

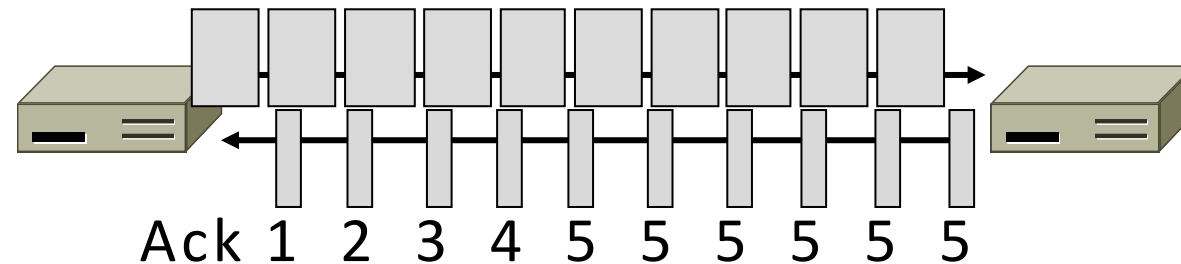
Fast Recovery
(TCP Multiplicative Decrease)

Inferring Loss from ACKs

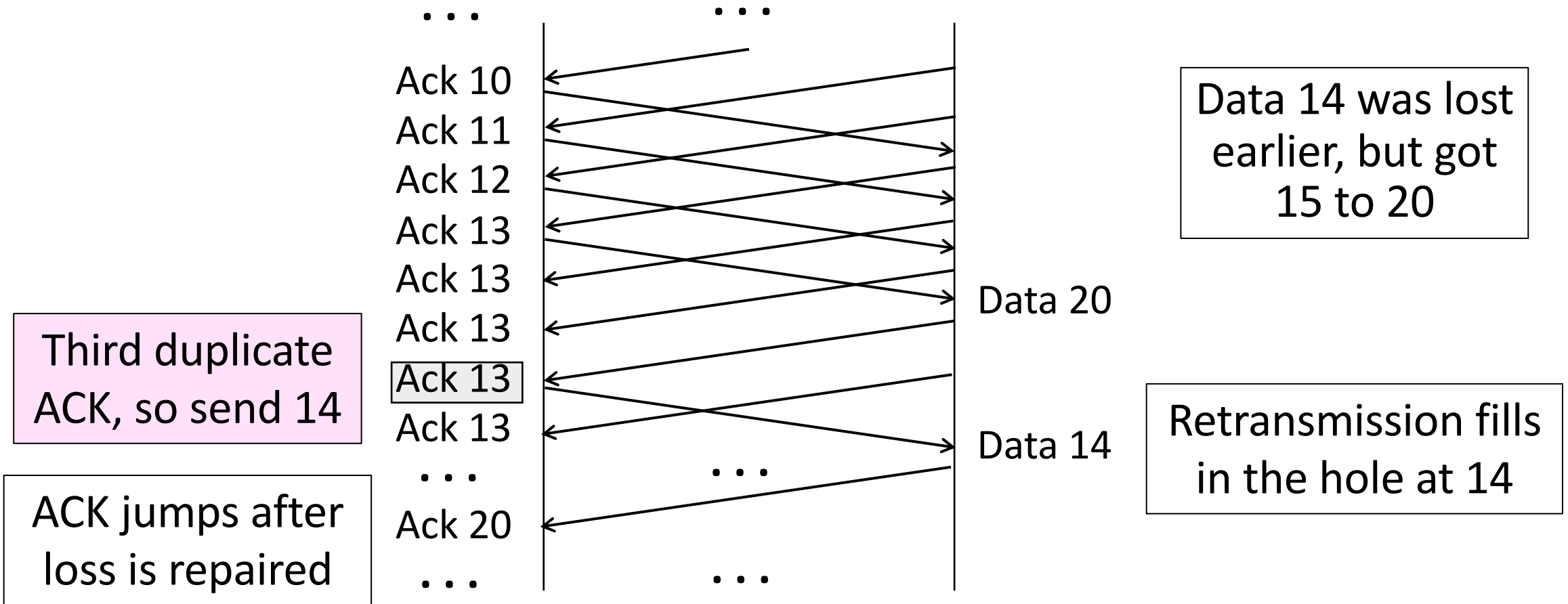
- TCP uses a cumulative ACK
 - Carries highest in-order seq. number
 - Normally a steady advance
- Duplicate ACKs give us hints about what data hasn't arrived
 - Tell us some new data did arrive, but it was not next segment
 - Thus the next segment may be lost

Fast Retransmit

- Treat three duplicate ACKs as a loss
 - Retransmit next expected segment
 - Some repetition allows for reordering, but still detects loss quickly



Fast Retransmit (2)



Fast Retransmit (3)

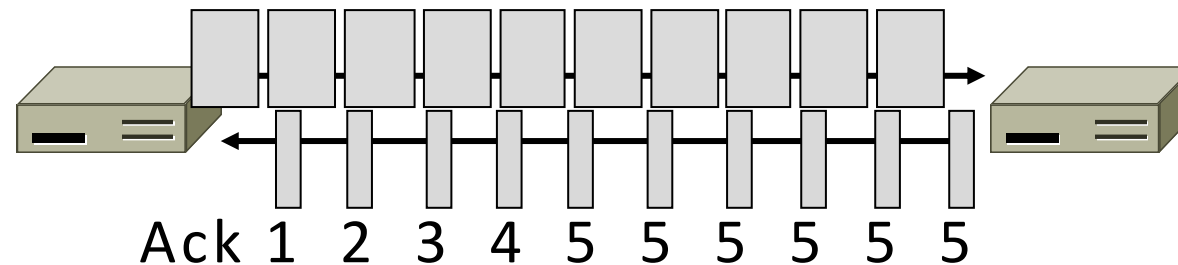
- It can repair single segment loss quickly, typically before a timeout
- However, we have quiet time at the sender/receiver while waiting for the ACK to jump
- And we still need to MD cwnd ...

Inferring Non-Loss from ACKs

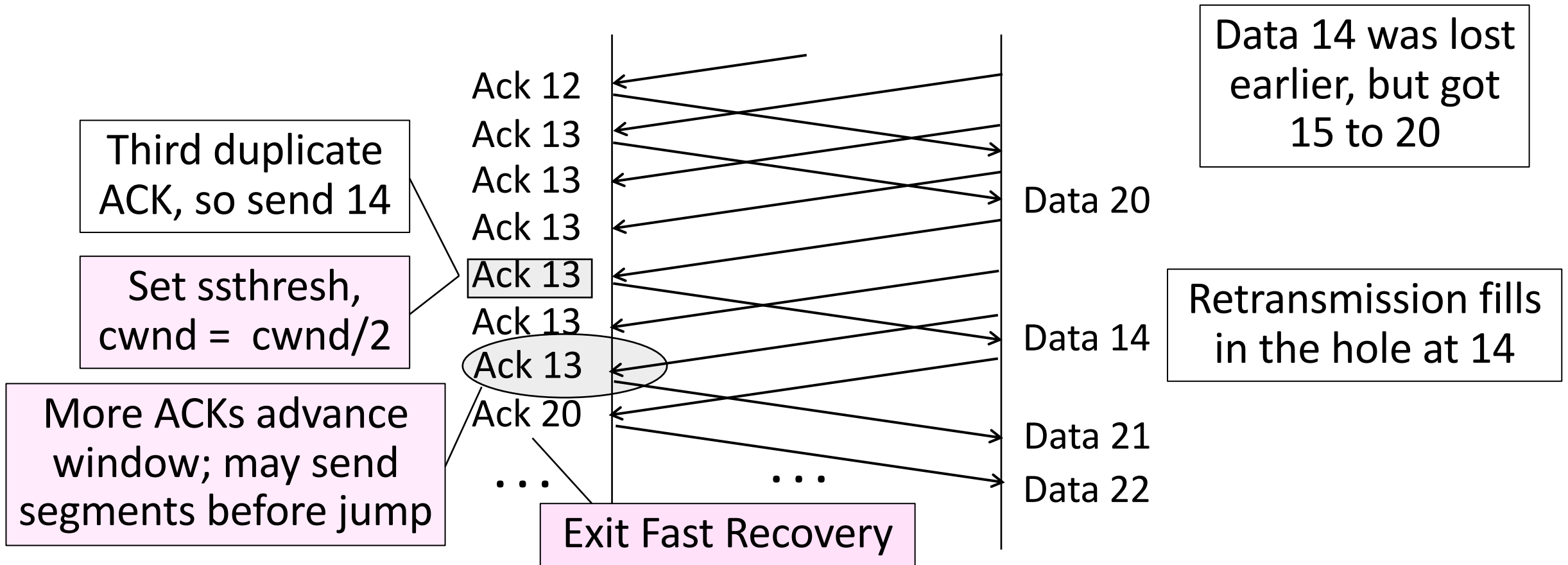
- Duplicate ACKs also give us hints about what data has arrived
 - Each new duplicate ACK means that some new segment has arrived
 - It will be the segments after the loss
 - Thus advancing the sliding window will not increase the number of segments stored in the network

Fast Recovery

- First fast retransmit, and MD cwnd
- Then pretend further duplicate ACKs are the expected ACKs
 - Lets new segments be sent for ACKs
 - Reconcile views when the ACK jumps



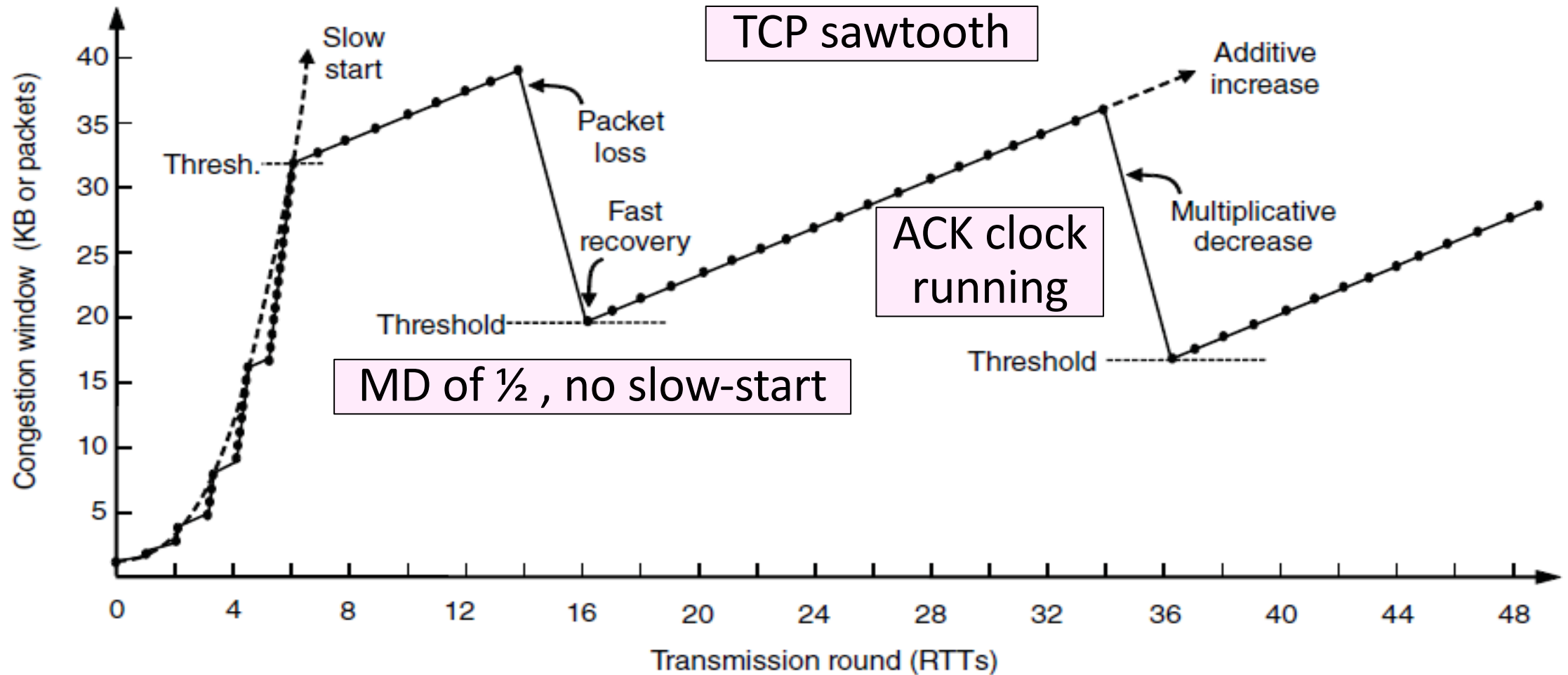
Fast Recovery (2)



Fast Recovery (3)

- With fast retransmit, it repairs a single segment loss quickly and keeps the ACK clock running
- This allows us to realize AIMD
 - No timeouts or slow-start after loss, just continue with a smaller cwnd
- TCP Reno combines slow-start, fast retransmit and fast recovery
 - Multiplicative Decrease is $\frac{1}{2}$

TCP Reno



TCP Reno, NewReno, and SACK

- Reno can repair one loss per RTT
 - Multiple losses cause a timeout
- NewReno further refines ACK heuristics
 - Repairs multiple losses without timeout
- Selective ACK (SACK) is a better idea
 - Receiver sends ACK ranges so sender can retransmit without guesswork

Network-Assisted Congestion Control

Congestion Avoidance vs. Control

- Classic TCP drives the network into congestion and then recovers
 - Needs to see loss to slow down
- Would be better to use the network but avoid congestion altogether!
 - Reduces loss and delay
- But how can we do this?

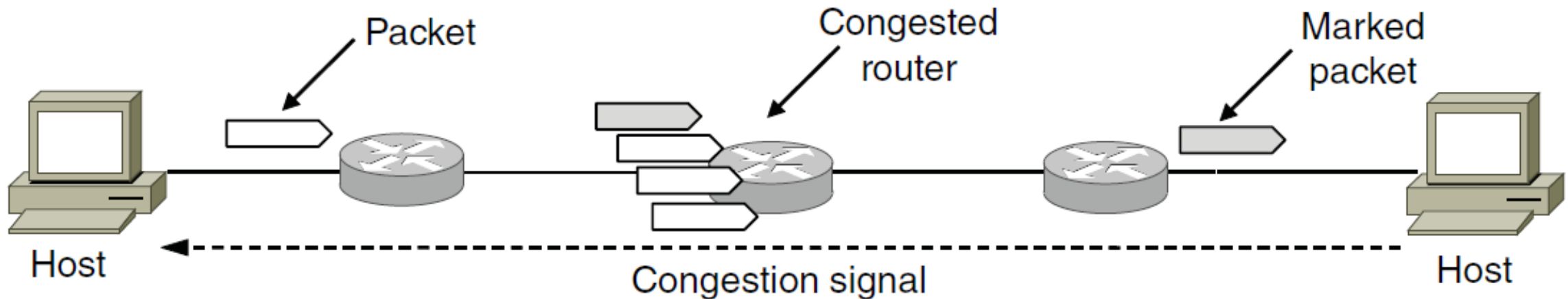
Feedback Signals

- Delay and router signals can let us avoid congestion

Signal	Example Protocol	Pros / Cons
Packet loss	Classic TCP Cubic TCP (Linux)	Hard to get wrong Hear about congestion late Other events can cause loss
Packet delay	Compound TCP (Windows)	Hear about congestion early Need to infer congestion
Router indication	TCPs with Explicit Congestion Notification	Hear about congestion early Require router support

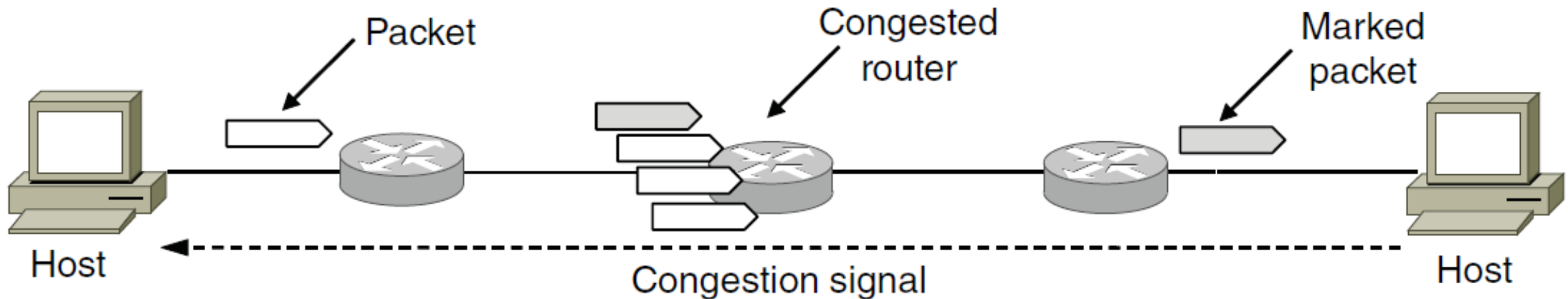
ECN (Explicit Congestion Notification)

- Router detects the onset of congestion via its queue
 - When congested, it marks affected packets (IP header)



ECN (2)

- Marked packets arrive at receiver
 - TCP receiver informs TCP sender of the congestion



ECN (3)

- **Advantages:**
 - Routers deliver clear signal to hosts
 - Congestion is detected early, no loss
 - No extra packets need to be sent
- **Disadvantages:**
 - Routers and hosts must be upgraded

What's new in transport protocols?

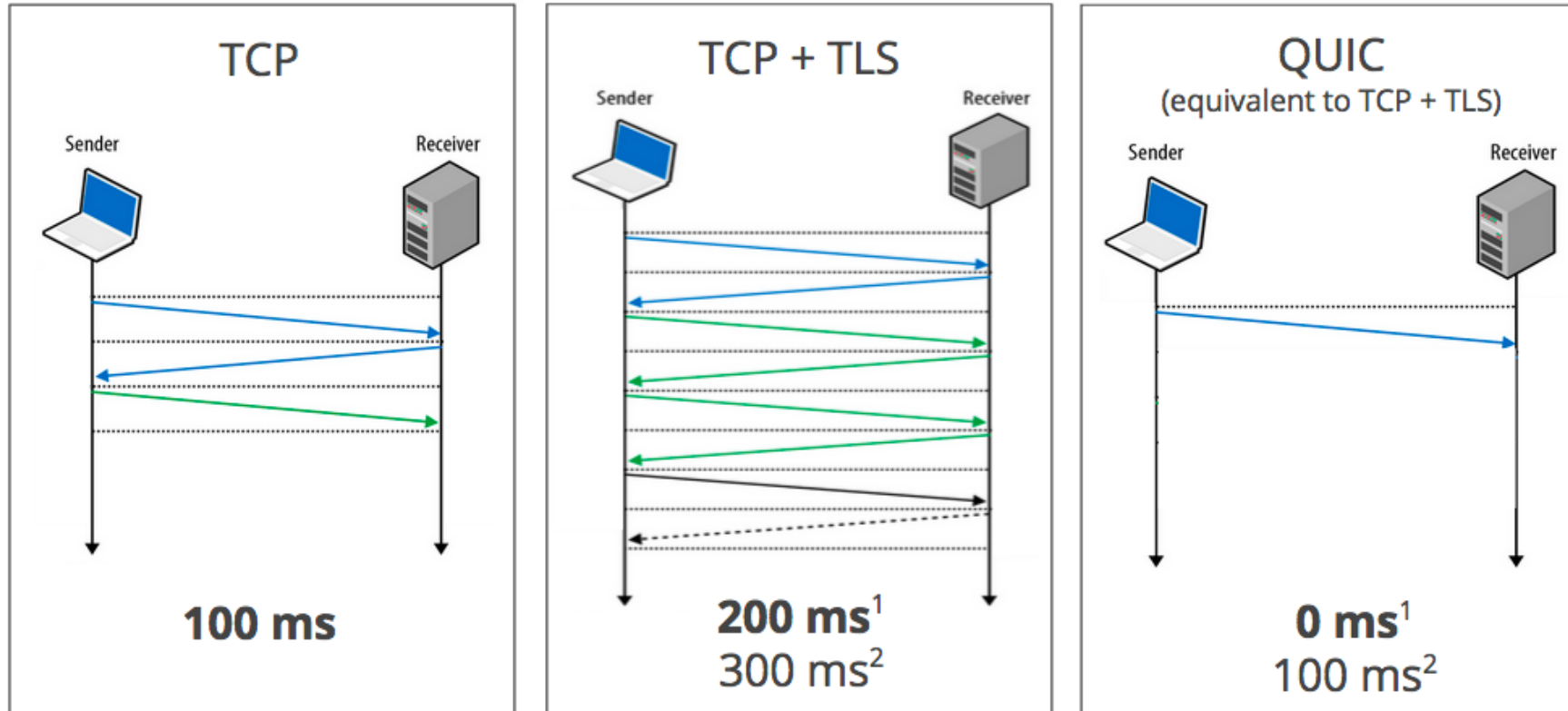
QUIC

MPTCP

BBR

QUIC

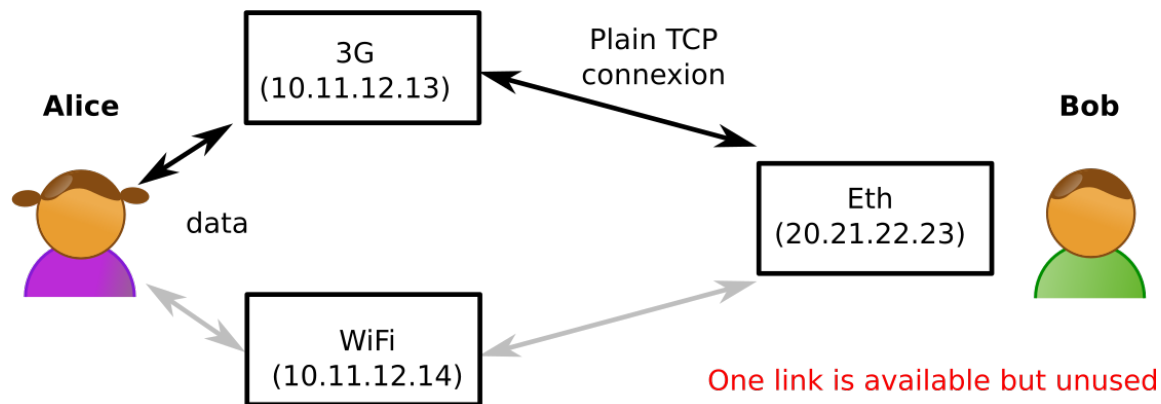
Zero RTT Connection Establishment



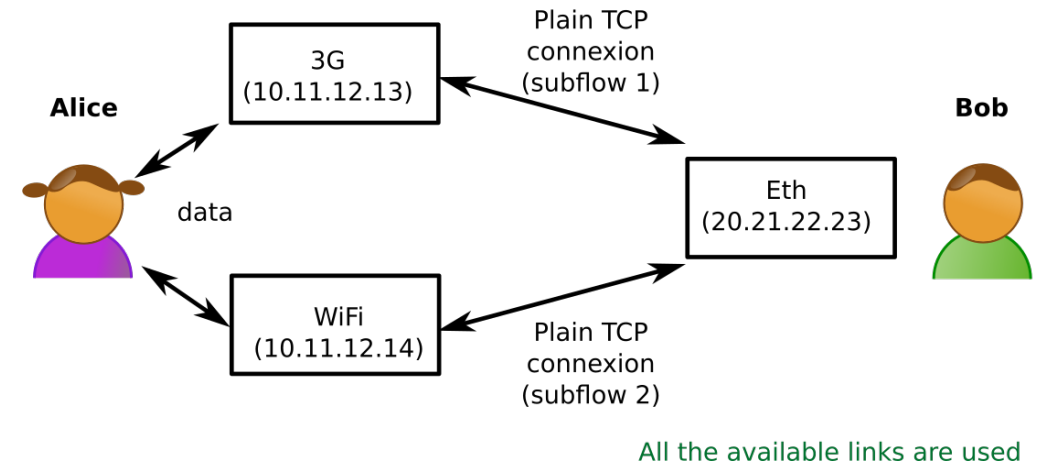
1. Repeat connection
2. Never talked to server before

MPTCP: Multipath TCP

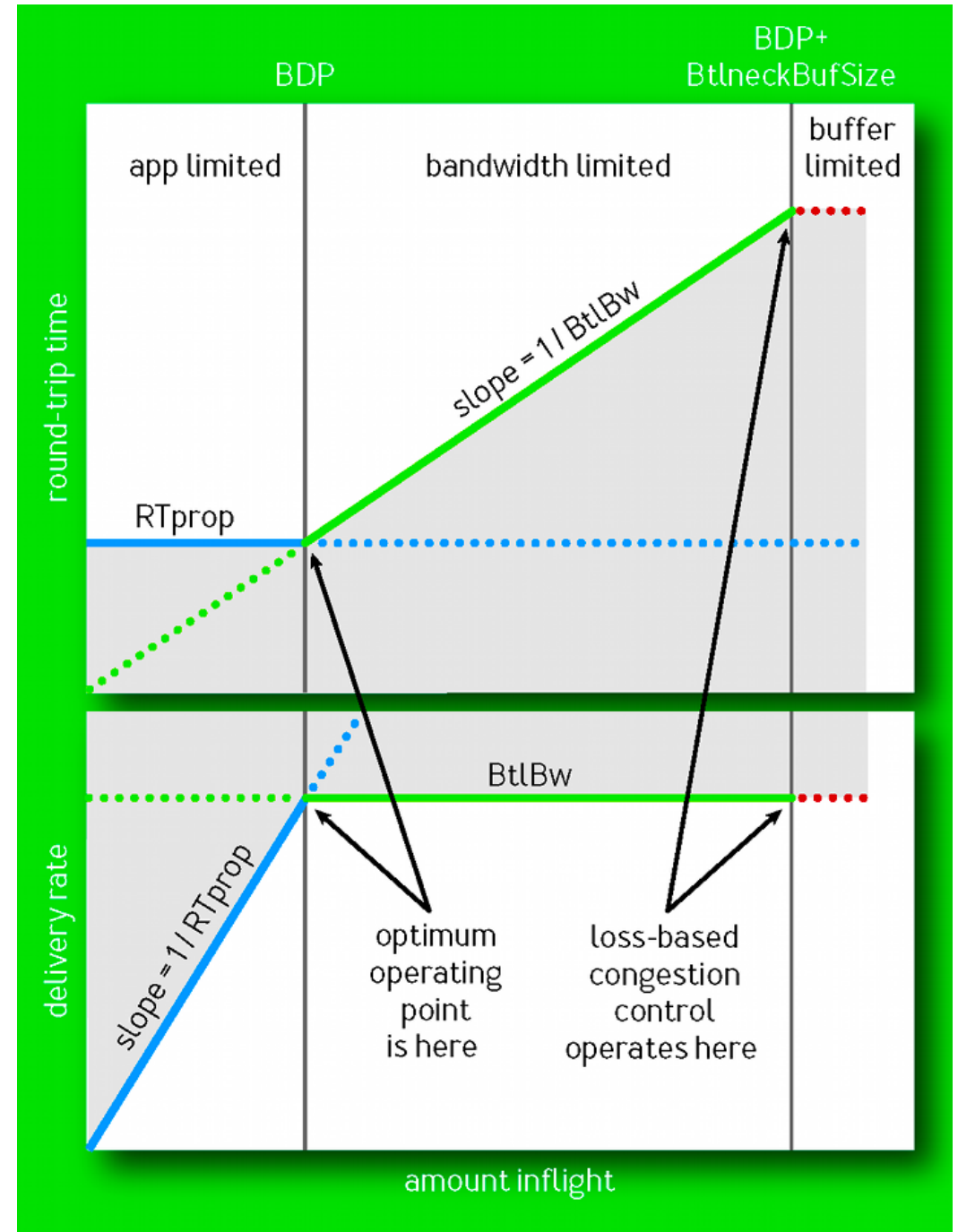
Data transmission with plain TCP



Data transmission with MPTCP



BBR: Bottleneck Bandwidth and Round trip propagation



Recap: Transport protocols

Goal: Provide end-to-end message delivery to applications

- Reliable (or not), messages vs streams

Challenges:

- Dealing with packet losses
- Dealing with slow receivers (flow control) and network (congestion control)
- Adapting to network conditions
 - Determine the right sending rate for yourself
 - Individual behaviors resulting in efficient and fair resource use

Toolbox

- Timeouts/retransmissions, sliding windows, max-min fairness, AIMD,