenly creates a lot of new connections to Host B it might still wrap around and try to create two connections with the same global ID. If this happened, the bytes from one connection might become confused with the bytes from another connection. This could happen, for example, if a TCP segment somehow lived for a long time in the network, stuck inside a router buffer or circulating in a temporary loop.

<click> To reduce the chances of confusion, the TCP connections initialize with a random initial sequence number to refer to bytes in the byte stream. While not totally fool proof, it does reduce the chances of confusion. When Host A initiates the connection to B, it includes the initial sequence number if will use in the stream of bytes from A to B. When B replies and initiates the connection from B to A, it supplies its own initial sequence number for the stream of bytes from B to A.

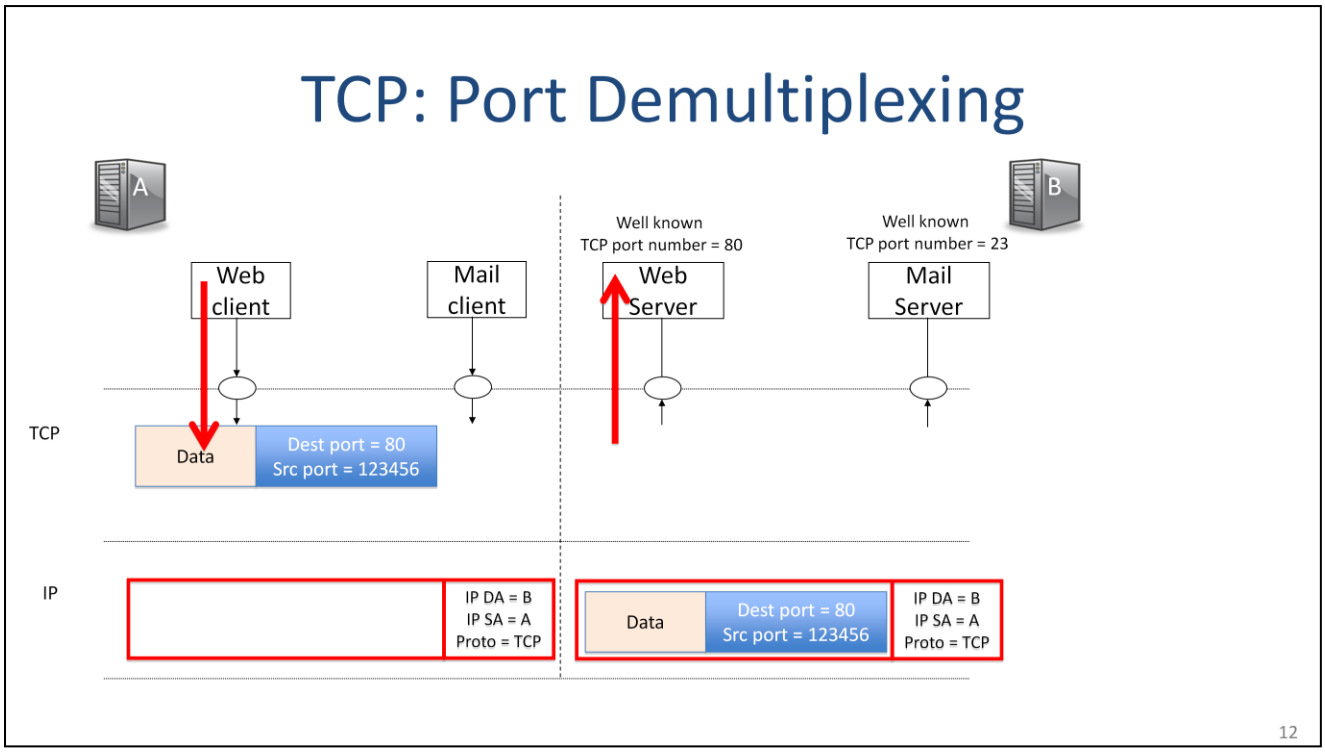


And so to summarize how sequence numbers work…..

<click> The sequence number in a segment from A to B includes the sequence number of the first byte, offset by the initial sequence number.

<click>

<click> The acknowledgment sequence number in the segment from B back to A tells us which byte B is expecting next, offset by A's initial sequence number.



Let's summarize how TCP port numbers work….

Imagine that Host B on the right offers two services: A Web Server and a Mail Server.

When the Web client – for example a Chrome browser on Host A – wants to request a page from the Web server on B, it sends the data <click> to TCP.

We'll assume TCP has already established a connection with B, so now it just needs to send the data. It creates a segment and uses destination port 80 to tell B it is requesting the data be sent to the web server.

Host A uses a locally generated source port number for B to use when sending data and acknowledgments back again.

<click> As usual, the TCP segment is encapsulated into an IP datagram and sent to B. The IP + TCP headers carry the unique ID of the TCP connection.

<click> When the IP datagram arrives at B, the TCP segment is removed. The TCP layer sees that the segment is for port 80 and sends the data to the web server <click>

TCP Sliding Window

You will learn about other TCP features in upcoming videos:

- Window-based flow control
- Retransmission and timeouts
- Congestion control

13

You'll learn about other features of TCP in upcoming videos.

You'll learn about window-based flow control and to stop us from overwhelming the receiver.
You'll learn about retransmissions and timeouts and different methods to accomplish it.
And you'll learn about Congestion Control in Unit 4.

Summary

TCP provides in-order, reliable delivery of a stream of bytes between application processes.

In summary, TCP provides in-order, reliable delivery of a stream of bytes between application processes.

<The End>