


Computer Networks

CMSC 417 : Spring 2024

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Topics:

A) TCP details (Textbook chapter 6)
B) Link layer: Introduction, Ethernet (Textbook chapter 2)

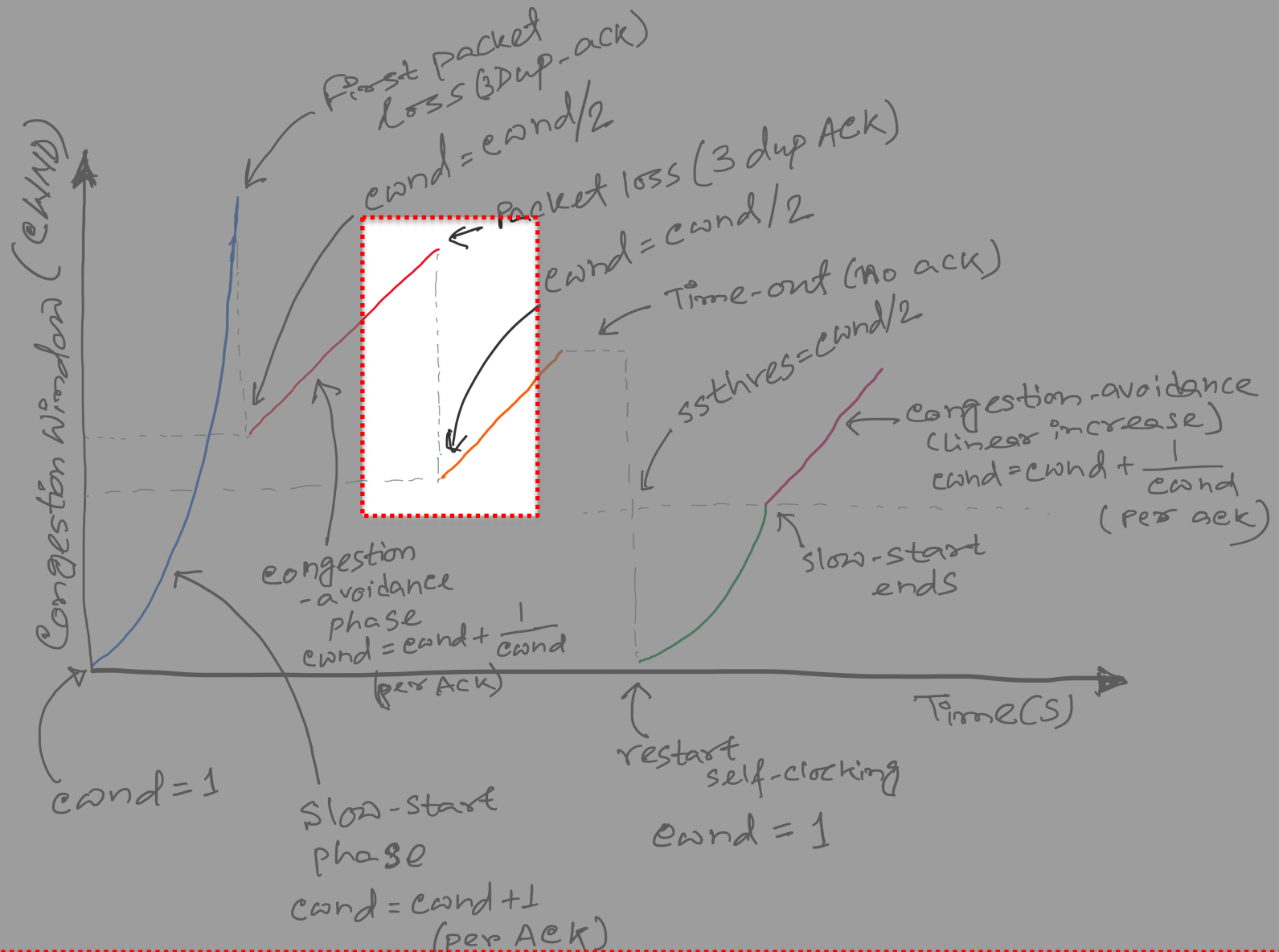
Nirupam Roy

Tu-Th 2:00-3:15pm
CSI 2117

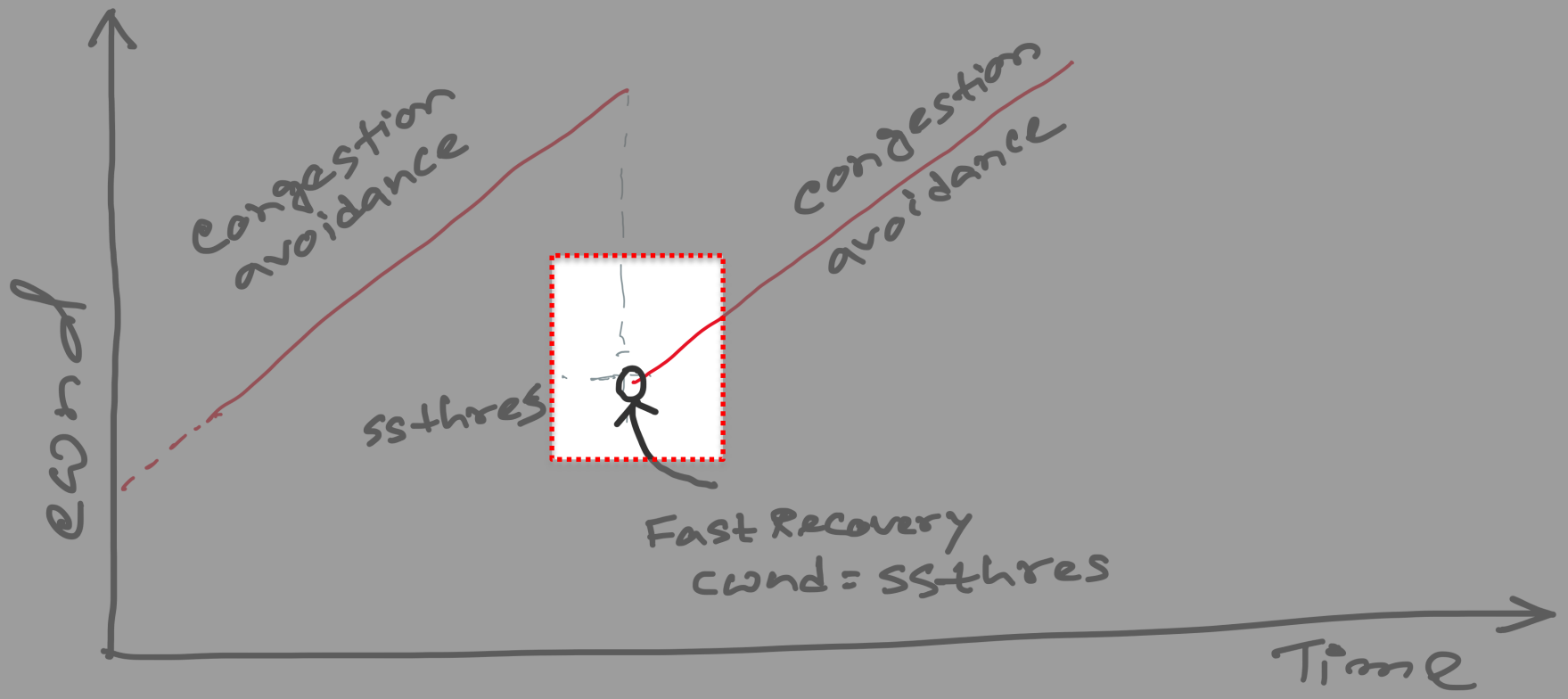
April 2nd, 2024



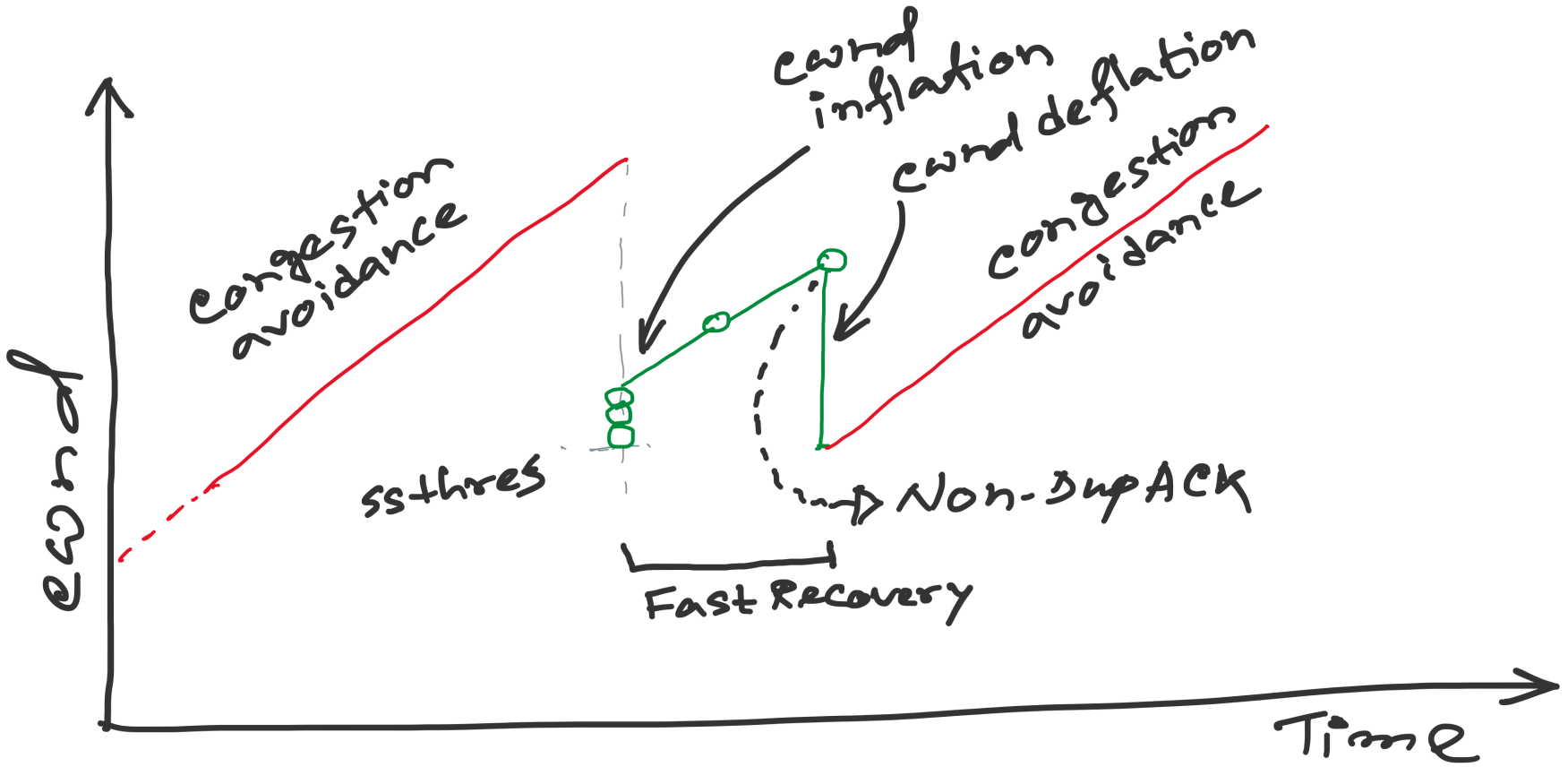

TCP Reno


First packet loss (3Dup-ack)
cwnd = cwnd/2
Packet loss (3 dup ACK)
cwnd = cwnd/2
Time-out (no ack)
ssthres = cwnd/2
congestion-avoidance (linear increase)
cwnd = cwnd + 1/cwnd
(per ack)
slow-start ends
congestion-avoidance phase
cwnd = cwnd + 1/cwnd
(per ACK)
Congestion window (cwnd)
Time(s)
cwnd = 1
slow-start phase
cwnd = cwnd + 1
(per Ack)
restart self-clocking
cwnd = 1

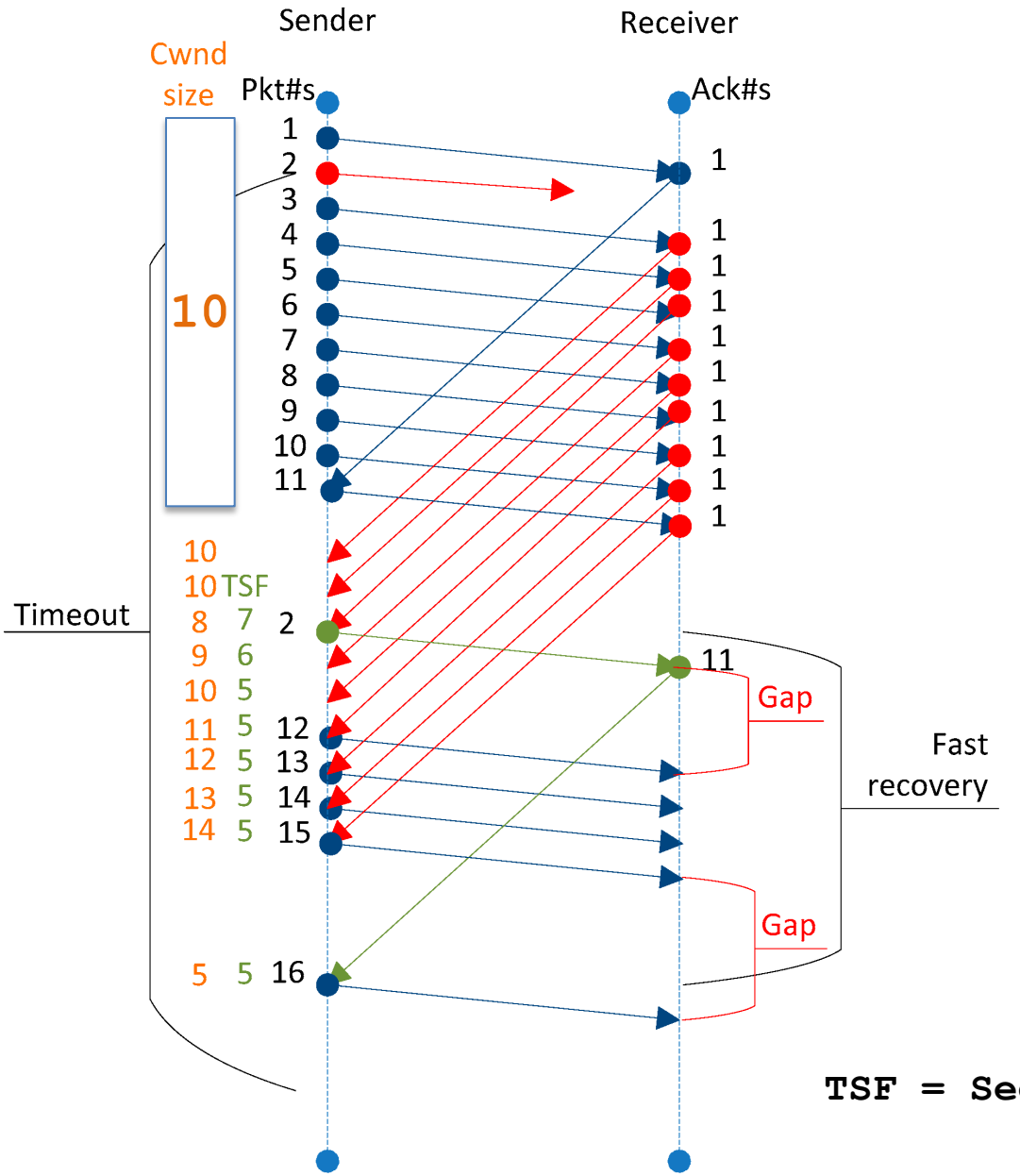
TCP Reno: details of Fast Recovery



TCP Reno: details of Fast Recovery



Fast Retransmit/Fast Recovery (FR/FR)



TSF = Segments in flight

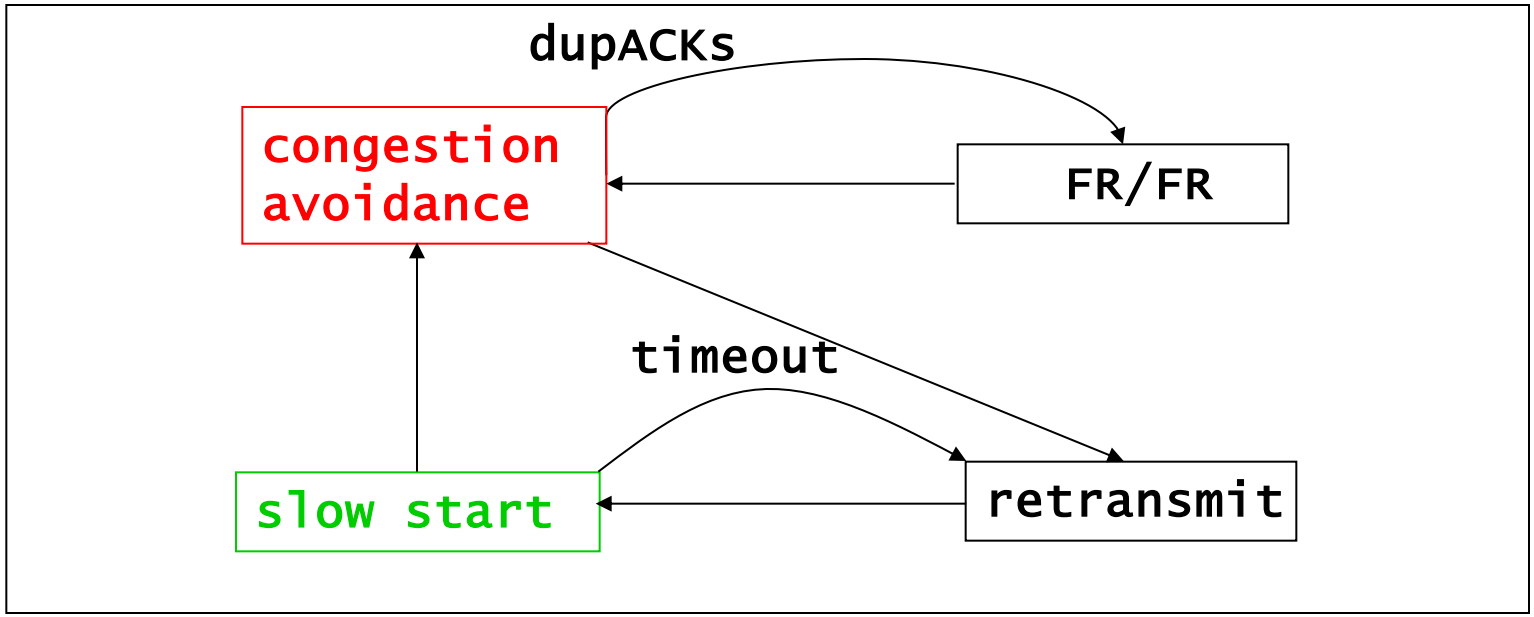
[Ref] https://www.mdpi.com/2076-3417/9/21/4698

Fast Retransmit/Fast Recovery (FR/FR)

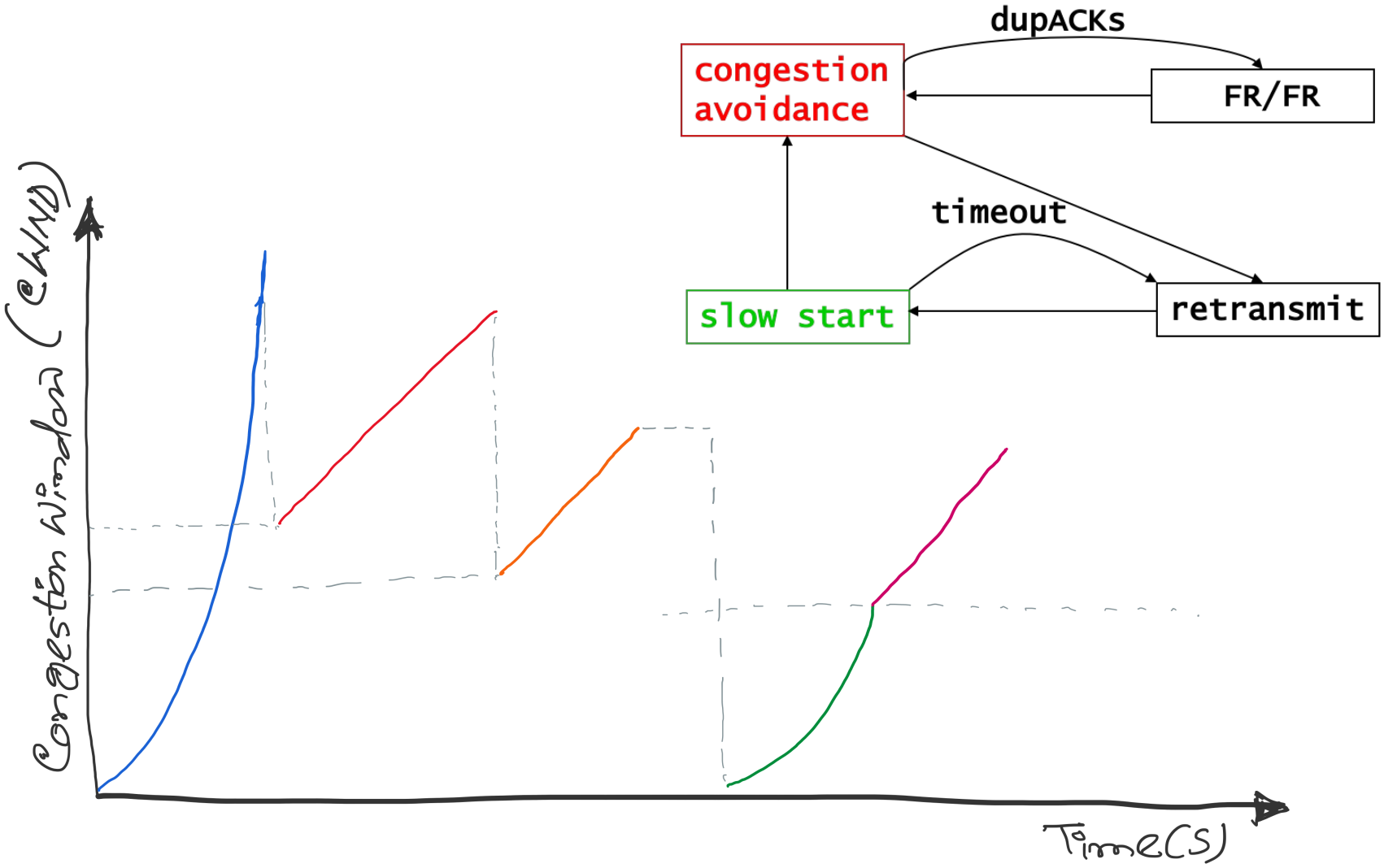
- Motivation: prevent `pipe' from emptying after fast retransmit
- Idea: each dupACK represents a packet having left the pipe (successfully received)
- Enter FR/FR after 3 dupACKs
  - Set ssthresh ← max(flightsize/2, 2)
  - Retransmit lost packet
  - Set cwnd ← ssthresh + ndup (window inflation)
  - Wait till W=min(awnd, cwnd) is large enough; transmit new packet(s)
  - On non-dup ACK (1 RTT later), set cwnd ← ssthresh (window deflation)
- Enter CA

Summary: Reno

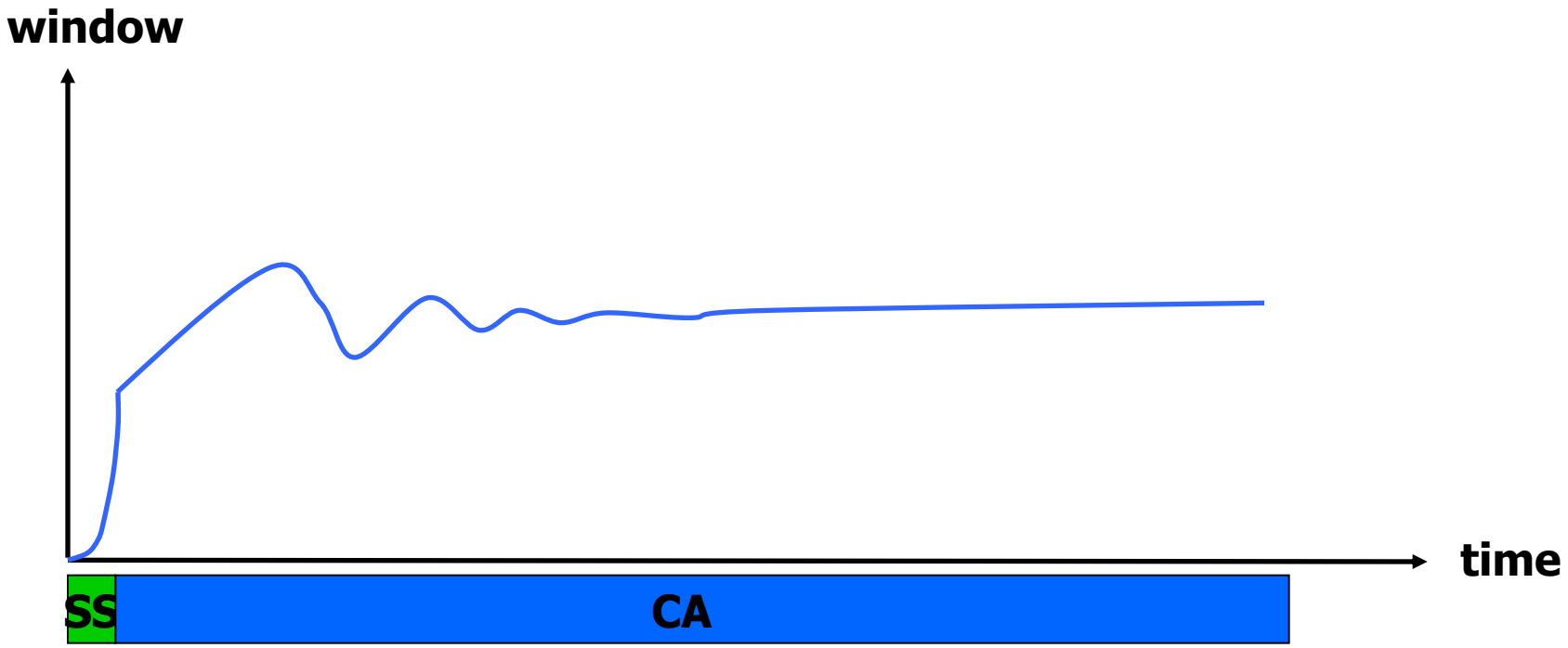
- Basic ideas
  - Fast recovery avoids slow start
  - dupACKs: fast retransmit + fast recovery
  - Timeout: fast retransmit + slow start



TCP congestion control: CWND graph



TCP Vegas (Brakmo & Peterson 1994)

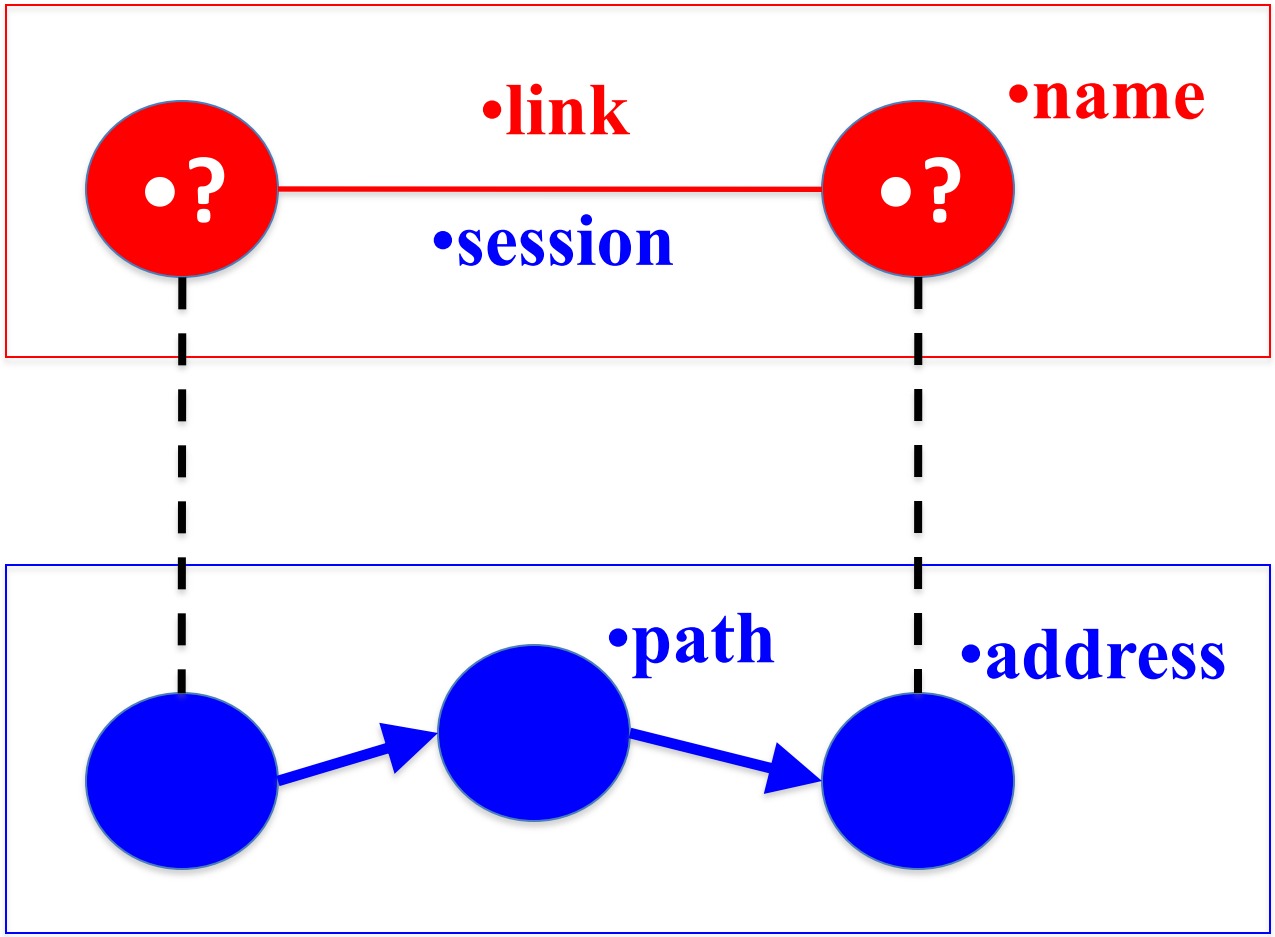


- Considers packet delay, not loss, as a hint for congestion
- Reno with a new congestion avoidance algorithm
- Converges (provided buffer is large) !

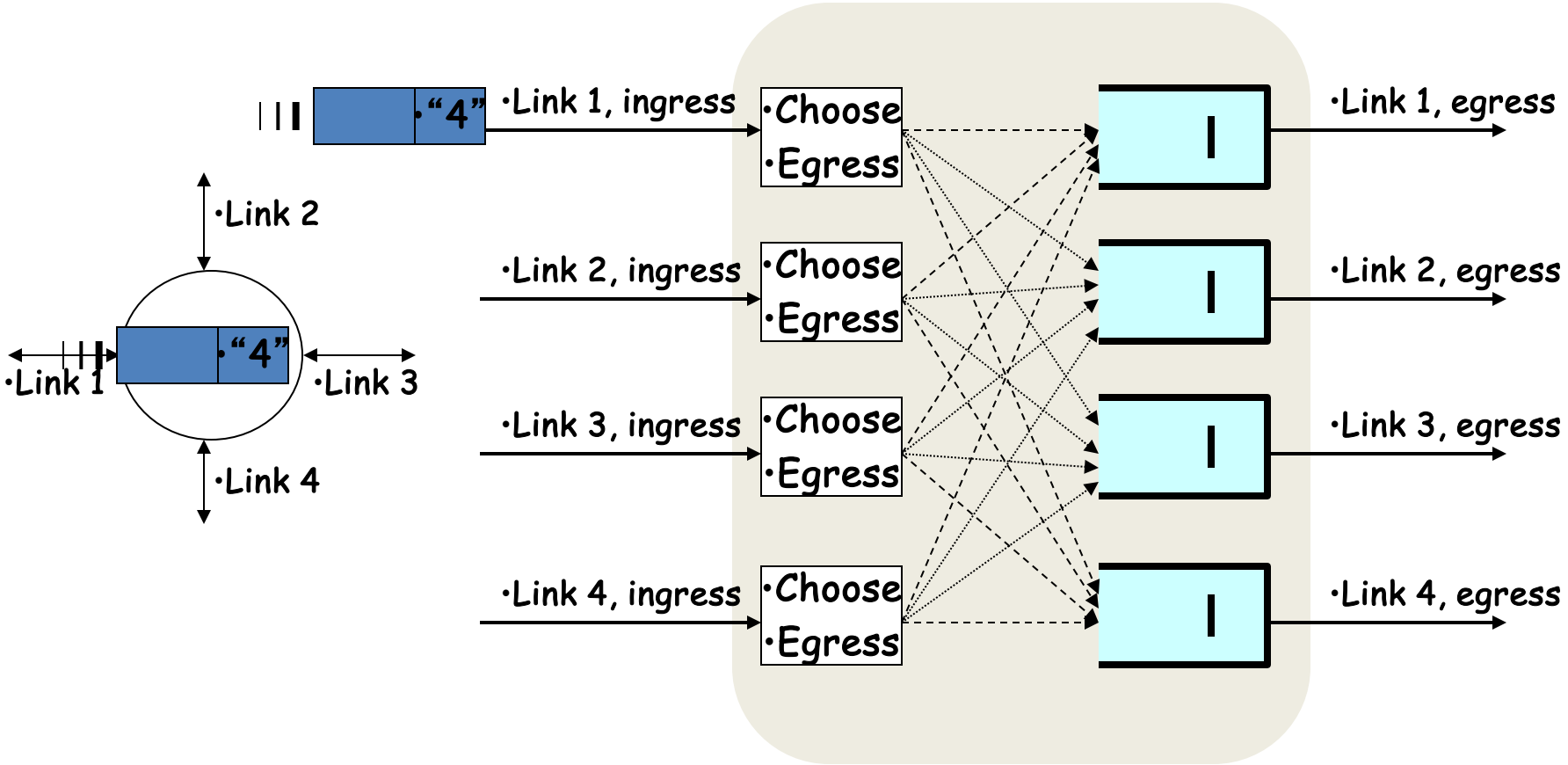
2. Router-assisted Congestion Control

Congestion Control discussed so far

- **What can the *end-points* do to collectively to make good use of shared underlying resources?**



Packet Switching and Forwarding



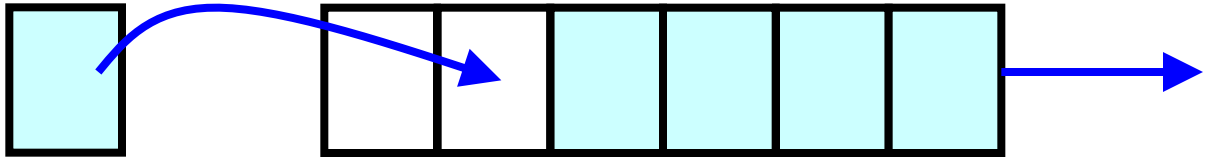
Queue Management Issues

- Scheduling discipline
  - Which packet to send?
  - Some notion of fairness? Priority?
- Drop policy
  - When should you discard a packet?
  - Which packet to discard?
- Goal: balance throughput and delay
  - Huge buffers minimize drops, but add to queuing delay (thus higher RTT, longer slow start, …)

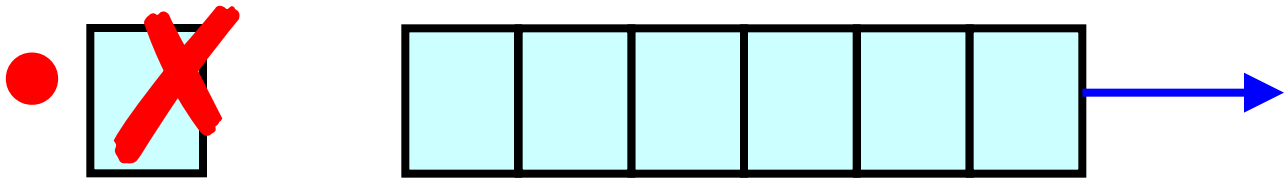
Queue Management 1: Drop Policy

FIFO Scheduling and Drop-Tail

- Access to the bandwidth: first-in first-out queue
  - Packets only differentiated when they arrive

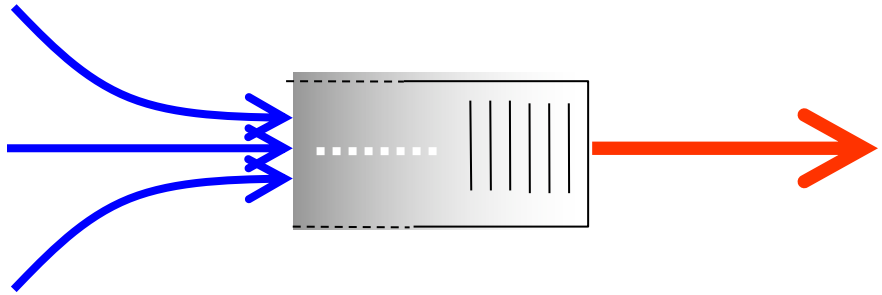


- Access to the buffer space: drop-tail queuing
  - If the queue is full, drop the incoming packet



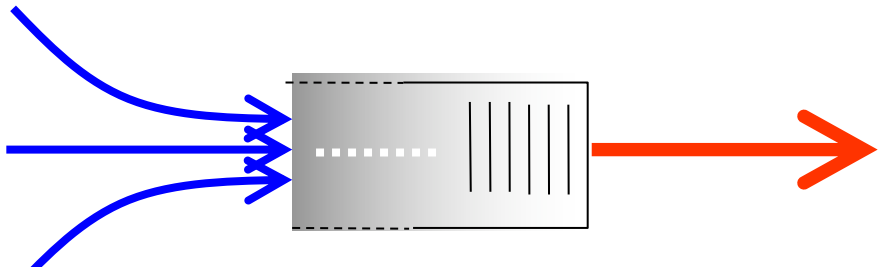
Bursty Loss From Drop-Tail Queuing

- TCP depends on packet loss
  - Packet loss is indication of congestion
  - TCP additive increase drives network into loss
- Drop-tail leads to *bursty* loss
  - Congested link: many packets encounter full queue
  - Synchronization: many connections lose packets at once



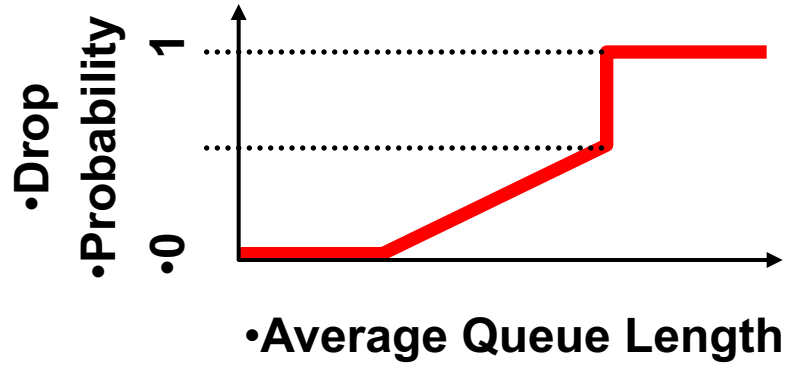
Slow Feedback from Drop Tail

- Feedback comes when buffer is completely full
  - … even though the buffer has been filling for a while
- Plus, the filling buffer is increasing RTT
  - … making detection even slower
- Better to give early feedback
  - Get 1-2 connections to slow down before it's too late!



Random Early Detection (RED)

- Router notices that queue is getting full
  - … and randomly drops packets to signal congestion
- Packet drop probability
  - Drop probability increases as queue length increases
  - Else, set drop probability *f*(avg queue length)



Properties of RED

- Drops packets before queue is full
  - In the hope of reducing the rates of some flows
- Drops packet in proportion to each flow's rate
  - High-rate flows selected more often
- Drops are spaced out in time
  - Helps desynchronize the TCP senders
- Tolerant of burstiness in the traffic
  - By basing the decisions on average queue length

Problems With RED

- Hard to get tunable parameters just right
  - How early to start dropping packets?
  - What slope for increase in drop probability?
  - What time scale for averaging queue length?
- RED has mixed adoption in practice
  - If parameters aren't set right, RED doesn't help
- Many other variations in research community
  - Names like "Blue" (self-tuning), "FRED"…

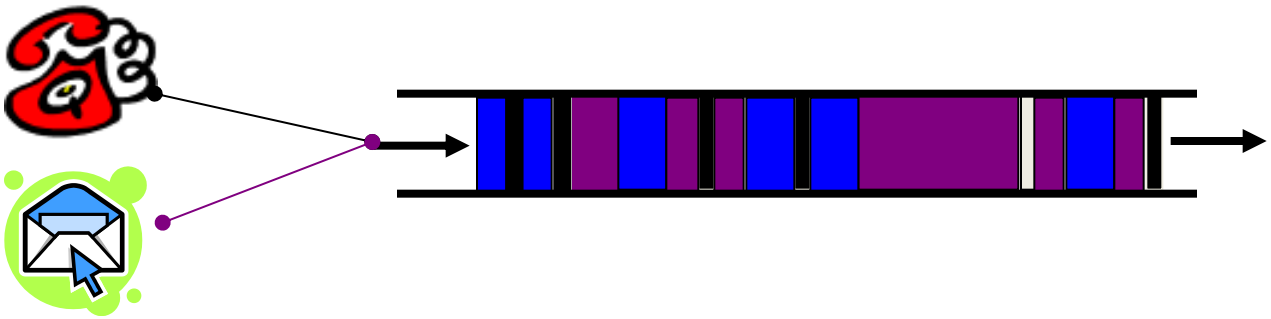
Feedback: From Loss to Notification

- Early dropping of packets
  - Good: gives early feedback
  - Bad: has to drop the packet to give the feedback
- Explicit Congestion Notification (ECN)
  - Router marks the packet with an ECN bit
  - Sending host interprets as a sign of congestion
  - Requires participation of hosts and the routers

Queue Management 2: Link Scheduling

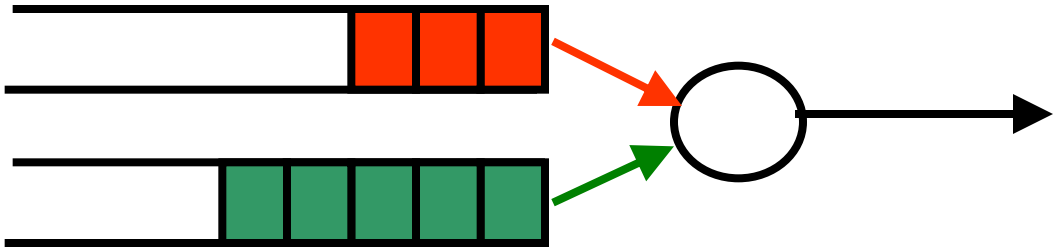
First-In First-Out Scheduling

- First-in first-out scheduling
  - Simple, but restrictive
- Example: two kinds of traffic
  - Voice over IP needs low delay
  - E-mail is not that sensitive about delay
- Voice traffic waits behind e-mail



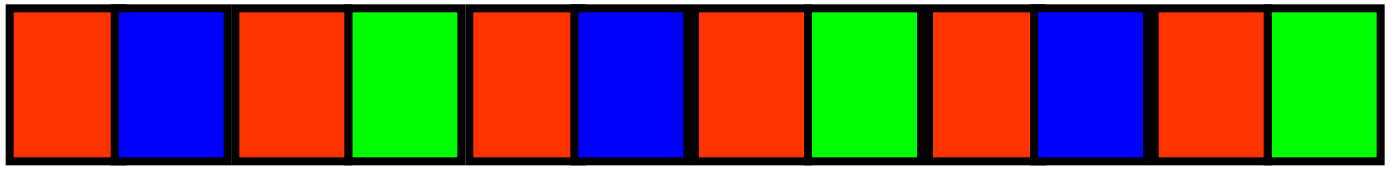
Strict Priority

- Multiple levels of priority
  - Always transmit high-priority traffic, when present
- Isolation for the high-priority traffic
  - Almost like it has a dedicated link
  - Except for (small) delay for packet transmission
- But, lower priority traffic may starve ☹



Weighted Fair Scheduling

- Weighted fair scheduling
  - Assign each queue a fraction of the link bandwidth
  - Rotate across queues on a small time scale



- 50% red, 25% blue, 25% green

- Work-conserving
  - Send extra traffic from one queue if others are idle

Implementation Trade-Offs

- FIFO
  - One queue, trivial scheduler
- Strict priority
  - One queue per priority level, simple scheduler
- Weighted fair scheduling
  - One queue per class, and more complex scheduler

TCP Congestion Control

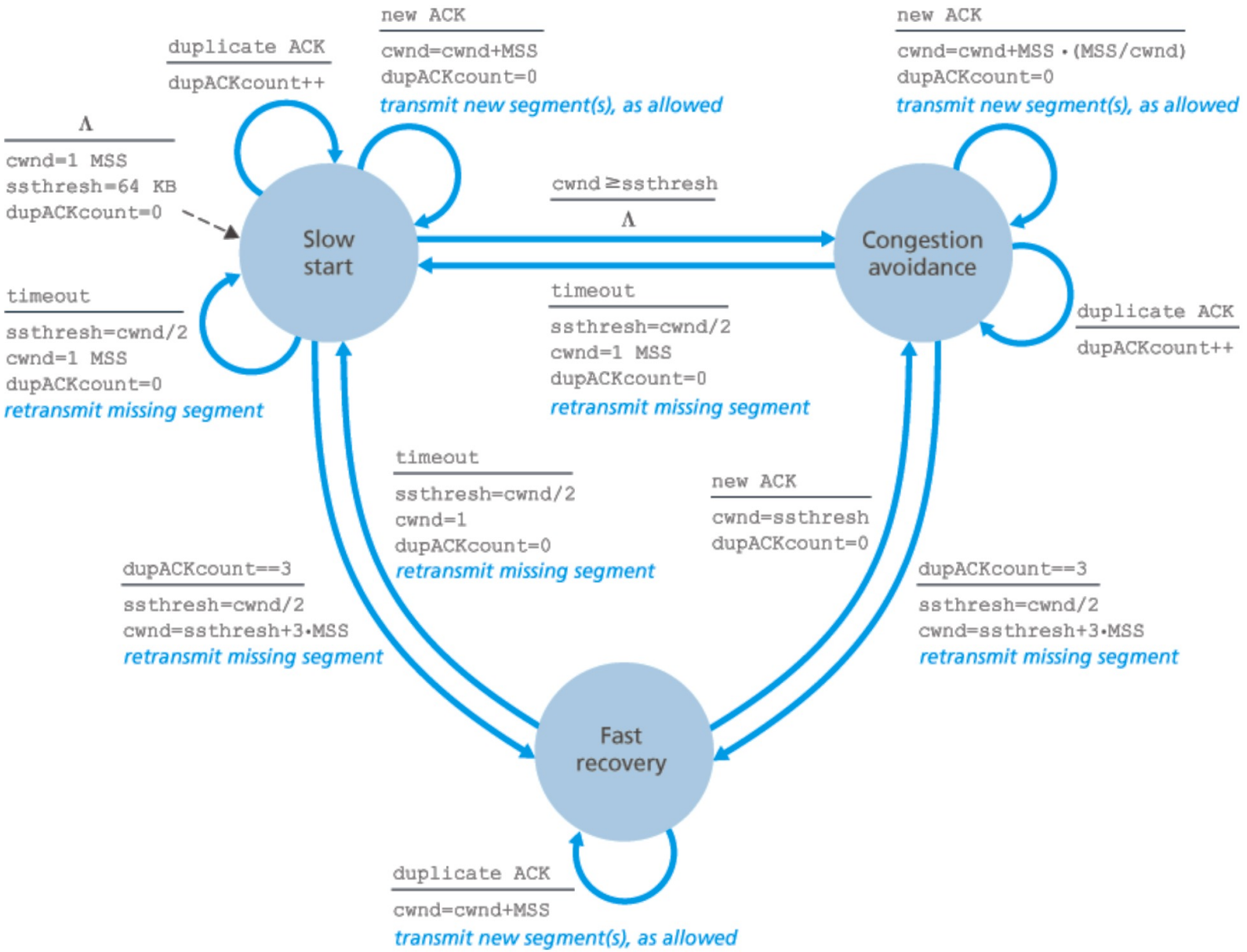
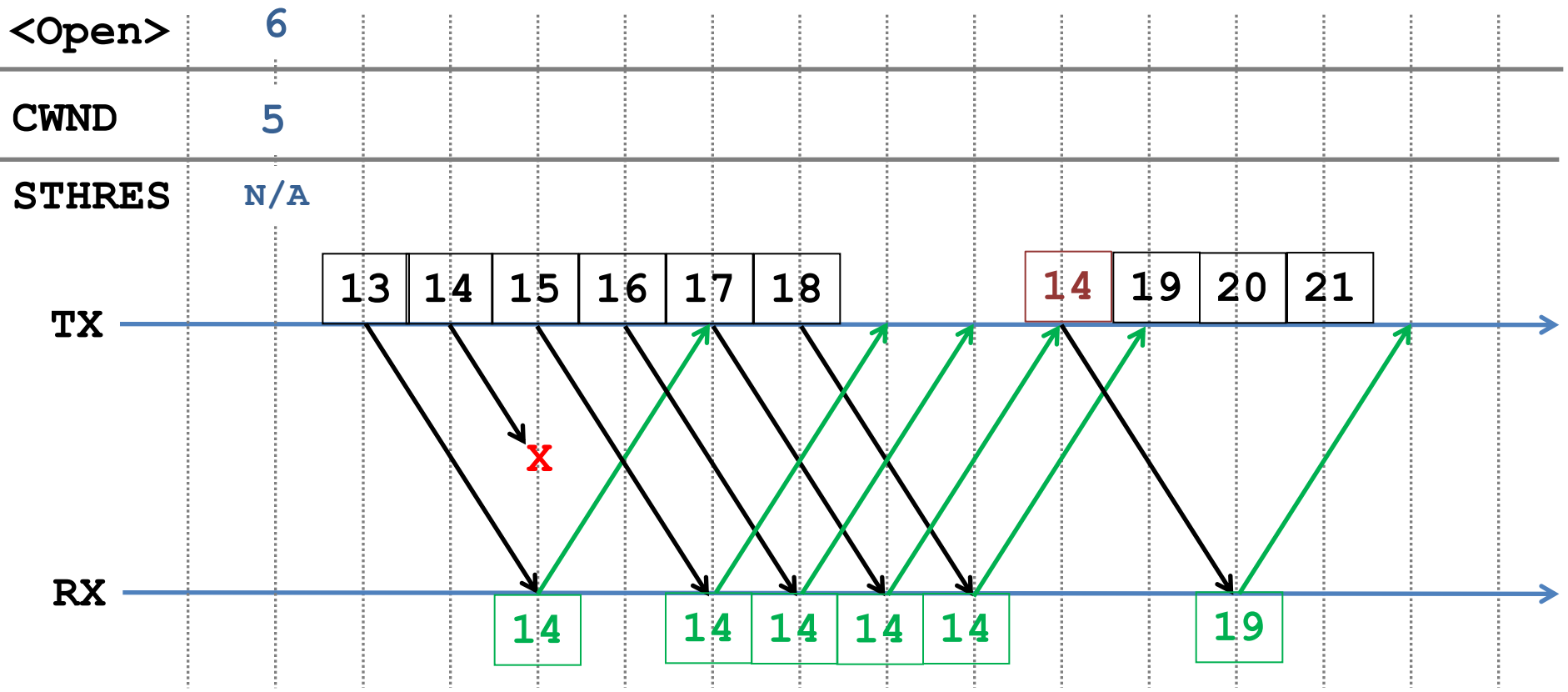
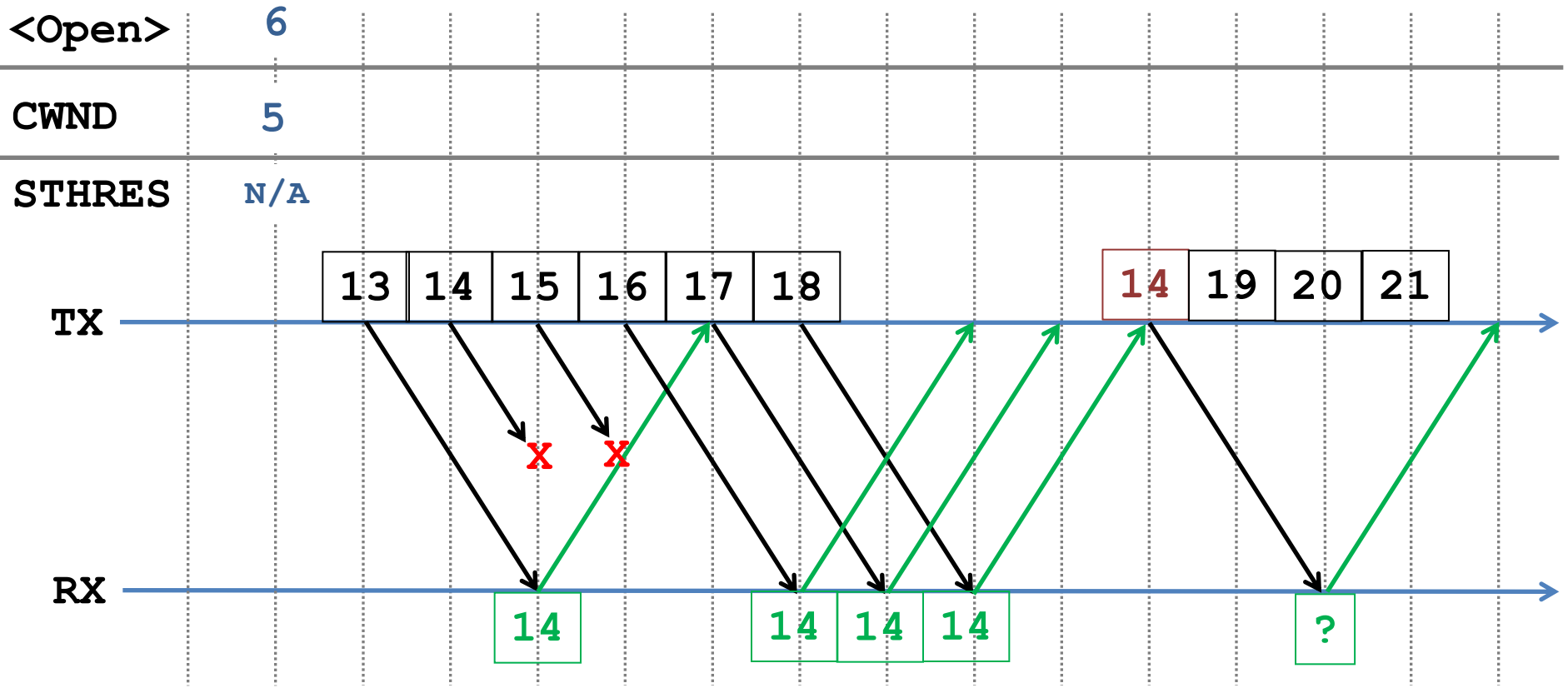


Figure 3.51 FSM description of TCP congestion control

Revisiting Dup ACKs (without Fast-Recovery)



Revisiting Dup ACKs (without Fast-Recovery)



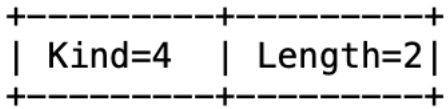
What if there are multiple losses in an RTT?

TCP Selective ACK (SACK): RFC 2018

https://www.rfc-editor.org/rfc/rfc2018

TCP Sack-Permitted Option:

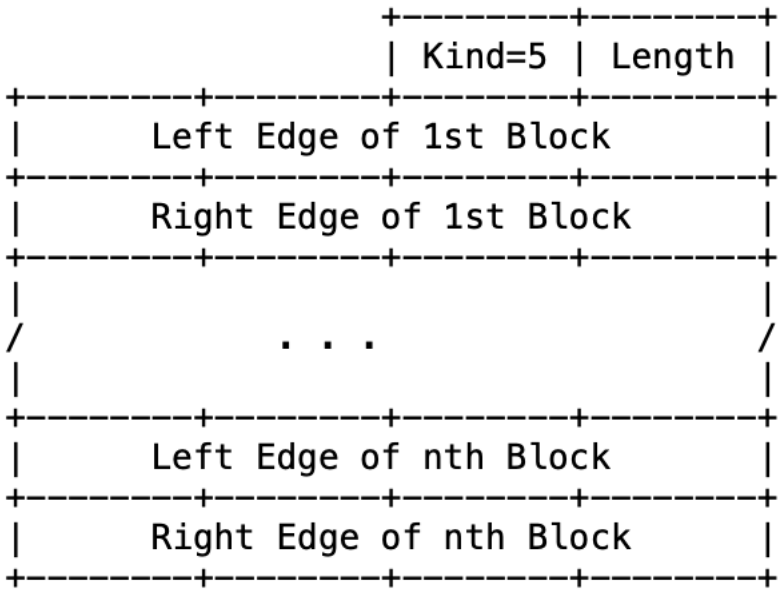
Kind: 4



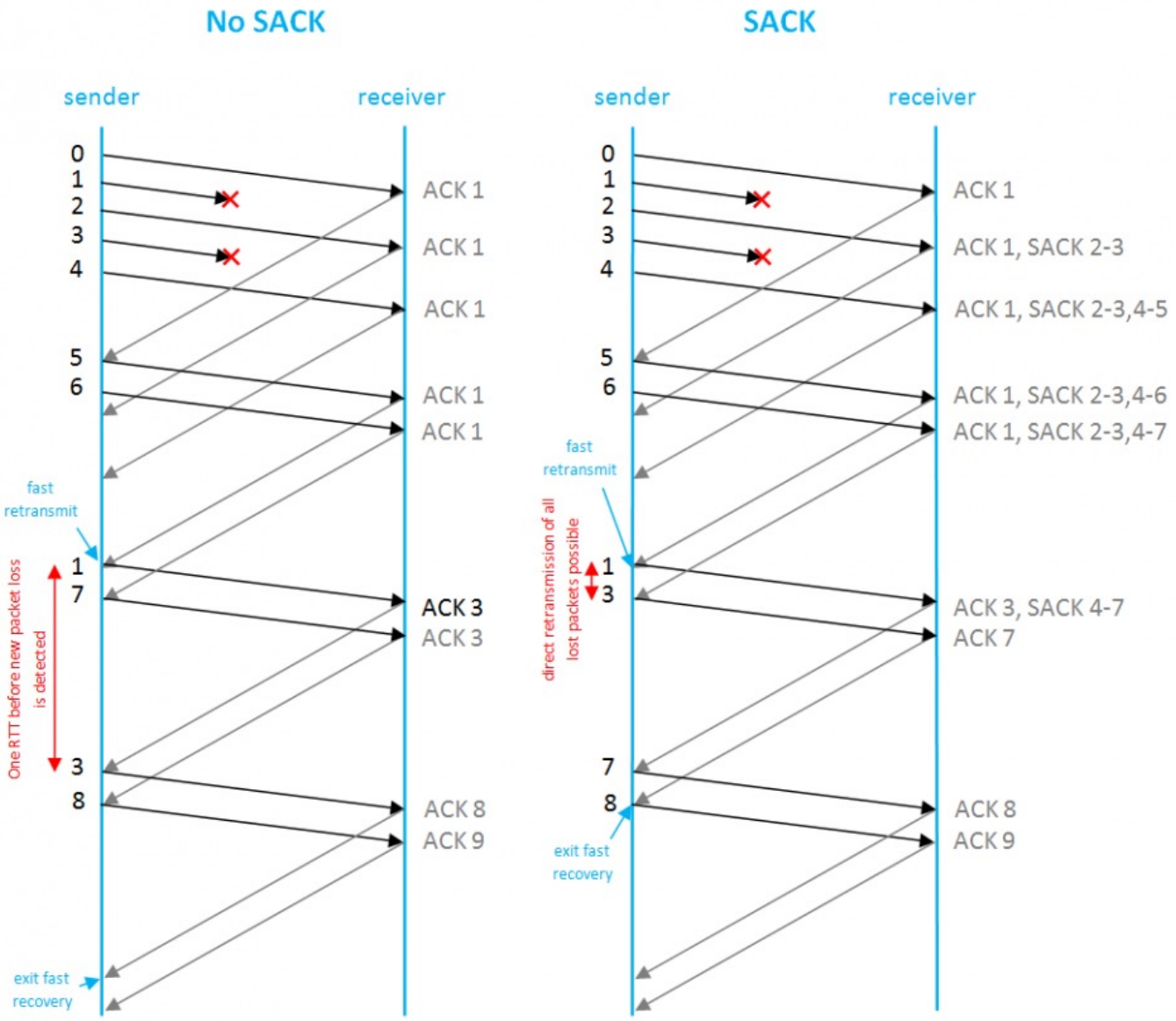
TCP SACK Option:

Kind: 5

Length: Variable



TCP Selective ACK (SACK): RFC 2018



[REF-1] https://www.rfc-editor.org/rfc/rfc2018

[REF-2] https://www.excentis.com/blog/measuring-throughput-effect-of-used-tcp-settings/

TCP Congestion Control with a Misbehaving Receiver

Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson
Department of Computer Science and Engineering
University of Washington, Seattle

Slides courtesy: https://cseweb.ucsd.edu/classes/wi01/cse222/lectures/attack-18.pdf

The problem

- Bandwidth sharing on the Internet
  - Hosts **voluntarily** limit own data rate
  - Mechanism implemented in TCP
  - “Fair” rate determined by testing the network
  - Relies on **cooperation** between endpoints
- Why doesn’t everyone cheat?

One explanation

- Cheating requires motive **and** opportunity
- Senders (e.g., Web servers)
  - Have opportunity (could send too fast)
  - Limited motive (economic incentive to share)
- Receivers (e.g., Web clients)
  - Have competitive motive (faster Web surfing)
  - No opportunity (only receive data)… right?

What if receivers misbehave?

- A client can **implicitly** control the data rate of a remote server
  - This is not an implementation error
  - It is a weakness in the TCP specification
  - TCP's design does not consider that senders and receivers might have disjoint interests
- The vulnerability is significant…

Why the Web is faster in Seattle