

Transmission Control Protocol

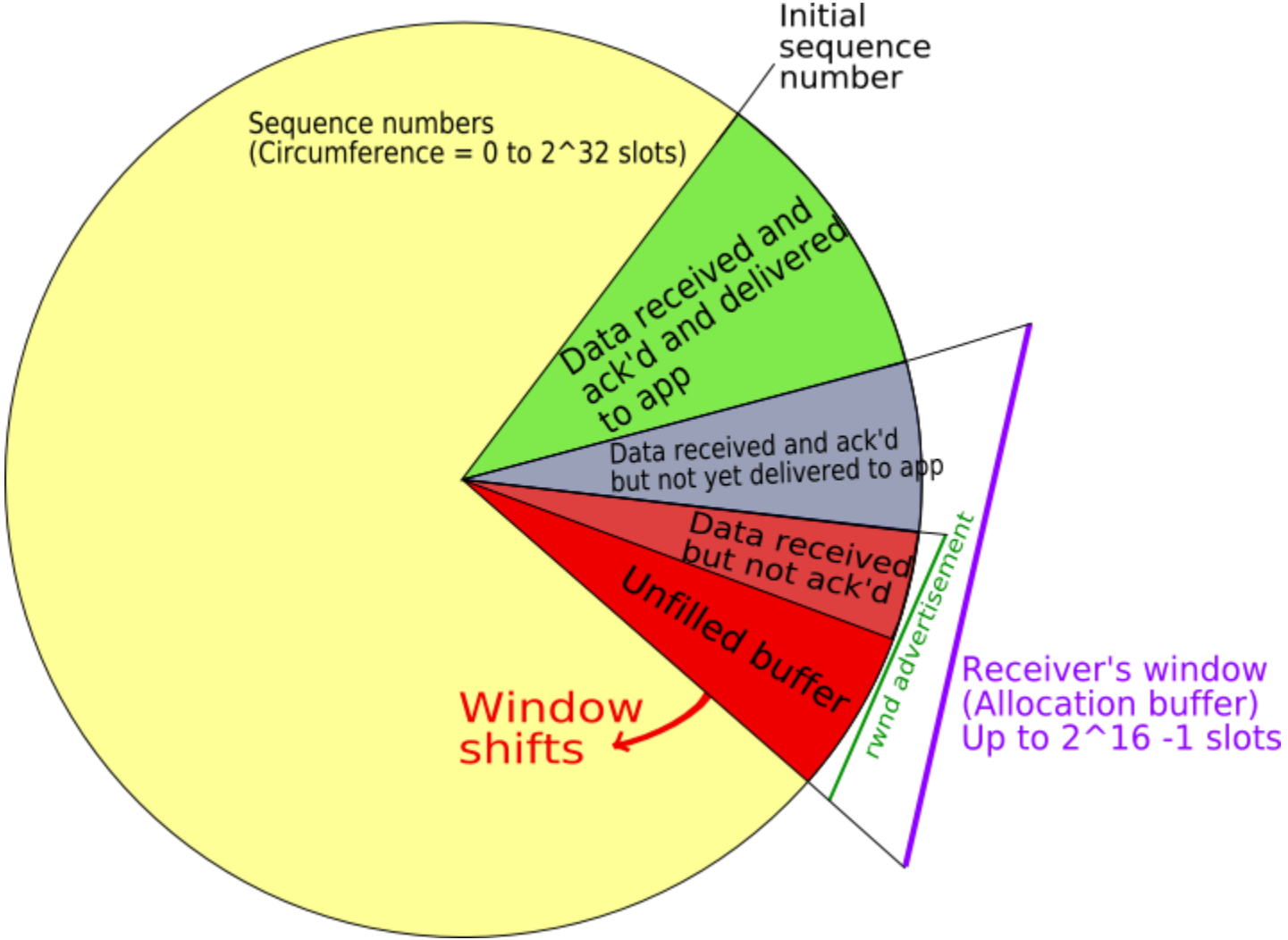


TCP Properties

- Highlights from the last lecture
 - Reliability
 - In-order delivery
 - Guaranteed delivery
 - Optimized for accuracy rather than latency
 - Good for non-real-time applications
 - Uses sliding window and positive acknowledgements to ensure proper packet delivery



Another view of Sliding Window



TCP Header Format

Byte offset	0	1	2	3
0	Source port		Destination port	
4	Sequence number			
8	Acknowledgment number			
12	Data offset	Resvd	Flags	Window Size
16	Checksum		Urgent pointer	
20	Options (optional)			
20+	Data			



TCP Header Format

- Source Port: sending port number
- Destination Port: receiving port number
- Sequence number: sequence number for bytes
- ACK number: if ACK set, acknowledged sequence number
- Data offset: size of TCP header / 4
- Flags: next slide
- Window: size of the receive window in bytes
- Checksum: Internet checksum
- Urgent pointer: if URG set, offset from sequence number for last urgent data byte
- [Options]



TCP Header Format

- Flags
 - CWR: Congestion Window Reduced (ECE ACK)
 - ECE (ECN-Echo): Indicates congestion
 - URG: Urgent pointer field significant
 - ACK: Ack field significant
 - PSH: Push function
 - RST: Reset the connection
 - SYN: Synchronize sequence numbers
 - FIN: No more data from sender

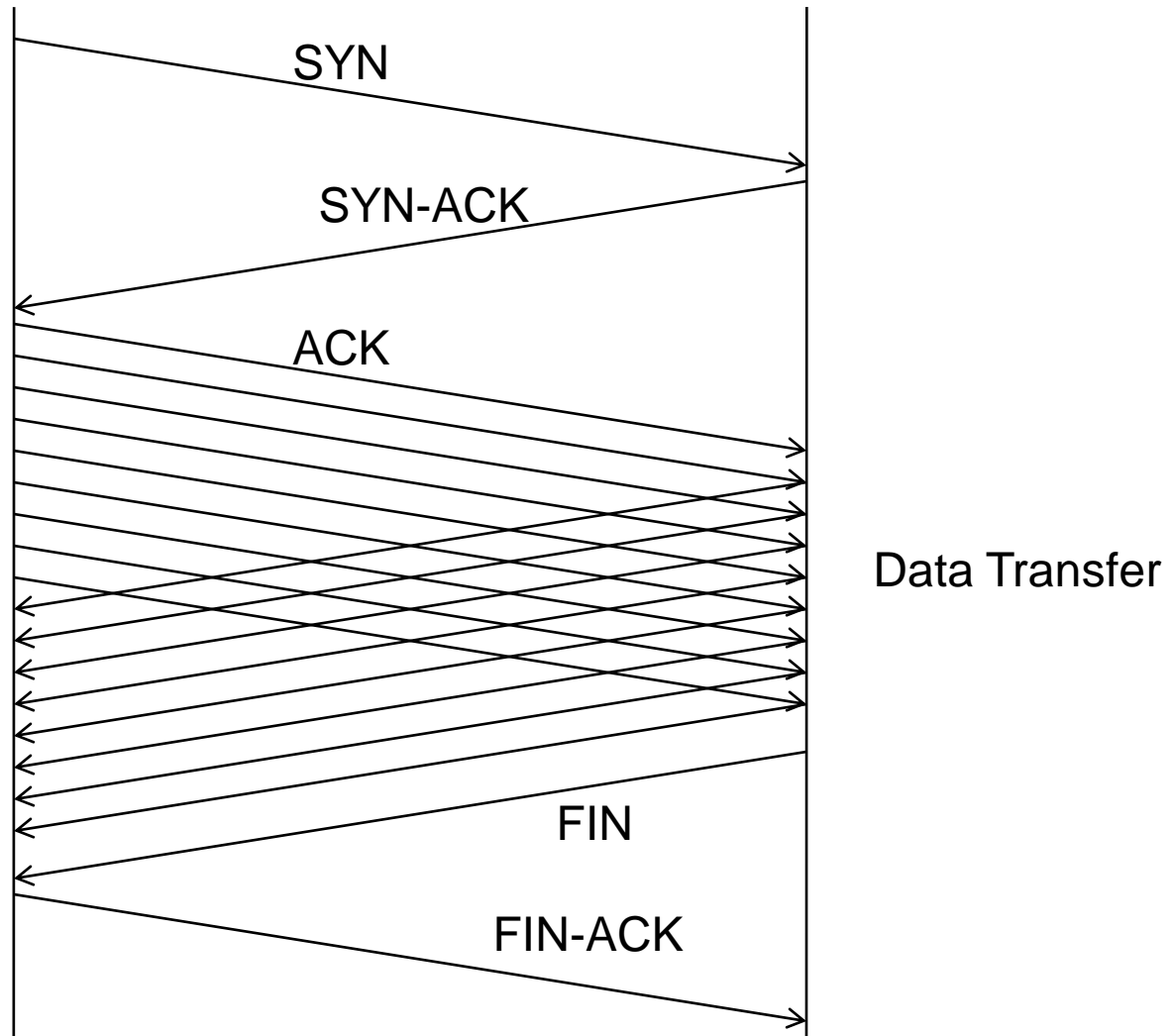


TCP Header Format

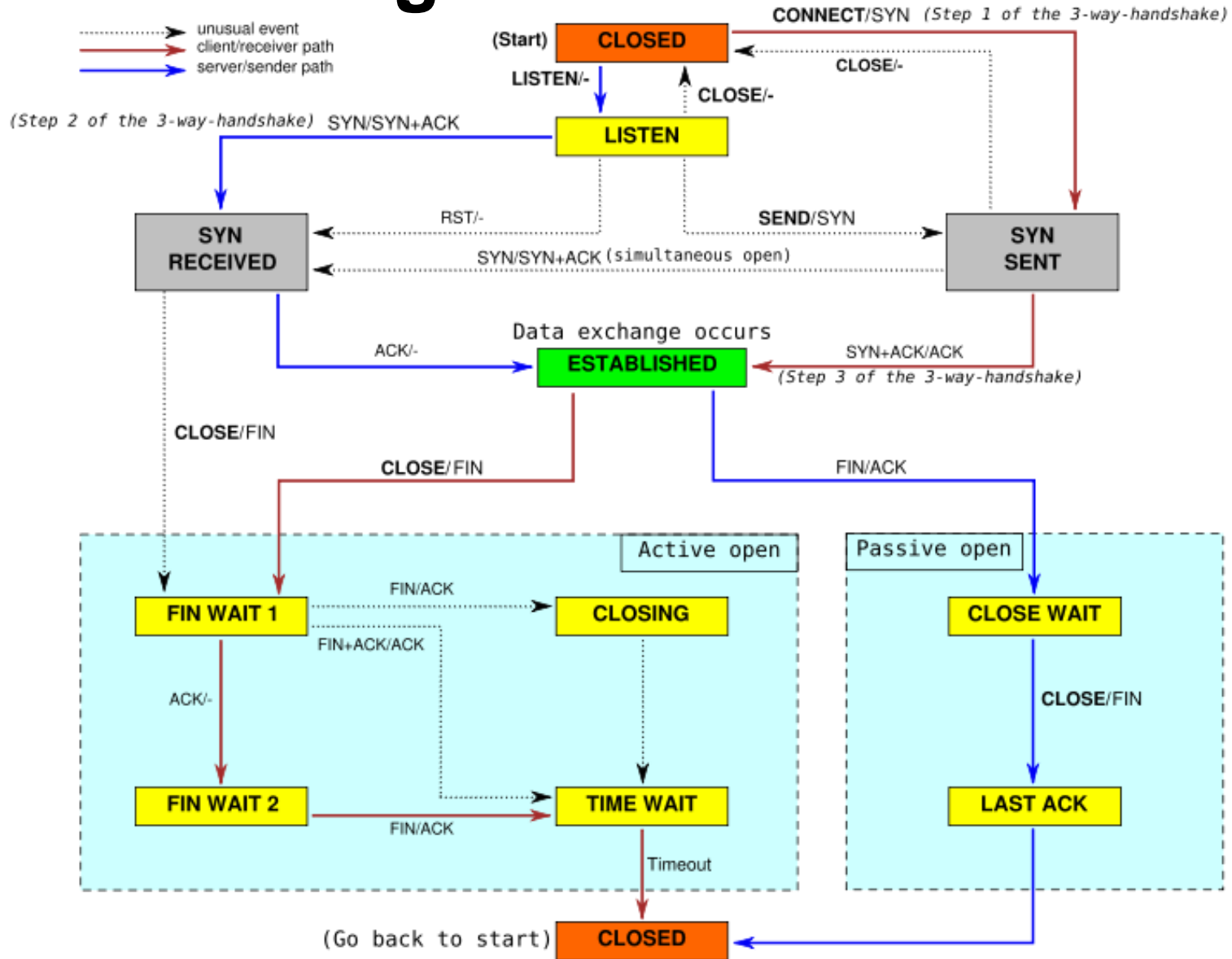
- Options
 - Maximum segment size (MSS)
 - Sender/receiver negotiate Path MTU to avoid IP fragmentation
 - Window scale
 - Allows window sizes > 65535
 - Specifies “units” of the window size field
 - Selective ACK ok
 - Timestamp
 - Useful for computing RTT
 - Useful for detecting sequence number wrap-around



TCP Control Packets



TCP State Diagram



Flow Control & Congestion Control

- Flow control
 - Regulate speed so end host can process it fast enough
 - Advertise receive window = 0 to pause traffic flow
 - “Silly window syndrome” results if repeatedly advertise small window size, overhead > data
 - Controlled by receiver
- Congestion control
 - Regulate speed so network can support traffic flow
 - Controlled by sender



Congestion Control

- Added to TCP/IP in 1980s as Internet was facing collapse due to congestion
- TCP self regulating
 - Sender enforced congestion control
 - Receiver enforced flow control
- Added new window size
 - rwnd = receiver window size (flow control)
 - cwnd = congestion window size (cong control)
 - Effective window size: $wnd = \min(rwnd, cwnd)$
 - Receiver doesn't know cwnd, only that the actual window size is no larger than rwnd



TCP Congestion Control

- Additive Increase, Multiplicative Decrease
- Additive Increase
 - Every time a window's worth of packets are successfully received, $cwnd = cwnd + MSS$
- Multiplicative Decrease
 - Every time a packet is lost, $cwnd = cwnd / 2$
- Effective window: $wnd = \min(cwnd, rwnd)$
- Called TCP *Congestion Avoidance*



Issues

- When do you decide a packet is lost?
 - After **timeout** occurs and ACK not received (traditional approach)
 - After **multiple ACKs** for the previous packet received (called *fast retransmit*)
- Ramp up period at the beginning can take a long time
- Serious gaps in performance when timeout occurs

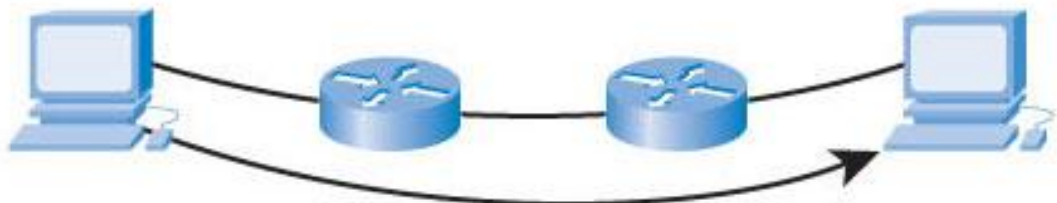


Slow Start

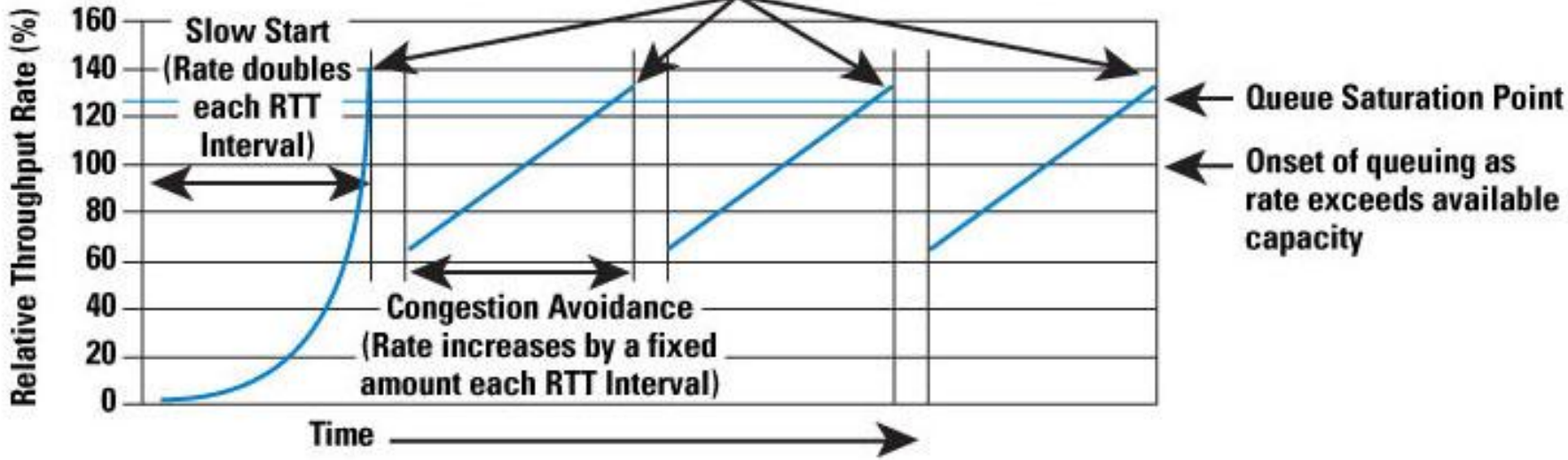
- Exponential growth rather than linear to get to optimal window size faster
 - Start with $cwnd=1$
 - For every distinct ACK, increase window size by the amount of data ACK-ed (doubles window size each RTT)
 - Stops when either a packet is lost or *slow-start threshold* is reached
 - If threshold, do additive increase (linear)
 - If loss
 - Cut $cwnd$ in half and start additive increase
 - Set this halved $cwnd$ as the new slow-start threshold



Slow Start



Duplicate ACKs received. Halve *cwnd* to recover.

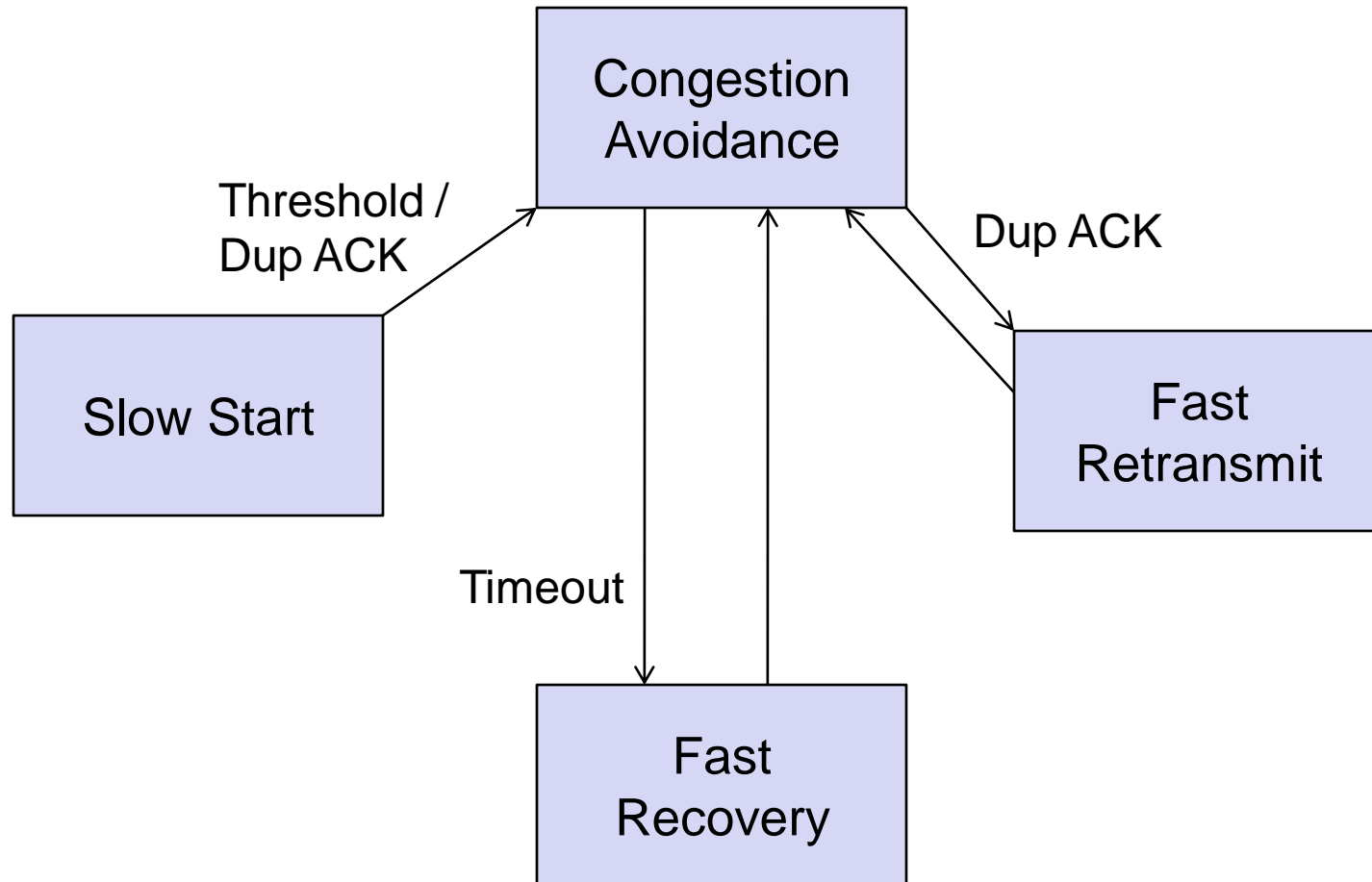


Fast Recovery

- Two loss events can occur
 - Packet loss detected via duplicate ACKs
 - addressed with *fast retransmit*
 - Packet loss detected via timeout
 - Typically requires $cwnd=1$, start over (possibly with slow start)
 - *Fast recovery* resets $cwnd$ to slow start threshold and resumes congestion avoidance



TCP Established States



TCP Flavors

- TCP Tahoe
 - Implemented congestion avoidance in TCP
 - Implemented Slow Start
 - Losses detected by timeouts
- TCP Reno
 - Losses detected by 3 dup ACKs
 - Implemented Fast Retransmit
- TCP Vegas
 - Losses detected by timeouts, but better stats (time average)
 - Used additive decrease, rather than multiplicative decrease
- TCP New Reno
 - Implements Fast Recovery



Explicit Congestion Notification (ECN)

- Intermediate routers can set bit in IP header when packet dropping is imminent
 - Set ECN bit in packets queued toward end of router buffers
 - Warns that congestion is going to occur
- TCP detects ECN bit and receiver signals to sender in ACK that congestion may occur soon, and that cwnd should be decreased
- Not well supported on the Internet

