Congestion Control

Acknowledgements
These Slides have been adapted from the originals made available by J. Kurose and K. Ross
All material copyright 1996-2009
J.F Kurose and K.W. Ross, All Rights Reserved

Goals

- Present principles of congestion control
- Instantiation in existing netorks
  - ATM Networks
  - Internet (TCP protocol)

Roadmap

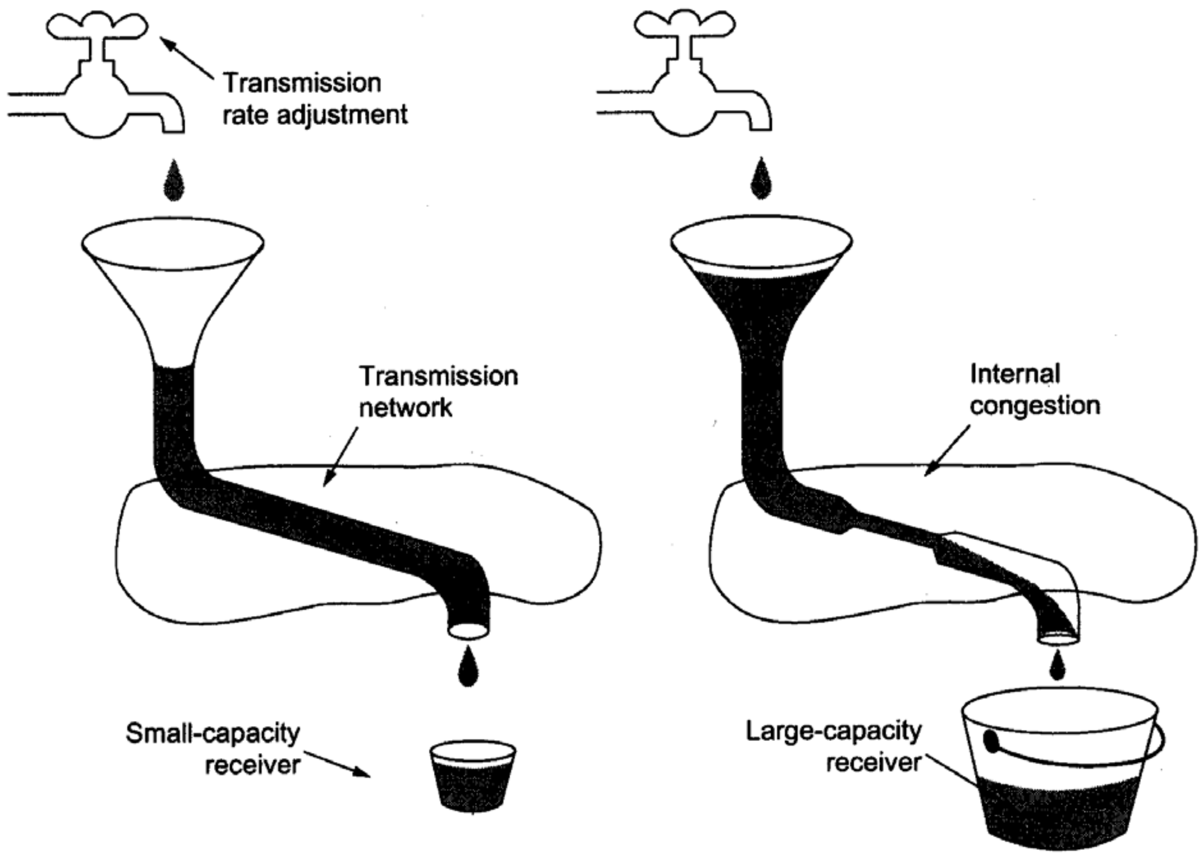
- Principles of congestion control
- Congestion Control in ATM Networks
    - ABR Congestion Control
- Congestion Control in Internet
    - TCP congestion control algorithm

Principles of Congestion Control

Congestion:

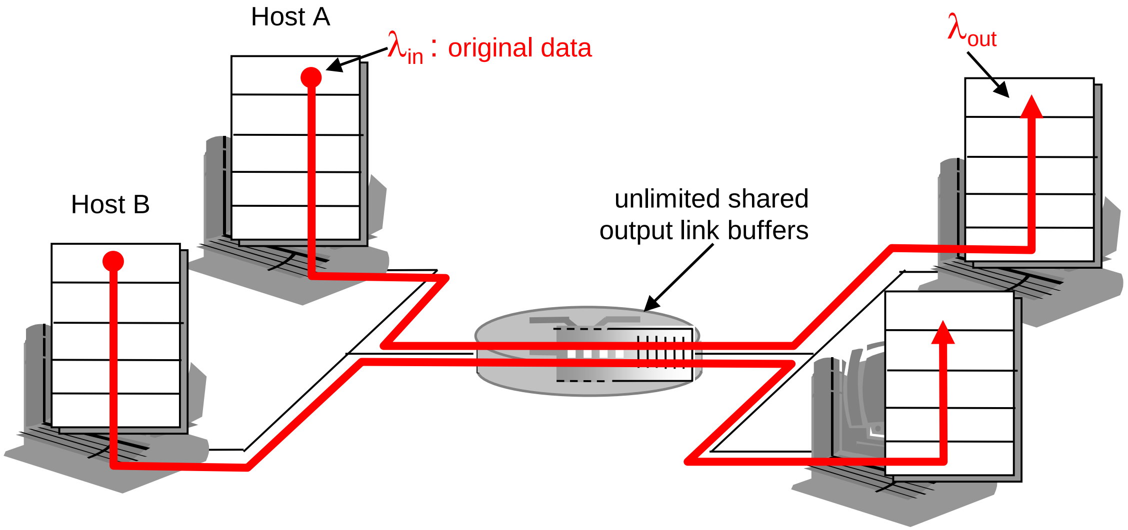
- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- different from flow control!

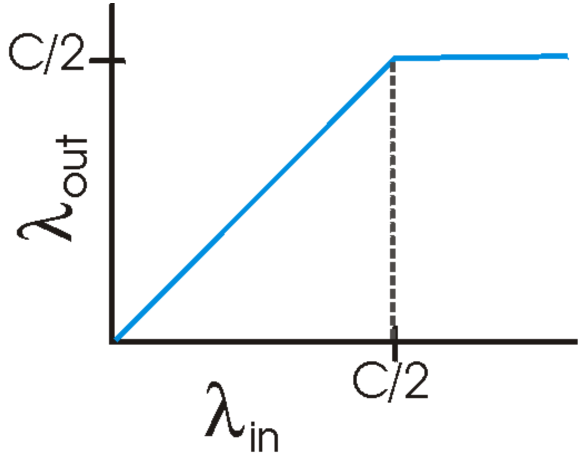
Flow Control vs. Congestion Control

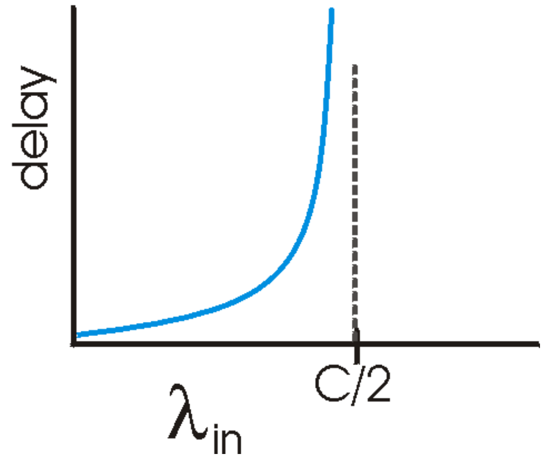


Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission



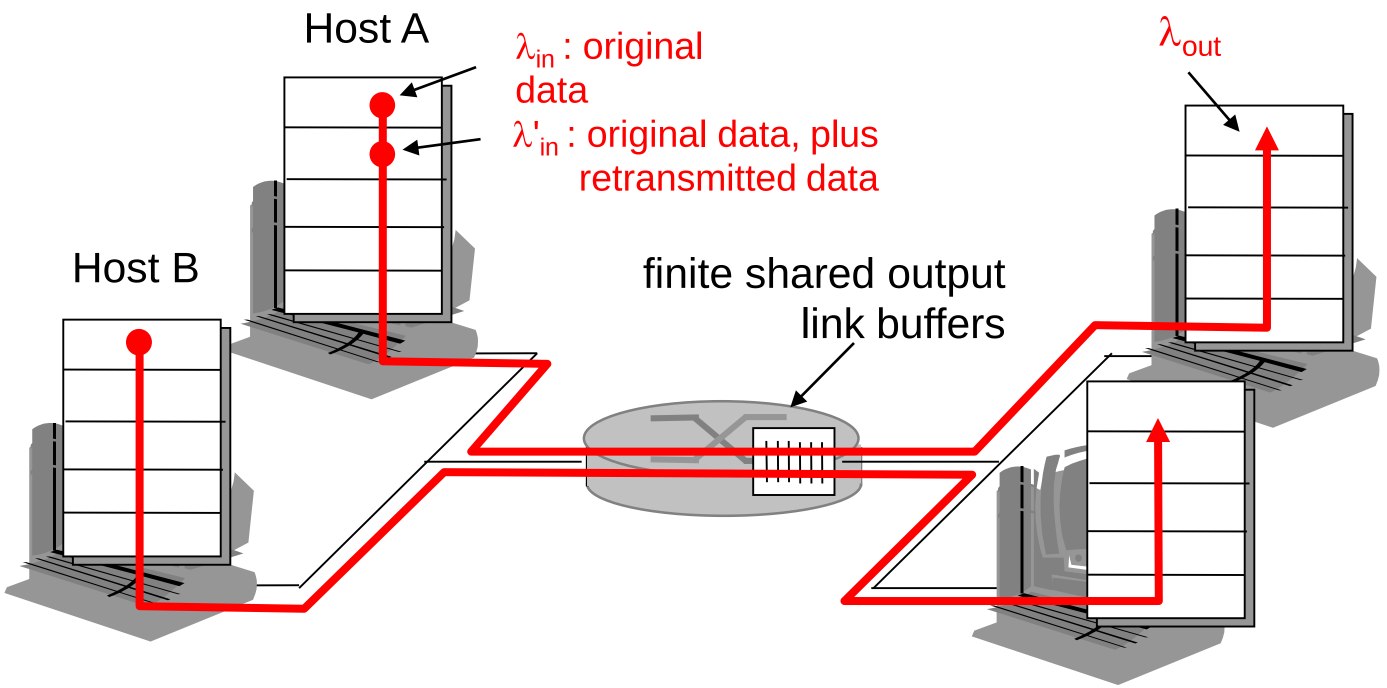




- large delays when congested
- maximum achievable throughput

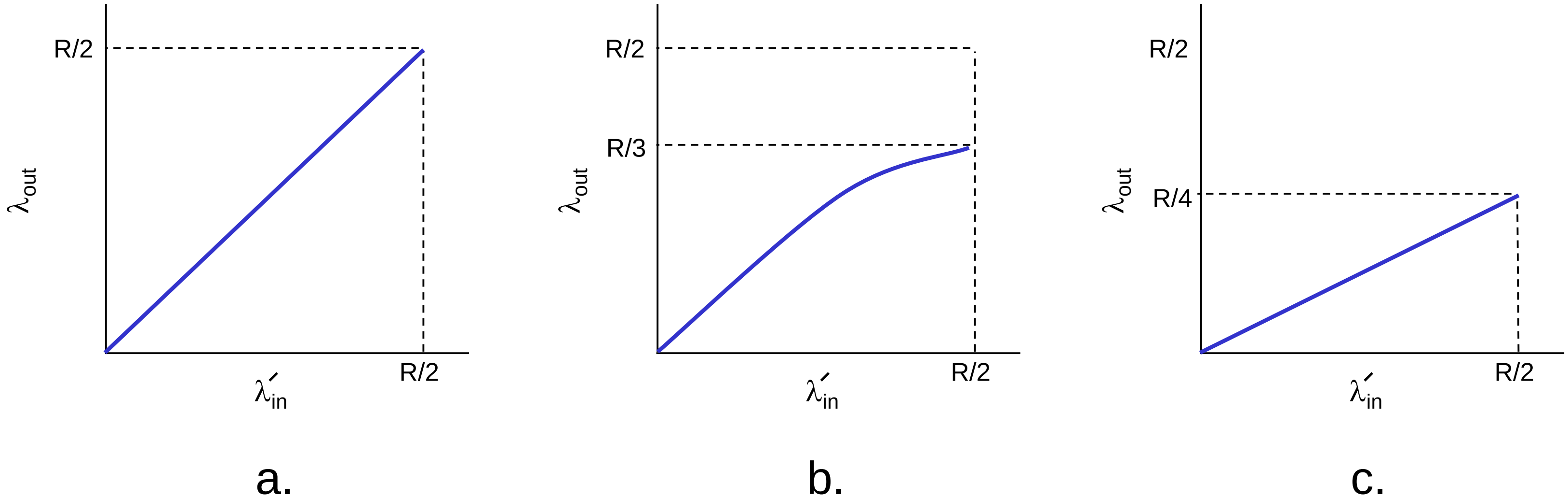
Causes/costs of congestion: scenario 2

- one router, *finite* buffers
- sender retransmission of lost packet



Causes/costs of congestion: scenario 2

- No loss (ideal case): $\lambda_{in} = \lambda_{out}$ (goodput)
- "perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



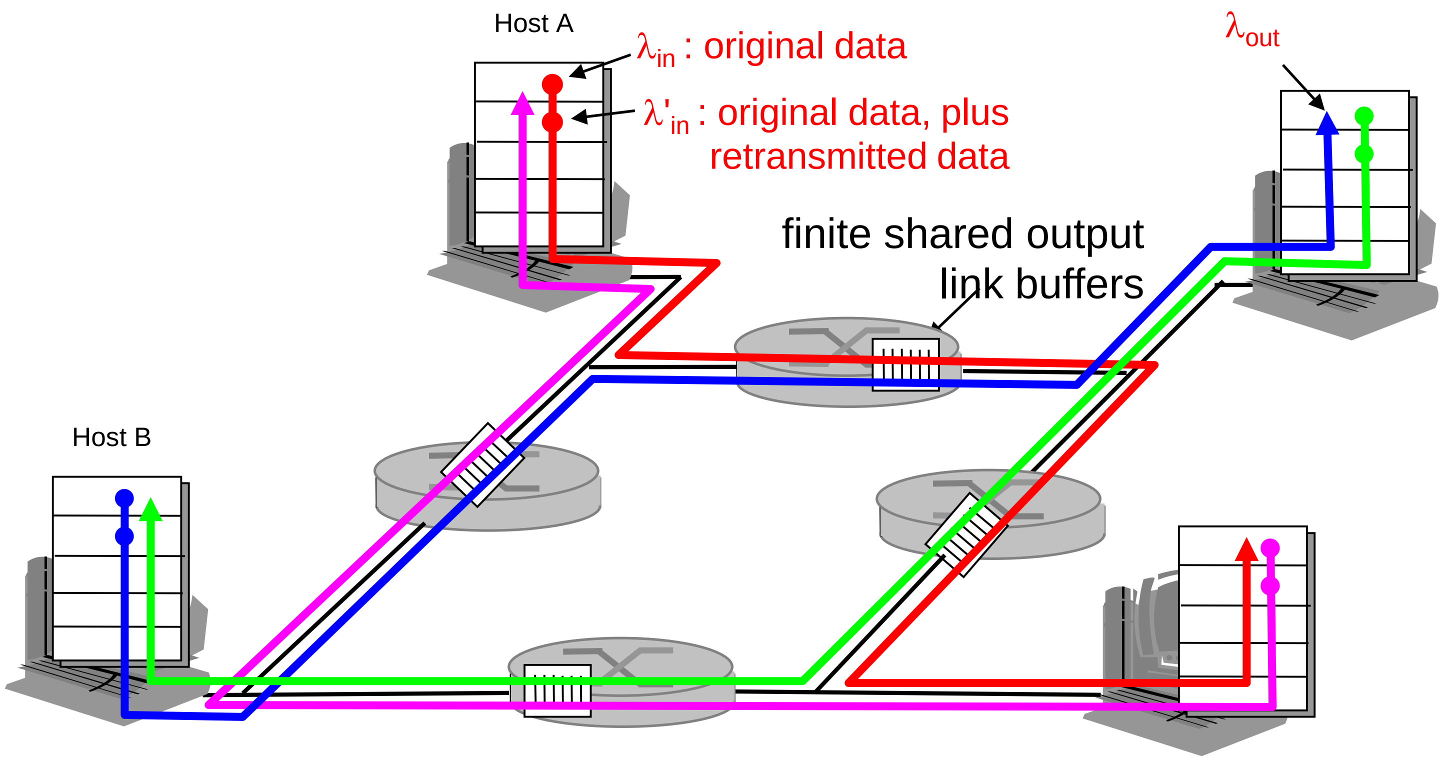
"costs" of congestion:

- more work (re-transmissions) for recovering lost packets
- unneeded retransmissions: link carries multiple copies of pkt

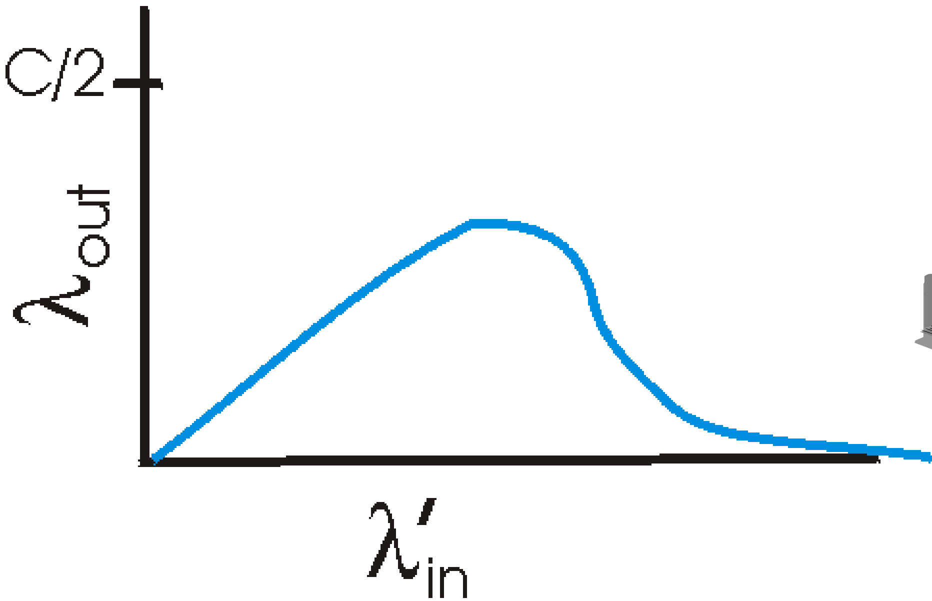
Causes/costs of congestion: scenario 3

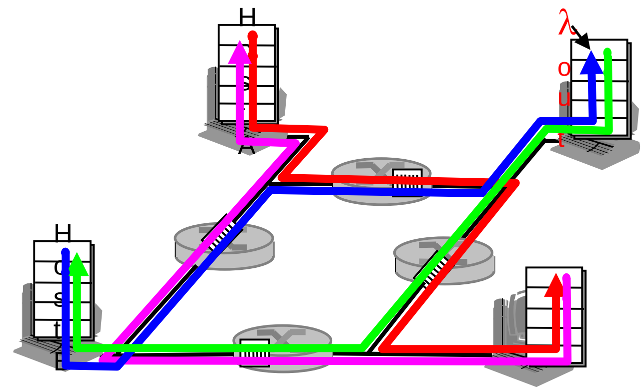
- four senders
- multihop paths
- timeout/retransmit

<u>Q:</u> what happens as λ_{in} and λ'_{in} increase ?



Causes/costs of congestion: scenario 3





another "cost" of congestion:

- when packet dropped, any "upstream transmission capacity used for that packet was wasted!

Approaches to Congestion Control

Network-assisted

- routers provide feedback to end systems
  - single bit indicating congestion
    - SNA,
    - DECnet,
    - ATM
    - TCP/IP ECN
  - explicit rate sender should send at
    - ATM
    - XCP (rated increase/decrease sent to sources)

End-to-end

- no explicit feedback from network
  - congestion inferred from end-system observed loss, delay
  - approach taken by TCP

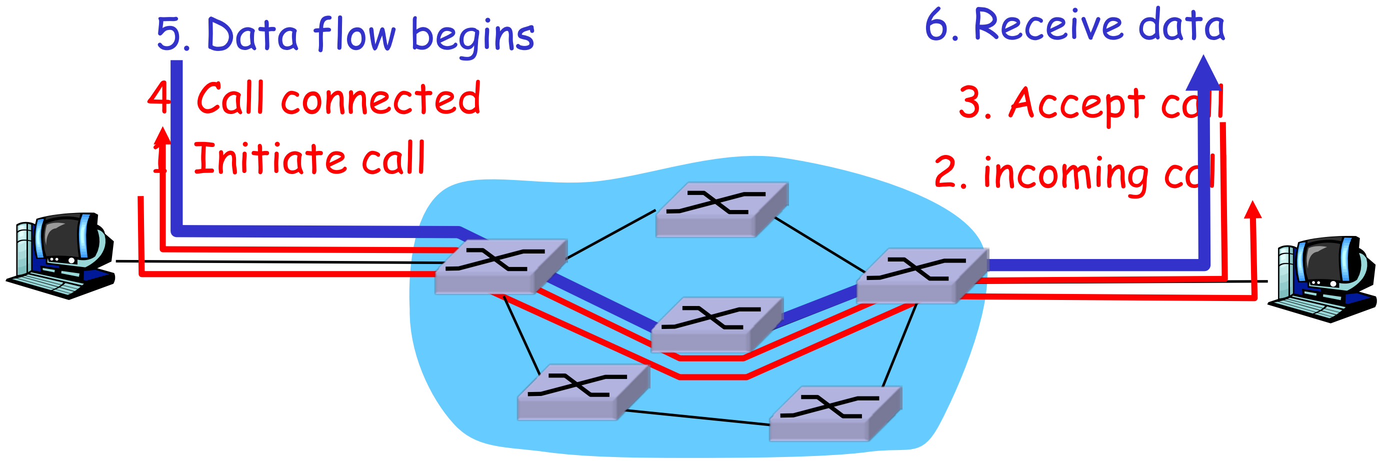
Roadmap

- Principles of congestion control
- Congestion Control in ATM Networks
  - ABR Congestion Control
- Congestion Control in Internet
  - TCP congestion control algorithm

Asynchronous Transfer Mode: ATM

- **1990's/00 standard for high-speed** (155Mbps to 622 Mbps and higher) *Broadband Integrated Service Digital Network* architecture
- <u>Goal:</u> *integrated, end-end transport of carry voice, video, data*
  - meeting timing/QoS requirements of voice, video (versus Internet best-effort model)
  - "next generation" telephony: technical roots in telephone world
  - packet-switching (fixed length packets, called "cells") using virtual circuits

VC setup (and teardown)



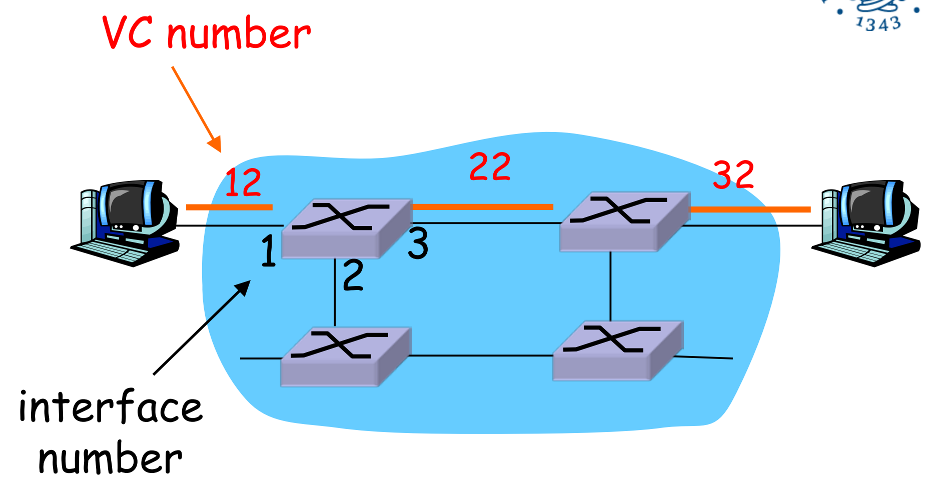
VC implementation

a VC consists of:

1. path from source to destination
2. VC numbers, one number for each link along path
3. entries in forwarding tables in routers along path

- packet belonging to VC carries VC number (rather than dest address)
- VC number can be changed on each link.
  - New VC number comes from forwarding table

Forwarding table



Forwarding table in
A switch

Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
…	…	…	…

Switches maintain connection state information!

ATM Service Classes

- Constant Bit Rate (CBR)
- Variable Bit Rate (VBR)
- Available Bit Rate (ABR)
- Unspecified Bit Rate

ATM ABR Service

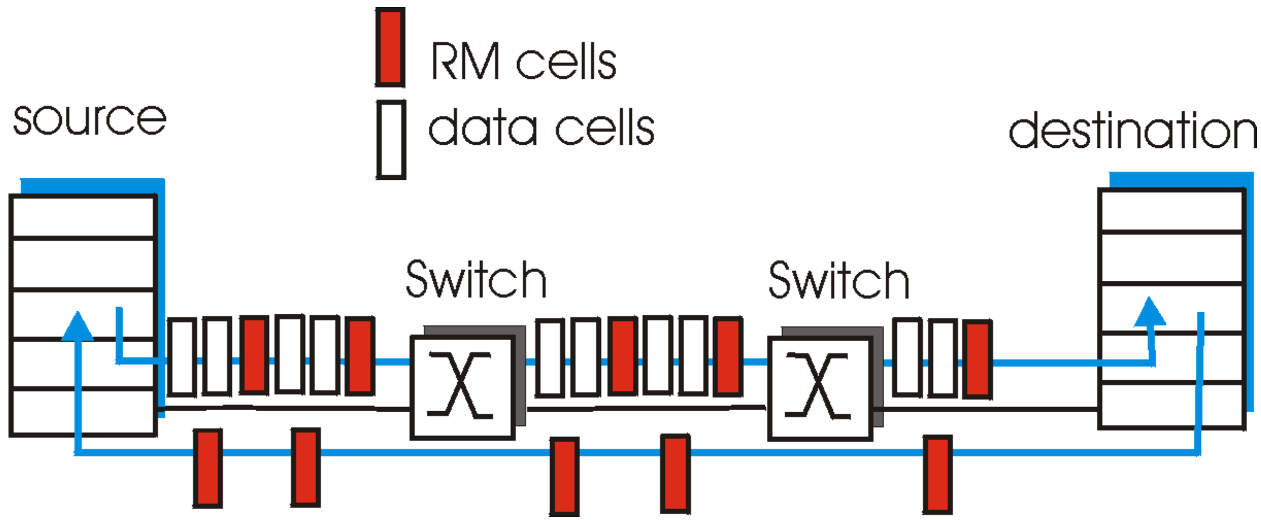
- "elastic service"
  - guaranteed minimum rate
- if sender's path "underloaded":
  - sender should use available bandwidth
- if sender's path congested:
  - sender throttled to minimum guaranteed rate

ATM ABR congestion control

RM (resource management) cells:

- sent by sender, interspersed with data cells
- bits in RM cell set by switches ("*network-assisted*")
  - NI bit: no increase in rate (mild congestion)
  - CI bit: congestion indication
- RM cells returned to sender by receiver
  - possibly after modifying the contents

ATM ABR congestion control



- EFCI bit in data cells: set to 1 in congested switch
  - if data cell preceding RM cell has EFCI set, sender sets CI bit in returned RM cell
- two-byte ER (explicit rate) field in RM cell
  - congested switch may lower ER value in cell
  - sender' send rate thus maximum supportable rate on path

Roadmap

- Principles of congestion control
- Congestion Control in ATM Networks
  - ABR Congestion Control
- Congestion Control in Internet
  - TCP congestion control algorithm

TCP Congestion Control

- GOAL: TCP sender should transmit as fast as possible, but without congesting network

Three Fundamental Questions

- How the sender *limit* its rate based on perceived congestion?
- How the sender *perceive* congestion?
- How the sender *adjust* the rate based on perceived congestion?

Rate Limitation: Congestion Window

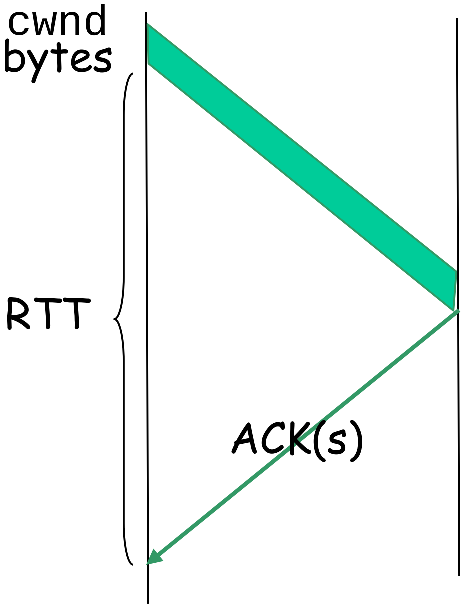
- sender limits rate by limiting number of unACKed bytes "in pipeline":

  `LastByteSent-LastByteAcked` $\leq$ `cwnd`

  - **`cwnd`**: differs from **`rwnd`** (how, why?)
  - sender limited by **`min(cwnd,rwnd)`**
- roughly,

$$\boxed{\text{rate} = \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}}$$

- **`cwnd`** is dynamic, function of perceived network congestion



How Congestion is Perceived?

Each TCP sender sets its own rate, based on *implicit* feedback

- *ACK:* segment received (a good thing!), network not congested, so increase sending rate
- *Lost segment:* assume loss due to congested network, so decrease sending rate
  - Time-out
  - 3 duplicate acks

Congestion Control Algorithm

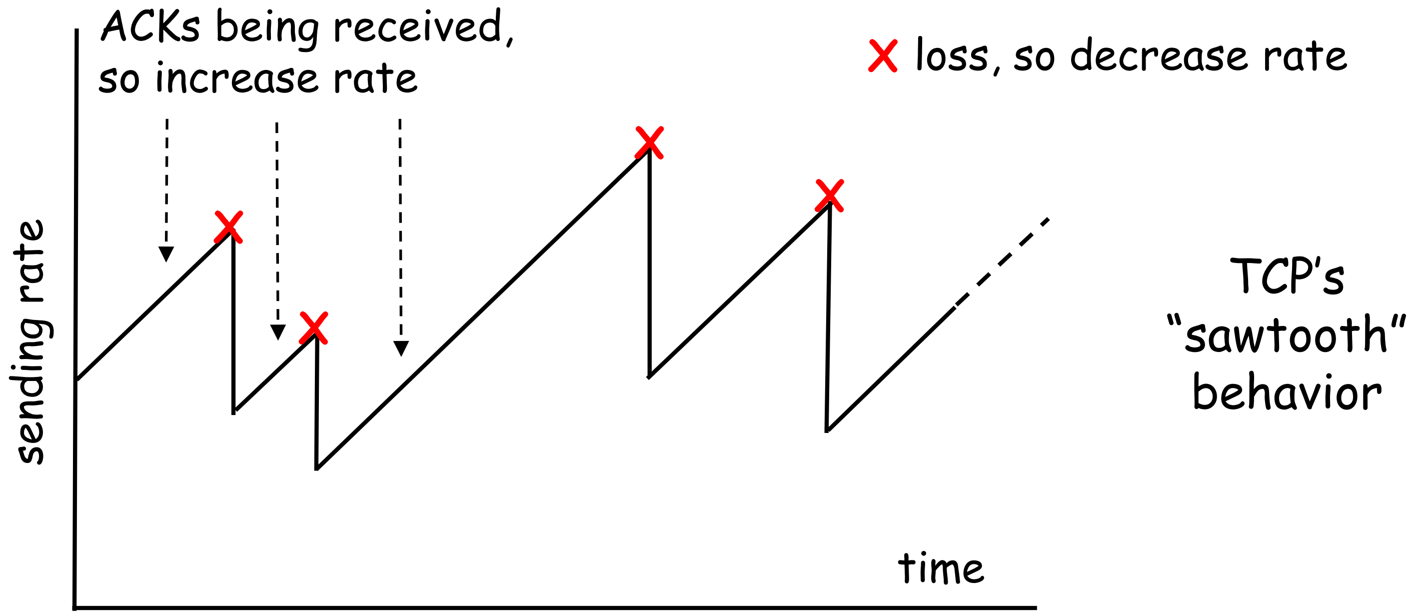
Basic idea

- "probing for bandwidth"

  increase transmission rate on receipt of ACK, until eventually loss occurs, then decrease transmission rate
  - continue to increase linearly on ACK (additive increase)
  - decrease on loss
    - half of the current value (multiplicative decrease)

Congestion Control Algorithm

Probing for bandwidth



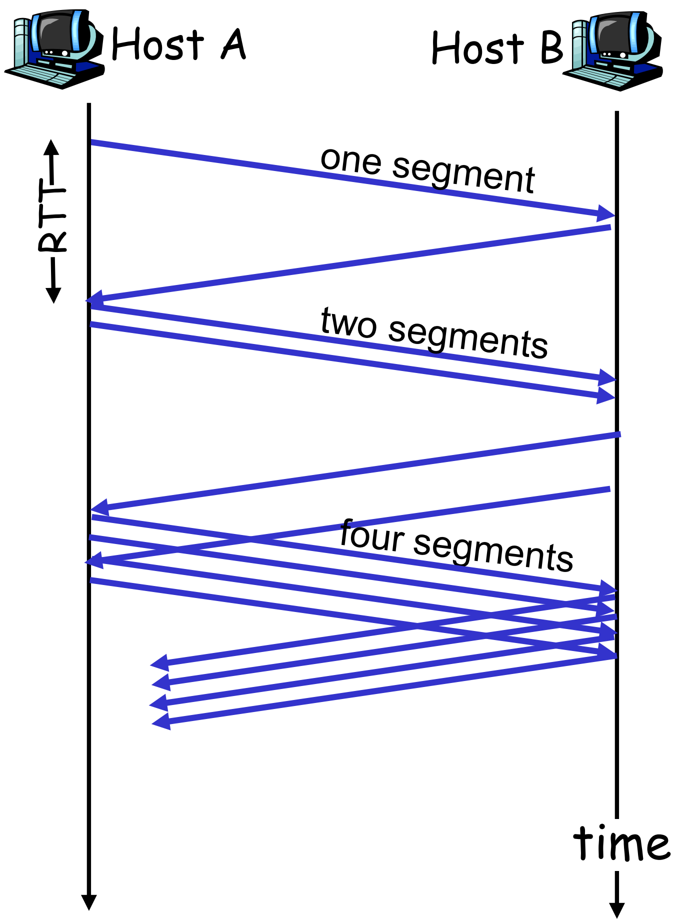
Congestion Control Algorithm

Phases

- Slow Start
- Congestion Avoidance
- Reaction to Timeout Events

Slow Start Phase

- when connection begins:
  cwnd = 1 MSS
  rate= MSS/RTT
- available bandwidth may be
  >> MSS/RTT
- increase rate exponentially until first loss event or when threshold reached
  - double **cwnd** every RTT
  - done by incrementing **cwnd** by 1 for every ACK received



Transitioning out of slowstart

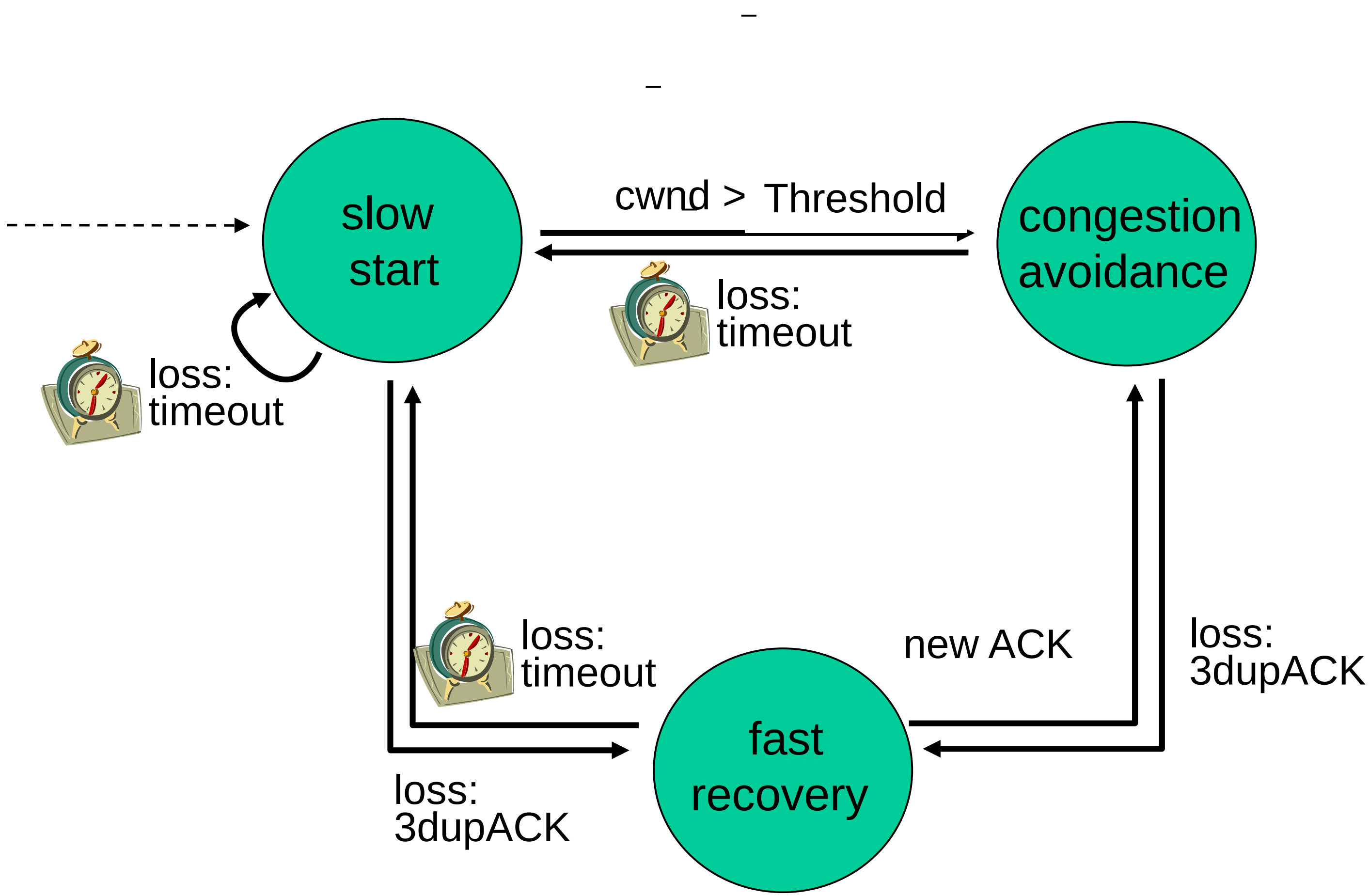
`Threshold:` **`cwnd`** threshold maintained by TCP

- **`If(cwnd >= Threshold) then`** transition from slow start to congestion avoidance phase
- In the congestion avoidance phase **`cwnd`** is increased linearly

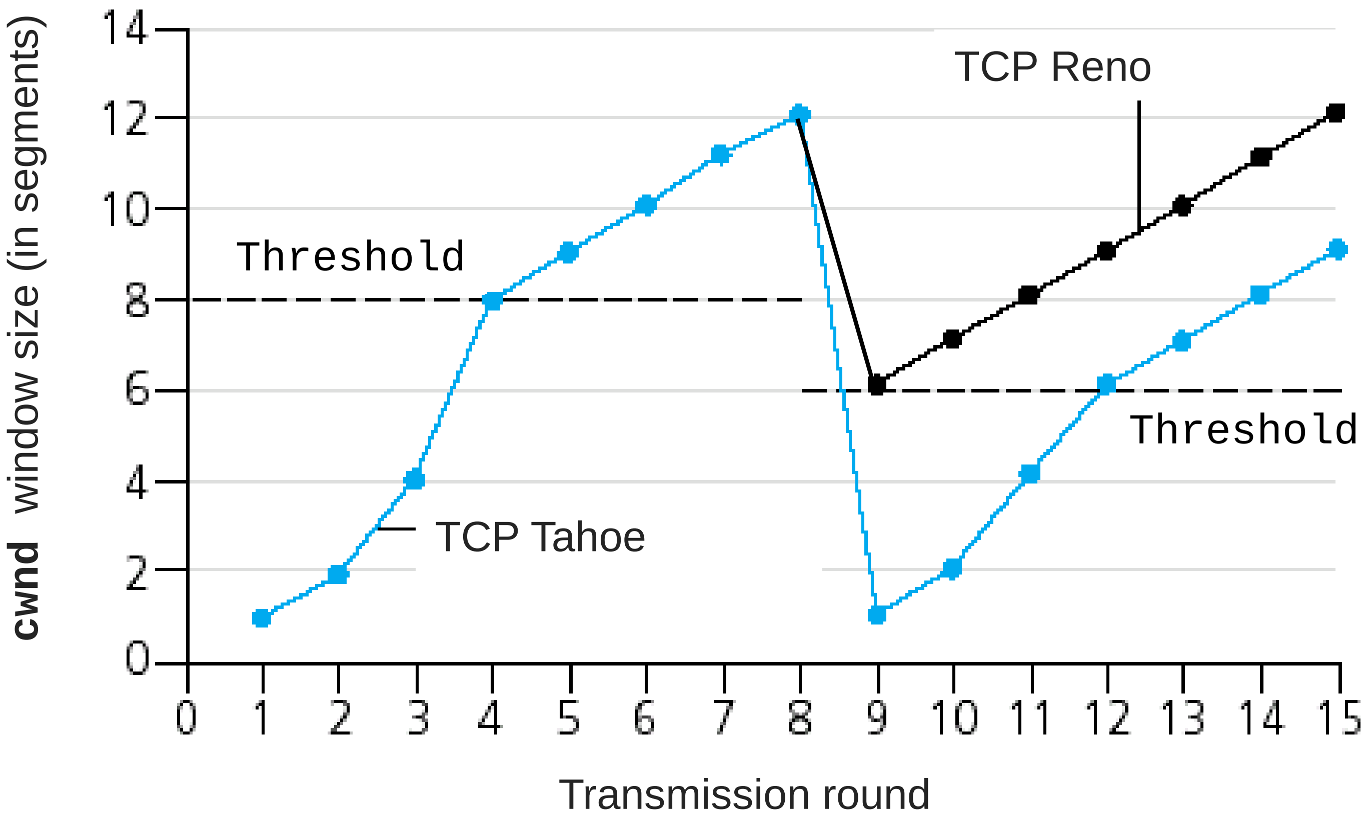
Reaction to Loss

- 3 Duplicate ACKs
  - **`Threshold=Cwind/2`**
  - **`Cwind=Cwind/2 + 3 MSS`**
  - Congestion avoidance (**`cwind`** increases linearly)
    - Fast Recovery
- Timeout
  - **`Threshold=Cwind/2`**
  - **`Cwind=1`**
  - Slow Start (**`cwind`** increases exponetially)

Congestion Control FSM (TCP Reno)



Popular “flavors” of TCP



TCP Congestion Control: Summary

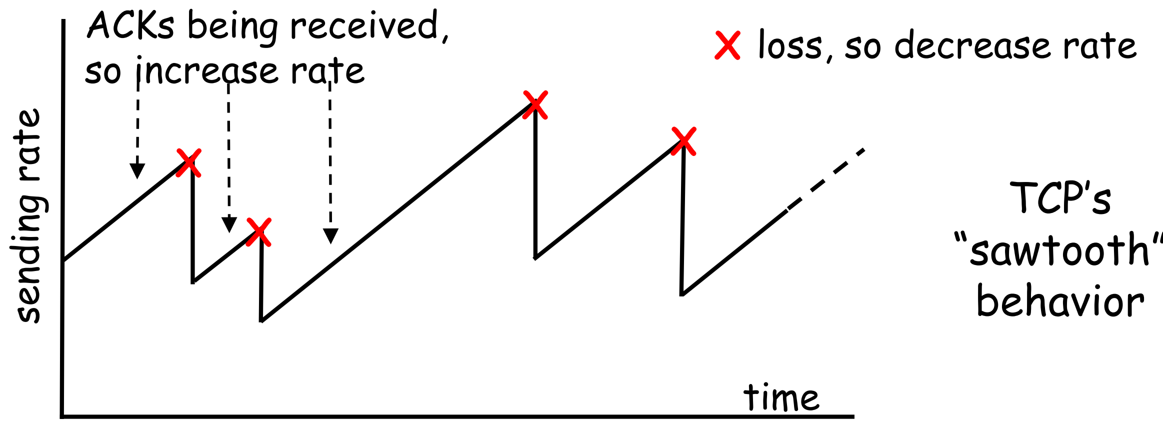
- when **`cwnd < Threshold`**, sender in slow-start phase, window grows exponentially.
- when **`cwnd >= Threshold`**, sender is in congestion-avoidance phase, window grows linearly.
- when triple duplicate ACK occurs, **`Threshold`** set to **`cwnd/2`**, **`cwnd`** set to ~ **`Threshold`**
- when timeout occurs, **`Threshold`** set to **`cwnd/2`**, **`cwnd`** set to 1 MSS.

TCP with lossy links

- The TCP CC assumes that packet loss is due to congestion
- This assumption is not true with lossy links
  - Wireless links
  - Networks with mobile nodes
- TCP CC misinterprets these losses as congestion signals and decreases the rate
  - Explicit congestion notification suggested

TCP Throughput

What's average throughout of TCP as function of window size, RTT?



- let W be window size when loss occurs.
  - when window is W, throughput is W/RTT
  - just after loss, window drops to W/2, throughput to W/2RTT.
  - average throughout: .75 W/RTT

TCP Futures: TCP over “long, fat pipes”

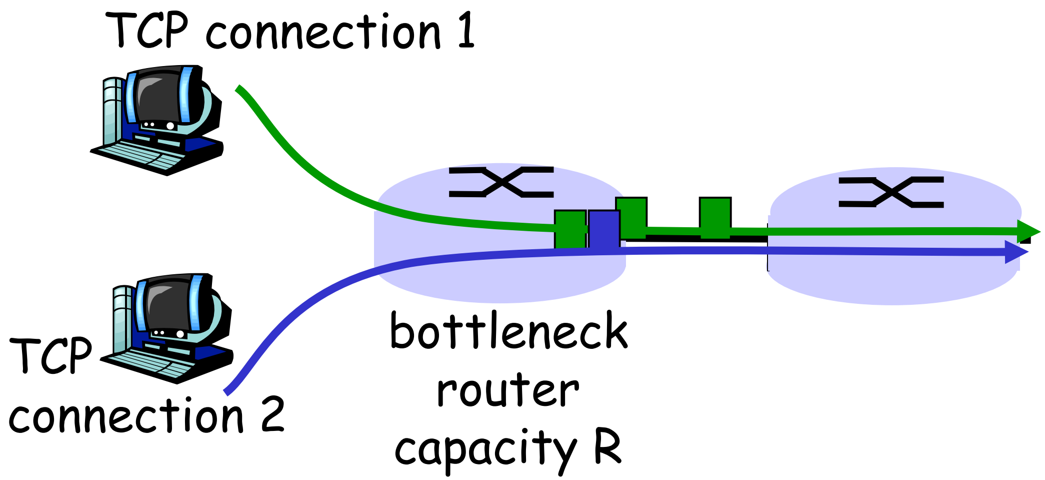
- example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- requires window size W = 83,333 in-flight segments
- throughput in terms of loss rate:

$$\frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

- ➜ $L = 2 \cdot 10^{-10}$ *Wow*
- new versions of TCP for high-speed

TCP Fairness

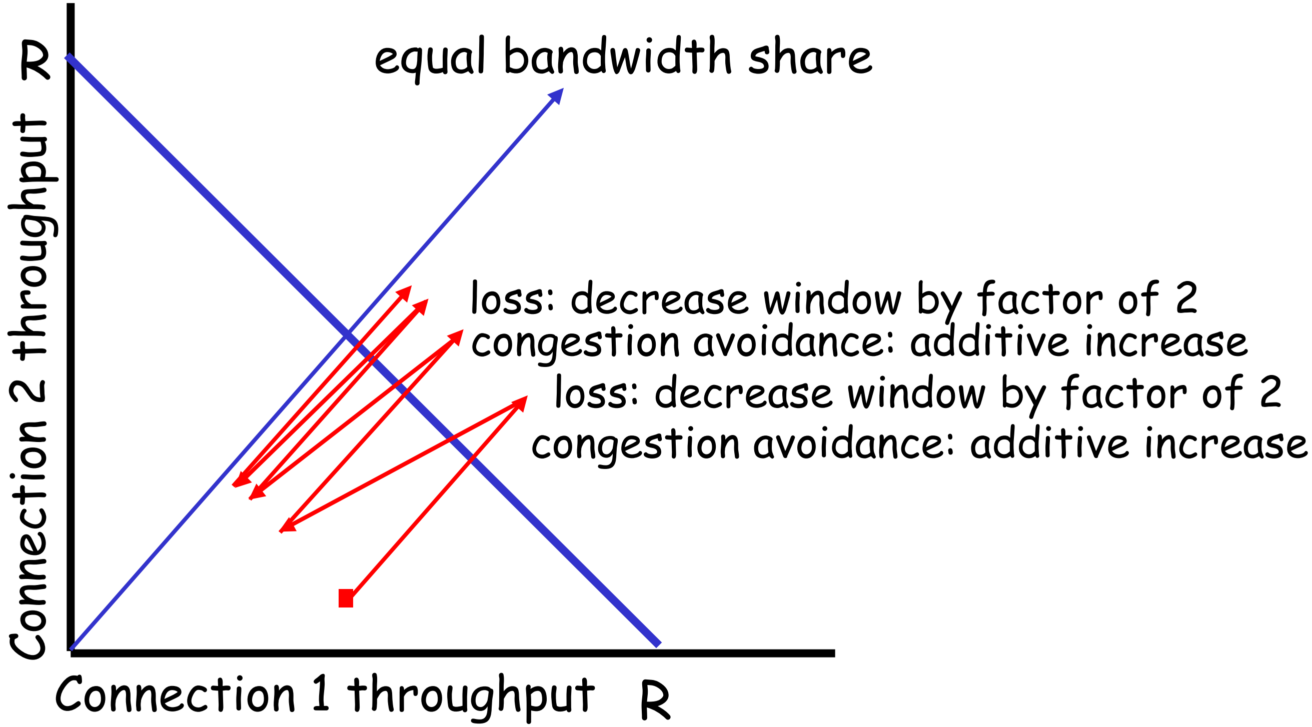
fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

Two competing sessions:

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



Fairness (more)

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.
- web browsers do this
- example: link of rate R supporting 9 connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets more than R/2 !

Summary

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
  - message structure
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP congestion control

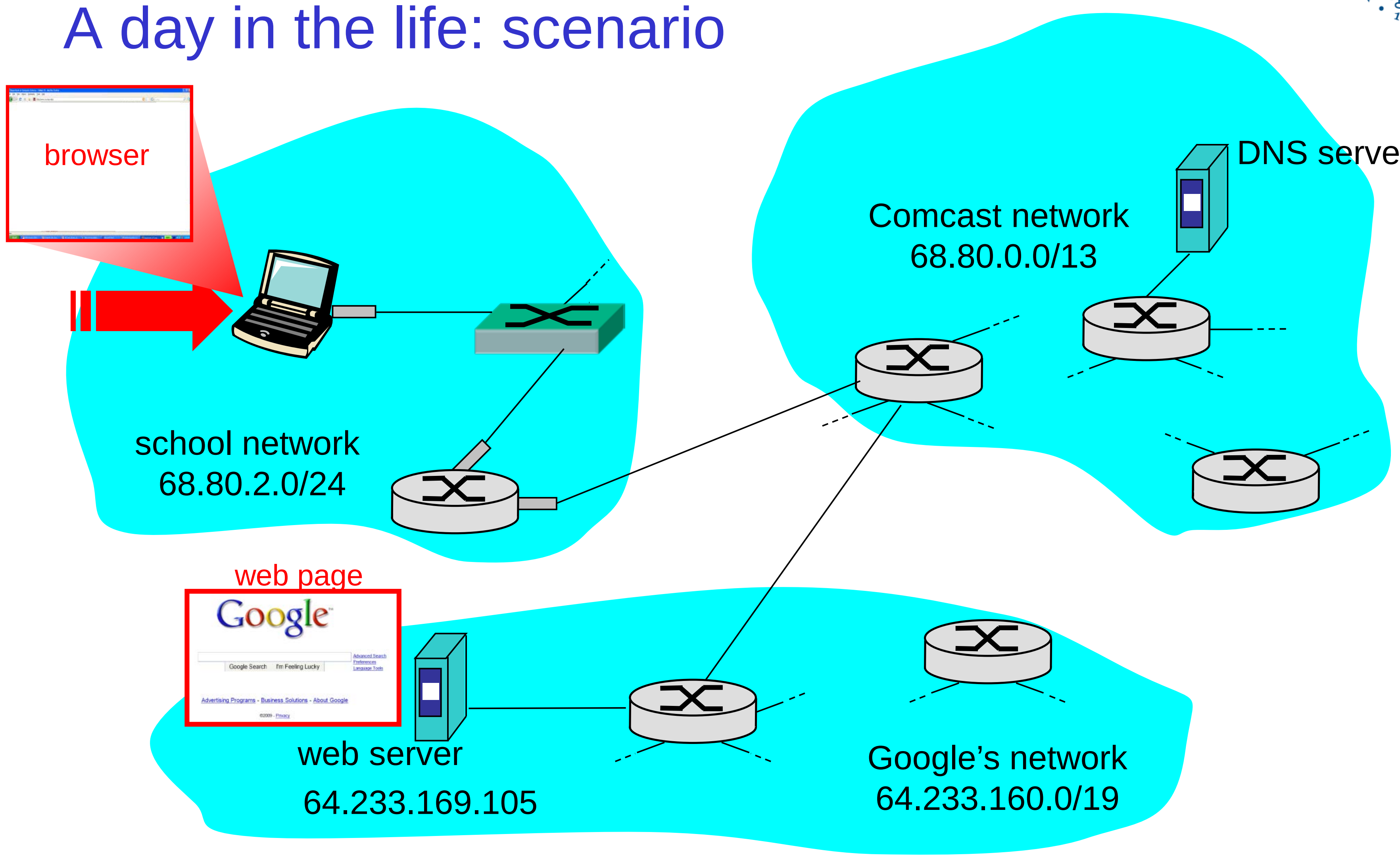
Summary

- Process-to-process data delivery is now possible
- Synthesis: a day in the life of a web request

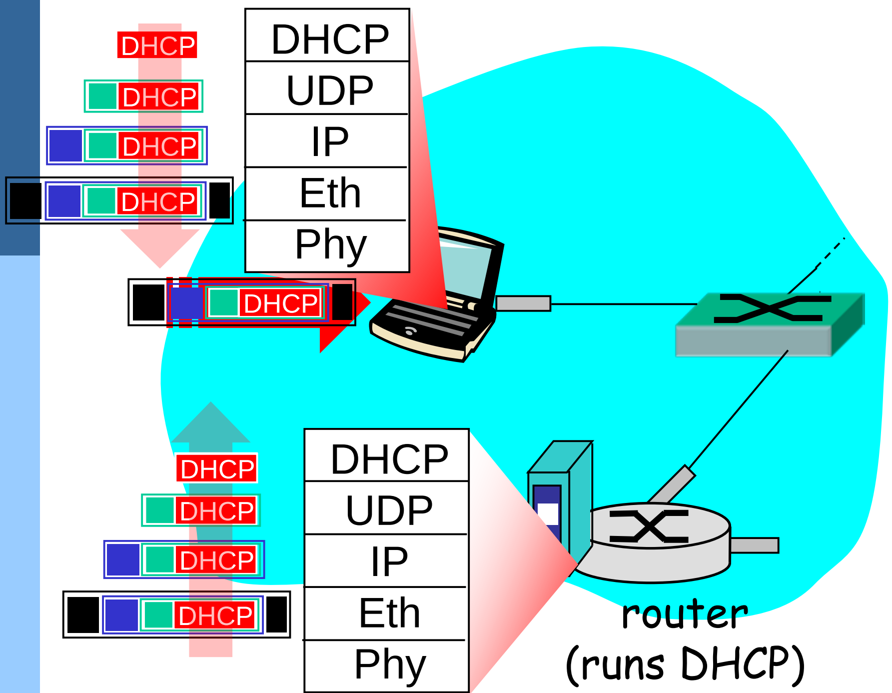
A day in the life of a web request

- journey down protocol stack complete!
  - application, transport, network, link
- putting-it-all-together: synthesis!
  - *goal:* identify, review, understand protocols (at all layers) involved in seemingly simple scenario: requesting www page
  - *scenario:* student attaches laptop to campus network, requests/receives www.google.com

A day in the life: scenario

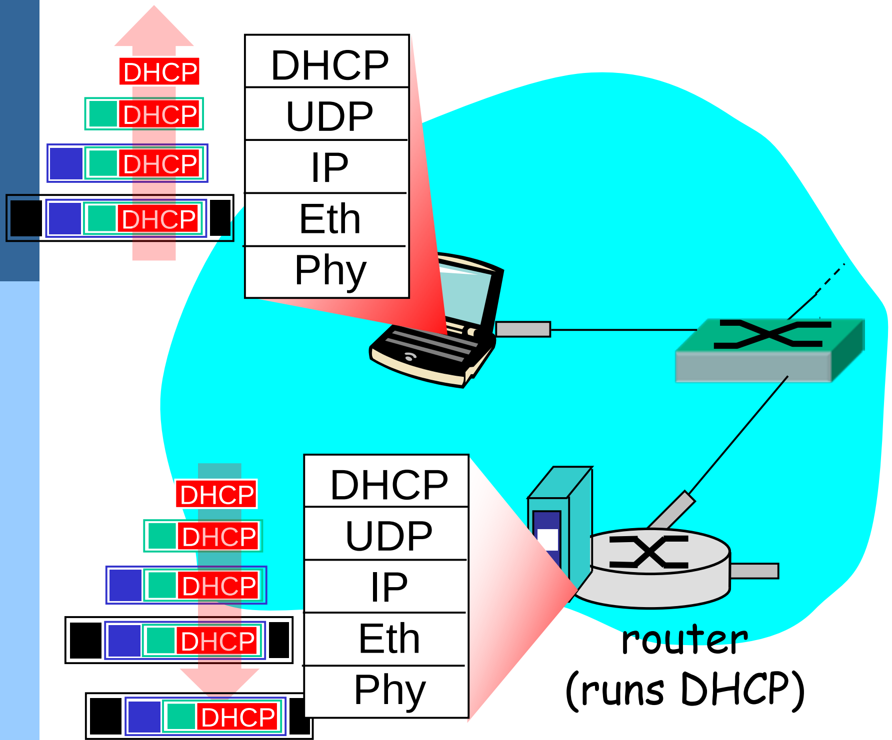


A day in the life… connecting to the Internet



- connecting laptop needs to get its own IP address, addr of first-hop router, addr of DNS server: use ***DHCP***
- DHCP request **encapsulated** in **UDP**, encapsulated in **IP**, encapsulated in **802.3** Ethernet
- Ethernet frame **broadcast** (dest: FFFFFFFFFFFF) on LAN, received at router running **DHCP** server
- Ethernet **demux'ed** to IP demux'ed, UDP demux'ed to DHCP

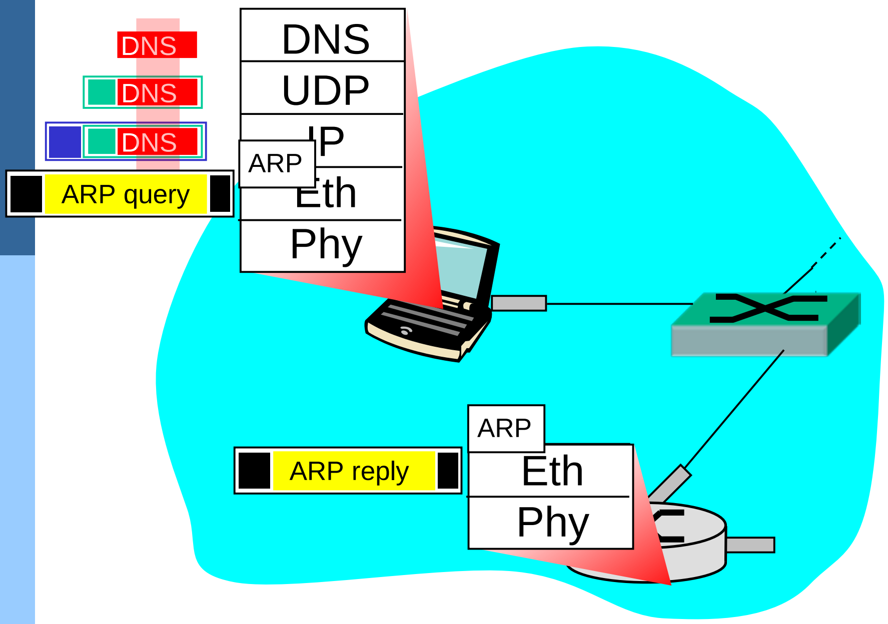
A day in the life… connecting to the Internet



- DHCP server formulates ***DHCP ACK*** containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation at DHCP server, frame forwarded (**switch learning**) through LAN, demultiplexing at client
- DHCP client receives DHCP ACK reply

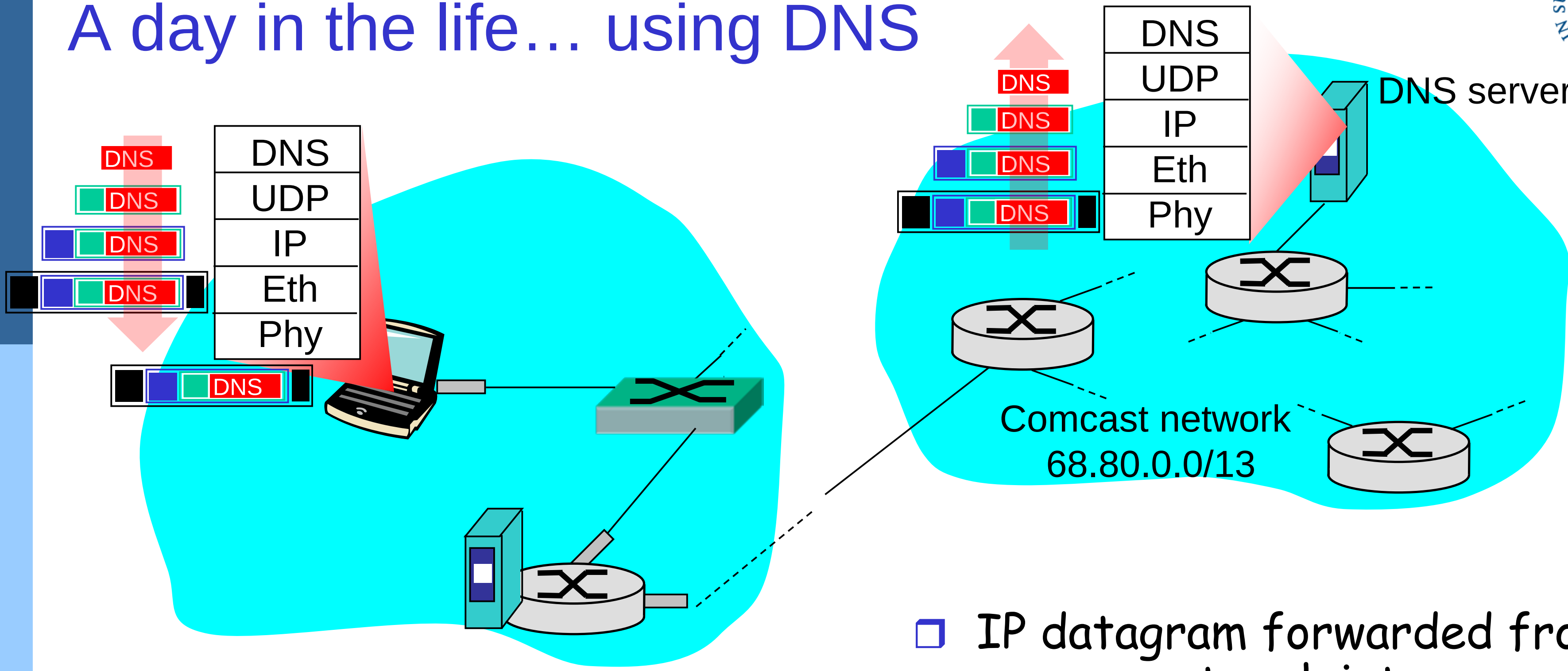
Client now has IP address, knows name & addr of DNS server, IP address of its first-hop router

A day in the life… ARP (before DNS, before HTTP)



- before sending ***HTTP*** request, need IP address of www.google.com: ***DNS***
- DNS query created, encapsulated in UDP, encapsulated in IP, encasulated in Eth. In order to send frame to router, need MAC address of router interface: ***ARP***
- **ARP query** broadcast, received by router, which replies with **ARP reply** giving MAC address of router interface
- client now knows MAC address of first hop router, so can now send frame containing DNS query

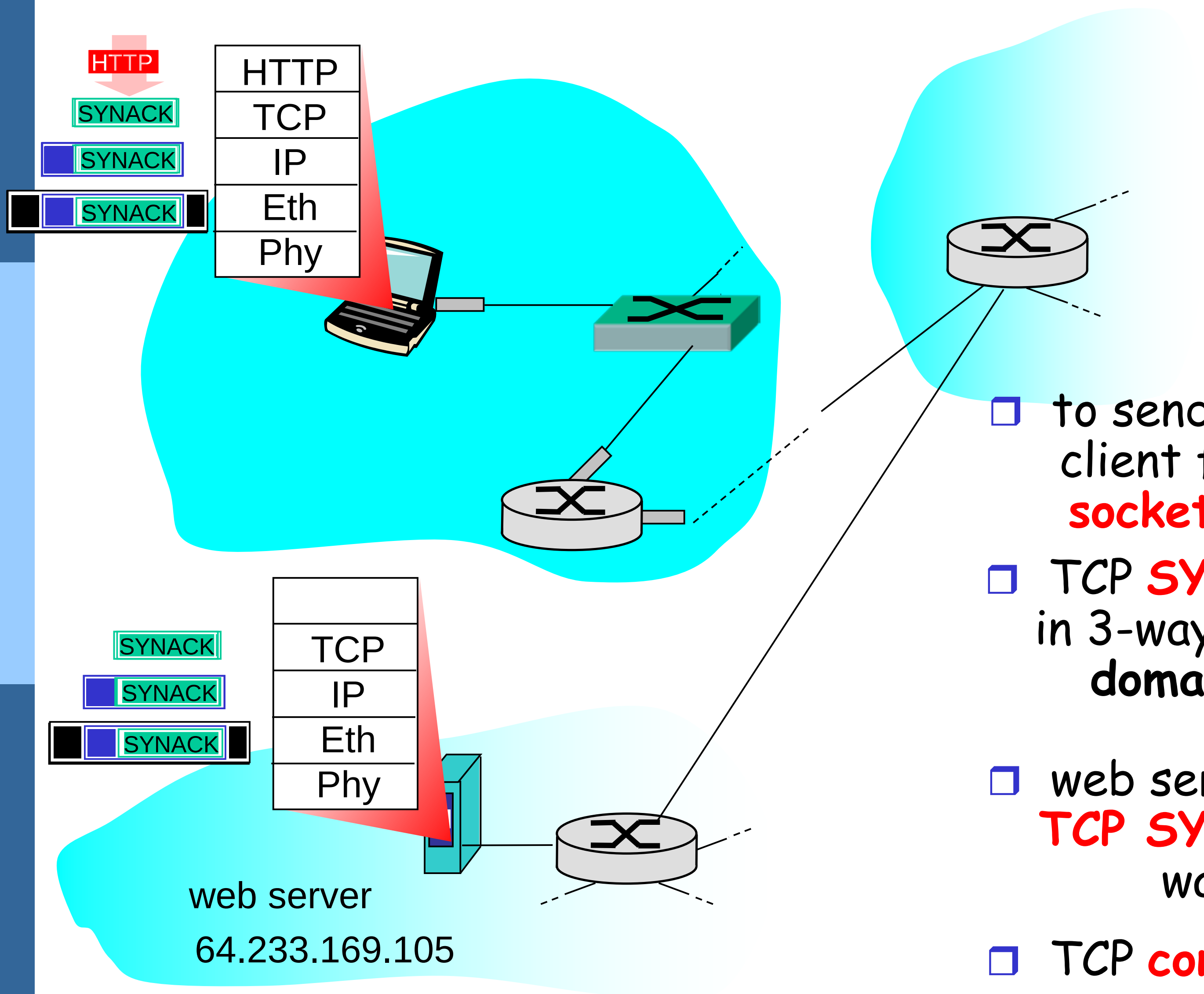
A day in the life… using DNS



- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router

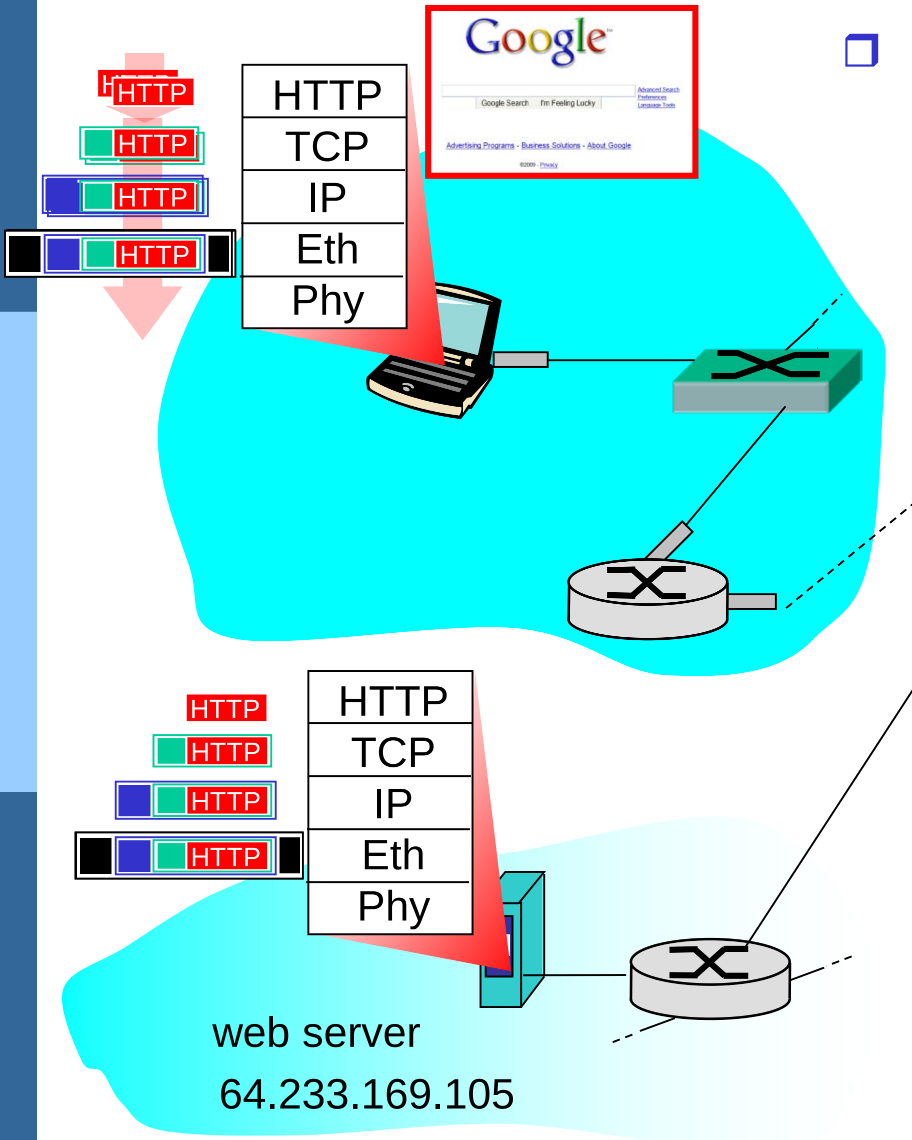
- IP datagram forwarded from campus network into comcast network, routed (tables created by **RIP**, **OSPF**, **IS-IS** and/or **BGP** routing protocols) to DNS server
- demux'ed to DNS server
  - DNS server replies to client with IP address of www.google.com

A day in the life… TCP connection carrying HTTP



- to send HTTP request, client first opens **TCP socket** to web server
- TCP **SYN segment** (step 1 in 3-way handshake) **inter-domain routed** to web server
- web server responds with **TCP SYNACK** (step 2 in 3-way handshake)
- TCP **connection established!**

A day in the life… HTTP request/reply



- web page **finally (!!!)** displayed

- **HTTP request** sent into TCP socket
- IP datagram containing HTTP request routed to www.google.com
- web server responds with **HTTP reply** (containing web page)
- IP datgram containing HTTP reply routed back to client

Let's take a breath

- journey down protocol stack *complete* (except PHY)
- solid understanding of networking principles, practice
- ….. could stop here …. but *lots* of interesting topics!
  - security
  - wireless
  - multimedia