Transport Layer

The transport layer is used for:

- Demultiplexing: decide which application should receive the data.

  This is accomplished using port numbers, which are in the transport header and designate the application (Web server is port 80 for example).

  UDP (User Datagram Protocol) does only this.

  A unit in the transport layer is called a segment.

- If segments get lost, retransmission has to be done
- If the routes change, we should reorder the segments to get the original order the segments were sent in (multiple segments may constitute a single file)

TCP does the above two.

↳ Transmission Control Protocol

→ It does not send segment (n+1) until segment (n) has been received

The issue is that routers do not cooperate to ensure reliable data transfer. We should be able to infer packet loss!

If we want something between UDP and TCP, we should use UDP and build our own protocol on top of it.

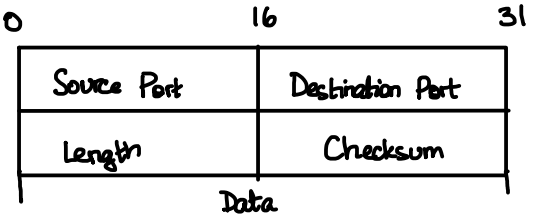
The mechanism via which we drop packets at a router if the queue becomes full is called a drop-tail mechanism. (queue becomes full = congestion)

TCP is able to infer that congestion has occurred and does congestion control: the segment sending rate is reduced. Enough of the input links back off to allow the queue to clear up.

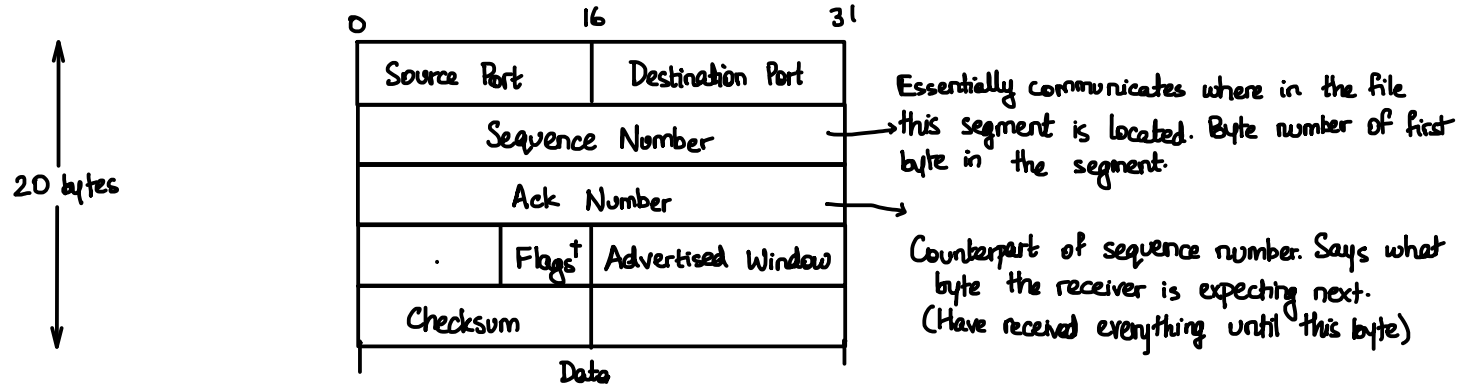
TCP also does flow control, which is just congestion control at the destination. This is different because the source and destination can communicate. The destination may take time to process information, and may not have read all the information from the transport layer yet (there is a buffer).

Van Jacobson and Sally Floyd did a lot of work on TCP in the 80s.

The UDP header is just



The TCP header on the other hand, looks like



† SYN, FIN, RESET, PUSH, URG, ACK (6 bits)

- SYN: Open Connection
- FIN: Close connection
- RESET: Reset connection
- ACK: Ack number is present in segment (shows validity)

The protocol field in the IP layer says whether the transport layer is using TCP (6) or UDP (17).

Lecture 23 TCP: Opening and Closing a Connection

Recall that there is a sequence number and an ACK number in TCP.
This allows to send (data + ACK) together.

How is a connection established?
Suppose there is a web server, which is a passive participant, listening on some port.
Then, there is an active client that is trying to open a connection.

→ First, a SYN flagged packet is sent. Suppose it has a seq. no. X.
→ The server responds with a (SYN+ACK) flagged packet with ack. no. X+1. Suppose it has seq. no. Y.
→ The client responds with ACK Y+1.

This is called a three-way handshake.
While SYN/FIN packets don't have data, they are considered to have one byte.
Usually, the value X is randomly selected.
ACK z acknowledges that we have received everything from start to z-1.

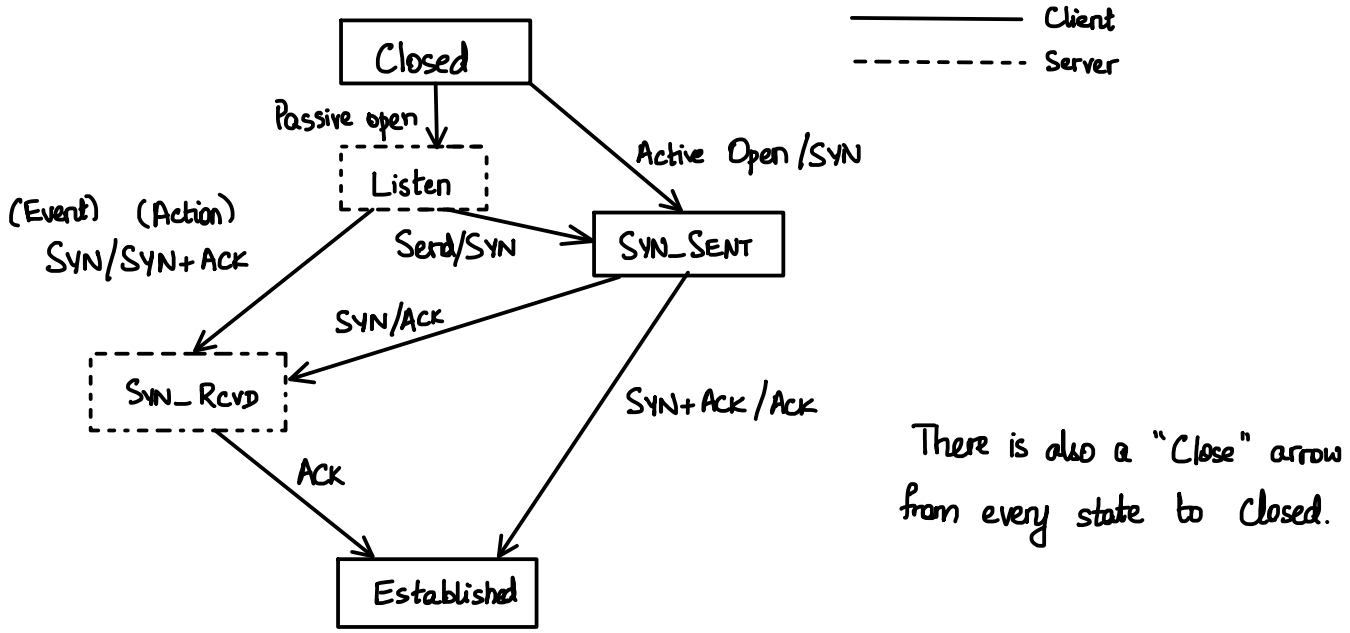
Why do we want a random X and Y?

Suppose we want to transmit two files of size N and M and we start with seq. no. at 0. If some segment from the first packet arrives late, then it could cause problems. If the port numbers are the same, we could interpret it to belong to the second file, which we do not want.

We choose the initial seq. no. randomly in the hopes that the sequence number spaces do not overlap.

If the file is large enough, it is possible to wrap around from $(2^{32}-1)$ to 0.

Let us draw the state transition diagram.



There is also a "Close" arrow from every state to Closed.

The above is complicated because we do not know what the other side is doing. We have to account for them sending a SYN at the same time as us. The actual diagram is in fact far more complicated.

It is better to actively terminate a connection because this tells the other side that they can clear their buffer and any state-related data.

To do this, we use a FIN-flagged packet.

1. Half-close: Closed by one end at a time. The other end can continue to send some data, but it need not receive a response. After it is done, it sends a FIN flag as well, and receives an ACK.

2. 3-Way Handshake Termination: Both sides close at the same time.
The client sends a FIN, and the server responds with (FIN + ACK) together. The client responds with an ACK (to the FIN).

Lecture 24 TCP Timeout

When the sender sends a packet, they expect a response by a certain time
If we do not receive an ACK within the timeout, we retransmit.

What do we set the timeout to?
A couple of issues are.

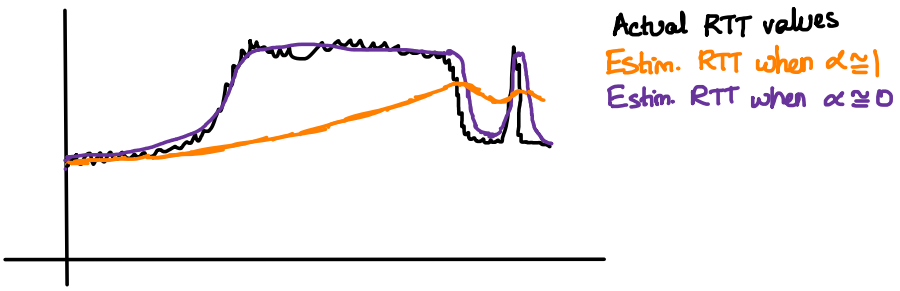
1. The RTT can vary quite a lot depending on the network path.
2. The RTT can vary across time as well due to the queue lengths. (even for the same path)
3. Packet loss can occur. When we receive an ACK, could it be due to a previous transmission of the same packet?

We would like to get an average RTT.
↳ in what sense?

Let the "sample RTT" be the most recent RTT.
We take the new estimated RTT as a weighted average

$$\alpha \times \text{Estim. RTT} + (1-\alpha) \times \text{Sample RTT.}$$

For example,



Both the extreme values of α are not very good.

The old algorithm uses

$$\text{Timeout} = 2 \times \text{Estimated RTT}$$

In the new algorithm, we try to take the variation of RTT into account

Suppose we have a Gaussian distribution $N(\mu, \sigma)$. If the RTT follows this distribution, then $\mu + 3\sigma$ serves as a good value for the timeout.

Measure Estim. RTT (mean) as before.

Let Diff = Sample RTT − Estim. RTT

Since standard deviation is tricky due to the square root, we use the mean deviation instead.

That is, we let

$$Dev = (1-\beta) \times Dev + \beta \times |Diff|$$

The timeout is then set as

$$\mu \times \text{Estim. RTT} + \phi \times Dev.$$

(μ: usually 1; ϕ: usually 4)

We usually take $\alpha = 7/8$ and $\beta = 1/4$.

The initial timeout value is quite large.

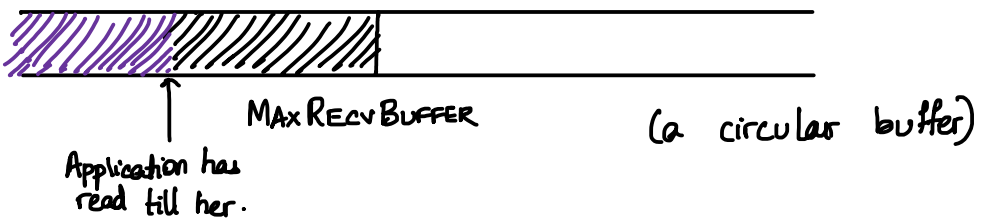
What about the packet loss issue?

We do <u>not</u> use the RTT measurement for retransmitted packets.

Now, let us move on to congestion control.

Initially, because the internet was for military purposes, TCP was kept to not have state information (it is localized to the end nodes).

Consider the receiver buffer.



The left and right sections are free. If congestion is at the receiver, we can send this buffer information.

What if congestion is at an intermediate router? There is no direct communication so TCP must infer this.

(Rarely, we use Explicit Congestion Notification (ECN))

How does TCP infer this?

Flow Control deals with congestion at the receiver

The advertisement window field in the TCP header is equal to the space left in the receiver buffer.

The sender has a window, which is the maximum amount of data (in bytes) which is outstanding – sent but not ACKed.

If we send window many bytes, the data rate is $\simeq \frac{\text{window}}{\text{RTT}}$

First, the window must be less than the advertisement window.

(to ensure that we take care of flow control)

We calculate the congestion window and set

$$\text{window} = \min\{\underbrace{\text{congestion window}}_{\text{result of inference about congestion at routers}}, \text{adv. window}\}$$

This is approximately equal to

$$\underbrace{\text{DataRate} \times \text{RTT}}_{\text{(maximum possible)}} \rightarrow \text{Delay-Bandwidth Product}$$

It remains to choose the congestion window