

Transport Layer (IV)

TCP Congestion Control

- Receiver congestion control

- **Window Size field** - explicitly reported by the receiver
- TCP Window Scale Option
- Calculated window size = Window size value x Window size scaling factor

```

► Flags: 0x018 (PSH, ACK)
Window size value: 2061
[Calculated window size: 131904]
[Window size scaling factor: 64]

```

- Window size value: raw window size value read directly from TCP header
- Calculated window size: calculated based on whether the scaling is allowed or not. If scaling is not allowed, scaling factor = 1. (The receiver window size may increment to calculated window size.)

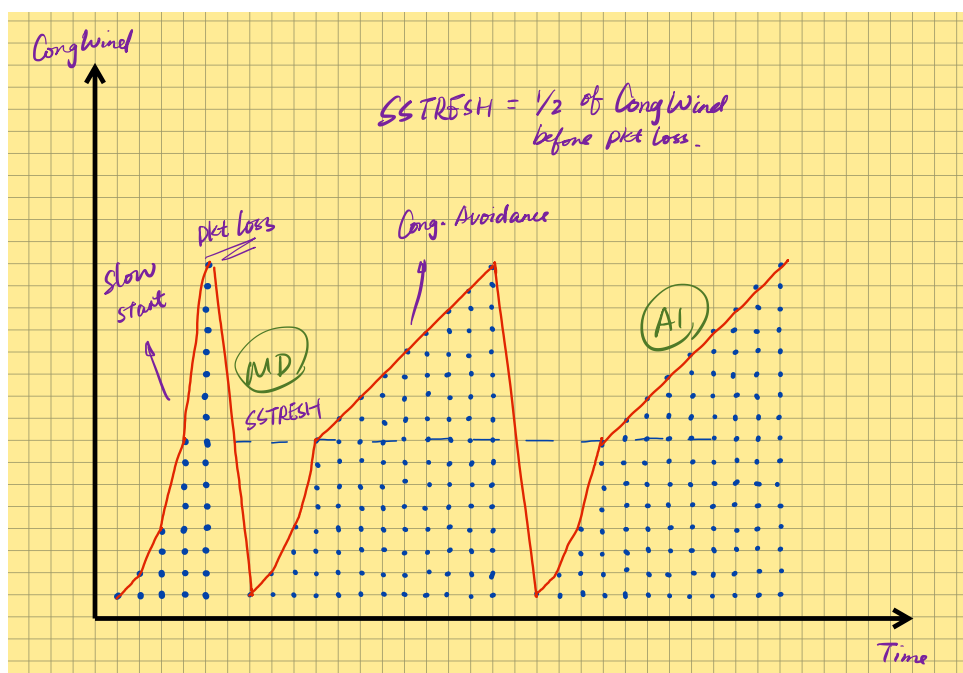
- Network congestion control

- **Retransmission timeout** - based on observed RTT
 - It's not a static value.
 - The timeout value will be exponentially increment after a timeout to alleviate the network congestion.
 - Why this makes sense?
 - The timer expiration is most likely caused by congestion in the network, that is, too many packets arriving at one/more router queues in the path between the source and destination, causing packets to be dropped and/or long queuing delays.
 - In times of congestion, if the sources continue to retransmit packets persistently, the congestion may get worse.
 - Instead, TCP acts more politely, with each sender retransmitting after longer and longer intervals.
- **Transmission window** - based on detected packet loss
 - Source observes **RTT** and **packet loss** and **adjusts transmission rate** according to its estimate of the congestion state of the network
 - Transmission window provides a **way to control the transmission rate**
- Transmission **window size is proportional** to the **maximum transmission rate**

TransWind = min(**RecvWind**, **CongWind**)

- **TransWind** - used in transmission

- **RecvWind** - from Window Size field of the TCP header
- **CongWind** - transmitter's estimate of how many unacknowledged packets can be pushed onto the network without causing congestion
 - RFC 2001 defines the algorithm, **Additive Increase Multiplicative Decrease** (AIMD), to manage CongWind — better safe than sorry
 - **Additive increase**: increase **CongWind** by 1 message every RTT until loss detected
 - **Multiplicative decrease**: cut **CongWind** in half after loss
- **Slow Start** - initial growth of **CongWind**
 - Initially, the **CongWind** is 1 packet. After the receiver receives an acknowledgement, it increases the **CongWind** by 1 for each acknowledgement.
 - Once a packet loss is detected (timeout), the sender reset the **CongWind** to 1 and restart the slow start process.
 - The sender also set the **ssthresh** (slow start threshold) to **CongWind**/2, the value of **CongWind** is the value when the loss was last detected.
 - Once the **CongWind** reach at the **ssthresh**, it enters into Congestion Avoidance mode.
- **Congestion Avoidance** - AIMD-based “search” for optimal rate
 - In entry to the Congestion Avoidance mode, the congestion could be just around the corner!
 - TCP increase **CongWind** value by 1 packet every RTT.
 - Once the timeout occur (packet loss), Congestion Avoidance enters into Slow Start state.



UDP

- UDP does **not** have **built-in congestion control**.
- Many **multimedia applications** (Skype use both TCP and UDP, but prefers UDP for voice transmission,) often **run over UDP**.
- When running over **UDP**, applications can **pump** their **audio and video** into network at a **constant rate** and occasional loose packets, rather than reduce their rates to “fair” levels at times of congestion and not lose any packet.
- From the **perspective of TCP**, the **multimedia applications** running **over UDP** are **not** being **fair** - they do not cooperate with other connections or adjust their transmission rates appropriately.
- Because TCP congestion control will decrease its transmission rate, while UDP sources need not, it is possible for **UDP** sources to **crowd out TCP traffic**.
- Google already uses QUIC in its Chrome in early 2010s, and any connection to Google’s umbrella is established uses QUIC. QUIC = TCP + TLS + HTTP/2 implemented on UDP.
- User Datagram Protocol (RFC 768)
 - A wrapper protocol for IP to add port numbers
 - 8 bytes

Source Port	Destination Port
Length	Checksum

Disclaimer: Notes adapted from the textbook and online resources.