Comp3331 Week 5 Lecture 2

Transport Layer TCP

# 3.5 connection-oriented transport: TCP

Table

Description automatically generatedWhy? Ensure the sender doesn't overrun the receiver.

**Receiver** controls sender, so sender won’t overflow receiver’s buffer by transmitting too much, too fast

the receiver uses receiver window in the TCP header to tell sender the current size of the free buffer window size

what if the rwnd is filled up? sender would stop sending data. sender keeps sending TCP segments **with one data** byte to the receiver (minimum data). These segments are dropped but ACK by the receiver with a zero-window size. Eventually when the buffer is freed up, data transfer can begin again.

## 3-way handshake

Diagram

Description automatically generated

Server initially goes into listening state -> when client initiates the socket, we move into SYNSENT state, it creates and sends SYNBIT, which consumes 1 sequence number. Server responds back with SYNACK and SYNBIT, which doesn’t contain any data but consumes 1 sequence number. This is what is called SYNRCVD state. The client receives SYNACK indicates server is live, send ACK for SYNACK.

Diagram

Description automatically generated

**What if the SYN packet gets lost?** Sender sets a timer and waits for the SYN-ACK

**what's the value of our timer?** Most implementation use between 1 to 3 seconds

# Team down process

Diagram

Description automatically generated

Suppose that the client application decides it wants to close the connection. (Note that the server could also choose to close the connection.) This causes the client TCP to send a TCP segment with the FIN bit set to 1 and to enter the FIN\_WAIT\_1 state. While in the FIN\_WAIT\_1 state, the client TCP waits for a TCP segment from the server with an acknowledgment. When it receives this segment, the client TCP enters the FIN\_WAIT\_2 state. While in the FIN\_WAIT\_2 state, the client waits for another segment from the server with the FIN bit set to 1; after receiving this segment, the client TCP acknowledges the server’s segment and enters the TIME\_WAIT state.

The TIME\_WAIT state lets the TCP client resend the final acknowledgment in case the ACK is lost. The time spent in the TIME\_WAIT state is implementation-dependent, but typical values are 30 seconds, 1 minute, and 2 minutes. We need a timer in the end to make sure the connection close properly. After the wait, the connection formally closes and all resources on the client side (including port numbers) are released.

## RESET (if something goes wrong)

Diagram

Description automatically generated

A sends a RESET (RST) to B E.g., because application process on A crashed

1. B does not ack the RST
2. Thus, RST is not delivered reliably
3. And: any data in flight is lost
4. But: if B sends anything more, will elicit another RST

# 3.6 principles of congestion control

Note the difference between congestion control and flow control, here we are talking about the congestion control of the network, if there is too much data in the network without congestion control, the entire network will be slowed down.

Diagram

Description automatically generated

Increased traffic from S2 causes Link A to become congested. S1 starts retransmitting.

Basically, congestion can back propagate the entire network, as illustrated in the graph above. Although S1 is not congested initially, S2’s traffic starts retransmitting on top of what is was sending. This increases traffic in link A to become congested, then congestion manifest and S1 becomes congested, it will also start retransmitting, and this will manifest to the entire network very quickly. After a while, we can go into the yellow zones below and the congestion collapse happens. The entire network will go down.

Diagram

Description automatically generated

## TCP congestion control

1. duplicated ACKs, fast retransmission
2. timeout: not enough dup ACKs.

Recall from lecture 1 week 5

Diagram

Description automatically generated

**how sender detects traffic goes back to normal**

1. Upon receipt of ACK (of new data): increase rate
2. Upon detection of loss: decrease rate

# Three windows

Graphical user interface, text

Description automatically generated with medium confidence

## Slow Start and AIMD (very important for the exam)

**Slow Start Phase**

when connection begins, initially cwnd = 1 MSS, then double cwnd every RTT (full ACKs), cwnd += 1 for each ACK

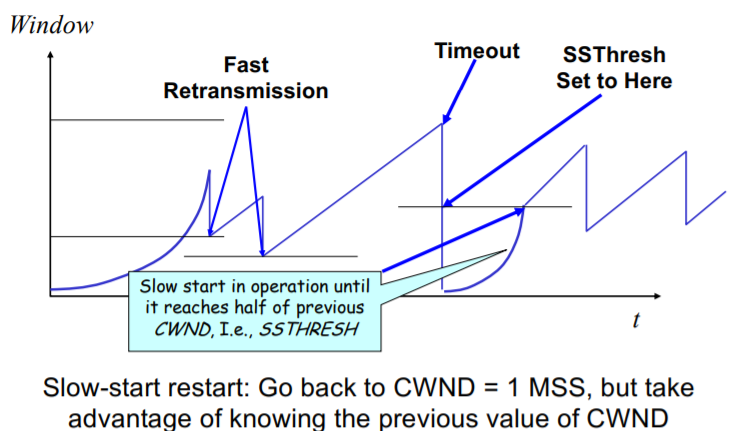
**Additive Increase Multiplicative Decrease” (AIMD)**

sender increases transmission rate (window size), probing for usable bandwidth, until another congestion event occurs. For each successful RTT (all ACKS)

cwnd = cwnd +1

Diagram

Description automatically generated



**TCP-Tahoe**

cwnd =1 on triple dup ACK & timeout

**TCP-Reno**

cwnd =1 on timeout, cwnd = cwnd/2 on triple dup ACK