Chapter 3
Transport Layer

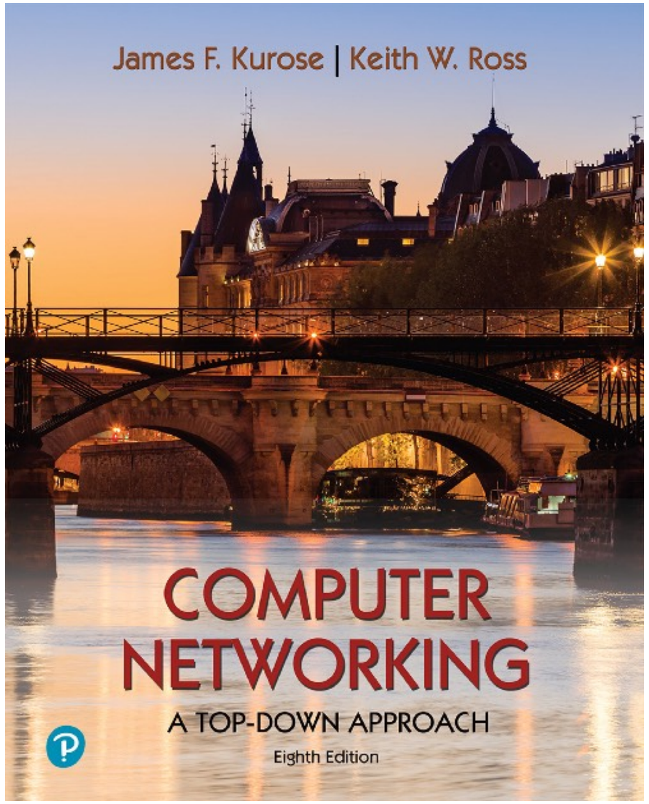
A note on the use of these PowerPoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Day 14: Evolution of TCP and Transport

CSEE 4119
Computer Networks
Ethan Katz-Bassett

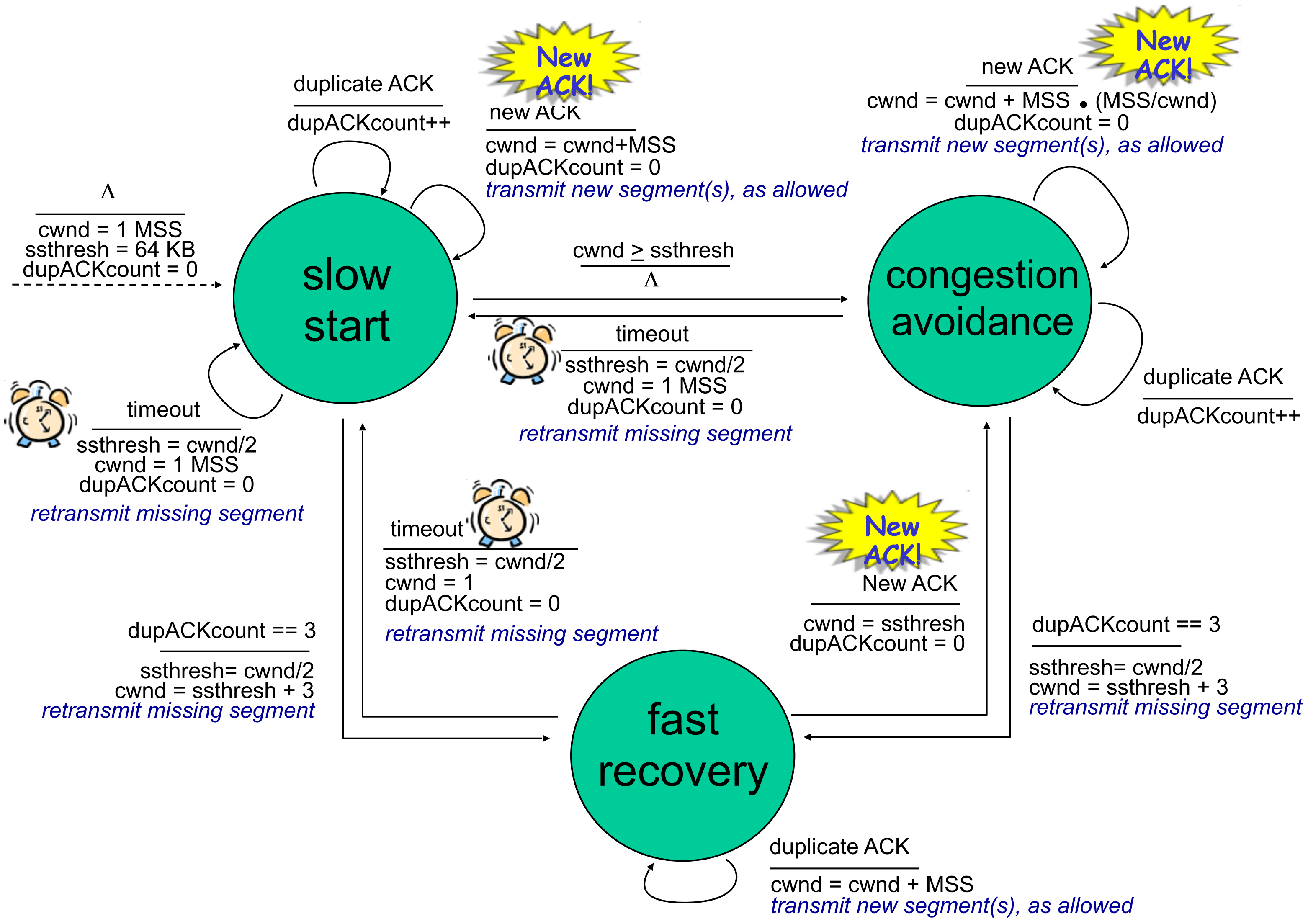
COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

Oct 18 admin

- Masks ***strongly recommended***, over nose and mouth
- Attendance & participation ***not*** required
- If you are not feeling well or were exposed to COVID, please stay home
- Videos of lectures available

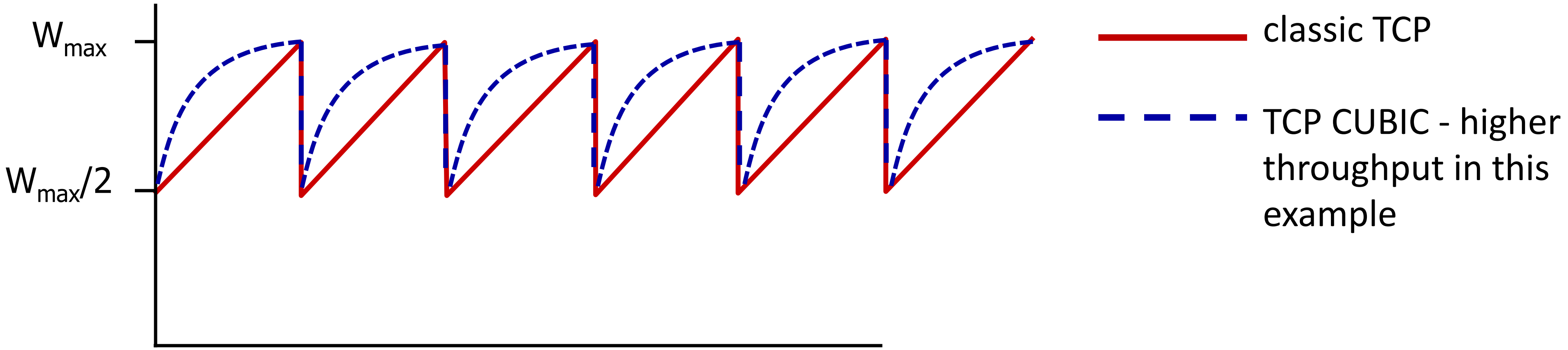
- HW2 due yesterday
- 1 extra slip day granted (5 total)
- HW3 assigned, due October 31
    - ***If submitted LATE, SCORE=0, NO exceptions***
- Project 1 Final Stage released this week
- Next week: guest lectures, Henning on Tuesday, Tom on Thursday
    - I'll be at the Internet Measurement Conference

Recap: TCP (Reno) congestion control



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*

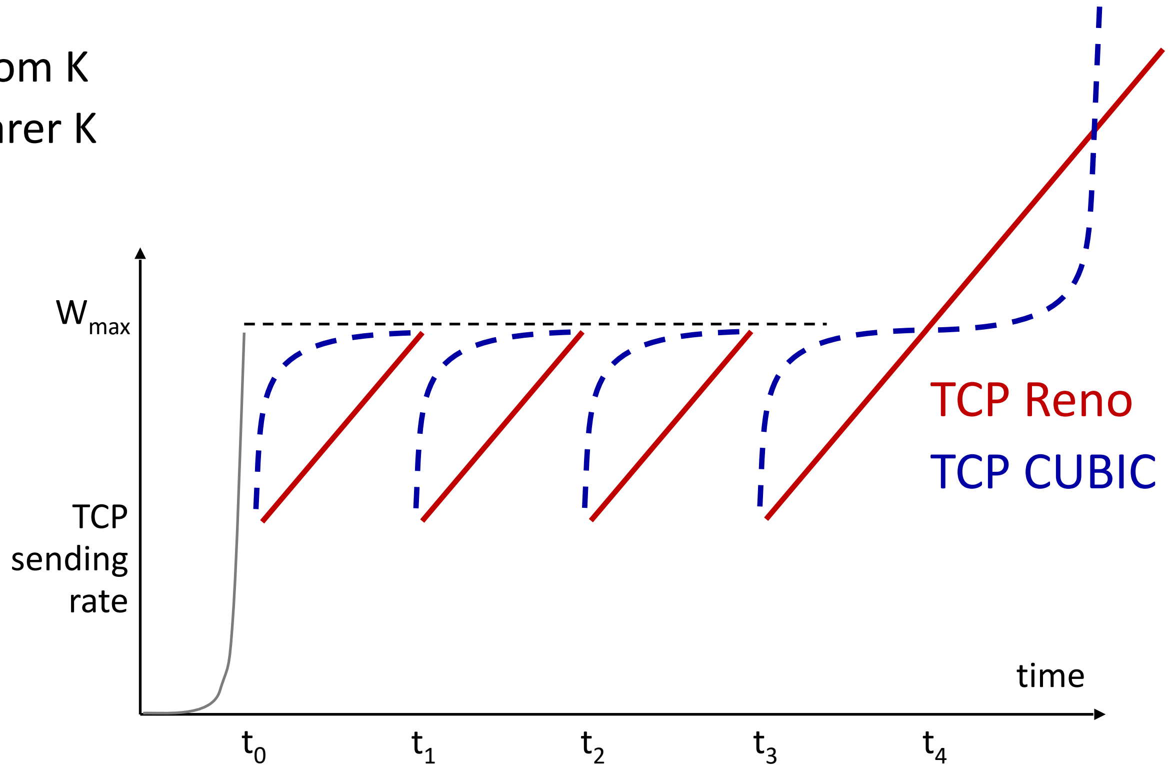


TCP CUBIC

- K: point in time when TCP window size will reach W_{max}
  - K is configurabletunable

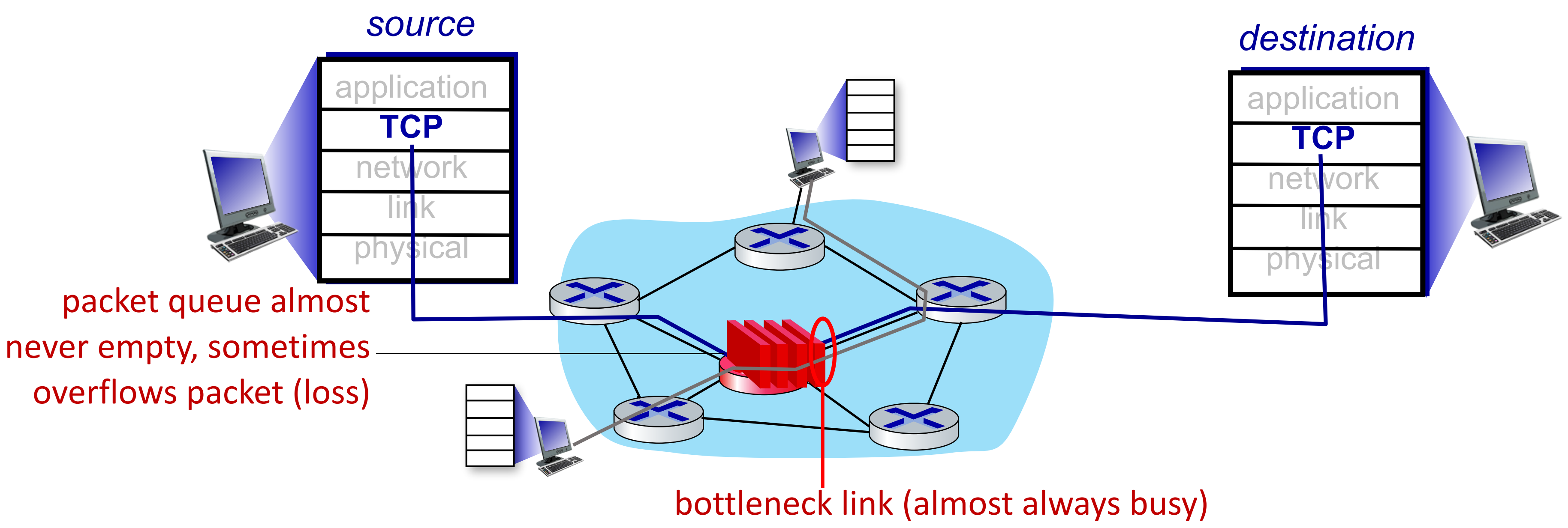
- increase W as a function of the *cube* of the distance between current time and K
  - larger increases when further away from K
  - smaller increases (cautious) when nearer K

- TCP CUBIC default in Linux
  - As of 2020, most popular TCP for popular Web servers
  - Today:
    - Netflix uses Reno
    - Google, YouTube, Akamai, Amazon, DropBox use BBR (next! And on HW!)



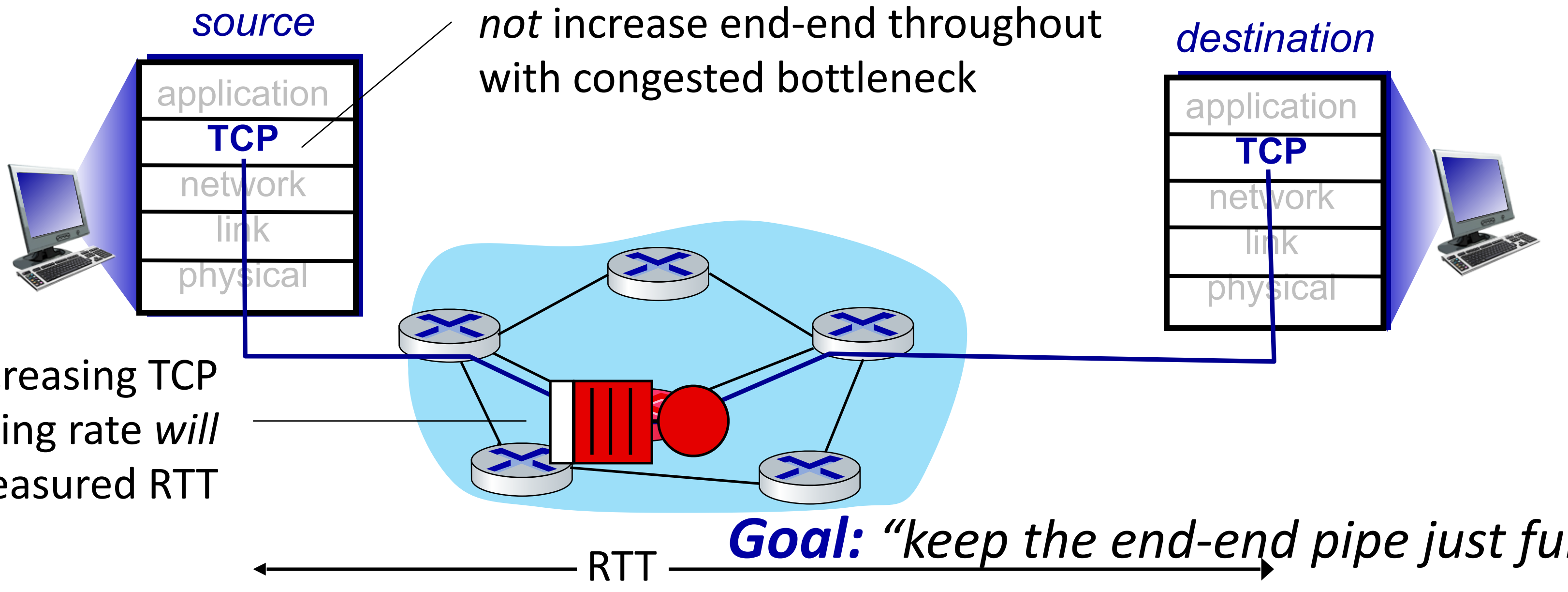
TCP and the congested "bottleneck link"

- 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



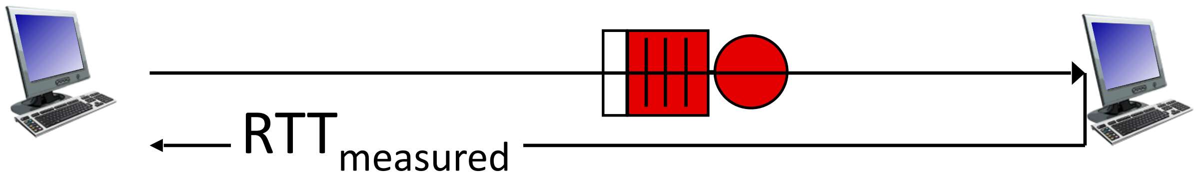
insight: increasing TCP sending rate will *not* increase end-end throughout with congested bottleneck

insight: increasing TCP sending rate *will* increase measured RTT

Goal: *"keep the end-end pipe just full, but not fuller"*

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



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

Delay-based approach:

- RTT_{min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window `cwnd` is cwnd/RTT_{min}

if measured throughput "very close" to uncongested throughput

increase `cwnd` linearly /* since path not congested */

else if measured throughput "far below" uncongested throughout

decrease `cwnd` linearly /* since path is congested */

Delay-based TCP congestion control

- congestion control without inducing/forcing loss
- maximizing throughout (“keeping the just pipe full… ”) while keeping delay low (“…but not fuller”)
- What’s the problem if you just use delay…and others use Reno (loss)?
- a number of deployed TCPs take a delay-based approach
  - BBR developed and deployed by Google uses a modified delay-based approach

BBR: a CC written from scratch at Google

The story of BBR goes back to 2013...

- Many Google services complained about TCP performance...
  - Internal B4 backbone TCP throughput often < 10Mbps
  - Youtube.com: sometimes terrible video quality, with RTT > 10 secs
  - Google.com: poor latency in developing regions
- Services started to “work around” TCP
  - Use parallel connections, tweak TCP knobs, add more buffer to the network, ...

What was the root cause of these problems?

- TCP congestion control at Google in 2013 was CUBIC (Linux default)
  - CUBIC is a loss-based congestion control algorithm
  - Packet loss was the sole signal

The problem: loss-based congestion control

- Loss-based congestion control Reno (NetFlix/FreeBSD), CUBIC (Linux/Apple/Windows)
  - Keeps sending faster until it sees a loss

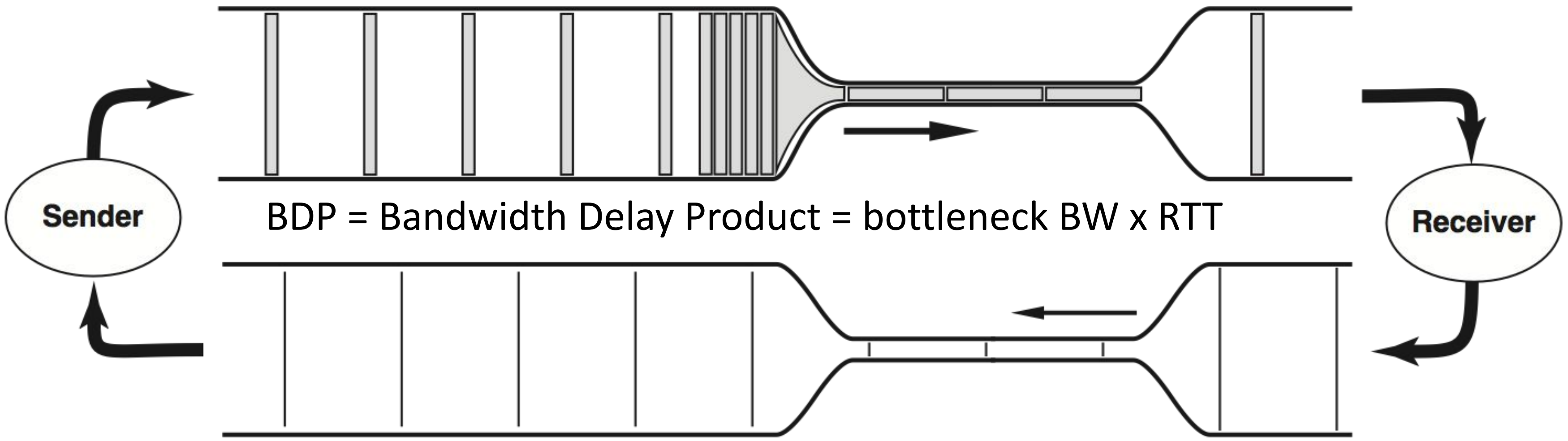
- But packet loss **alone** is **not** a good proxy for congestion

- If loss comes **before** sustained congestion, loss-based CC gets low throughput
  - 10Gbps over 100ms RTT needs < 0.0000029% (2.9e-8) packet loss (infeasible)
  - 1% loss (feasible) over 100ms RTT gets < 3Mbps

- If loss comes **after** congestion, loss-based CC bloats buffers, suffers high delays

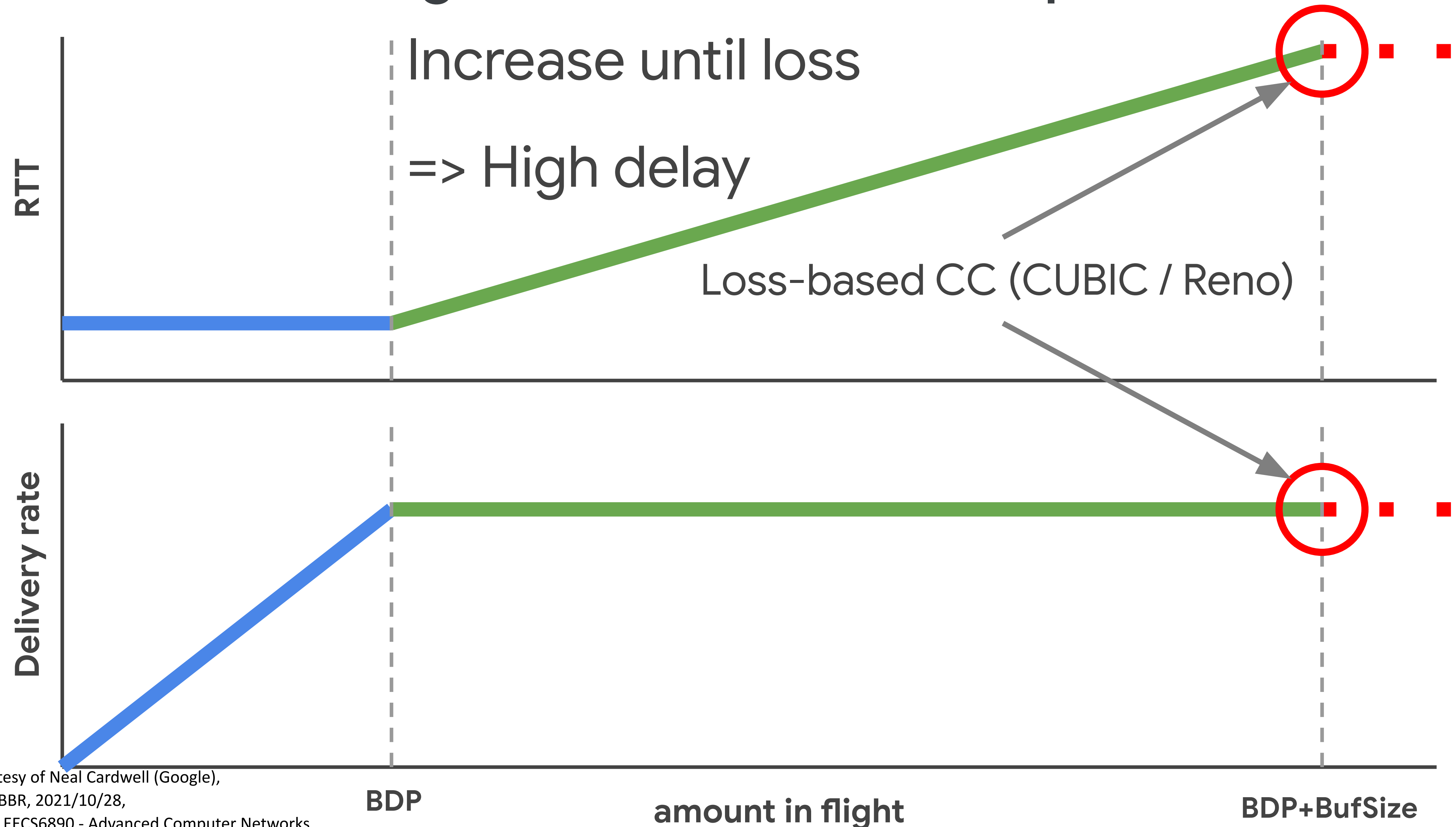
Slide courtesy of Neal Cardwell (Google), author of BBR, 2021/10/28, Columbia: EECS6890 - Advanced Computer Networks

Network congestion and bottlenecks: a model



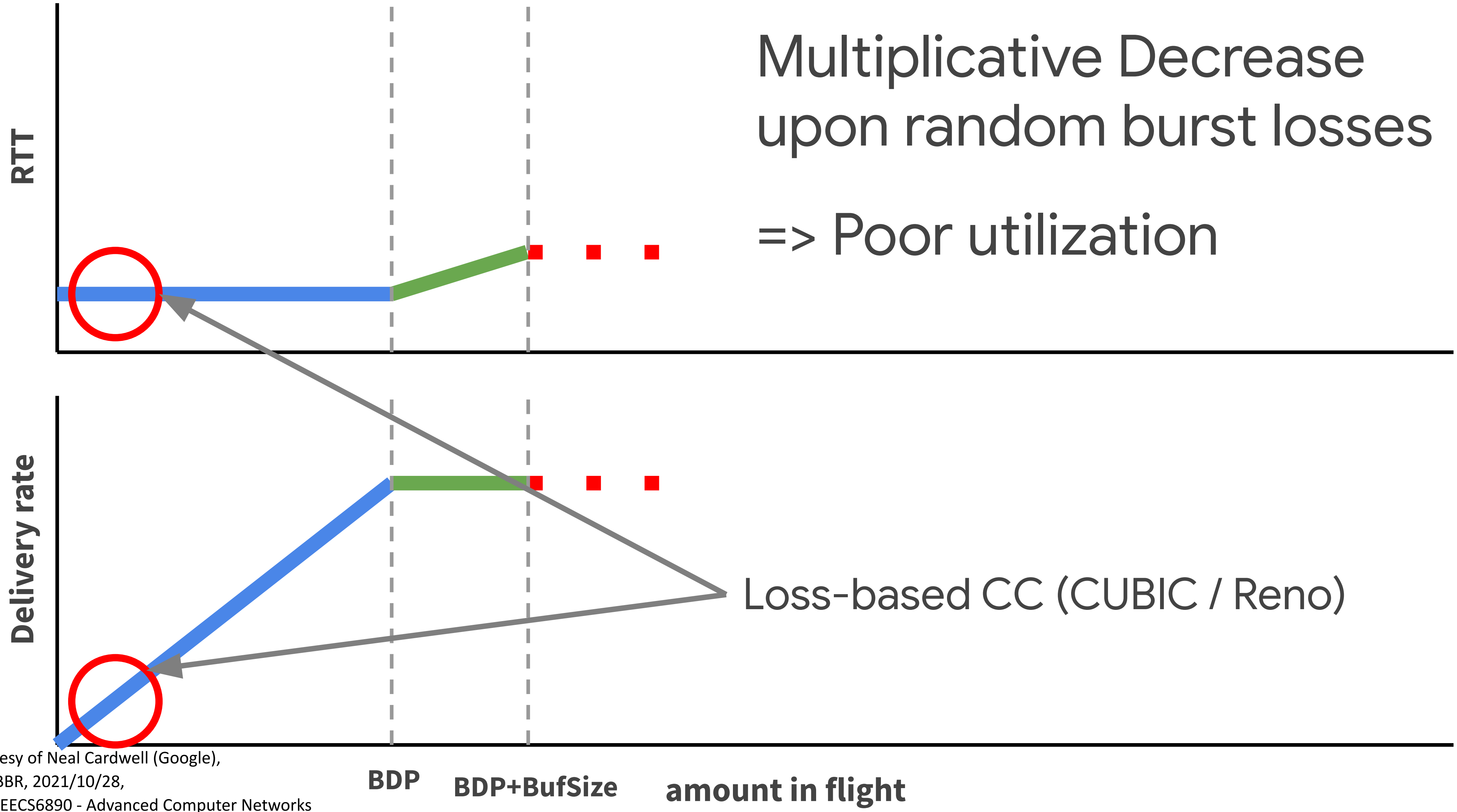
Slide courtesy of Neal Cardwell (Google),
author of BBR, 2021/10/28,
Columbia: EECS6890 - Advanced Computer Networks

Loss-based congestion control in deep buffers



Slide courtesy of Neal Cardwell (Google), author of BBR, 2021/10/28,
Columbia: EECS6890 - Advanced Computer Networks

Loss-based congestion control in shallow buffers

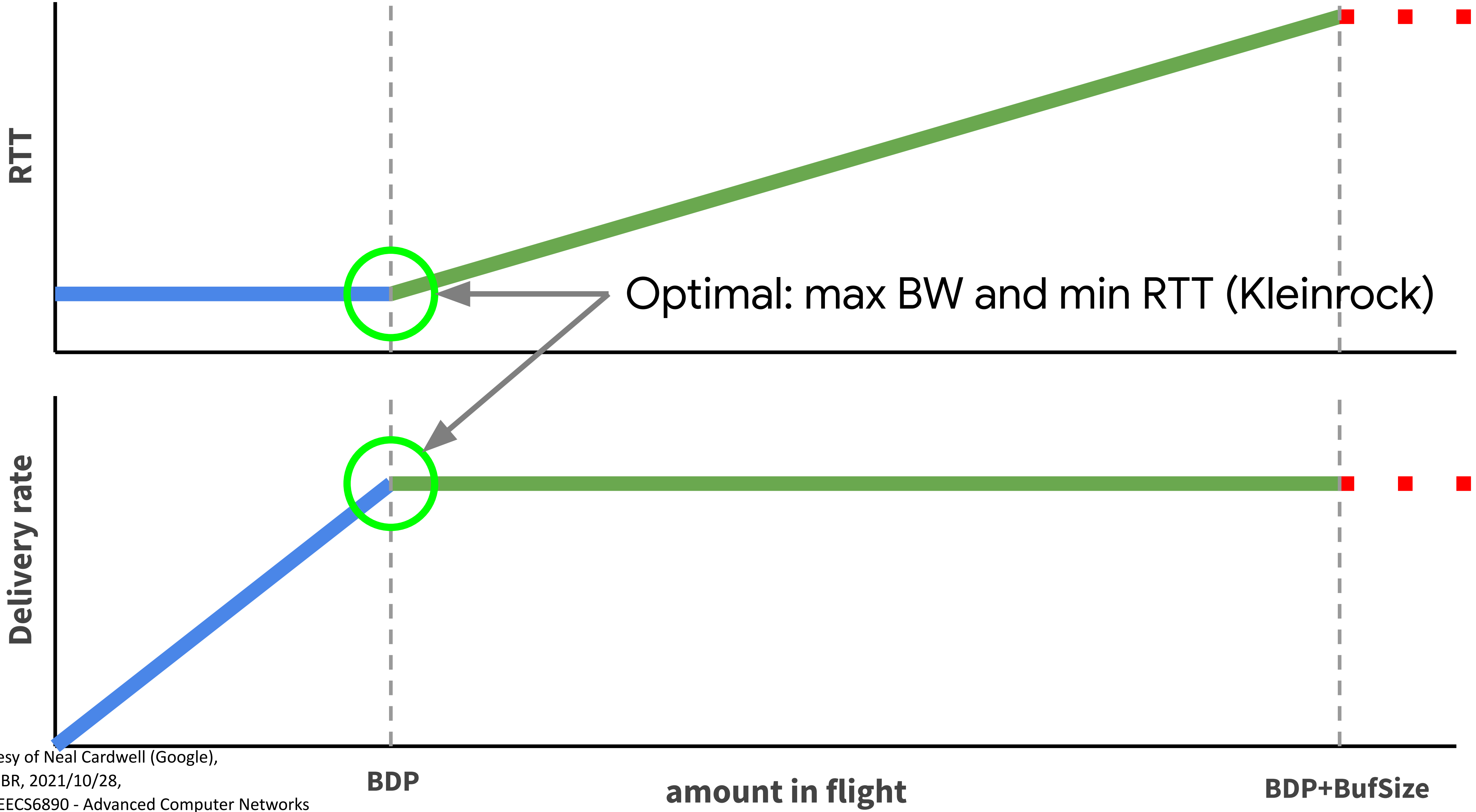


Multiplicative Decrease upon random burst losses

=> Poor utilization

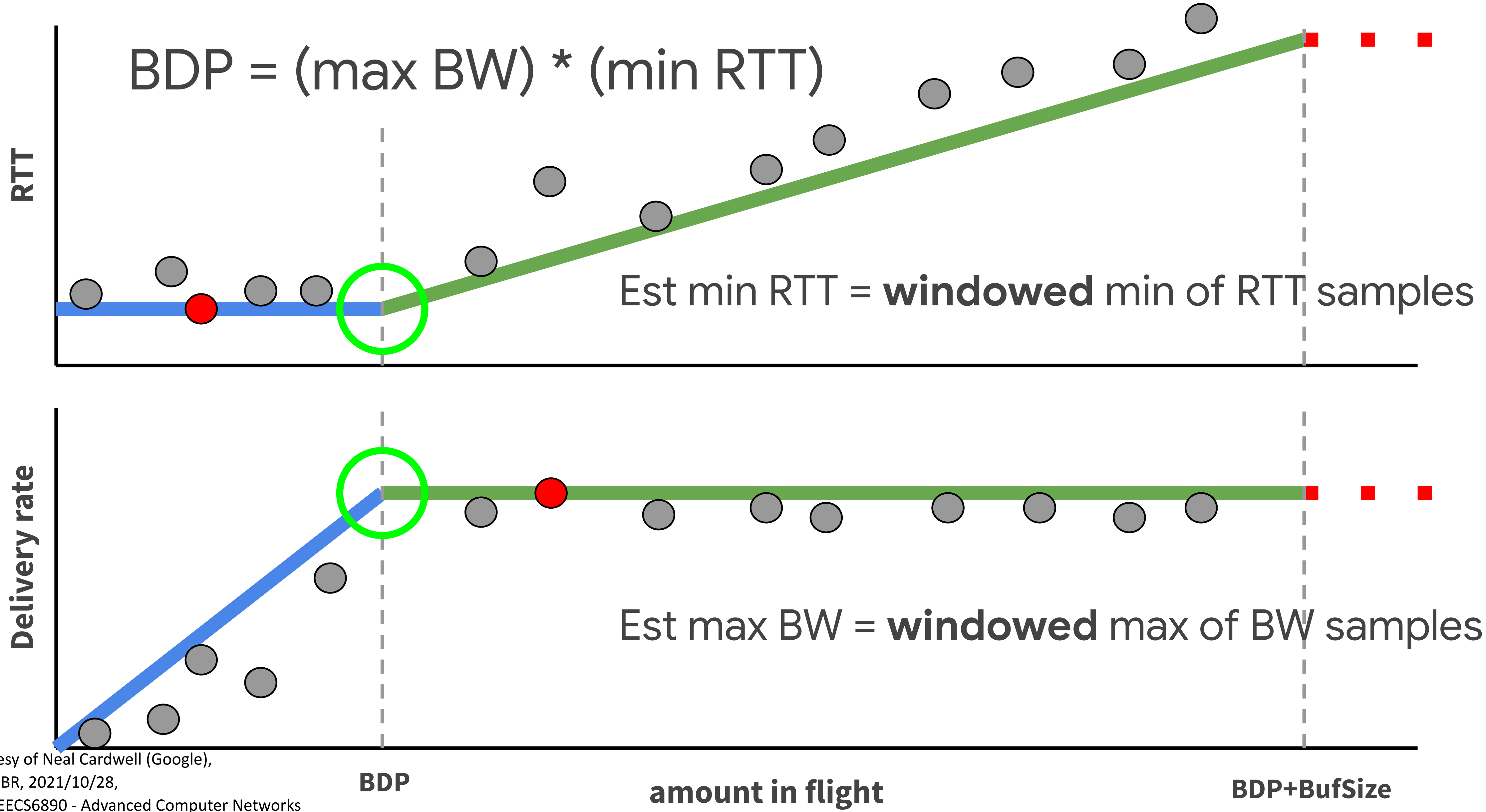
Slide courtesy of Neal Cardwell (Google), author of BBR, 2021/10/28,
Columbia: EECS6890 - Advanced Computer Networks

What is the optimal operating point?



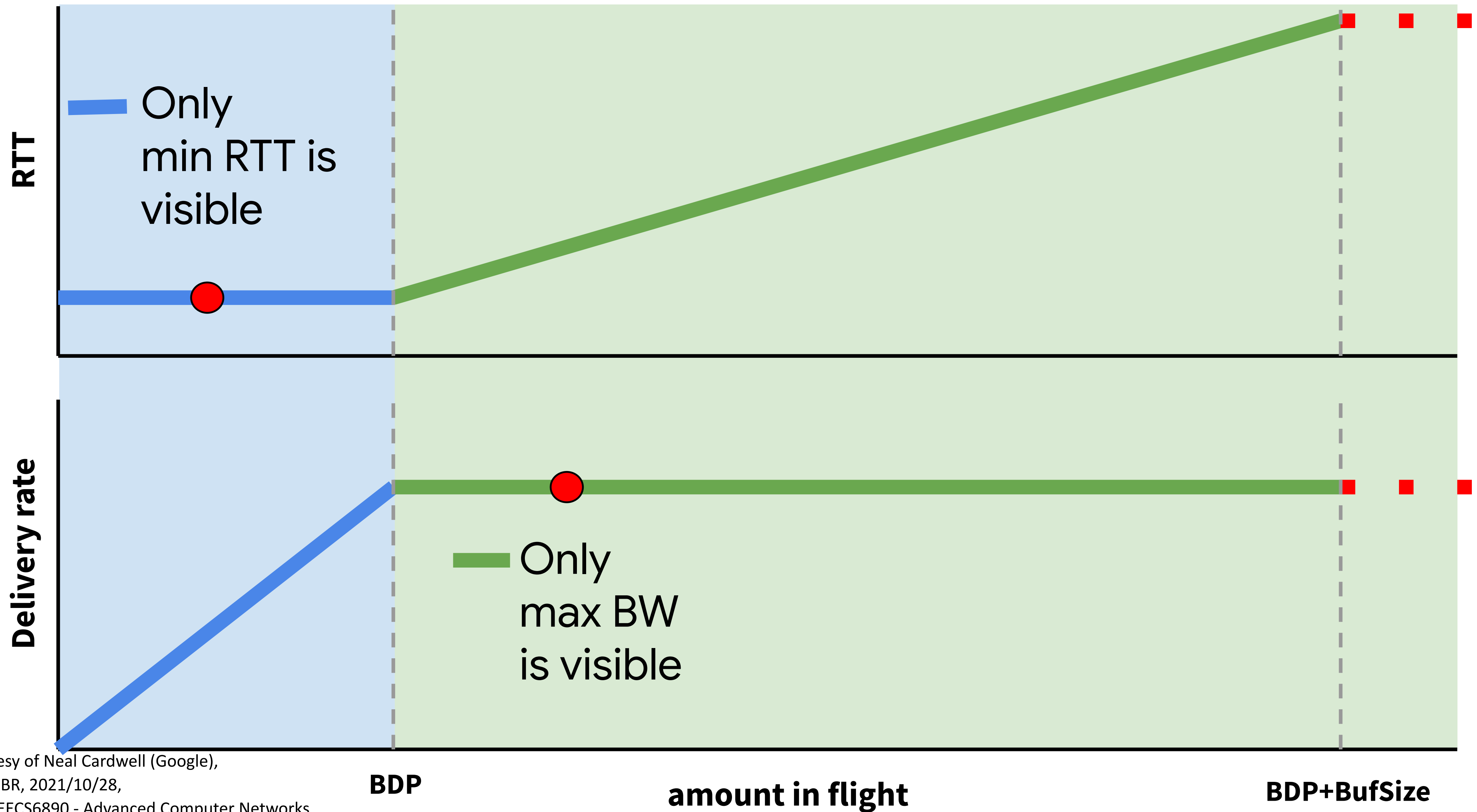
Slide courtesy of Neal Cardwell (Google), author of BBR, 2021/10/28, Columbia: EECS6890 - Advanced Computer Networks

Estimating optimal point (max BW, min RTT)



Slide courtesy of Neal Cardwell (Google), author of BBR, 2021/10/28, Columbia: EECS6890 - Advanced Computer Networks

To see max BW, min RTT: probe both sides of BDP

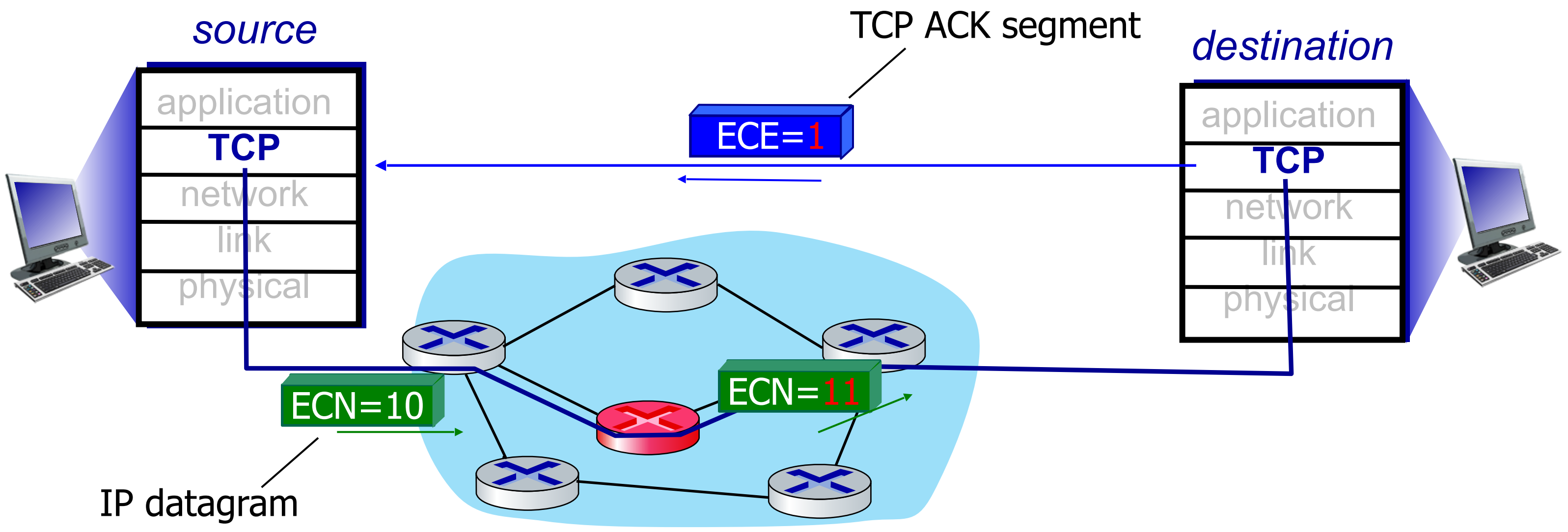


Slide courtesy of Neal Cardwell (Google),
author of BBR, 2021/10/28,
Columbia: EECS6890 - Advanced Computer Networks

Explicit congestion notification (ECN)

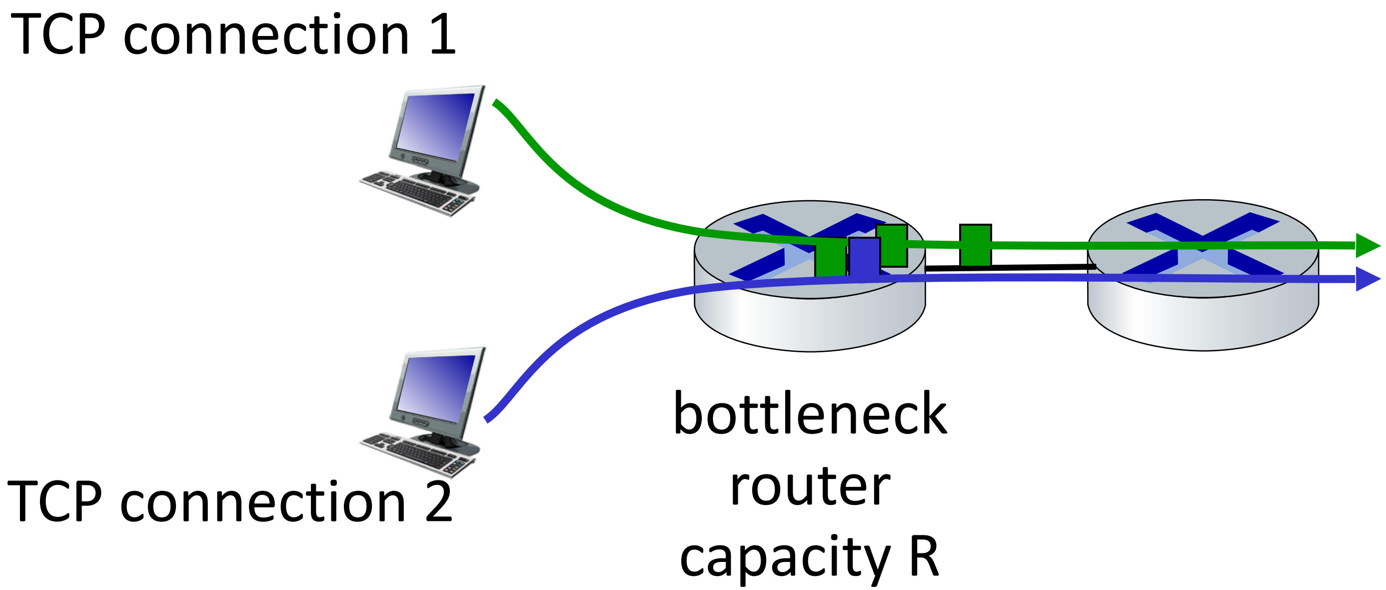
TCP deployments often implement *network-assisted* congestion control:

- 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 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

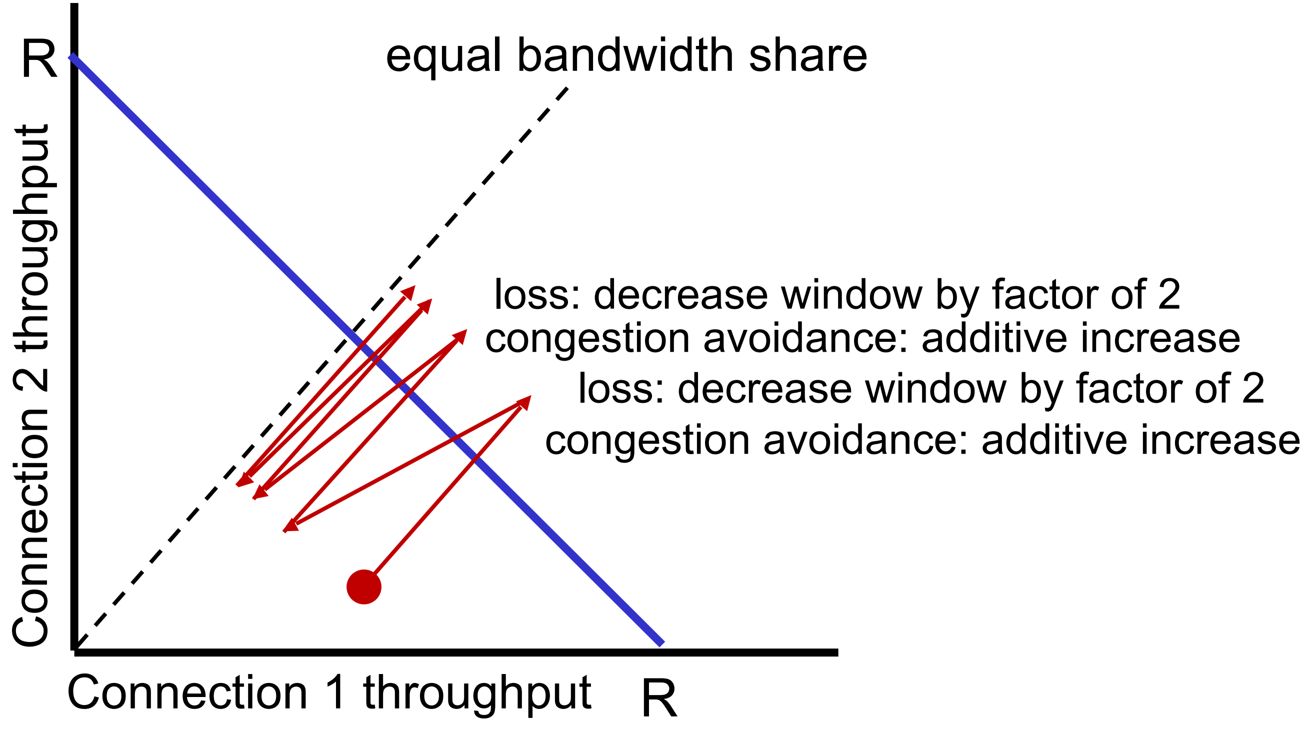
Fairness goal: if *K* TCP sessions share same bottleneck link of bandwidth *R*, each should have average rate of *R/K*



Q: is TCP Fair?

Example: two competing TCP sessions:

- additive increase gives slope of 1, as throughout 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, 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

TCP over "long, fat pipes"

- example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- requires W = 83,333 in-flight segments
- throughput in terms of segment loss probability, L [Mathis 1997]:

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\sqrt{L}}$$

➜ to achieve 10 Gbps throughput, need a loss rate of $L = 2\cdot 10^{-10}$ *– a very small loss rate!*

- versions of TCP for long, high-speed scenarios

Evolving transport-layer functionality

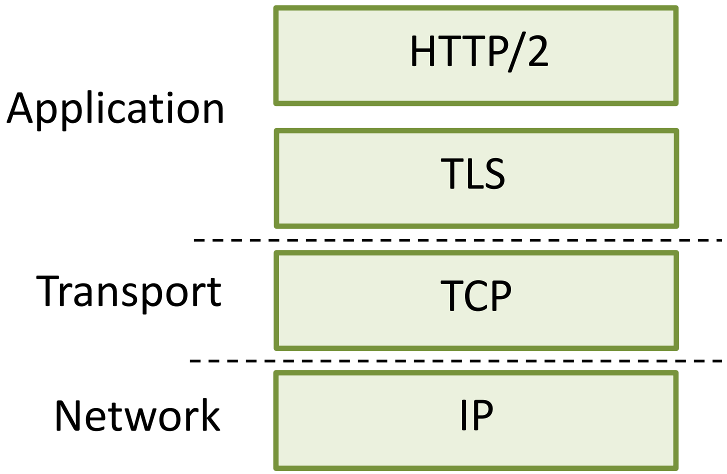
- TCP, UDP: principal transport protocols for 40 years
- different “flavors” of TCP developed, for specific scenarios:

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)



HTTP/2 over TCP

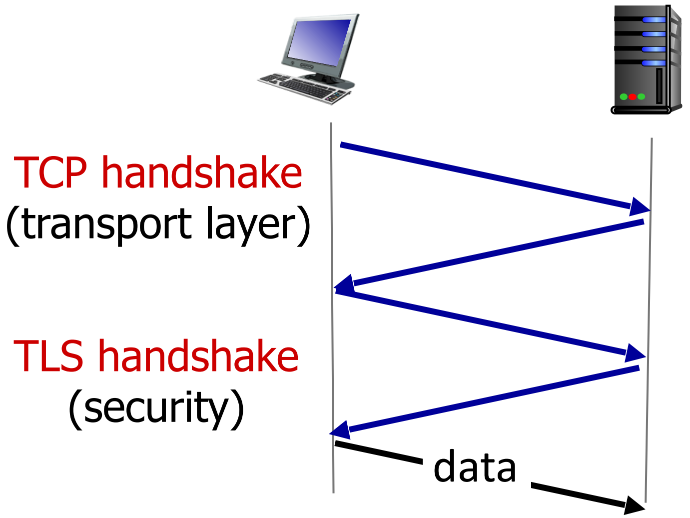
QUIC: Quick UDP Internet Connections

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

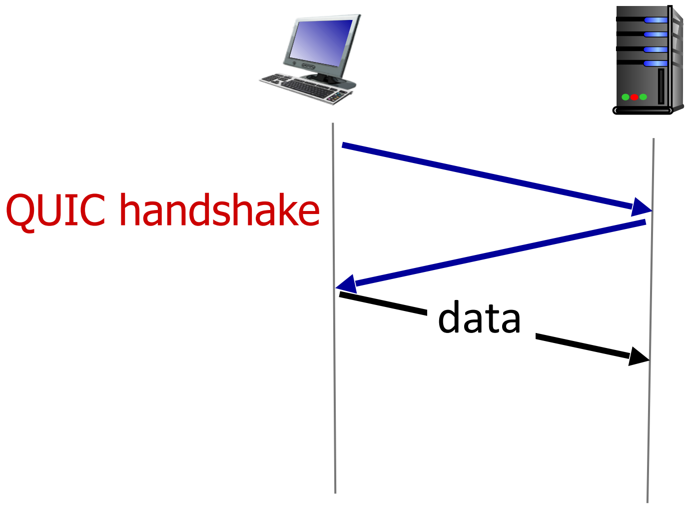
- multiple application-level "streams" multiplexed over single QUIC connection
  - separate reliable data transfer, security
  - common congestion control

QUIC: Connection establishment



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

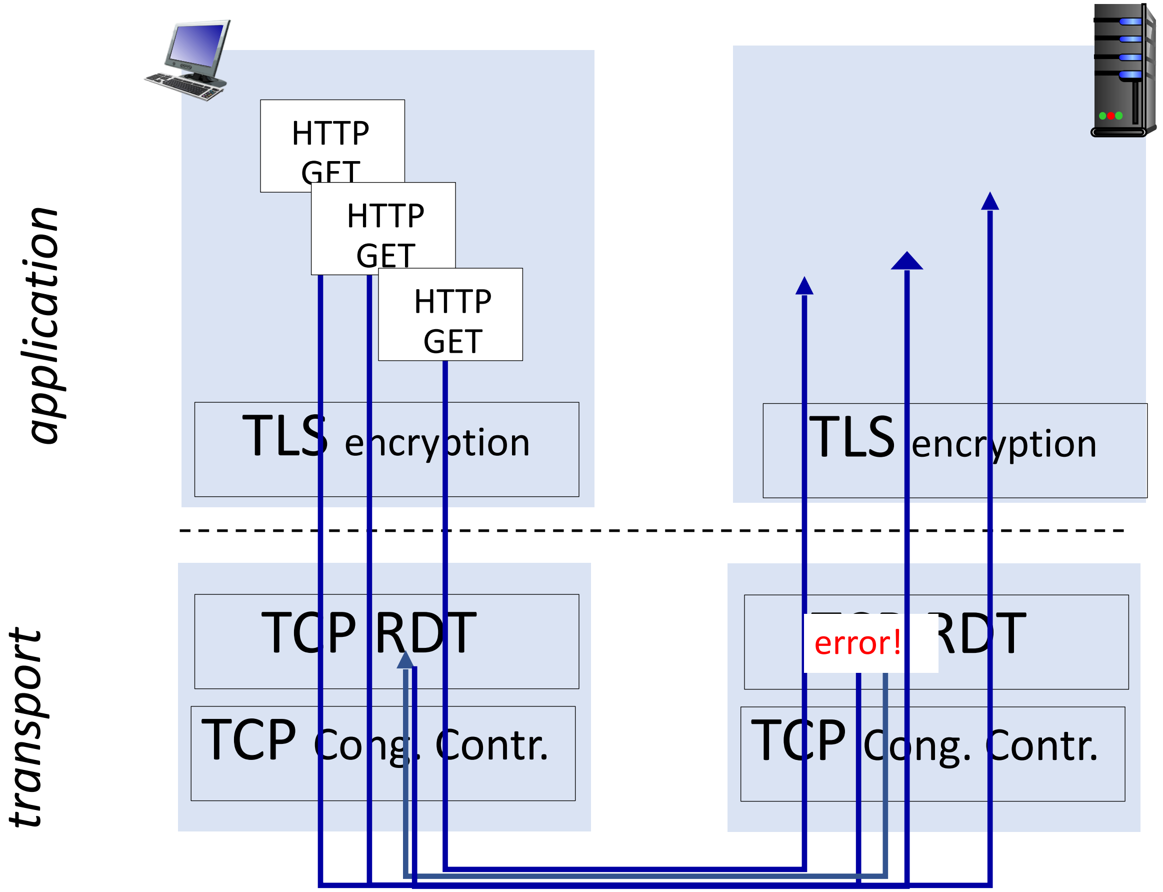
- 2 serial handshakes



QUIC: reliability, congestion control, authentication, crypto state

- 1 handshake

QUIC: streams: parallelism, no HOL blocking



(a) HTTP 1.1

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

DO NOT SHARE
SLIDES AND CLASS MATERIALS
ON ONLINE SITES

Uploading course materials to sites such as CourseHero, Chegg or Github is academic misconduct at Columbia (see pg 10 of Colu