TCP

- End-to-end protocol.
- Reliable byte stream.
- Connection oriented.
- Expected to operate under widely different round-trip times (RTT) and bandwidths.

TCP Functions

- Flow control.
- Congestion control.

Implemented using a variable size sliding window algorithm.

What is the difference between the two?

TCP Header Flags

When set, the flags indicate the following:

SYN, FIN: used for connection establishment and termination.

ACK: acknowledgment field valid.

URG: segment contains urgent data.

PUSH: sender invoked push operation.

RESET: receiver wishes to abort connection.

TCP Flow Control

Receiver:

$$LastByteRcvd - LastByteRead \leq MaxRcvBuffer$$

$$AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - LastByteRead)$$

Sender:

$$LastByteSent - LastByteAcked \leq AdvertisedWindow$$

$$EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)$$

To prevent overflow on sender side:

$$LastByteWritten - LastByteAcked \leq MaxSendBuffer$$

TCP Flow Control (contd.)

If $AdvertisedWindow == 0$ in ACK: sender cannot send any data.

How does sender know if receiver can accept more data?

- Sender sends 1-byte segments to trigger response from receiver.
- Consistent with *smart sender/dumb receiver* principle.

TCP RTT Calculation

- Original Algoirithm
  - $Estimated\,RTT = \alpha \times Estimated\,RTT + (1-\alpha) \times Sample\,RTT.$
  - $Timeout = 2 \times Estimated\,RTT.$
- Karn-Partridge Algorithm
  - Stop sampling RTT on retransmissions.
  - Exponential backoff (double timeout duration).
- Jacobson-Karels Algorithm
  - Timeout depends on $Estimated\,RTT$ and its variance.

TCP Congestion Control

- Introduced about eight years after TCP/IP deployed.
- Self-clocking: paced by ACKs.
- Additive increase and multiplicative decrease.
- Slow start.
- Fast retransmit and fast recovery.

Additive Increase and Multiplicative Decrease

$$MaxWindow = MIN(CongestionWindow, AdvertisedWindow)$$
$$EffectiveWindow = MaxWindow - (LastByteSent - LastByteAcked)$$

- *Congestion Window* halved on each timeout, but never below Max Segment Size (MSS).

- Increase *Congestion Window* by one MSS when a *Congestion Window* worth of data is acknowledged.

Slow Start

- Additive increase is too slow in the beginning.
- Expand *Congestion Window* exponentially in the beginning:
  - Increase *Congestion Window* by the amount of acknowledged data.
  - *Congestion Window* doubled every RTT.
- Slow start also used on timeouts following packet loss:
  - Drop *Congestion Window* to one MSS.
  - Slow start until half of previous *Congestion Window*.
  - Additive increase after that.

Fast Retransmit and Fast Recovery

Why wait until timeout to retransmit?

Is it possible to detect loss of packet(s)?

- Acknowledgment field always indicates next expected byte number.
- If three *duplicate acknowledgments* received:
  - High probability of packet loss.
  - *Fast Retransmission* of lost packet.
  - *Fast Recovery:* halve *Congestion Window* and resume additive increase.

Congestion Avoidance

Proposals:

- Increased functionality in routers to assist end nodes:
  - DECbit.
  - Random Early Detection (RED): two thresholds.
- Implemented only by end nodes:
  - Source-based congestion avoidance.

Fair Queuing

- *Flows:* sequence of packets with same source-destination pair and following the same route.

What if an application does not use TCP and floods its packets in the network?

- Routers need to ensure fair access to bandwidth.

- Solution should be work conserving.

Fair Queuing (contd.)

At a router, for each flow:

- A_i = arrival time of the i^{th} packet.
- F_{i-1} = departure time of $i-1^{st}$ packet.
- P_i = time needed to transmit the i^{th} packet.
- $F_i = max(F_{i-1}, A_i) + P_i$.

Next packet to be transmitted by the router is the one with the lowest F value.