

CSE 461: Computer networks

Spring 2021

Ratul Mahajan

Why is Bandwidth Allocation hard?

- Number of senders and their offered load changes
- Senders may be limited in other ways
 - Other parts of network or by applications
- Network is distributed; no single party has an overall picture of its state

Bandwidth Allocation Solution Context

In networks without admission control (e.g., Internet)
Transport and Network layers must work together

- Network layer sees congestion
 - Only it can provide direct feedback
- Transport layer causes congestion
 - Only it can reduce load

Bandwidth Allocation Solution Overview

- Senders adapt concurrently based on their own view of the network
- Design this adaptation so the network usage as a whole is efficient and fair
 - In practice, efficiency is more important than fairness
- Adaptation is continuous since offered loads continue to change over time

Bandwidth Allocation Models

- Open loop versus closed loop
 - Open: reserve bandwidth before use
 - Closed: use feedback to adjust rates
- Host versus Network support
 - Who is sets/enforces allocations?
- Window versus Rate based
 - How is allocation expressed?

TCP is a closed loop, host-driven, and window-based

Bandwidth Allocation Models (2)

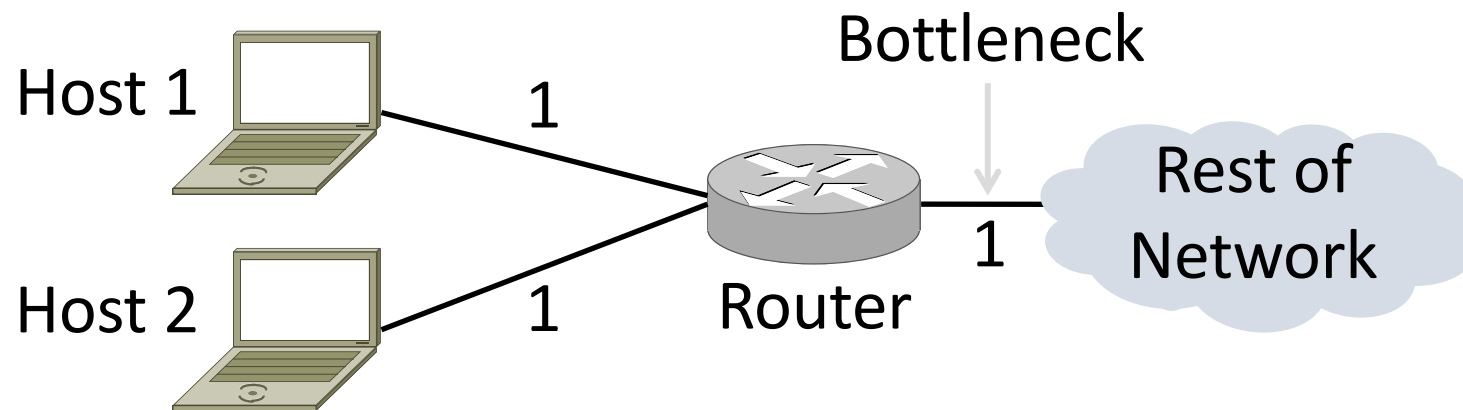
- We'll study closed-loop, host-driven, and window-based too
- Network layer returns feedback on current allocation to senders
 - At least tells if there is congestion
- Transport layer adjusts sender's behavior via window in response
 - How senders adapt is a control law

Additive Increase Multiplicative Decrease

- AIMD is a control law hosts can use to reach a good allocation
 - Hosts additively increase rate while network not congested
 - Hosts multiplicatively decrease rate when congested
 - Used by TCP
- Let's explore the AIMD game ...

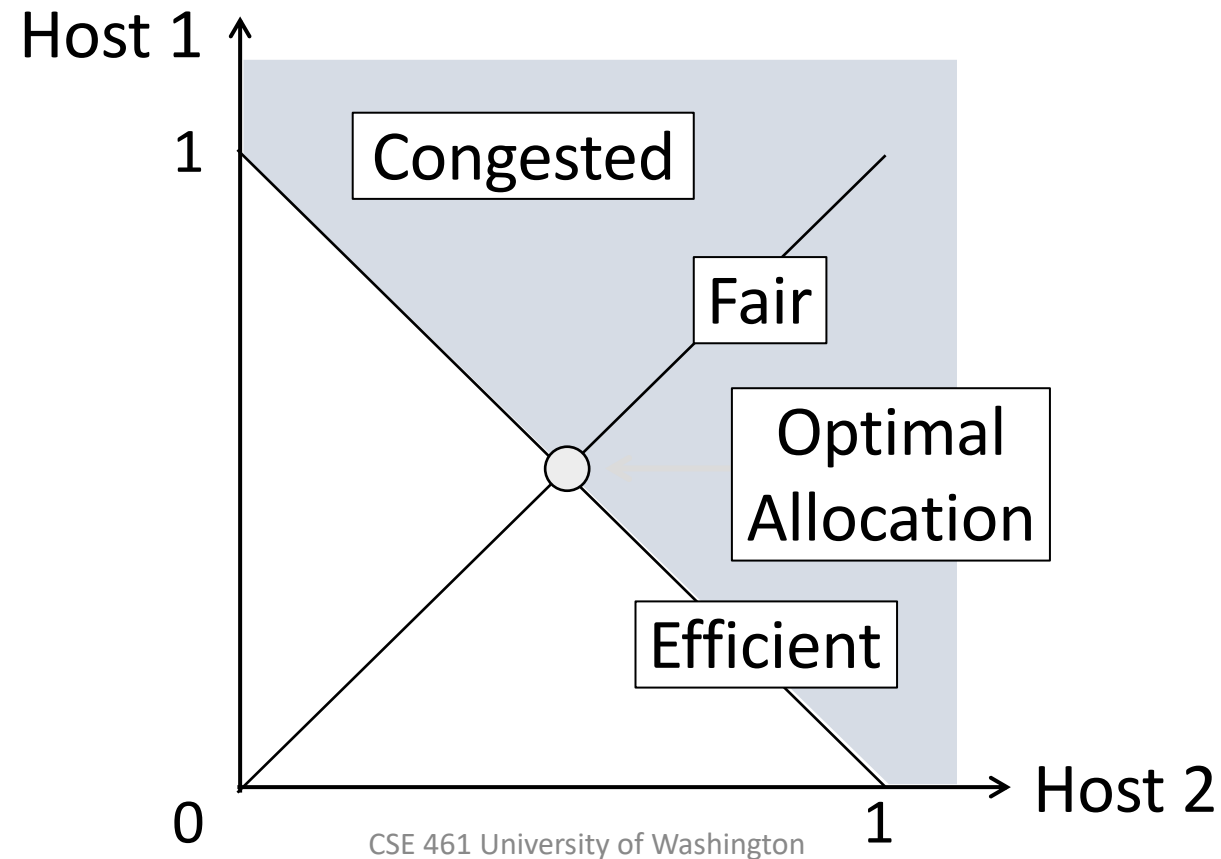
AIMD Game

- Hosts 1 and 2 share a bottleneck
 - But do not talk to each other directly
- Router provides binary feedback
 - Tells hosts if network is congested



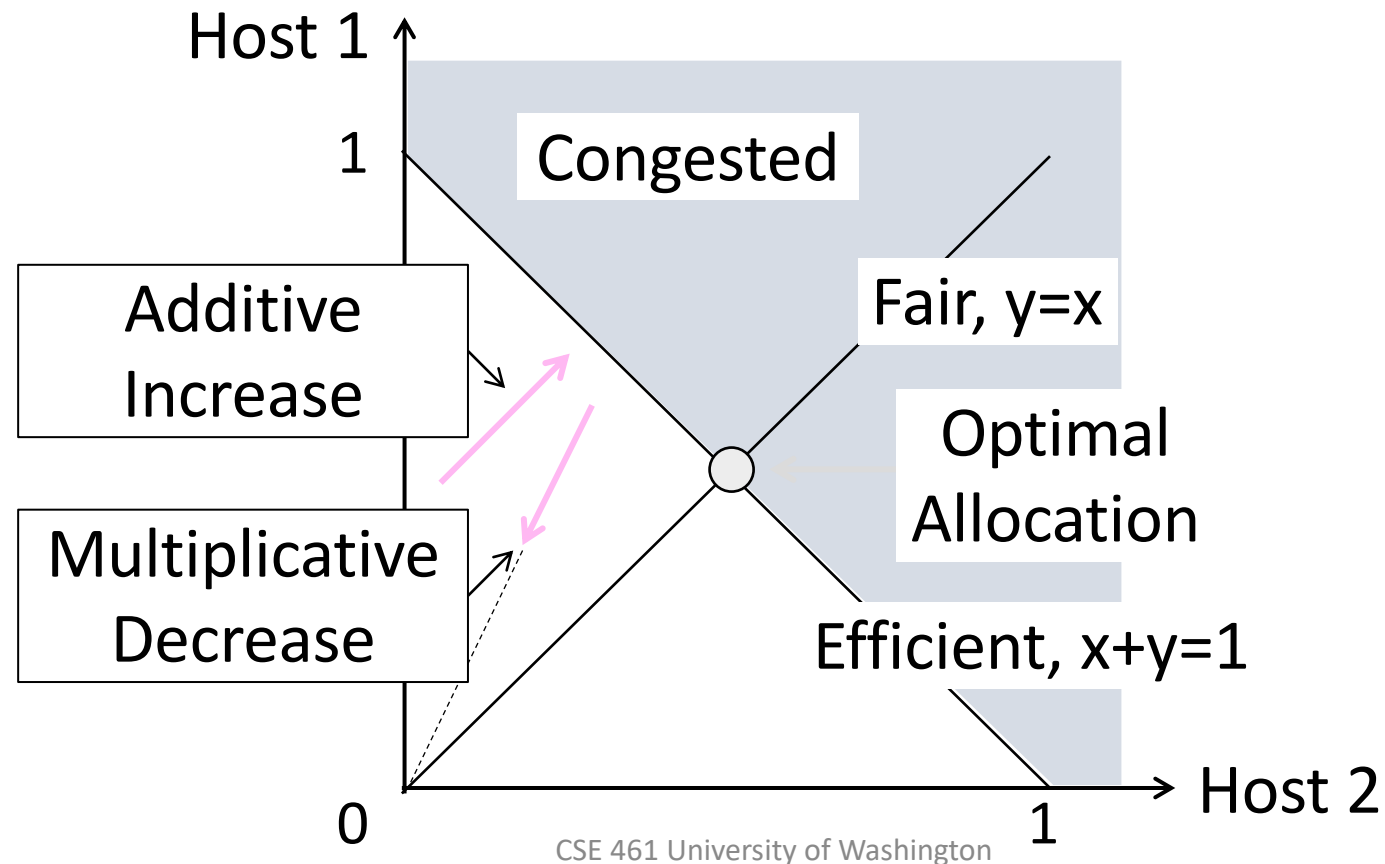
AIMD Game (2)

- Each point is a possible allocation



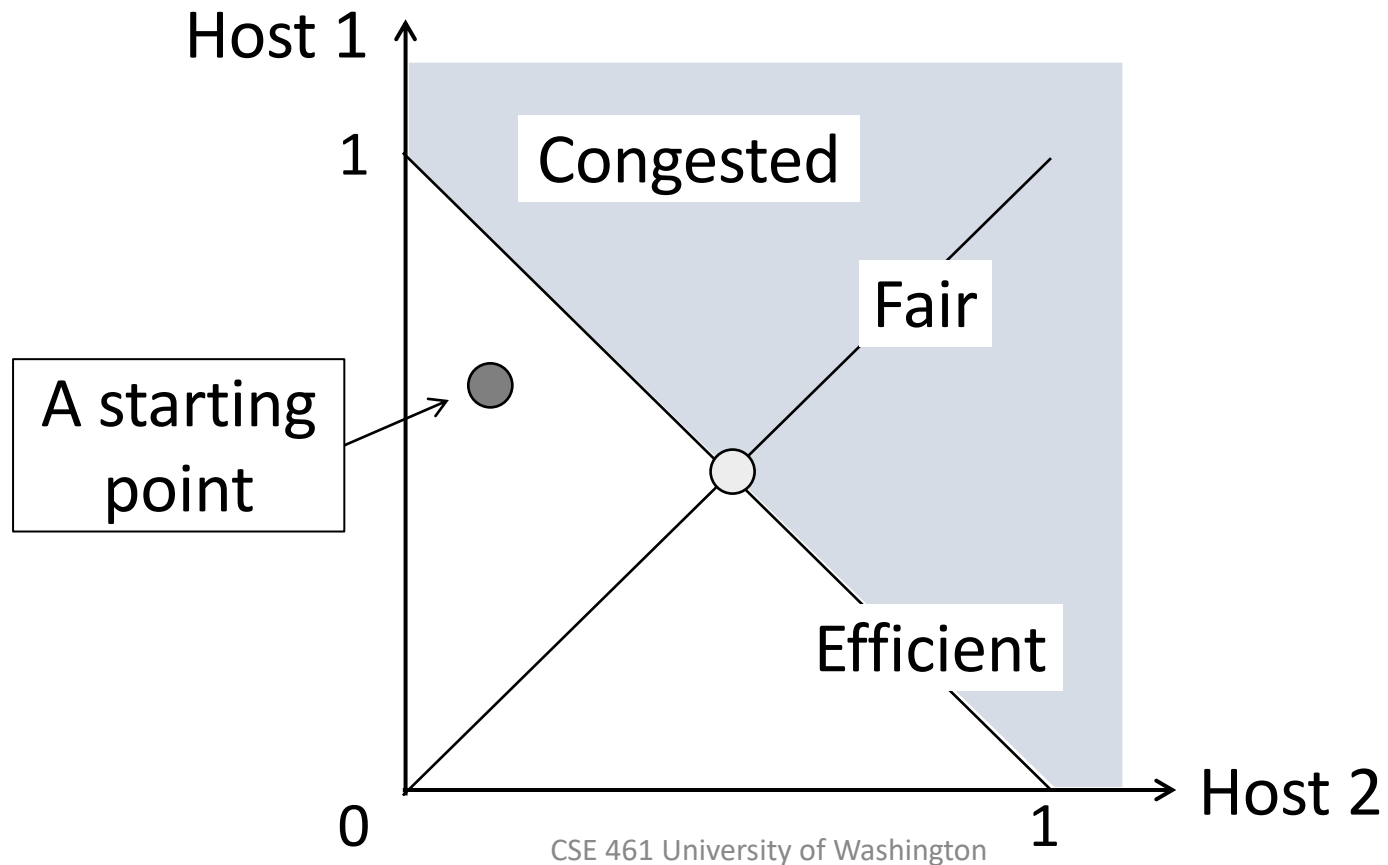
AIMD Game (3)

- AI and MD move the allocation



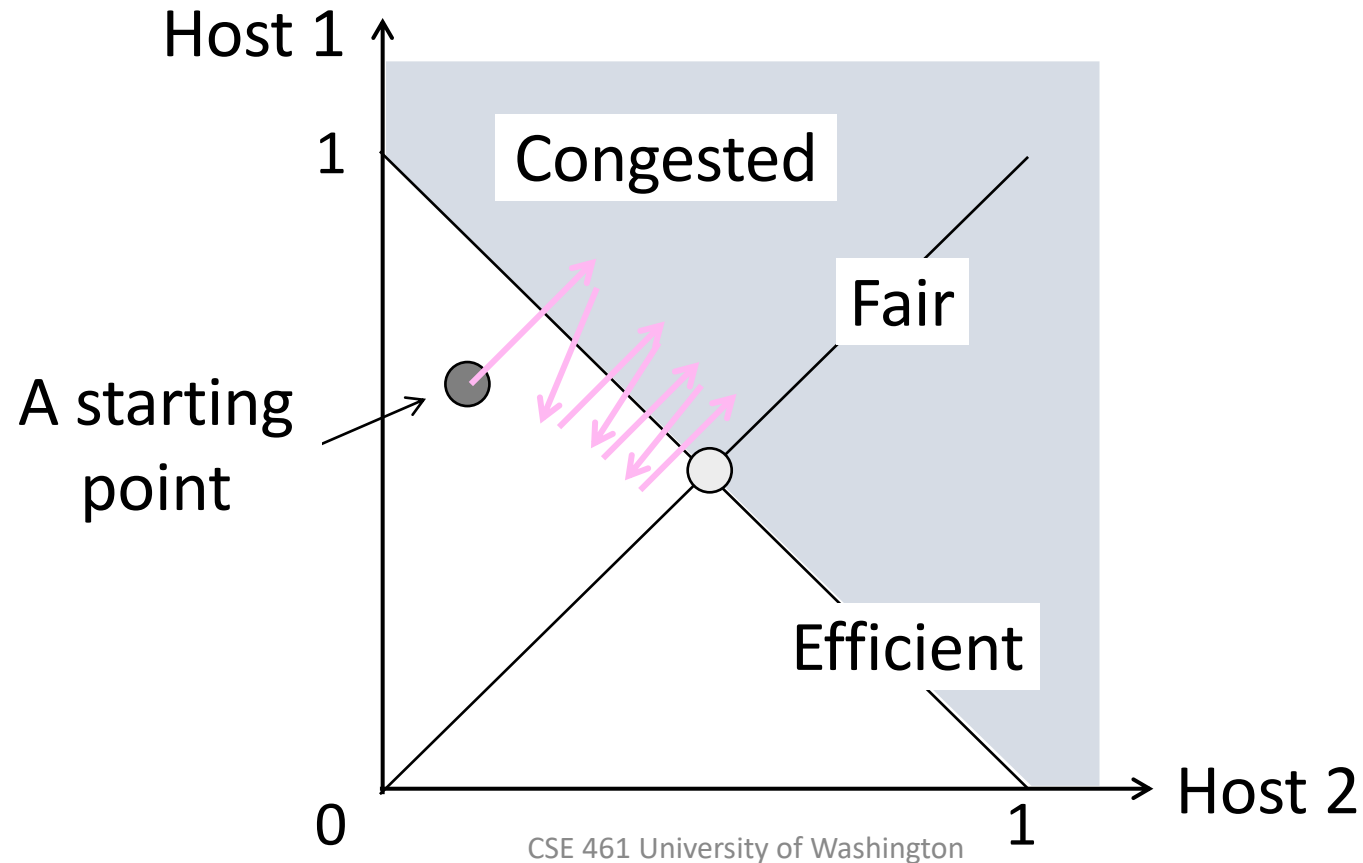
AIMD Game (4)

- Play the game!



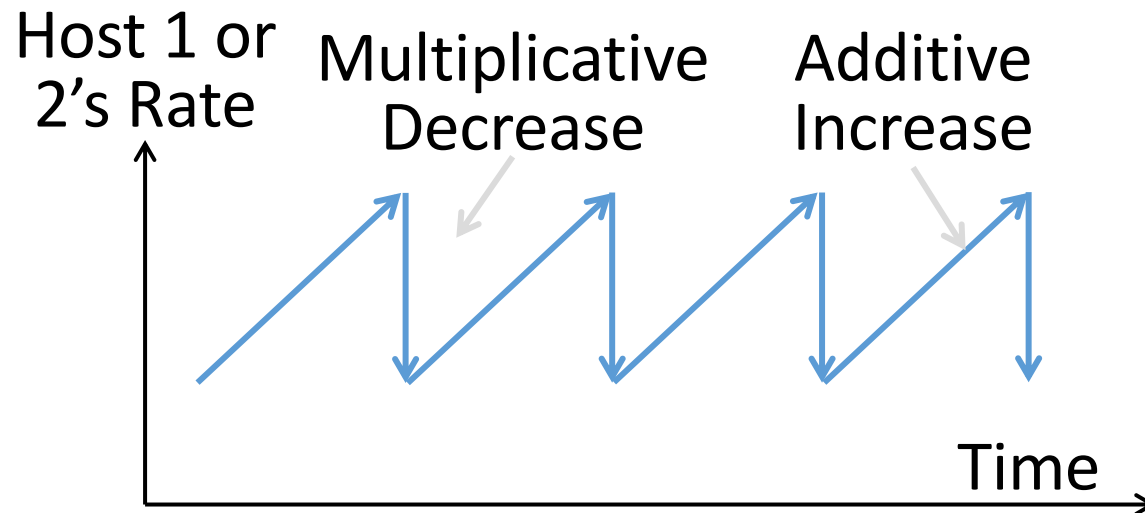
AIMD Game (5)

- Always converge to good allocation!



AIMD Sawtooth

- Produces a “sawtooth” pattern over time for rate of each host
 - This is the TCP sawtooth (later)



AIMD Properties

- Converges to an allocation that is efficient and fair when hosts run it
 - Holds for more general topologies
- Other increase/decrease control laws do not! (Try MIAD, MIMD, MIAD)
- Requires only binary feedback from the network

Feedback Signals

- Several possible signals, with different pros/cons
 - We'll look at classic TCP that uses packet loss as a signal

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

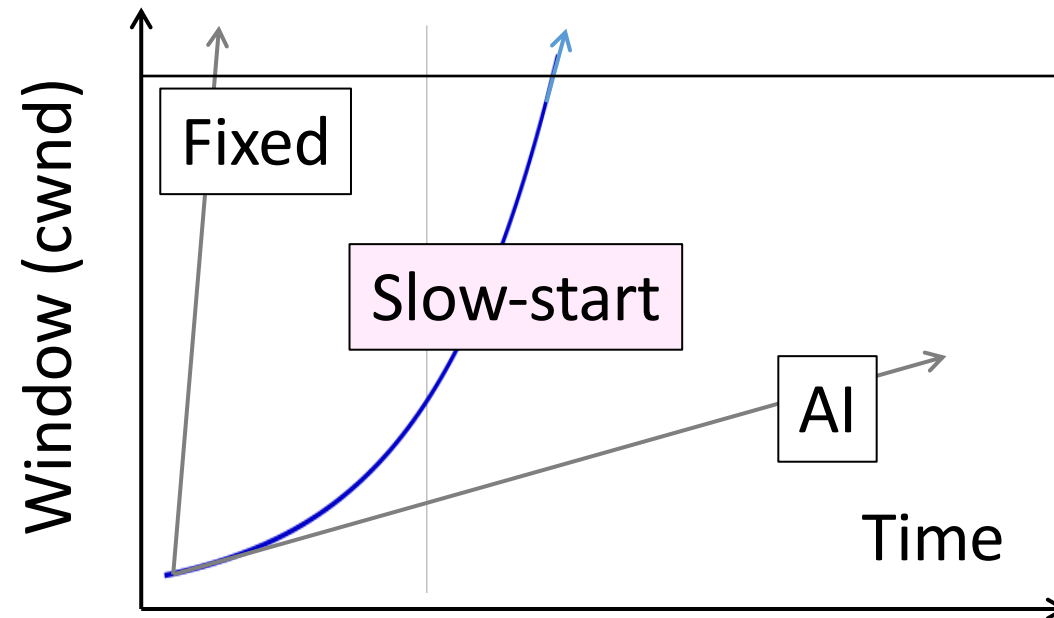
Slow Start (TCP Additive Increase)

TCP “Slow Start” Problem

- We want to quickly reach the right rate, $cwnd_{IDEAL}$, but it varies greatly
 - Fixed sliding window doesn't adapt and is rough on the network (loss!)
 - Additive Increase with small bursts adapts $cwnd$ gently, but might take a long time to become efficient

Slow-Start Solution

- Start by doubling cwnd every RTT
 - Exponential growth (1, 2, 4, 8, 16, ...)
 - Start slow, quickly reach large values

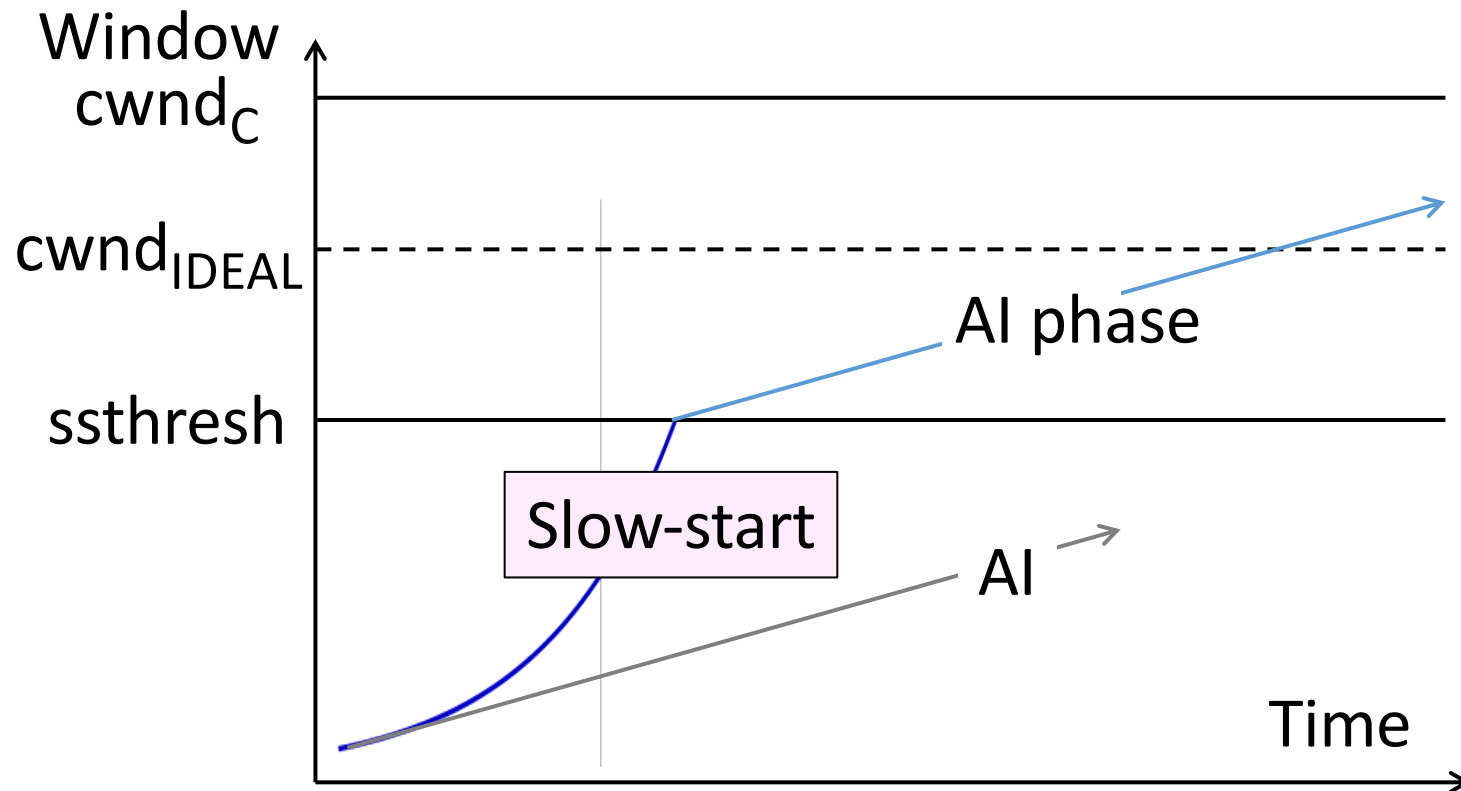


Slow-Start Solution (2)

- Eventually packet loss will occur when the network is congested
 - Loss timeout tells us cwnd is too large
 - Next time, switch to AI beforehand
 - Slowly adapt cwnd near right value
- In terms of cwnd:
 - Expect loss for $\text{cwnd}_c \approx 2BD + \text{queue}$
 - Use $\text{ssthresh} = \text{cwnd}_c / 2$ to switch to AI

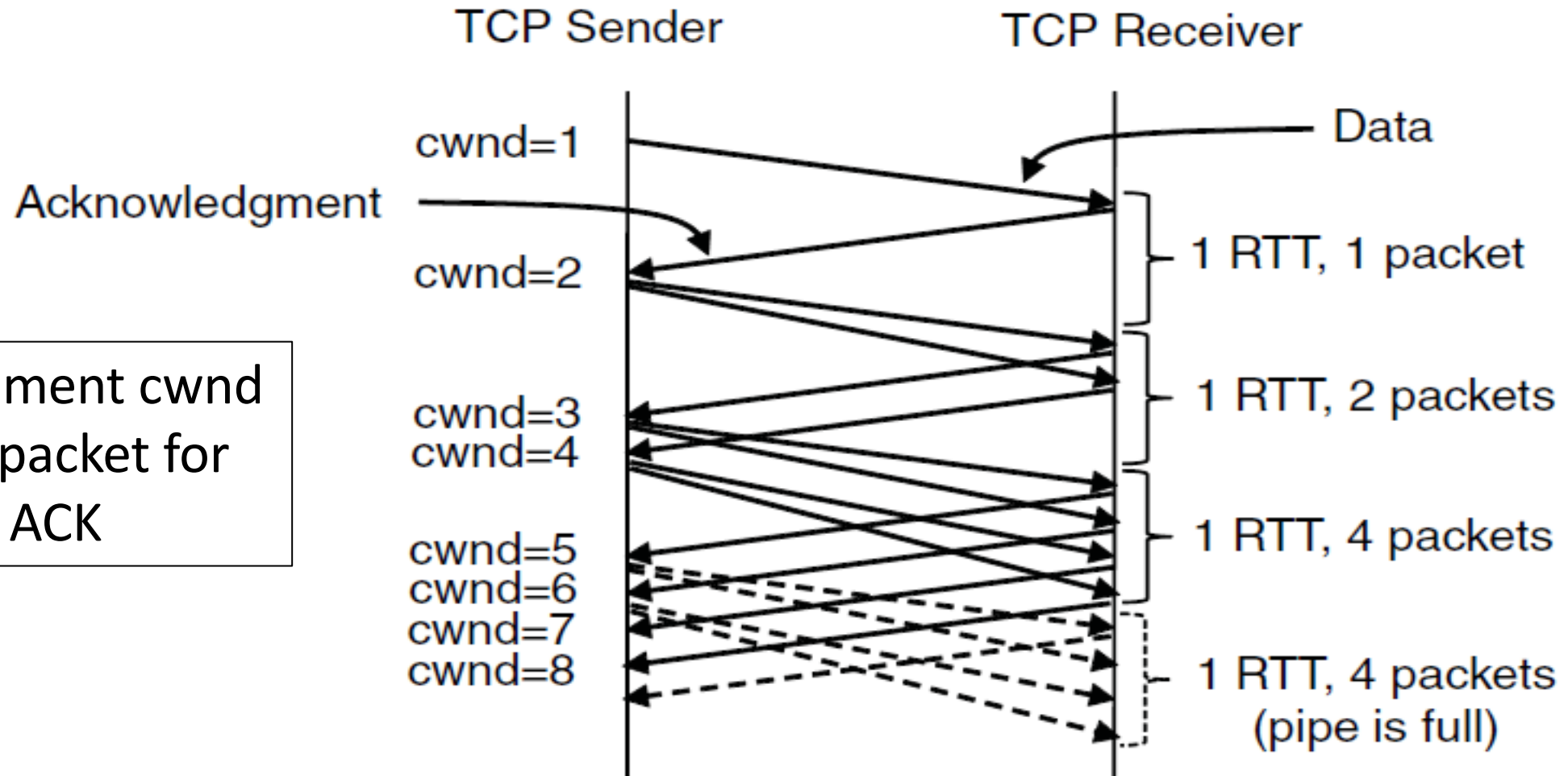
Slow-Start Solution (3)

- Combined behavior, **after first time**
 - Most time spent near right value



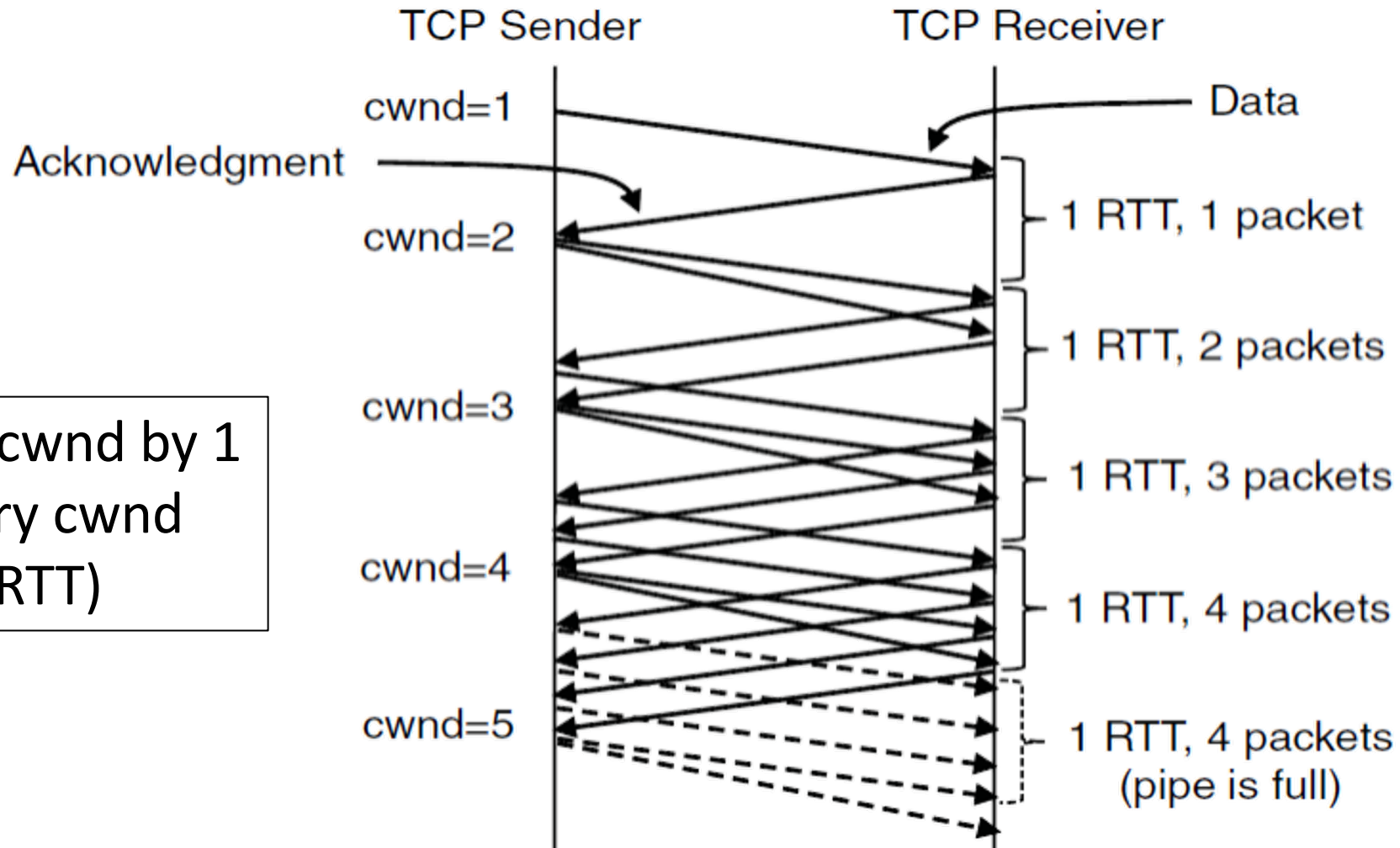
Slow-Start (Doubling) Timeline

Increment cwnd by 1 packet for each ACK



Additive Increase Timeline

Increment cwnd by 1 packet every cwnd ACKs (or 1 RTT)



TCP Tahoe (Implementation)

- Initial slow-start (doubling) phase
 - Start with $\text{cwnd} = 1$ (or small value)
 - $\text{cwnd} += 1$ packet per ACK
- Later Additive Increase phase
 - $\text{cwnd} += 1/\text{cwnd}$ packets per ACK
 - Roughly adds 1 packet per RTT
- Switching threshold (initially infinity)
 - Switch to AI when $\text{cwnd} > \text{ssthresh}$
 - Set $\text{ssthresh} = \text{cwnd}/2$ after loss
 - Begin with slow-start after timeout

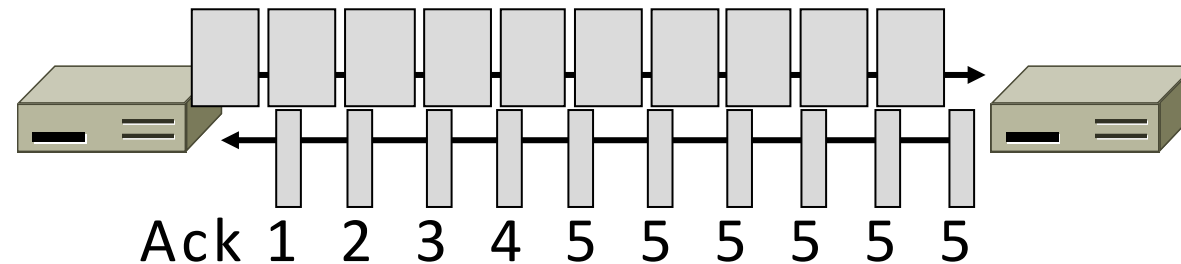
Fast Recovery
(TCP Multiplicative Decrease)

Inferring Loss from ACKs

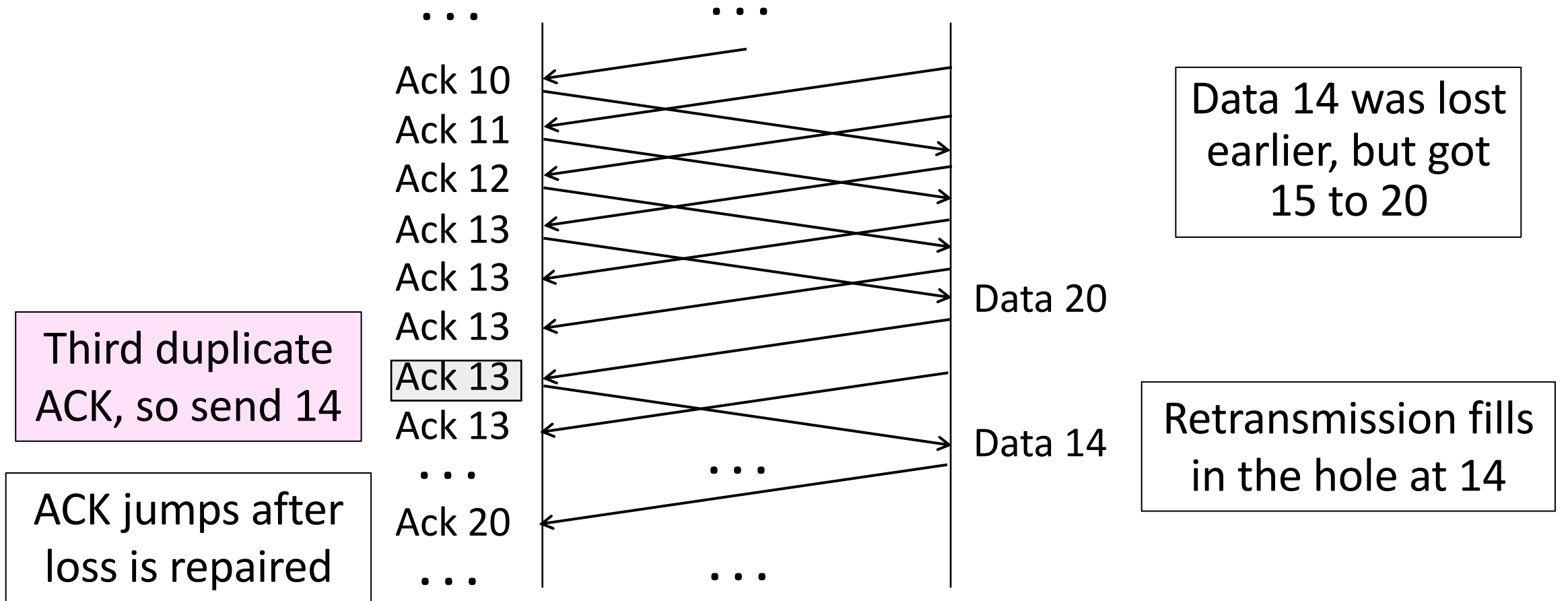
- TCP uses a cumulative ACK
 - Carries highest in-order seq. number
 - Normally a steady advance
- Duplicate ACKs give us hints about what data hasn't arrived
 - Tell us some new data did arrive, but it was not next segment
 - Thus the next segment may be lost

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)

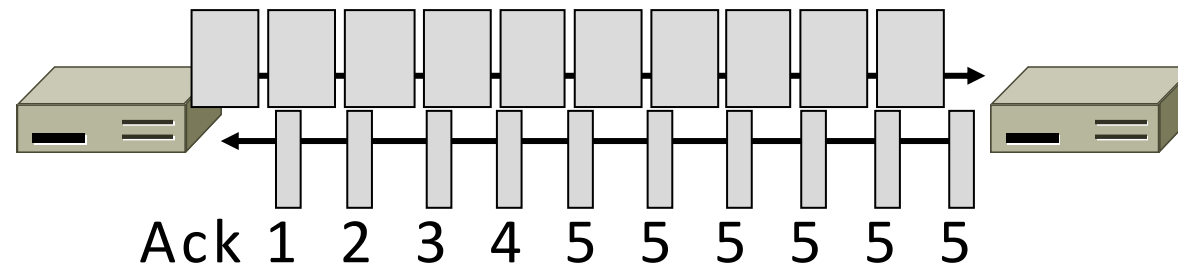
- It can repair single segment loss quickly, typically before a timeout
- However, we have quiet time at the sender/receiver while waiting for the ACK to jump
- And we still need to MD cwnd ...

Inferring Non-Loss from ACKs

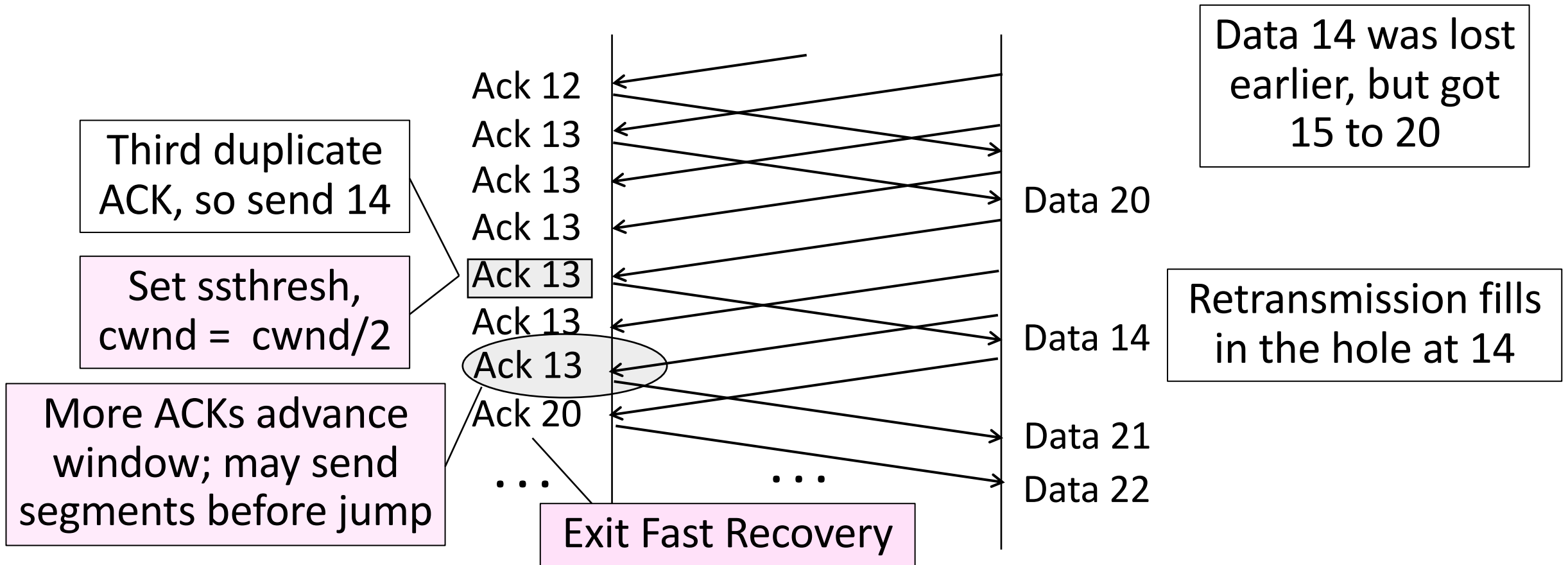
- Duplicate ACKs also give us hints about what data has arrived
 - Each new duplicate ACK means that some new segment has arrived
 - It will be the segments after the loss
 - Thus advancing the sliding window will not increase the number of segments stored in the network

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



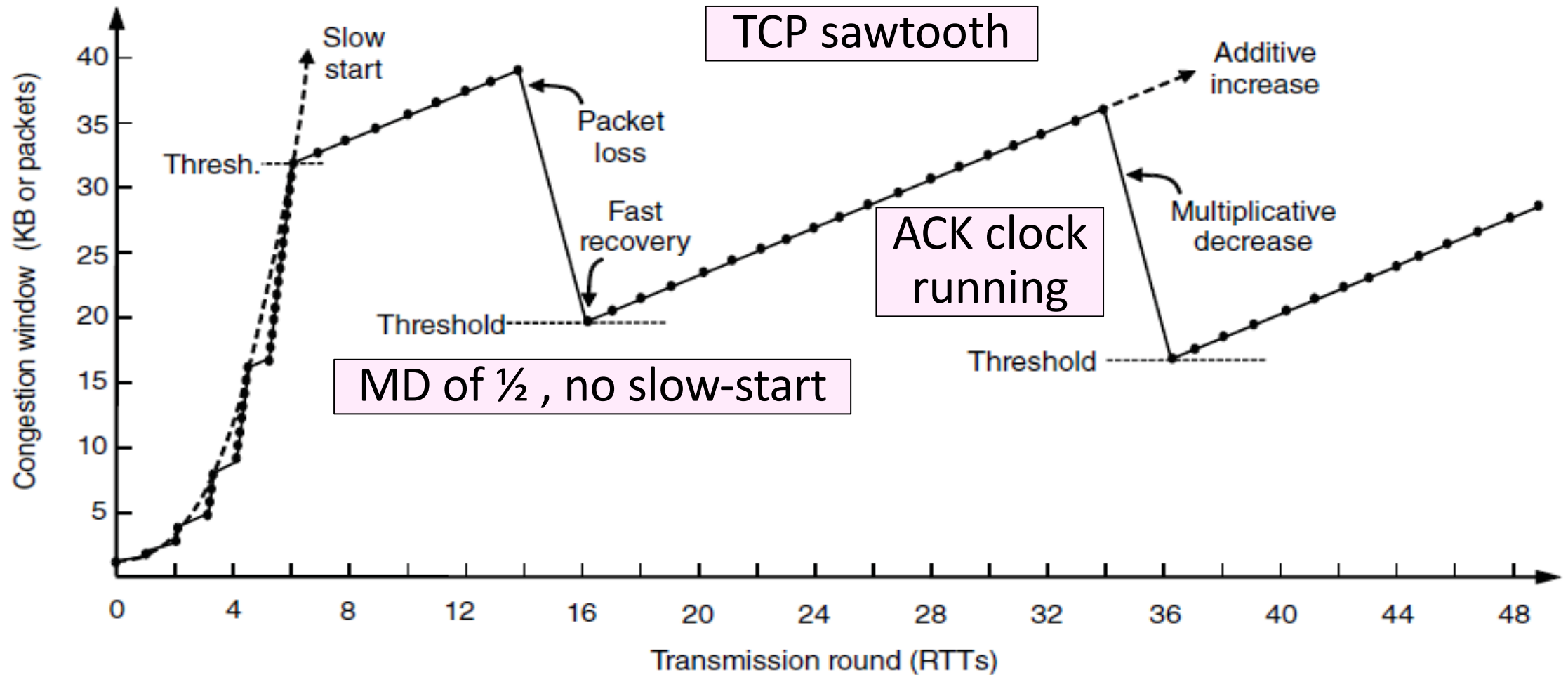
Fast Recovery (2)



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?

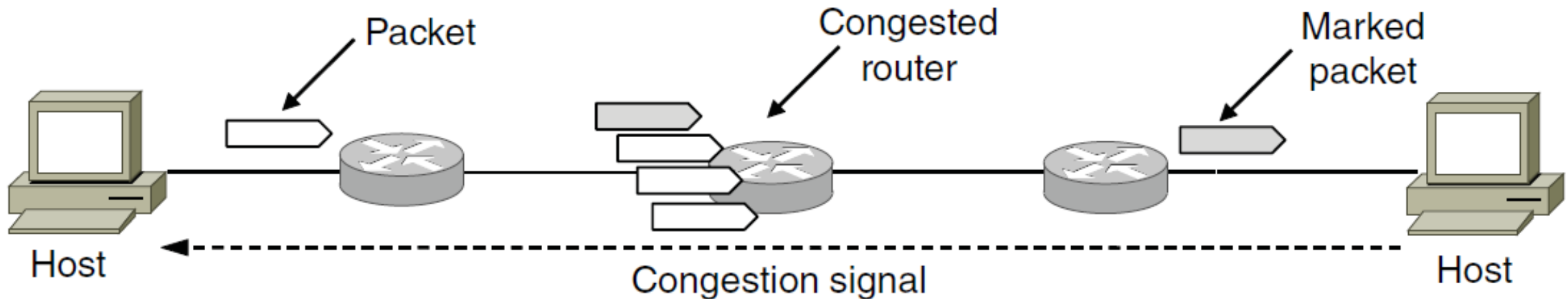
Feedback Signals

- 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

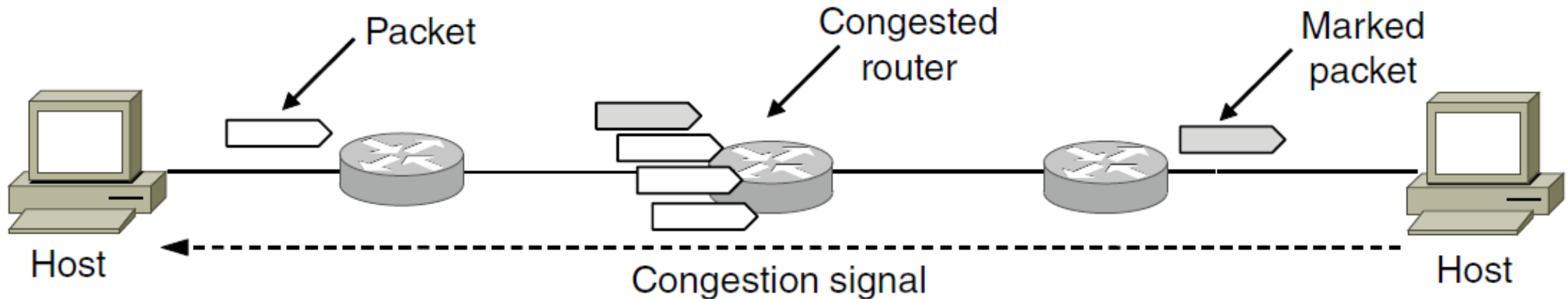
ECN (Explicit Congestion Notification)

- 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

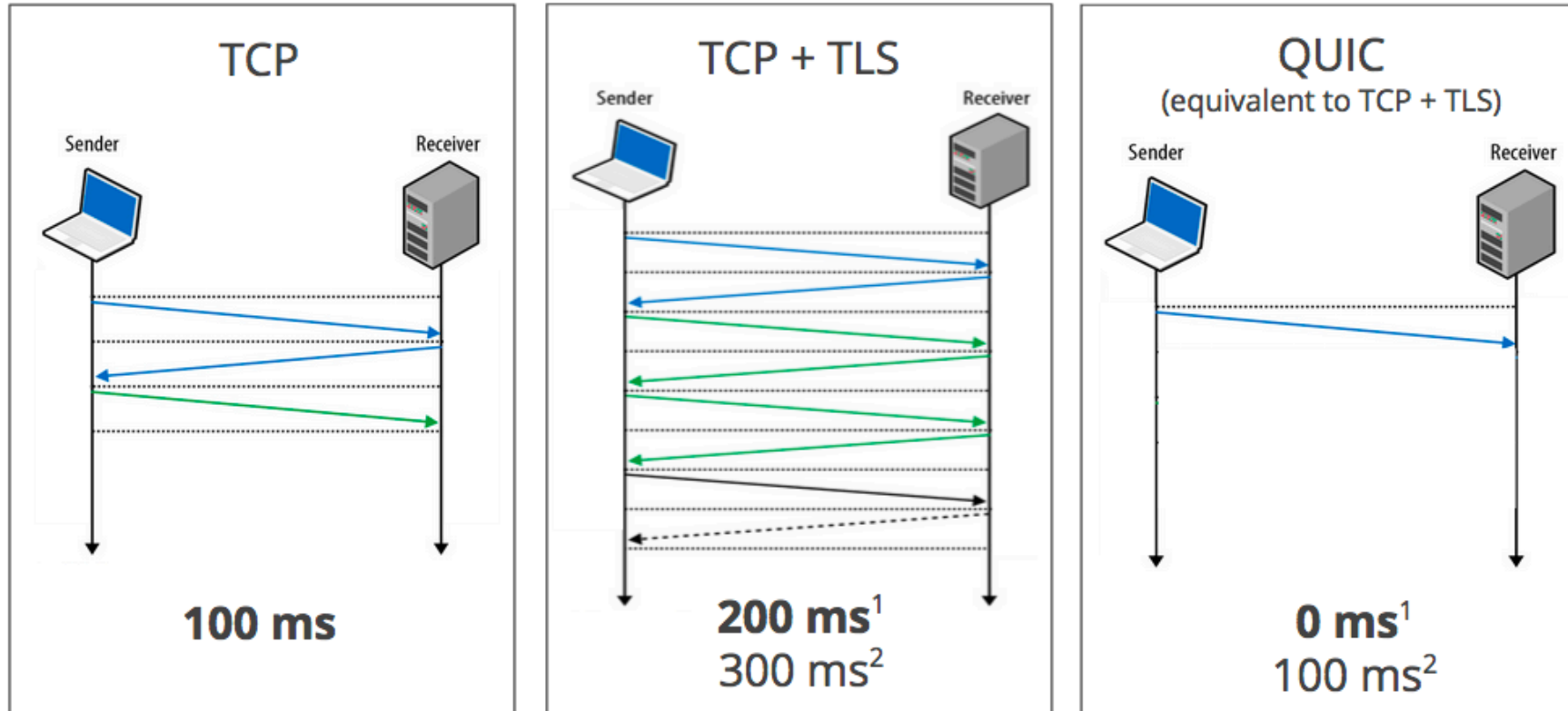
MPTCP

BBR

DCTCP

QUIC

Zero RTT Connection Establishment

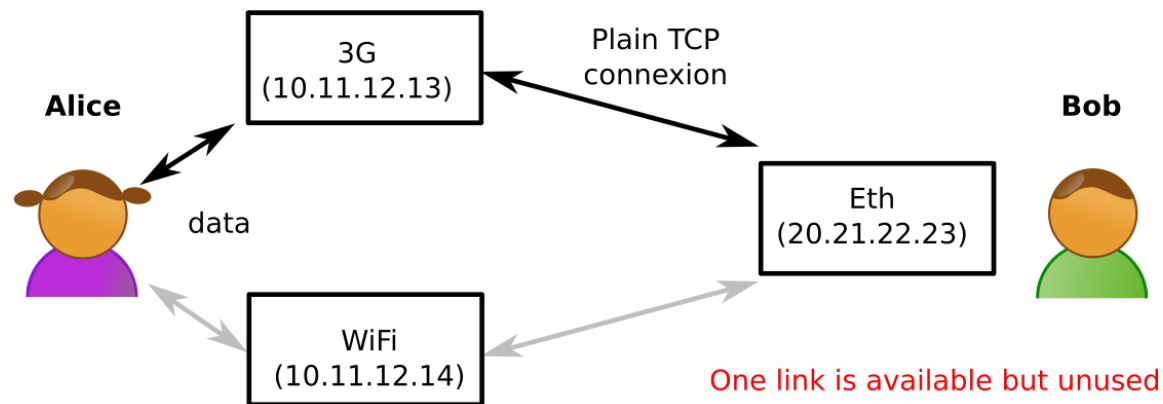


1. Repeat connection
2. Never talked to server before

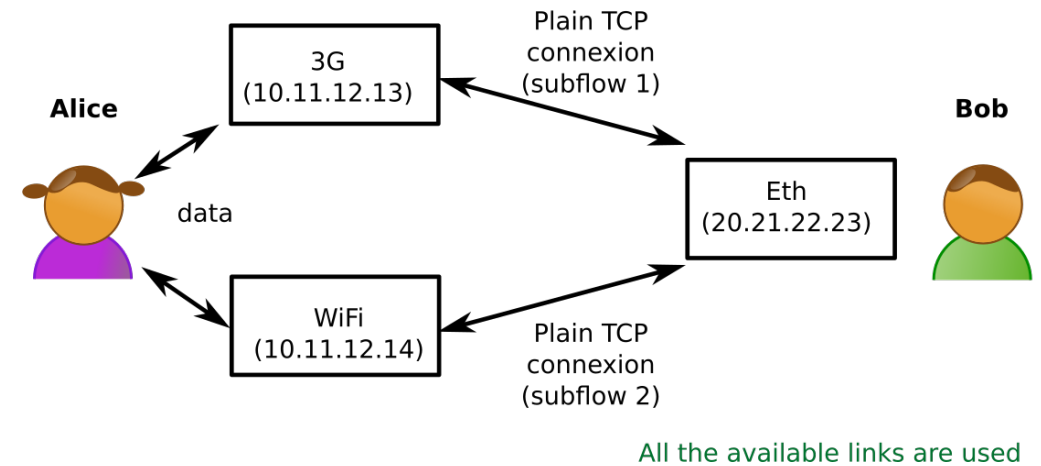
<https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html>

MPTCP: Multipath TCP

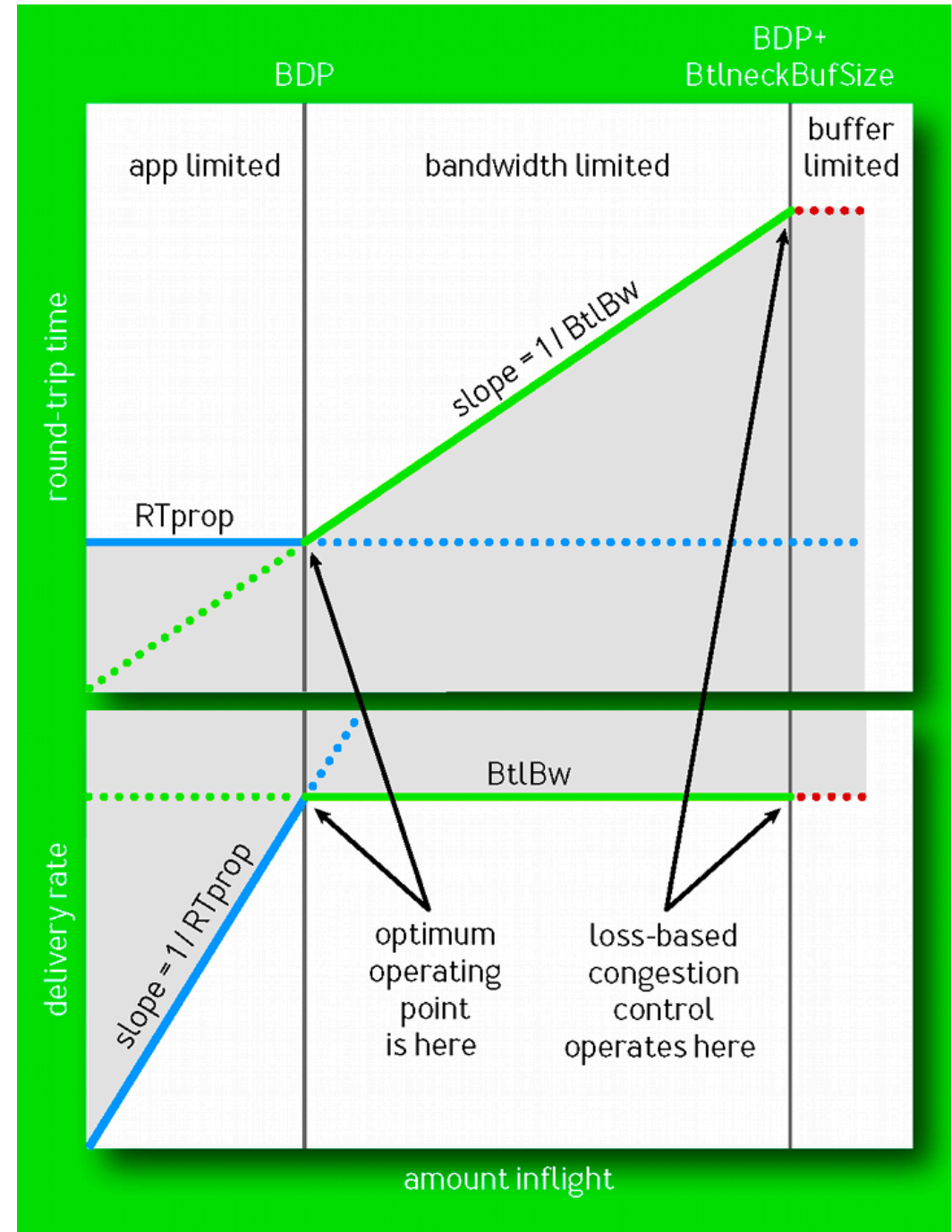
Data transmission with plain TCP



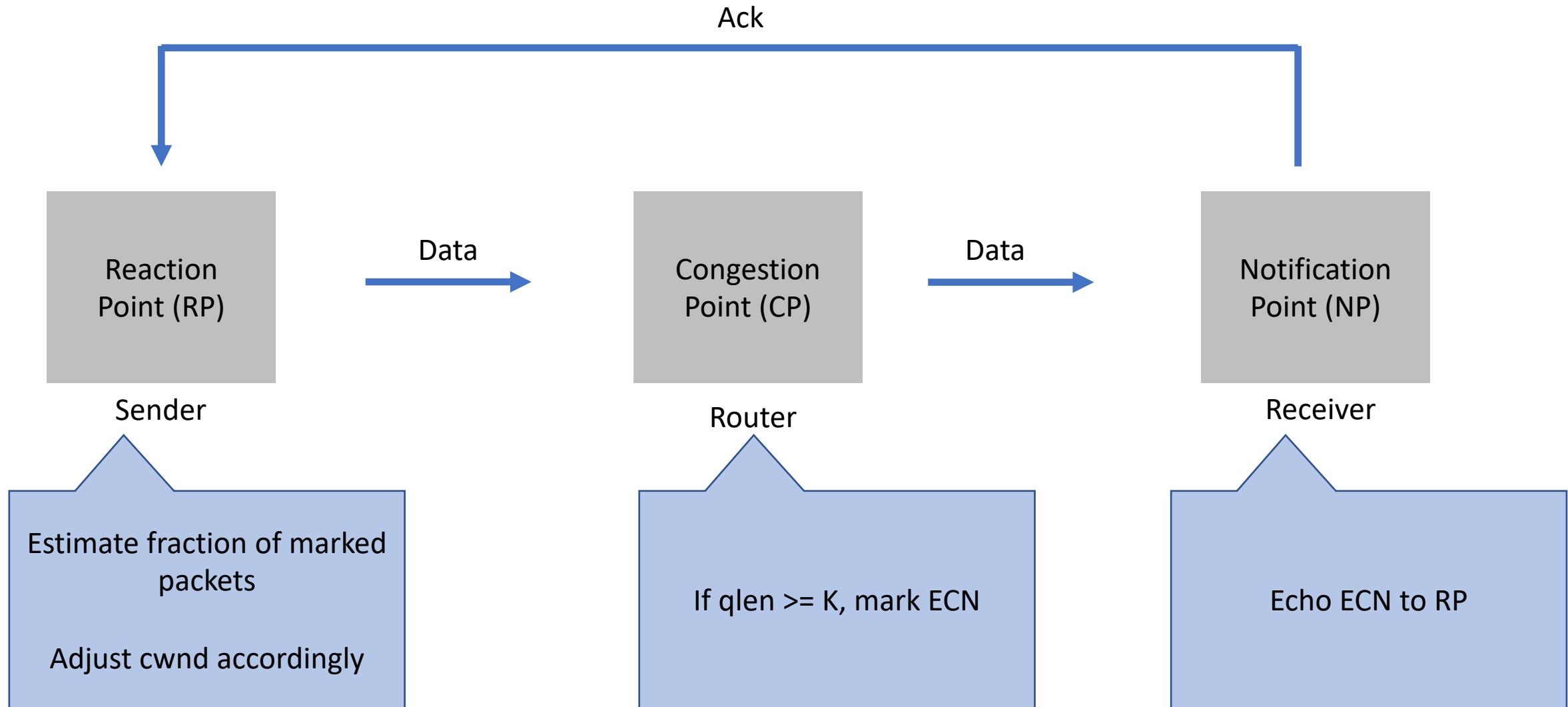
Data transmission with MPTCP



BBR: Bottleneck Bandwidth and Round trip propagation



DCTCP at a glance



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,