Transport Layer Part IV

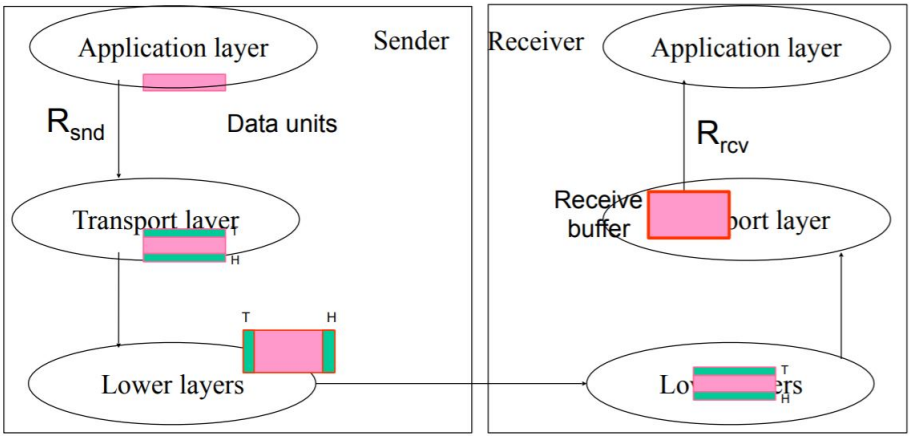
ct4wa

March 5th 2020

3.5 Connection-oriented transport: TCP

Flow Control

Flow control problem



Mini-review
Network layer: IP address in header, CRC in tail
Data link layer: MAC address in header, CRC in tail

When sending data, there is a Sender side and a Receiver side. In the image above, we can see

- the Sender sends data from the Application layer to the Transport layer at a rate R_{snd}
- On the Receiver side, the Application later receives data from the Transport layer at the rate R_{rcv}. This rate depends on how many processes are running on the machine.
- The Transport layer on the Receiver side has a receive buffer. This is where data is reordered if needed before sent to the Application layer.

With this buffer, there is a potential problem. If data is added to the buffer at a faster rate than it is consumed (or sent to the Application layer), then the buffer can get overwhelmed and end up dropping data segments. **We want data to be added to the buffer at a slower rate than consumed.**

This can be solved with flow control. Flow control is when the receiver controls the sender, so the sender won't overflow receiver's buffer by transmitting too much, too fast.

Stop and Wait Flow Control

- Sender has to stop after sending each frame and wait for an ACK
- ACK is sent back from the receiver after the higher layer depletes the receive buffer holding the single frame's payload
- Therefore the sender cannot overrun the receive buffer

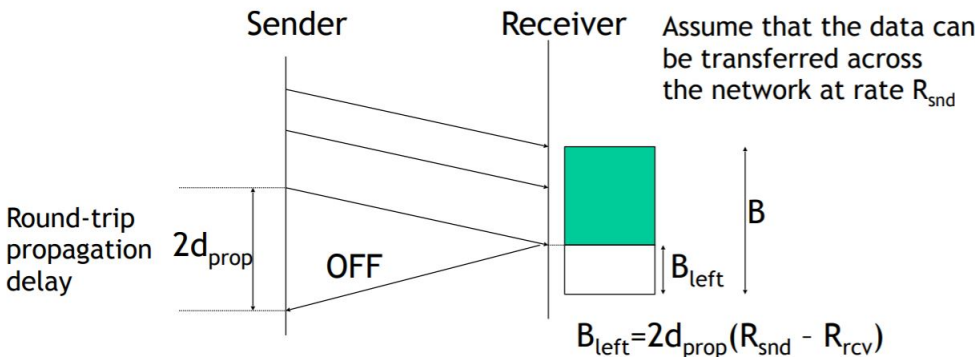
How can we optimize this?

- Stick number of things in flight based on window size
- Leave space in buffer on other end the size of the window

When would the frame advance?

- Wait for acknowledgements

ON/OFF flow control



How does it work?

- Leave space in buffer
- Know how much space is left and tell sender to stop sending when window size is reached

Challenge: Off signal could be lost (time of sending the off signal is the propagation rate)
How quickly receiver is filled up (factors)

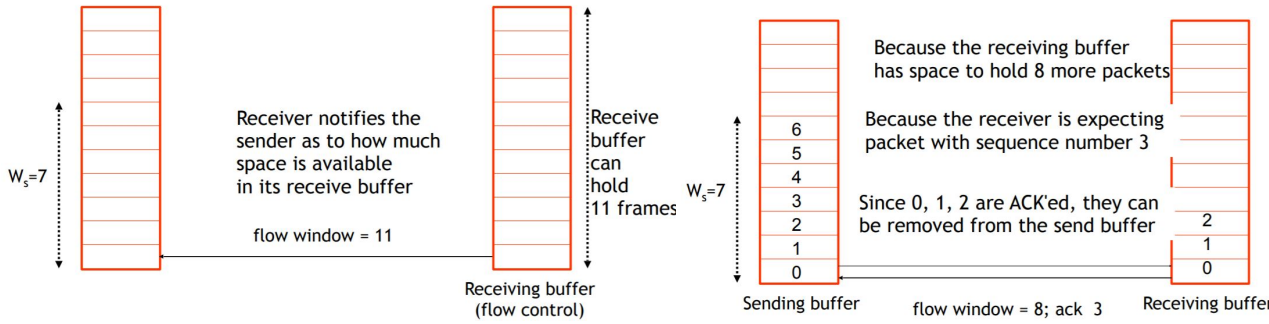
- Size of buffer
- Rate of removal
- Application layer: rate of send

When should the OFF signal be sent?

- It depends on the difference between rate of removal and rate of send. This determines how quickly the buffer fills up.
  - $B_{left} = 2d_{prop}(R_{snd} - R_{rcv})$
  - R_{snd} = rate at which the higher layer passes frames to the lower layer at the sender. But how do we know the rate of the sender?
  - R_{rsv} = rate at which the higher layer accepts frames from the lower layer at the receiver
- Round-trip propagation delay = $2d_{prop}$, meaning by the time the Sender receives the OFF signal, it will be sending another segment of data.

Sliding Window Flow Control

- Receiver has a receive buffer
- Receiver sends reports of available space in this buffer to the sender
- Sender uses this number to determine the maximum number of frames it can have outstanding (i.e., unacknowledged)



Flow window is the left-over space in the receiver's buffer. It is also called "advertised window" or "receiver's window"
Note: The frame numbers wrap around once they reach the window size number. E.g. once 0,1,2 are removed from the sending buffer, 7, 0, 1, will be added above the 6.

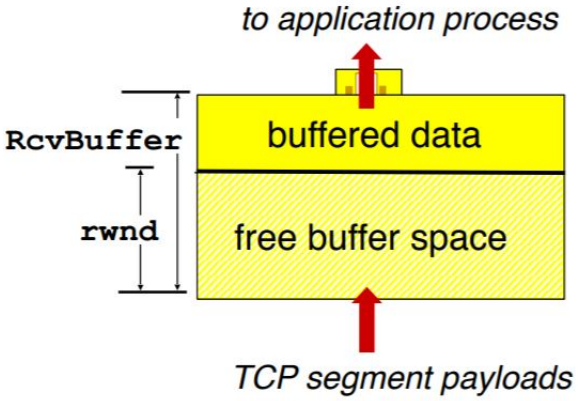
At any instant in time, what is the maximum number of frames that the sender is permitted to send? It is a minimum of sending window W_s, and the flow window indicated by the receiver
Three steps:

1. Find X = min (W_s, flow window)
2. Find Y = number of outstanding frames (already sent but unacknowledged)
3. Maximum number of frames that the sender is permitted to send = X - Y

Result: if X-Y frames are sent, then the number of outstanding frames will become X (which is the maximum limit for number of outstanding frames)

TCP Flow Control

How does the sender know how much space is left in the receiver's buffer (aka the flow window)? The receiver "advertises" free buffer space by including a rwnd value in TCP header of receiver-to-sender segments. The sender limits amount of unacked ("in-flight") data to receiver's rwnd value, guaranteeing it will not overflow the receiver's buffer.



receiver-side buffering

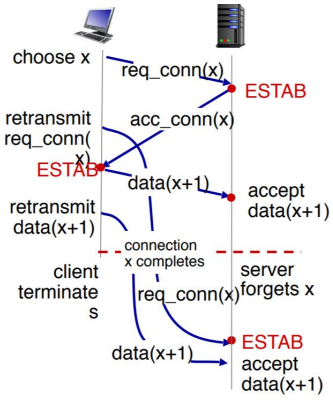
Connection Management

Before exchanging data, sender and receiver agree to establish connection (each knowing the other willing to establish connection) and agree on connection parameters. This is a **handshake**.

Agreeing to establish a connection

Issues with a 2-way handshake in a network:

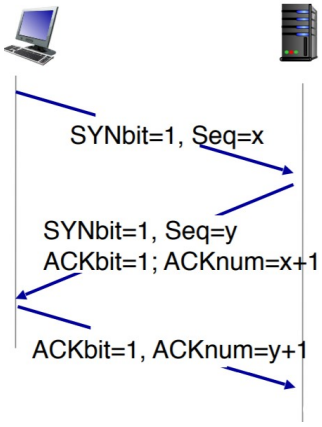
- Variable delays
- Re-transmitted messages due to loss
- Message reordering



A 2-way handshake can fail and create a half open connection. In simple terms, this happens when the client sends a hello, the server sends an ACK back, but the client doesn't receive the ACK in time, and so it sends another hello to the server. When the client eventually receives the initial ACK, it will send its data to the server. When the Server receives the second hello after the timer has expired, it has forgotten the client, but it will accept the data when the client sends it again since it has established it on the server side.

Solution: We need to acknowledge that the ACK was received

TCP 3-way handshake



In a 3-way handshake, the client sends the server a SYN message. When the client receives the message, it sends a SYN and an ACK back to the client. The client then sends an ACK to the server, and the server establishes the connection when it receives the ACK. This ensures that there will not be a half-open connection. In the figure, the bits and numbers of the messages can be seen.

TCP: Closing a connection

The client and server each close their side of connection by sending a TCP segment with FIN bit = 1. The other side responds to the received FIN with an ACK, which can be combined with that side's own FIN. The client and server can send FIN messages one at a time, or simultaneous FIN exchanges can be handled.

3.6 Principles of Congestion Control

Network congestion is too many sources sending too much data too fast for network to handle. It is different from flow control. Congestion can lead to:

- lost packets (buffer overflow at routers)
- long delays (queuing in router buffers)

Causes/costs of congestion

- Two senders, two receivers
- One router, infinite buffers
- Output link capacity: R
- No re-transmission