

Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- **TCP congestion control**
- Evolution of transport-layer functionality



TCP congestion control

- “Classic” TCP: loss-based, end-to-end

Additive increase, multiplicative decrease

“Slow” start and CUBIC

- Enhanced TCPs:

Delay-based congestion control TCP and Explicit congestion notification

- TCP fairness
- Back in the 1980s congestion collapse was actually beginning to be observed within the network.
- In 1988 a networking researcher Van Jacobson published a paper on congestion avoidance and control
- The same mechanism in place now for 35 years even as the amount of traffic carried by the Internet has gone up by more than factor of a billion.

TCP congestion control: AIMD

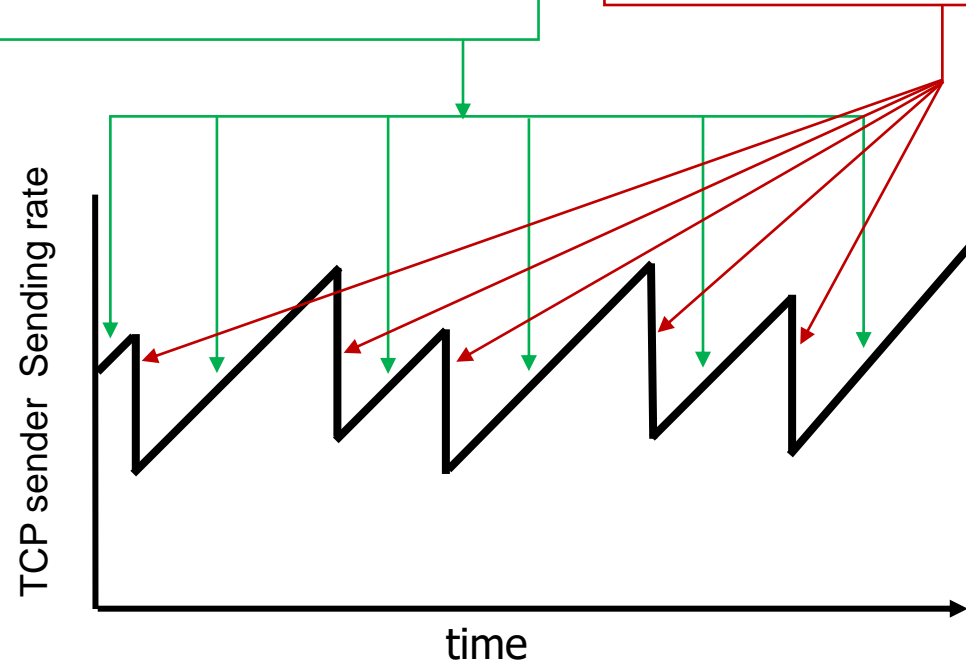
- *approach*: senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

TCP AIMD: more

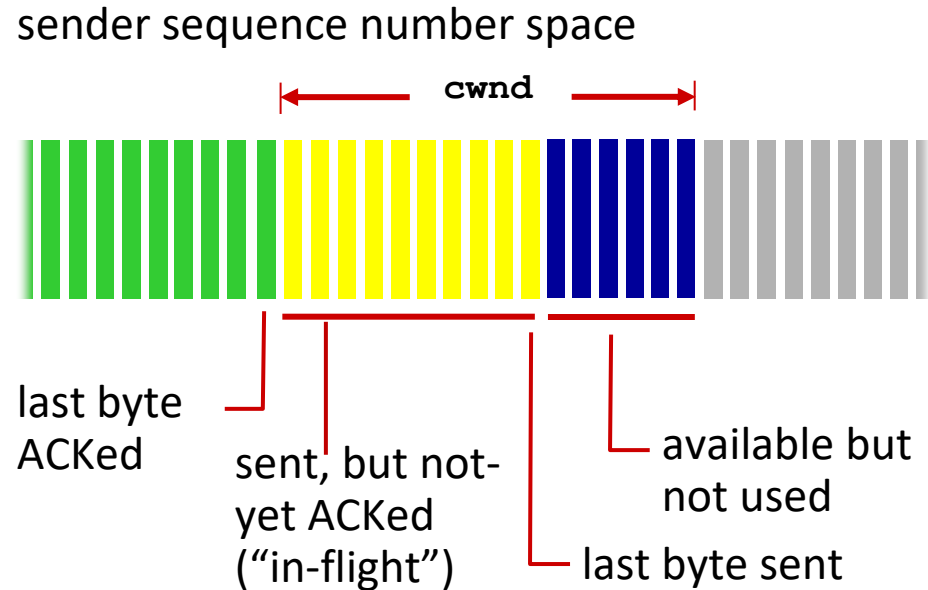
Multiplicative decrease detail: sending rate is

- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

Why AIMD?

- AIMD – a distributed, asynchronous algorithm – has been shown to:
 - optimize congested flow rates network wide!
 - have desirable stability properties

TCP congestion control: details



TCP sending behavior:

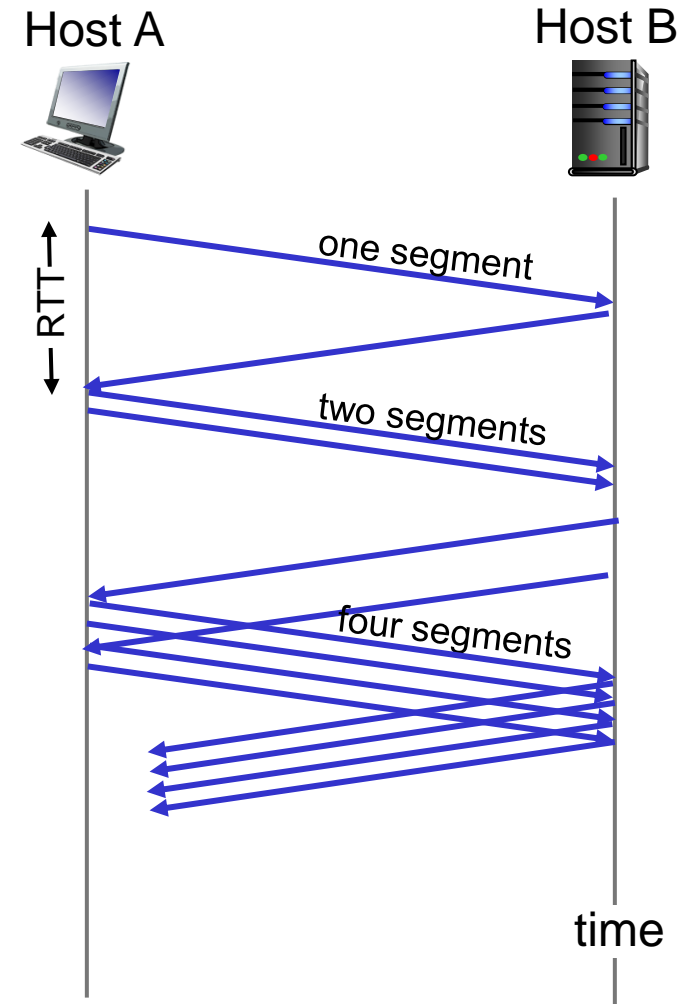
- *roughly*: send `cwnd` bytes, wait RTT for ACKS, then send more bytes

$$\text{TCP rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

- TCP sender limits transmission: $\text{LastByteSent} - \text{LastByteAcked} \leq \text{cwnd}$
- `cwnd` is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

TCP slow start

- when connection begins, increase rate exponentially until first loss event:
 - initially **cwnd** = 1 MSS
 - double **cwnd** every RTT
 - done by incrementing **cwnd** for every ACK received
- *summary*: initial rate is slow, but ramps up exponentially fast



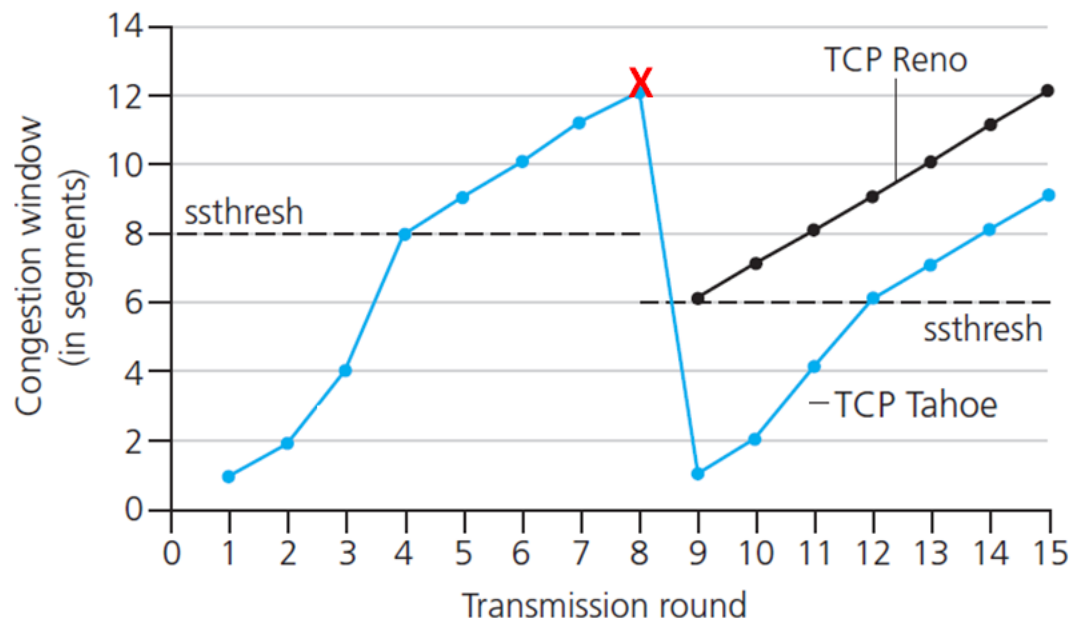
TCP: from slow start to congestion avoidance

Q: when should the exponential increase switch to linear?

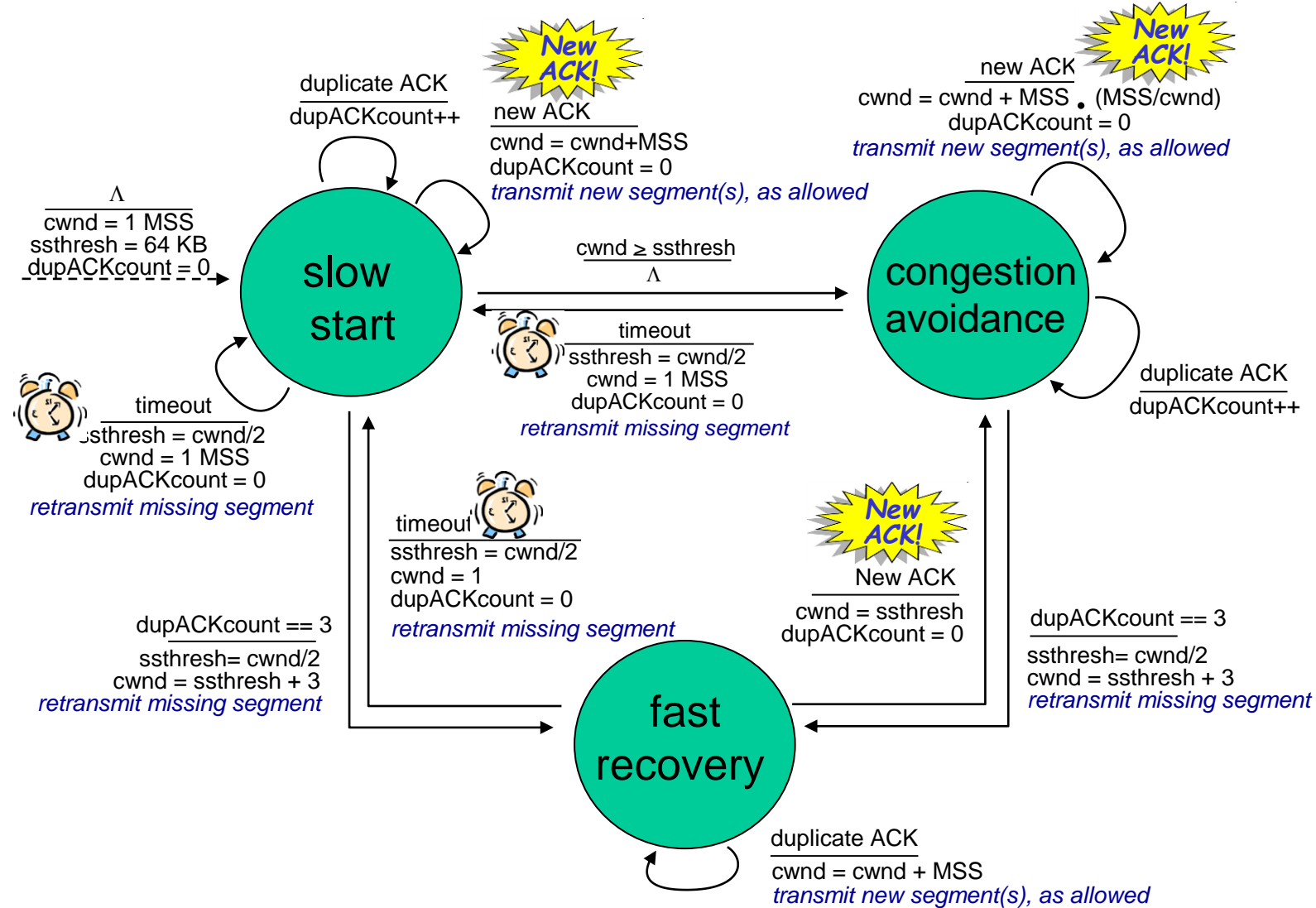
A: when **cwnd** gets to 1/2 of its value before timeout.

Implementation:

- variable **ssthresh**
- on loss event, **ssthresh** is set to 1/2 of **cwnd** just before loss event



Summary: TCP congestion control



TCP congestion control

- “Classic” TCP: loss-based, end-to-end

Additive increase, multiplicative decrease

“Slow” start and CUBIC

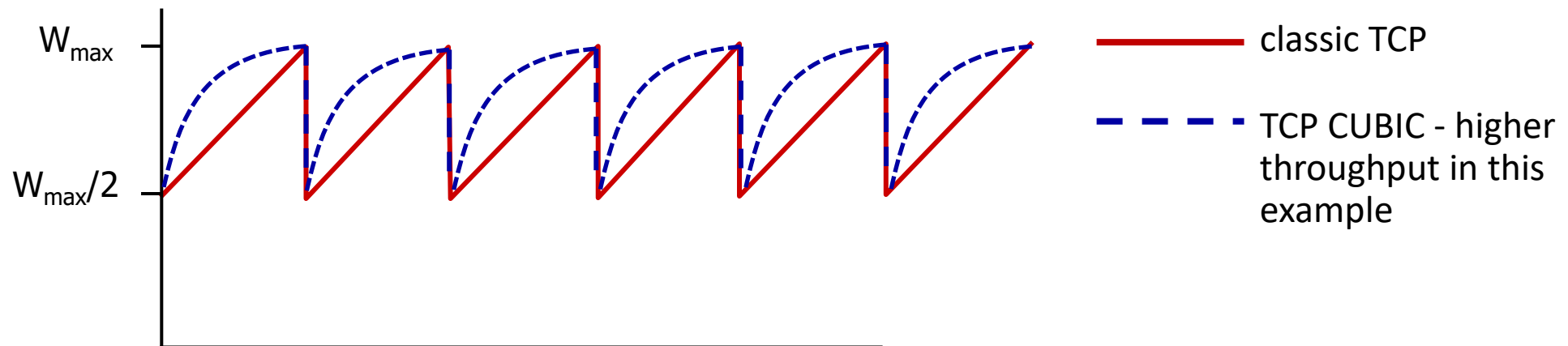
- Enhanced TCPs:

Delay-based congestion control TCP and Explicit congestion notification

- TCP fairness
- Evolution of transport-layer functionality
- Summary

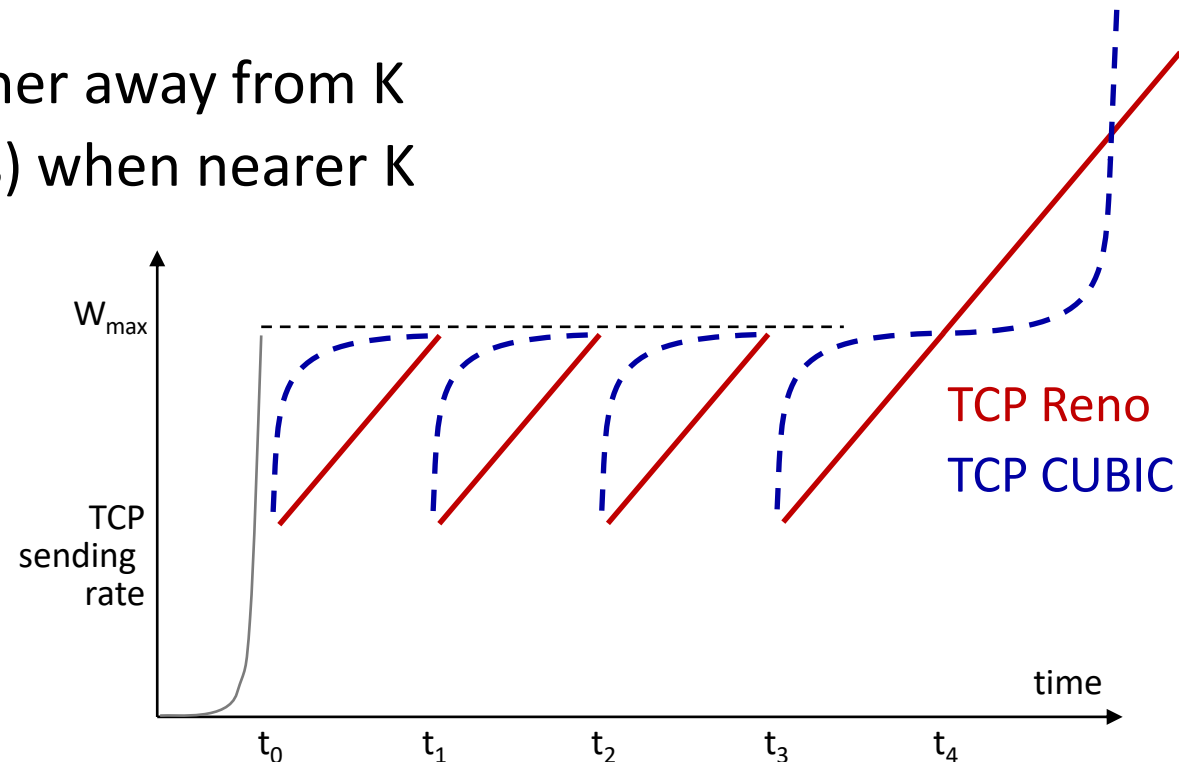
TCP CUBIC

- Is there a better way than AIMD to “probe” for usable bandwidth?
- Insight/intuition:
 - W_{\max} : sending rate at which congestion loss was detected
 - congestion state of bottleneck link probably (?) hasn't changed much
 - after cutting rate/window in half on loss, initially ramp to to W_{\max} *faster*, but then approach W_{\max} more *slowly*



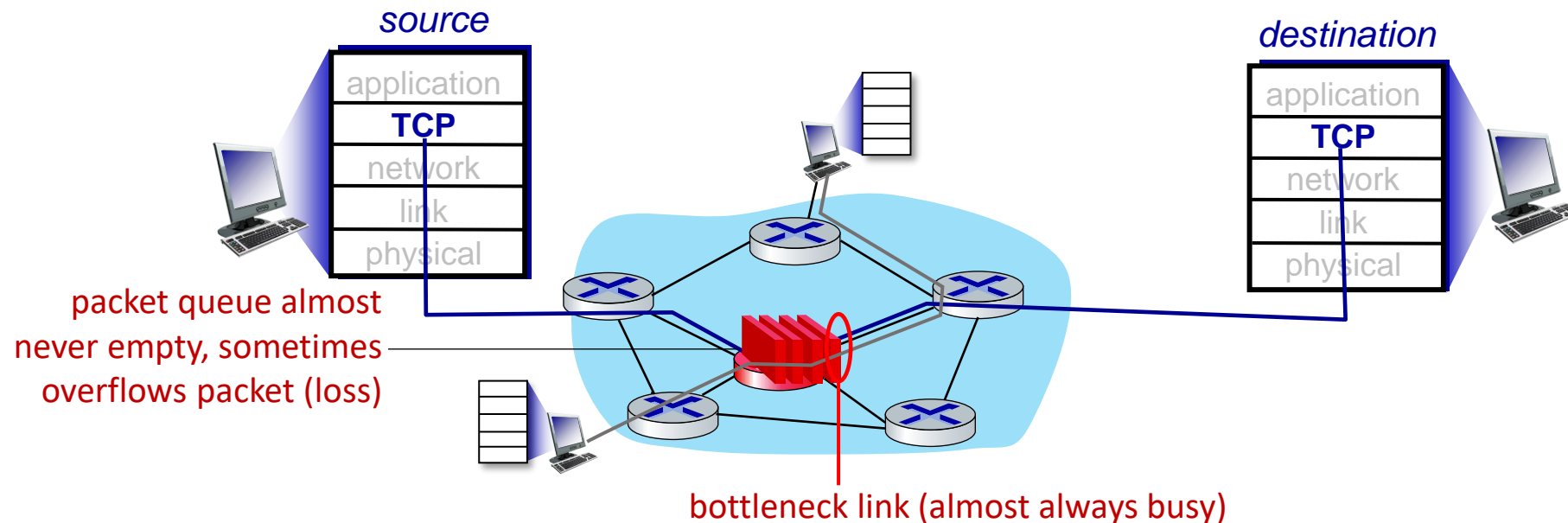
TCP CUBIC

- K: point in time when TCP window size will reach W_{\max}
 - K itself is tunable
- increase W as a function of the *cube* of the distance between current time and K
 - larger increases when further away from K
 - smaller increases (cautious) when nearer K
- TCP CUBIC default in Linux, most popular TCP for popular Web servers



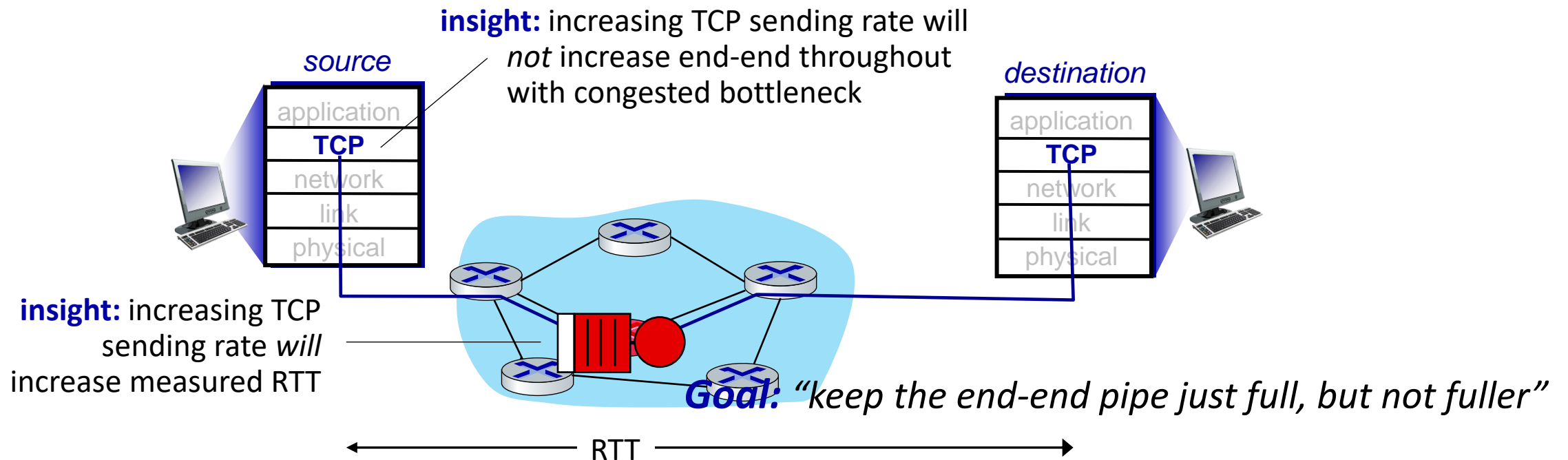
TCP and the congested “bottleneck link”

- TCP (classic, CUBIC) increase TCP’s sending rate until packet loss occurs at some router’s output: the *bottleneck link*



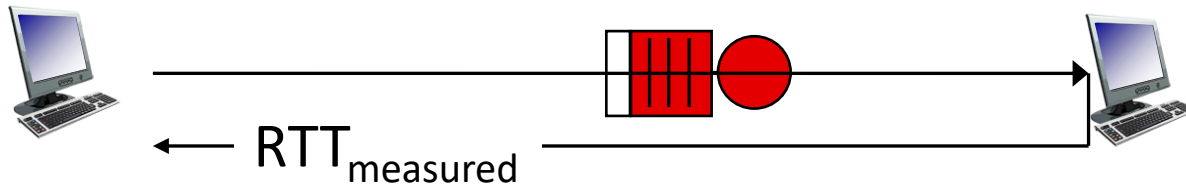
TCP and the congested “bottleneck link”

- TCP (classic, CUBIC) increase TCP’s sending rate until packet loss occurs at some router’s output: the *bottleneck link*
- understanding congestion: useful to focus on congested bottleneck link



Delay-based TCP congestion control

Keeping sender-to-receiver pipe “just full enough, but no fuller”: keep bottleneck link busy transmitting, but avoid high delays/buffering



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{\text{RTT}_{\text{measured}}}$$

Delay-based approach:

- RTT_{\min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window cwnd is $\text{cwnd}/\text{RTT}_{\min}$

if measured throughput “very close” to uncongested throughput
 increase cwnd linearly /* since path not congested */
else if measured throughput “far below” uncongested throughput
 decrease cwnd linearly /* since path is congested */

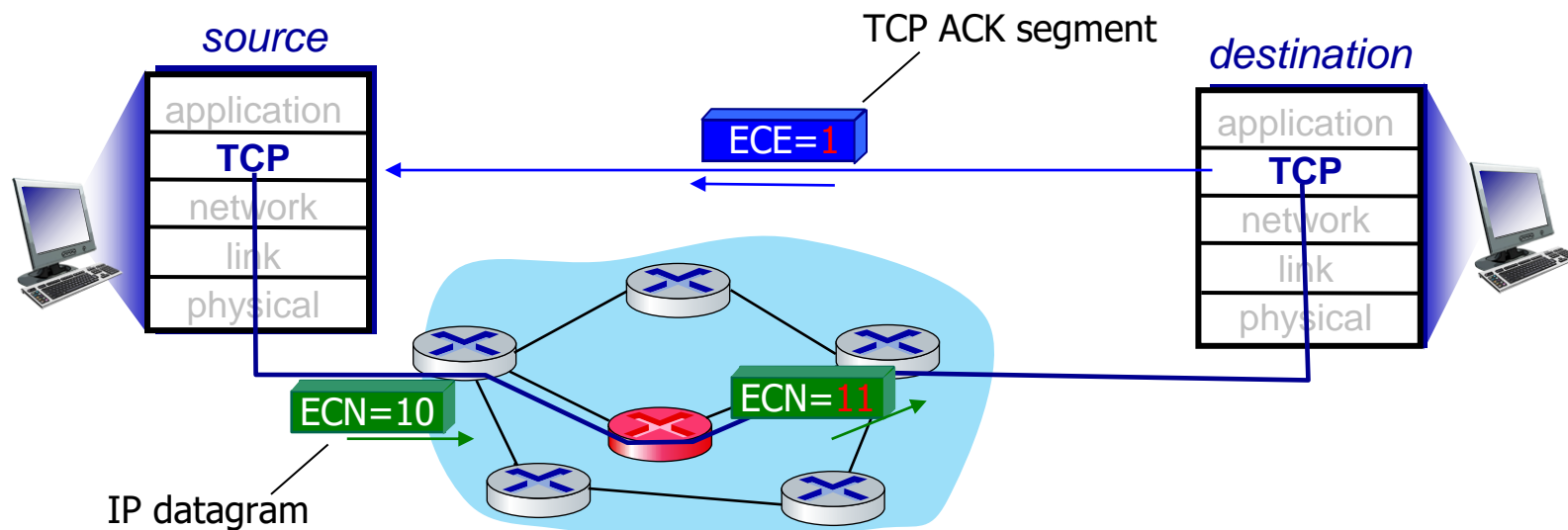
Delay-based TCP congestion control

- congestion control without inducing/forcing loss
- maximizing throughput (“keeping the just pipe full...”) while keeping delay low (“...but not fuller”)
- a number of deployed TCPs take a delay-based approach
 - BBR deployed on Google’s (internal) backbone network

Explicit congestion notification (ECN)

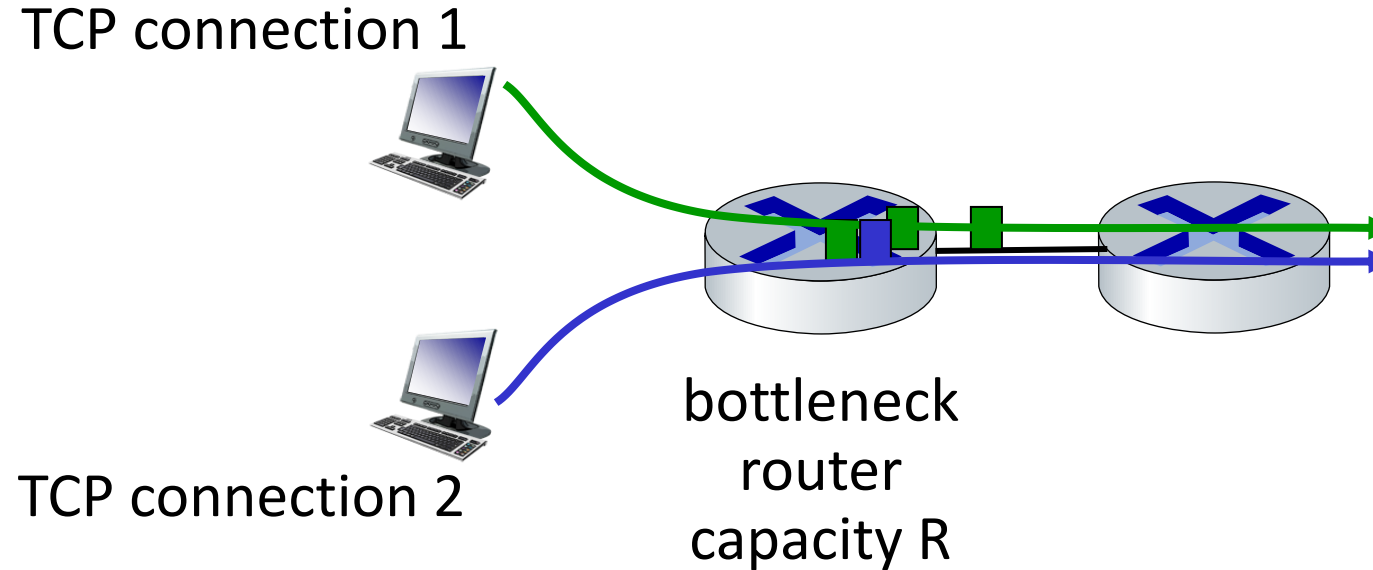
TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
 - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE(ECN-Echo) bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



TCP fairness

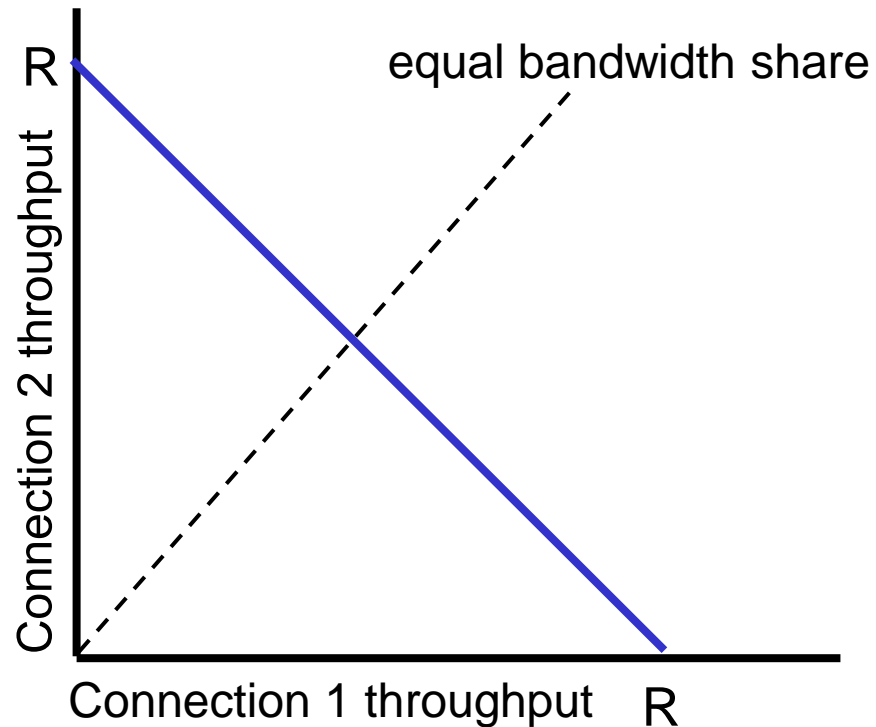
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R , each should have average rate of R/K



Q: is TCP Fair?

Example: two competing TCP sessions:

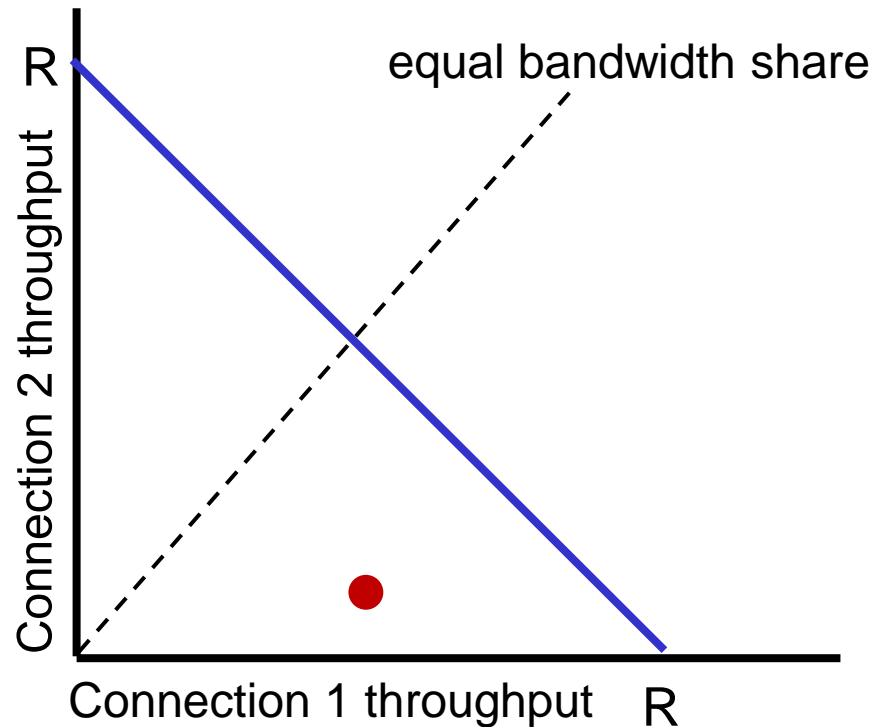
- additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



Q: is TCP Fair?

Example: two competing TCP sessions:

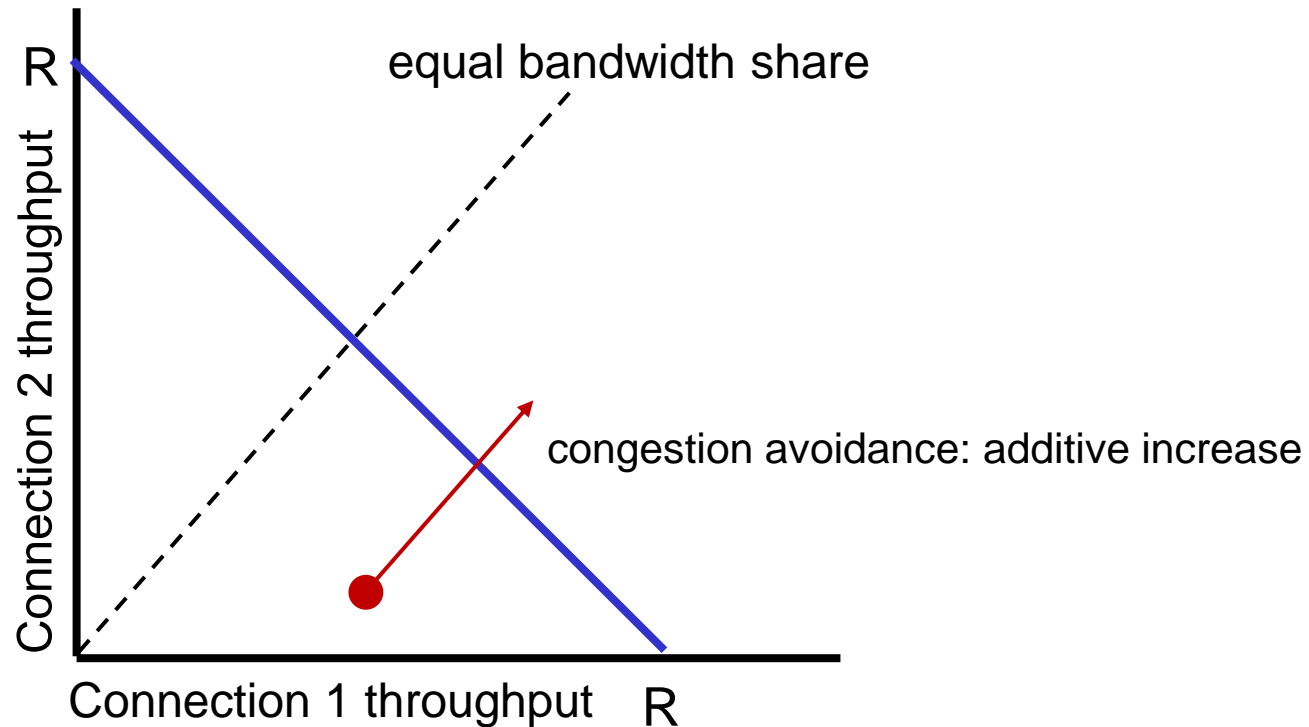
- additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



Q: is TCP Fair?

Example: two competing TCP sessions:

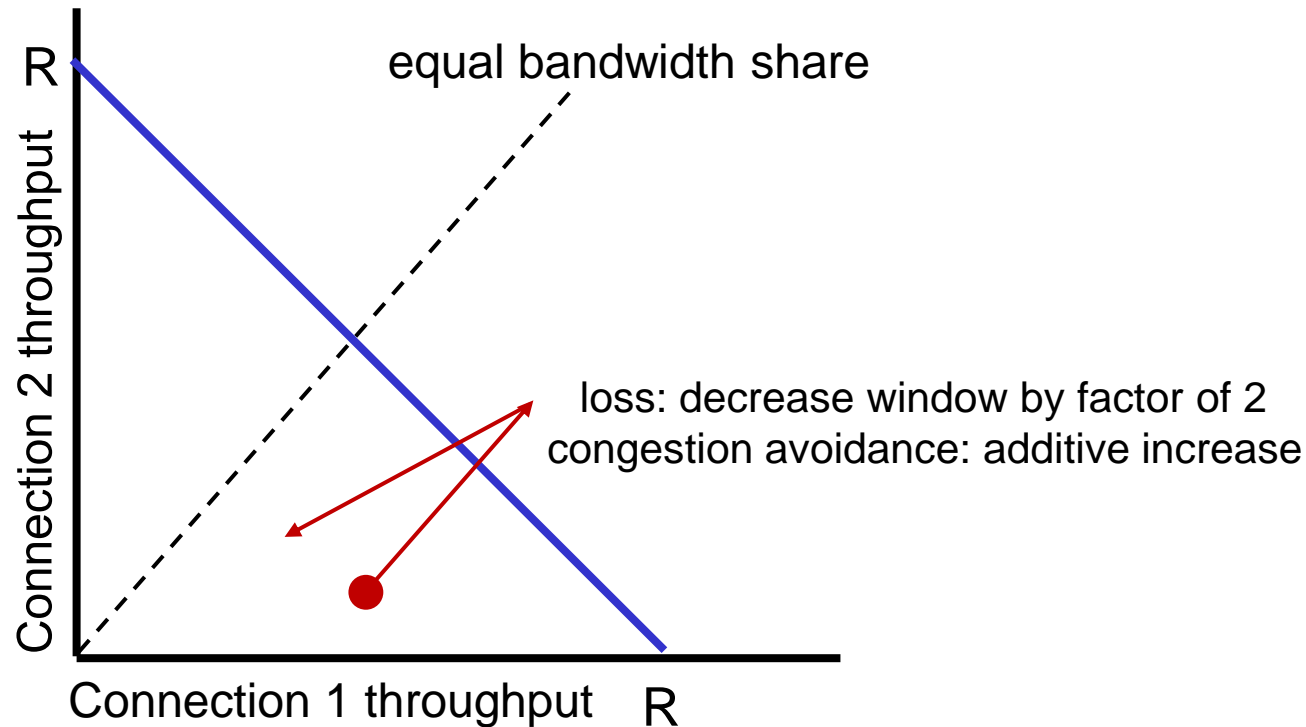
- additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



Q: is TCP Fair?

Example: two competing TCP sessions:

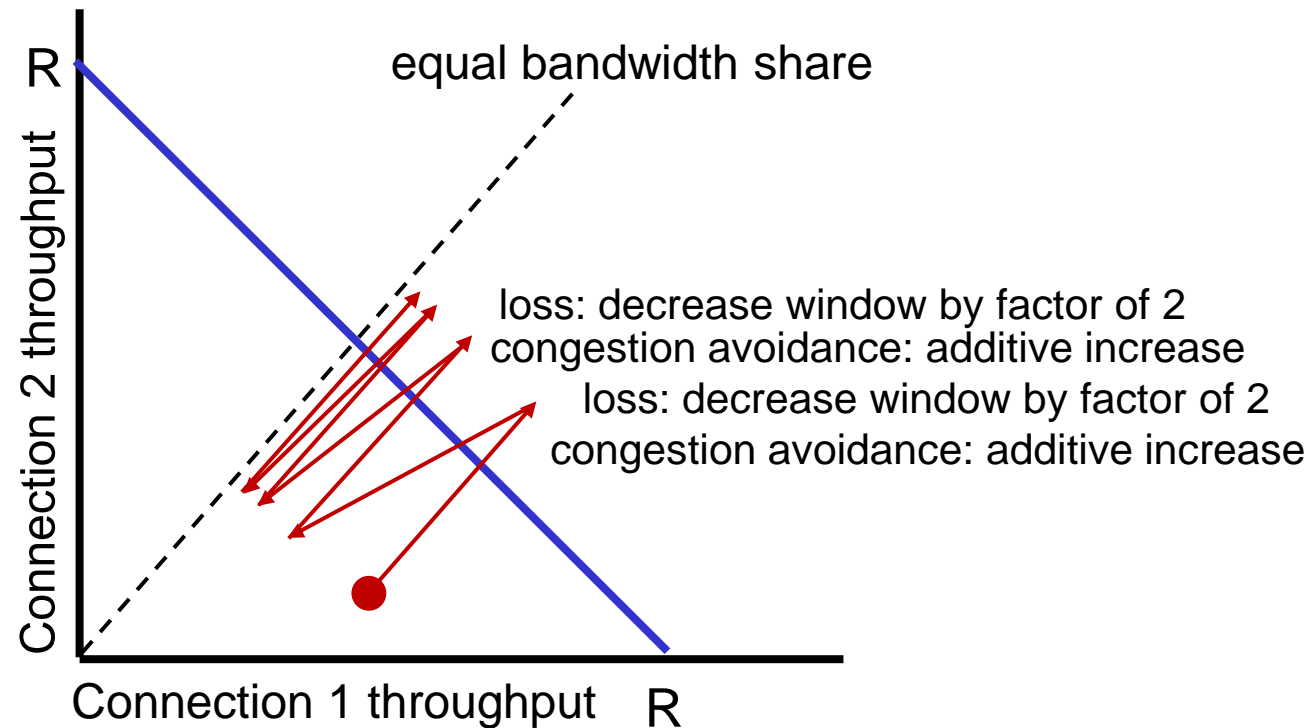
- additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



Q: is TCP Fair?

Example: two competing TCP sessions:

- additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



Is TCP fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions only in congestion avoidance

Fairness: must all network apps be “fair”?

Fairness and UDP

- multimedia apps often do not use TCP
 - do not want rate throttled by congestion control
- instead use UDP:
 - send audio/video at constant rate, tolerate packet loss
- there is no “Internet police” policing use of congestion control

Fairness: must all network apps be “fair”?

Fairness, parallel TCP connections

- application can open *multiple* parallel connections between two hosts
- web browsers do this , e.g., link of rate R with 9 existing connections:
 - new app asks for 1 TCP, gets rate $R/10$
 - new app asks for 11 TCPs, gets $R/2$

Transport layer: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality



Evolving transport-layer functionality

- TCP, UDP: principal transport protocols for 40 years



Evolving transport-layer functionality

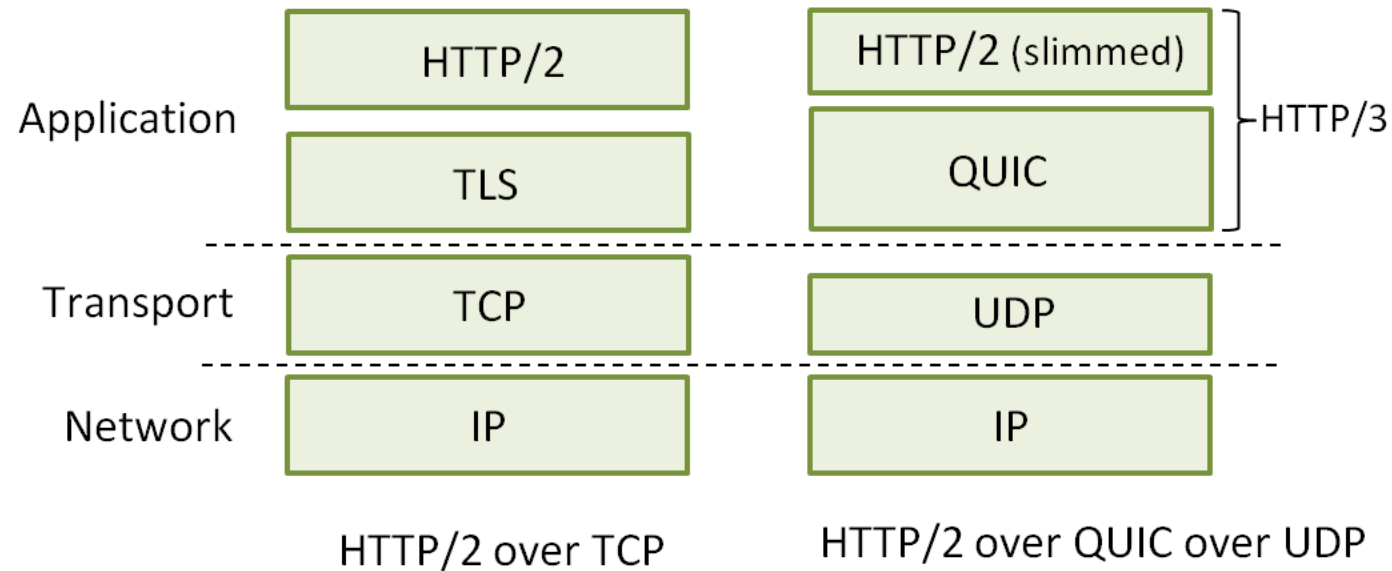
- TCP, UDP: principal transport protocols for 40 years
- different “flavors” of TCP developed, for specific scenarios:

Scenario	Challenges
Long, fat pipes (large data transfers)	Many packets “in flight”; loss shuts down pipeline
Wireless networks	Loss due to noisy wireless links, mobility; TCP treat this as congestion loss
Long-delay links	Extremely long RTTs
Data center networks	Latency sensitive
Background traffic flows	Low priority, “background” TCP flows

- moving transport–layer functions to application layer, on top of UDP
 - HTTP/3: QUIC

QUIC: Quick UDP Internet Connections

- application-layer protocol, on top of UDP
 - increase performance of HTTP
 - deployed on many Google servers, apps (Chrome, mobile YouTube app)

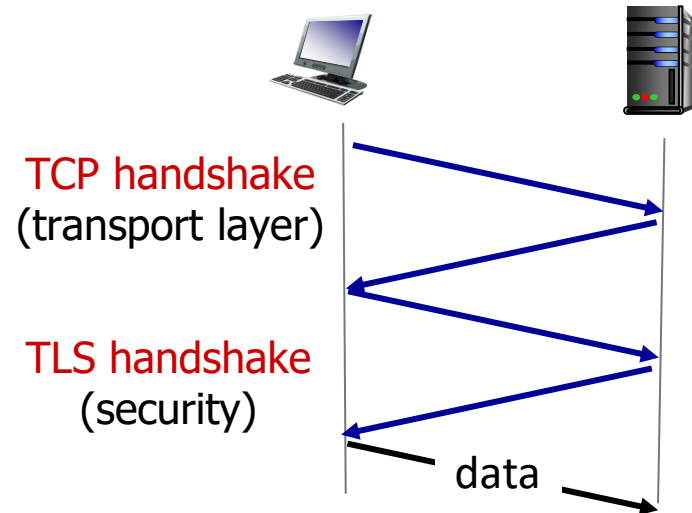


QUIC: Quick UDP Internet Connections

adopts approaches we've studied in this chapter for connection establishment, error control, congestion control

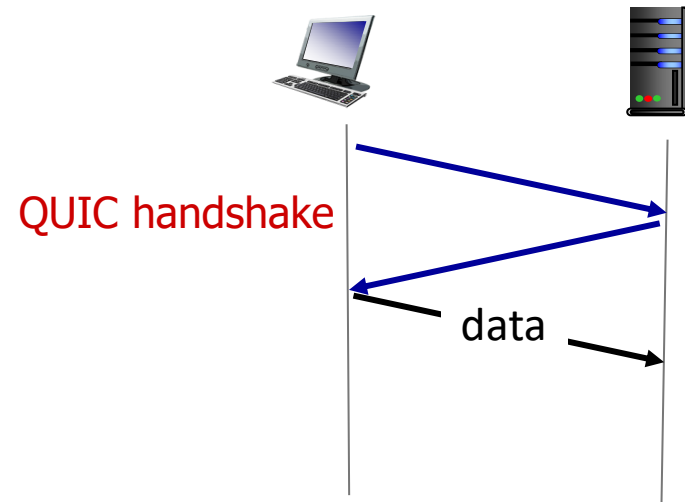
- **error and congestion control:** “Readers familiar with TCP’s loss detection and congestion control will find algorithms here that parallel well-known TCP ones.” [from QUIC specification]
- **connection establishment:** reliability, congestion control, authentication, encryption, state established in one RTT
- multiple application-level “streams” multiplexed over single QUIC connection
 - separate reliable data transfer, security
 - common congestion control

QUIC: Connection establishment



TCP (reliability, congestion control state) + TLS (authentication, crypto state)

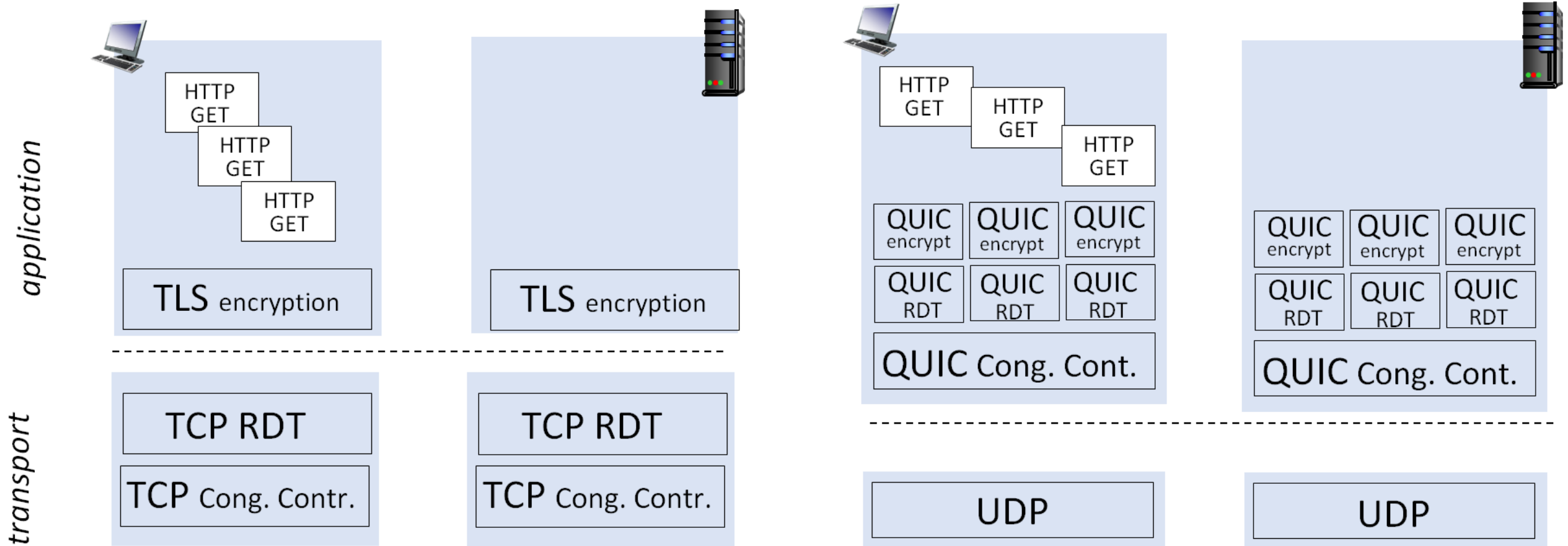
- 2 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake

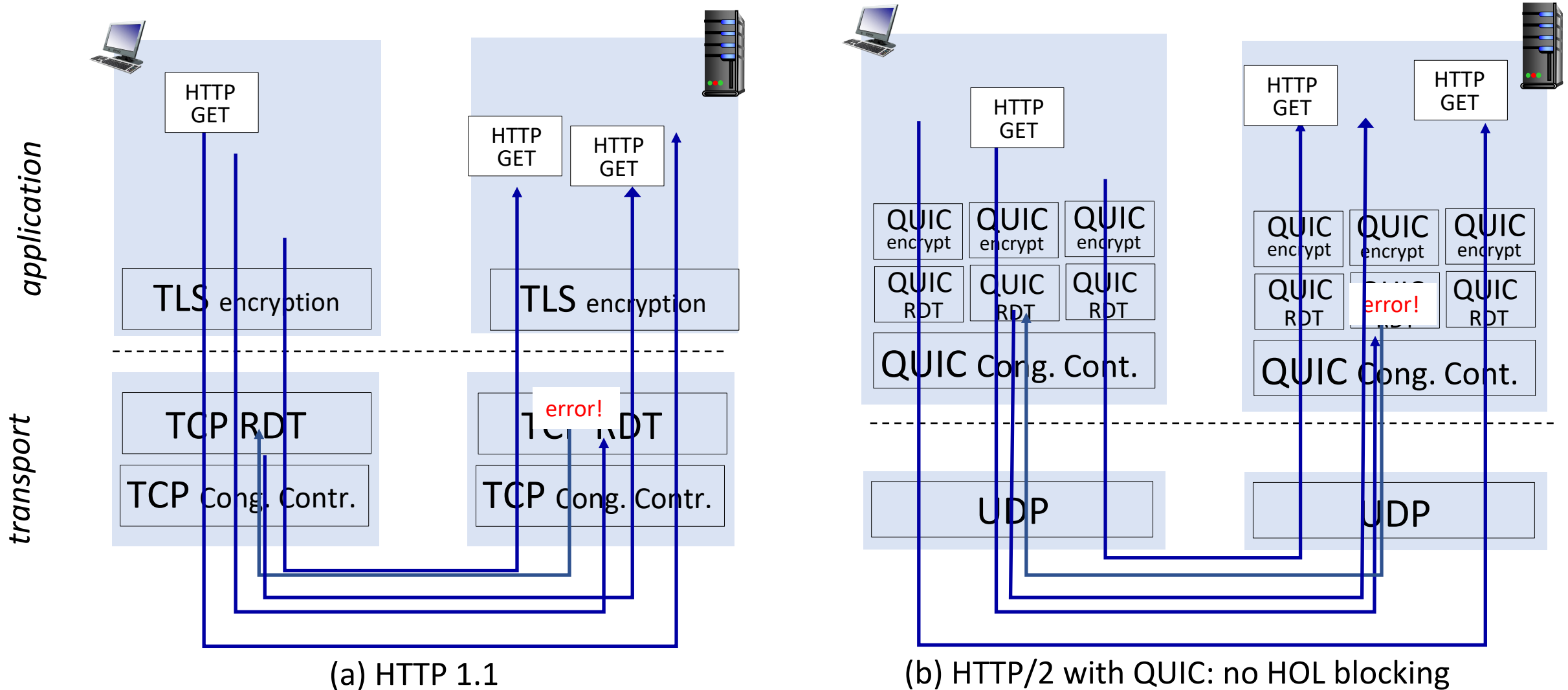
QUIC: streams: parallelism, no HOL blocking



(a) HTTP 1.1

(b) HTTP/2 with QUIC: no HOL blocking

QUIC: streams: parallelism, no HOL blocking



Chapter 3: summary

- principles behind transport layer services:
 - multiplexing, demultiplexing
 - reliable data transfer
 - flow control
 - congestion control
- instantiation, implementation in the Internet
 - UDP
 - TCP

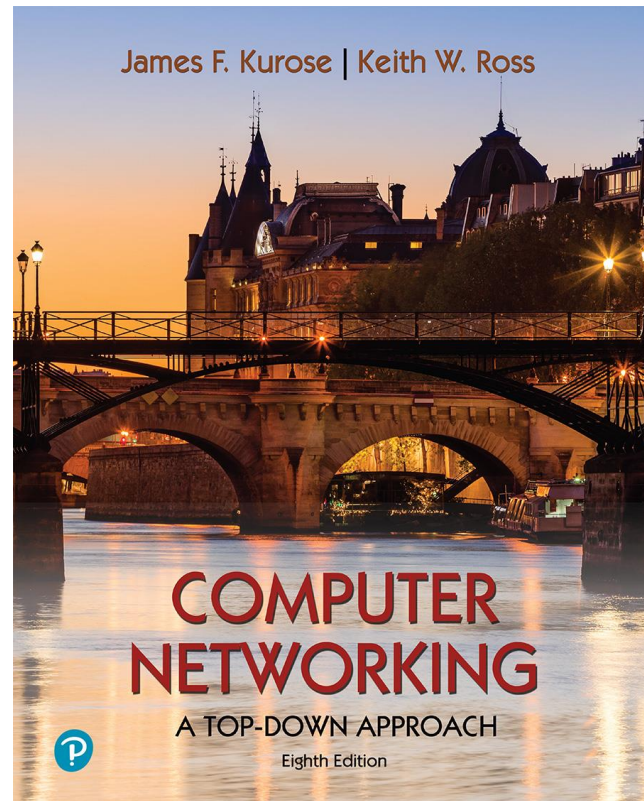
Up next:

- leaving the network “edge” (application, transport layers)
- into the network “core”
- two network-layer chapters:
 - data plane
 - control plane

Important Dates

- Exam1–07-05-2024(Friday)–11am to 12:15pm
- Please arrive by 10:50am and leave at least two seats vacant between you and other students while taking the exam.
- The exam1 will cover all the concepts
From LectureD_1_Chapter_1
To LectureD_10_Chapter_3
- The exam is closed-book, but one side of an 8 ½'' by 11'' sheet may be used, and the exam is worth 100 points.(Only formulas are allowed)
- The exam1 will consist of 4 main questions, each with different sub-questions.
- No other electronic devices are allowed except a calculator for the exam.

Copyright Information



Computer Networking: A Top-Down Approach

8th edition

Jim Kurose, Keith Ross
Pearson, 2020

The Slides are adapted from,

All material copyright 1996-2023
J.F Kurose and K.W. Ross, All Rights Reserved