Computer Networking

Michaelmas/Lent Term
M/W/F 11:00-12:00
LT1 in Gates Building

Slide Set 1

Andrew W. Moore
andrew.moore@cl.cam.ac.uk

2015-2016

Topic 1 Foundation

- Administrivia
- Networks
- Channels
- Multiplexing
- Performance: loss, delay, throughput

Course Administration

Commonly Available Texts

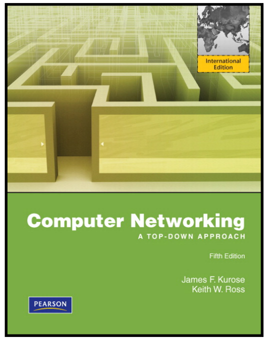
□ Computer Networking: A Top-Down Approach

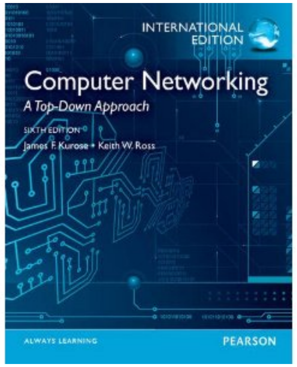
Kurose and Ross, 6th edition 2013, Addison-Wesley

(5th edition is also commonly available)

□ Computer Networks: A Systems Approach

Peterson and Davie, 5th edition 2011, Morgan-Kaufman





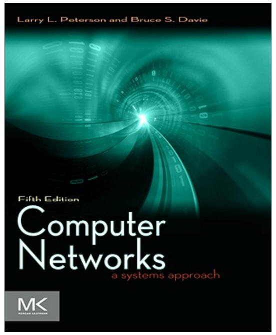
Other Selected Texts (non-representative)

□ Internetworking with TCP/IP, vol. I + II

Comer & Stevens, Prentice Hall

□ UNIX Network Programming, Vol. I

Stevens, Fenner & Rudoff, Prentice Hall



Thanks

- Slides are a fusion of material from

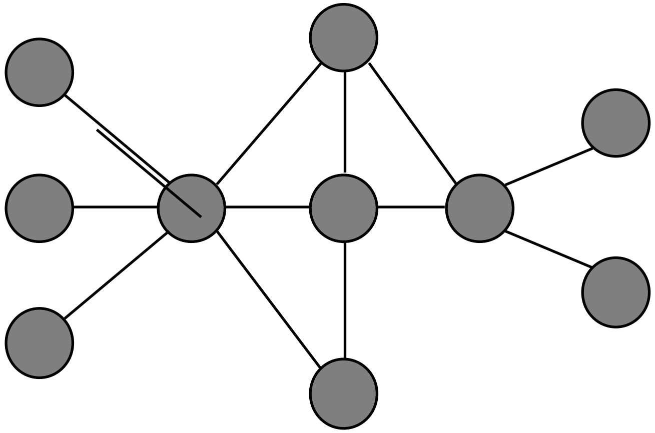
  Brad Smith, Ian Leslie, Richard Black, Jim Kurose, Keith Ross, Larry Peterson, Bruce Davie, Jen Rexford, Ion Stoica, Vern Paxson, Scott Shenker, Frank Kelly, Stefan Savage, Jon Crowcroft , Mark Handley, Sylvia Ratnasamy, and Adam Greenhalgh (and to those others I've forgotten, sorry.)
- Supervision material is drawn from

  Stephen Kell, Andy Rice, and the fantastic TA teams of 144 and 168
- Practical material will become available after a period of development; if you want to lead the practical networking assessment revolution – email me with

  Subject: *CompNet needs practicals*
- Finally thanks to the Part 1b students past and Andrew Rice for all the tremendous feedback.

What is a network?

- A system of “links” that interconnect “nodes” in order to move “information” between nodes



- Yes, this is very vague

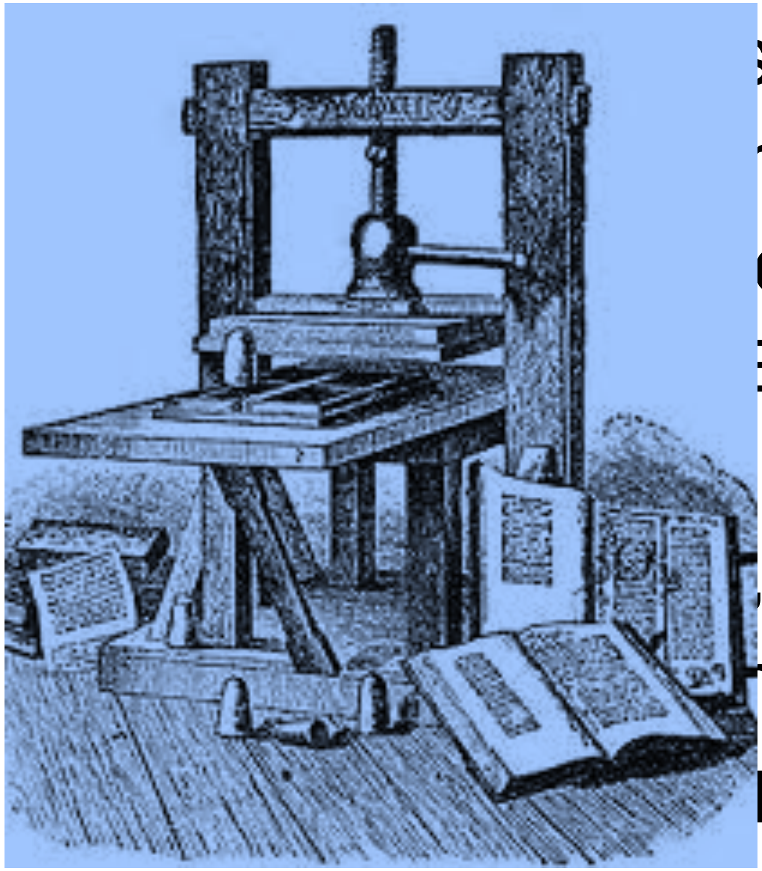
There are *many* different types of networks

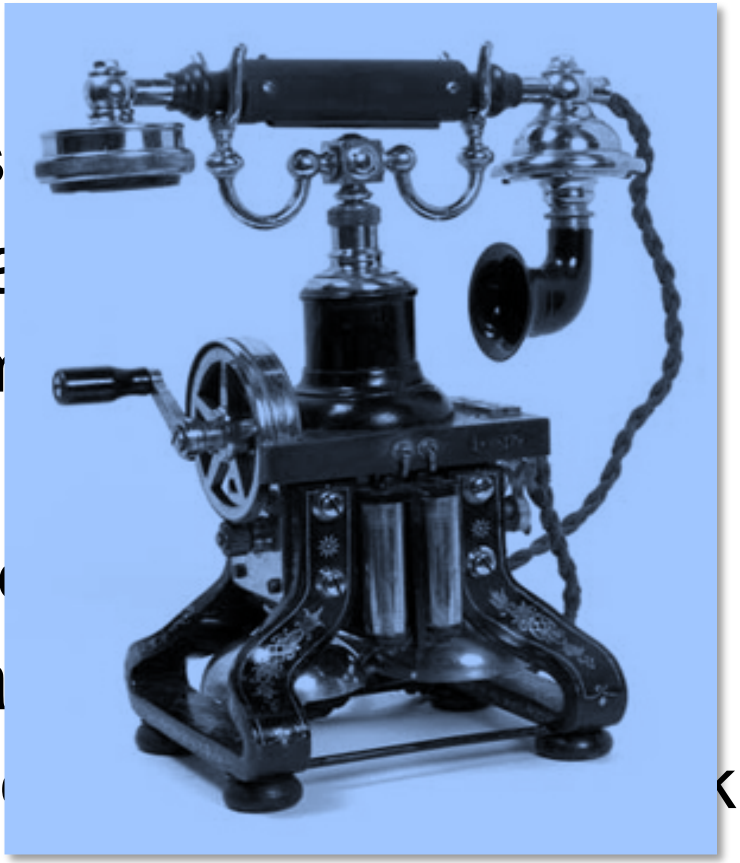
- Internet
- Telephone network
- Transportation networks
- Cellular networks
- Supervisory control and data acquisition networks
- Optical networks
- Sensor networks

We will focus almost exclusively on the Internet

The Internet is transforming everything

- sin
  rtis
- ela
  E-m
-
  , s
- n a
  p, ks





Took the dissemination of information to the next level

The Internet is big business

- Many large and influential networking companies
  - Cisco, Broadcom, AT&T, Verizon, Akamai, Huawei, …
  - $120B+ industry (carrier and enterprise alone)

- Networking central to most technology companies
  - Google, Facebook, Intel, HP, Dell, VMware, …

Internet research has impact

- The Internet started as a research experiment!
- 4 of 10 most cited authors work in networking
- *Many* successful companies have emerged from networking research(ers)

But why is the Internet *interesting*?

“What’s your formal model for the Internet?” -- *theorists*

“Aren’t you just writing software for networks” – *hackers*

“You don’t have performance benchmarks???” – *hardware folks*

“Isn’t it just another network?” – *old timers at AT&T*

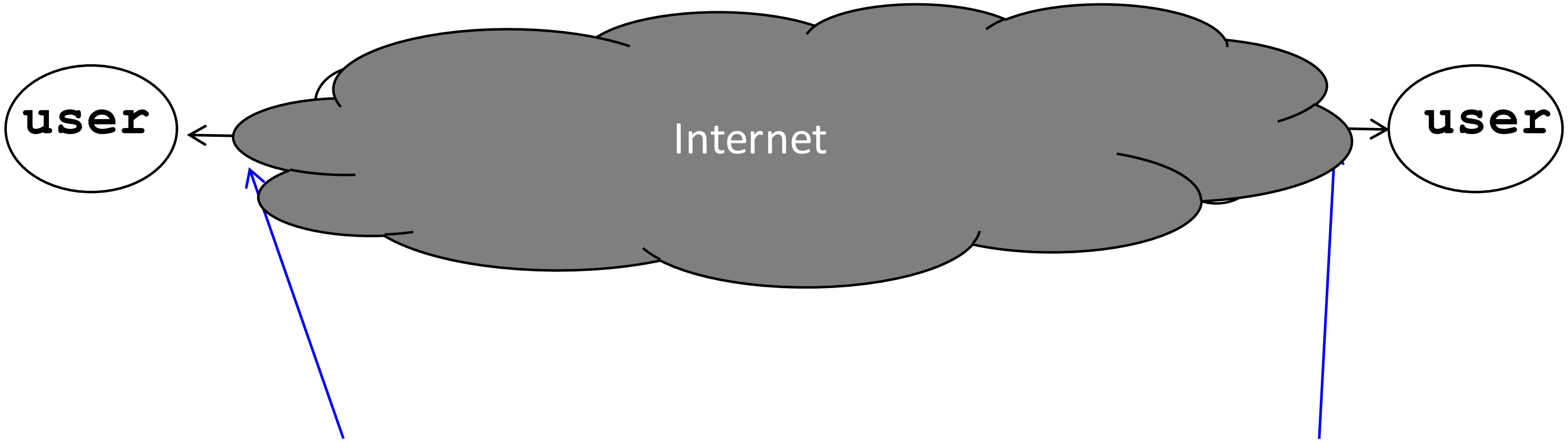
“What’s with all these TLA protocols?” – *all*

“But the Internet seems to be working…” – *my mother*

A few defining characteristics of the Internet

A federated system

- The Internet ties together different networks
  - >18,000 ISP networks



Tied together by IP -- the "Internet Protocol" : a single common interface between users and the network and between networks

A federated system

- The Internet ties together different networks
  - >18,000 ISP networks

- A single, common interface is great for interoperability…
- …but tricky for business

- Why does this matter?
  - ease of interoperability is the Internet’s most important goal
  - practical realities of incentives, economics and real-world trust drive topology, route selection and service evolution

Tremendous scale

- 3.17 Billion users (43% of world population)
- 1+ Trillion unique URLs from 1+ Billion web servers
- 215 Billion emails sent per day
- 1.86 Billion smartphones
- 1.44 Billion Facebook users
- 64 Billion WhatsApp messages per day
- 4 Billion YouTube videos watched per day
- 300 hours of video added per minute
- Routers that switch 322 Terabits/second
- Links that carry 400Gigabits/second

Enormous diversity and dynamic range

- Communication latency: microseconds to seconds (10^6)
- Bandwidth: 1Kbits/second to 100 Gigabits/second (10^7)
- Packet loss: 0 – 90%

- Technology: optical, wireless, satellite, copper

- Endpoint devices: from sensors and cell phones to datacenters and supercomputers
- Applications: social networking, file transfer, skype, live TV, gaming, remote medicine, backup, IM
- Users: the governing, governed, operators, malicious, naïve, savvy, embarrassed, paranoid, addicted, cheap …

Constant Evolution

1970s:

- 56kilobits/second “backbone” links
- <100 computers, a handful of sites in the US (and one UK)
- Telnet and file transfer are the “killer” applications

Today

- 100+Gigabits/second backbone links
- 10B+ devices, all over the globe
- *20M* Facebook apps installed per day

Asynchronous Operation

- Fundamental constraint: **speed of light**
- Consider:
  - How many cycles does your 3GHz CPU in Cambridge execute before it can possibly get a response from a message it sends to a server in Palo Alto?
    - Cambridge to Palo Alto: 8,609 km
    - Traveling at 300,000 km/s: 28.70 milliseconds
    - Then back to Cambridge: 2 x 28.70 = 57.39 milliseconds
    - 3,000,000,000 cycles/sec * 0.05739 = 172,179,999 cycles!
- Thus, communication feedback is always *dated*

Prone to Failure

- To send a message, **all** components along a path must function correctly
  - software, modem, wireless access point, firewall, links, network interface cards, switches,…
  - Including human operators
- Consider: 50 components, that work correctly 99% of time → 39.5% chance communication will fail
- Plus, recall
  - scale → lots of components
  - asynchrony → takes a long time to hear (bad) news
  - federation (**inter**net) → hard to identify fault or assign blame

An Engineered System

- Constrained by what technology is practical
  - Link bandwidths
  - Switch port counts
  - Bit error rates
  - Cost
  - …

Recap: The Internet is…

- A complex federation
- Of enormous scale
- Dynamic range
- Diversity
- Constantly evolving
- Asynchronous in operation
- Failure prone
- Constrained by what’s practical to engineer
- Too complex for theoretical models
- “Working code” doesn’t mean much
- Performance benchmarks are too narrow

Performance – not just bits per second

Second order effects

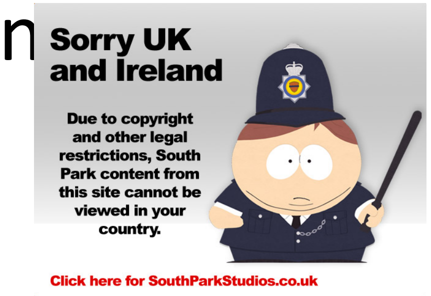
- Image/Audio quality

Other metrics…

- Network efficiency (good-put *versus* throughput)

- User Experience? (World Wide Wait)

- Network connectivity expectation



Sorry UK and Ireland

Due to copyright and other legal restrictions, South Park content from this site cannot be viewed in your country.

Click here for SouthParkStudios.co.uk

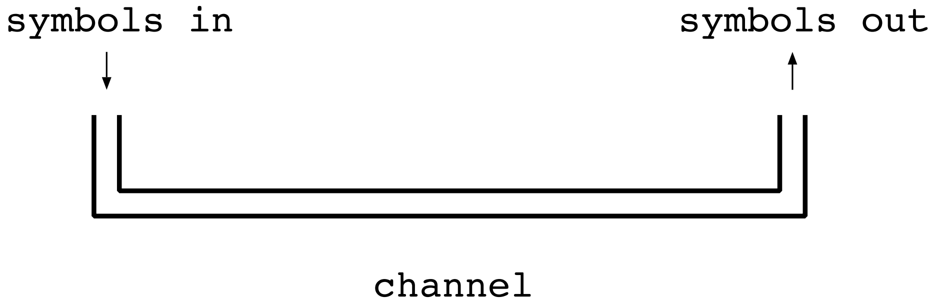
- Others?

Channels Concept

(This channel definition is very abstract)

- Peer entities communicate over channels
- Peer entities provide higher-layer peers with higher-layer channels

A channel is that into which an entity puts symbols and which causes those symbols *(or a reasonable approximation) to appear somewhere else at a later point in time.*



Channel Characteristics

Symbol type: bits, packets, waveform

Capacity: bandwidth, data-rate, packet-rate

Delay: fixed or variable

Fidelity: signal-to-noise, bit error rate, packet error rate

Cost: per attachment, for use

Reliability

Security: privacy, unforgability

Order preserving: always, almost, usually

Connectivity: point-to-point, to-many, many-to-many

Examples:

- Fibre Cable
- 1 Gb/s channel in a network
- Sequence of packets transmitted between hosts
- A telephone call (handset to handset)
- The audio channel in a room
- Conversation between two people

Example Physical Channels

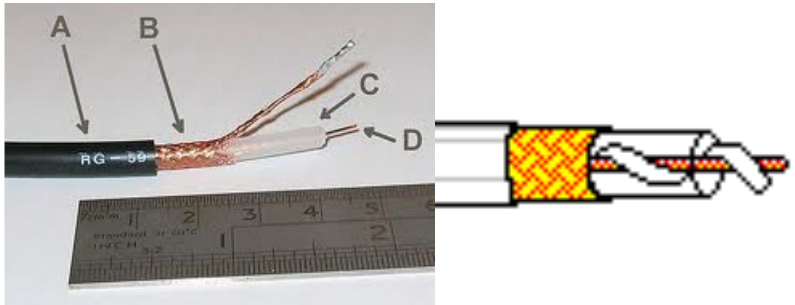
these example physical channels are also known as *Physical Media*

Twisted Pair (TP)

- two insulated copper wires
  - Category 3: traditional phone wires, 10 Mbps Ethernet
  - Category 6: 1Gbps Ethernet
- Shielded (STP)
- Unshielded (UTP)

Coaxial cable:

- two concentric copper conductors
- bidirectional
- baseband:
  - single channel on cable
  - legacy Ethernet
- broadband:
  - multiple channels on cable
  - HFC (Hybrid Fiber Coax)



Fiber optic cable:

- high-speed operation
- point-to-point transmission
- (10’s-100’s Gps)
- low error rate
- immune to electromagnetic noise

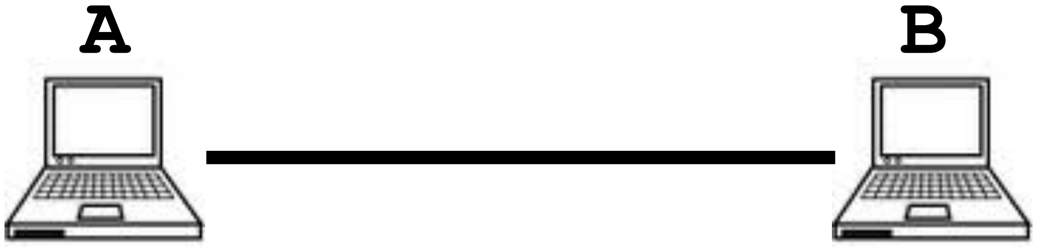
More Physical media: Radio

- Bidirectional and multiple access
- propagation environment effects:
  - reflection
  - obstruction by objects
  - interference

Radio link types:

- terrestrial microwave
  - e.g. 45 Mbps channels
- LAN (e.g., Wifi)
  - 11Mbps, 54 Mbps, 200 Mbps
- wide-area (e.g., cellular)
  - 4G cellular: ~ 4 Mbps
- satellite
  - Kbps to 45Mbps channel (or multiple smaller channels)
  - 270 msec end-end delay
  - geosynchronous versus low altitude

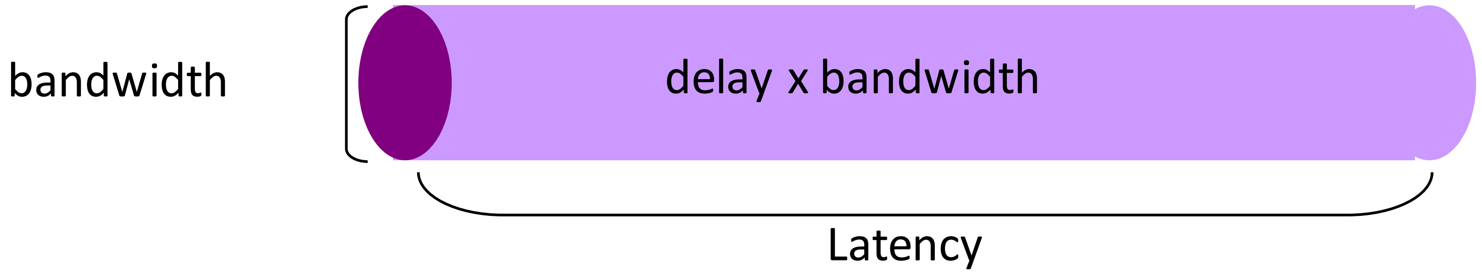
Nodes and Links



Channels = Links

Peer entities = Nodes

Properties of Links (Channels)



- Bandwidth (capacity): “width” of the links
  - number of bits sent (or received) per unit time (bits/sec or bps)
- Latency (delay): “length” of the link
  - propagation time for data to travel along the link(seconds)
- Bandwidth-Delay Product (BDP): “volume” of the link
  - amount of data that can be “in flight” at any time
  - propagation delay × bits/time = total bits in link

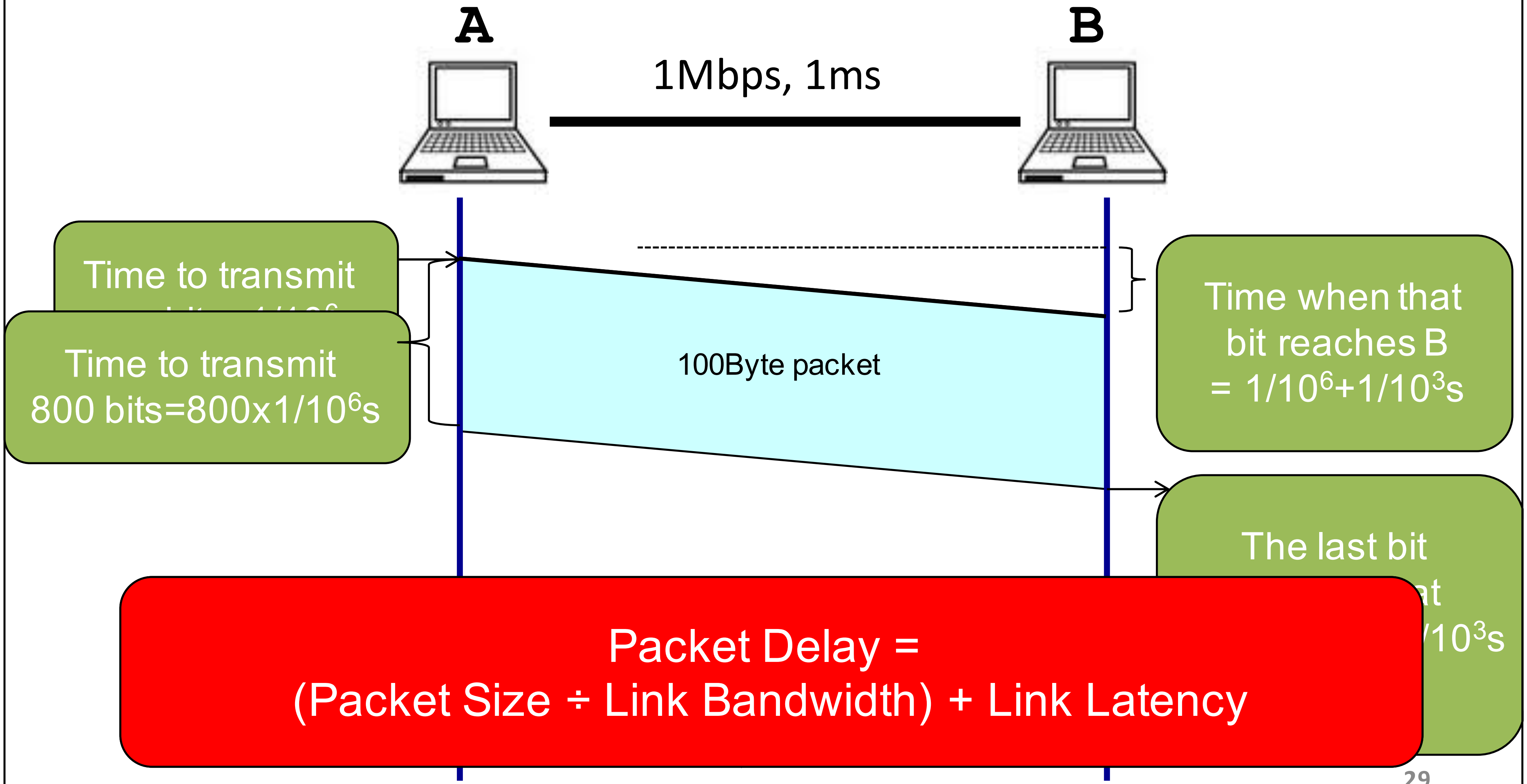
Examples of Bandwidth-Delay

- Same city over a slow link:
  - BW~100Mbps
  - Latency~0.1msec
  - BDP ~ 10,000bits ~ 1.25KBytes

- Cross-country over fast link:
  - BW~10Gbps
  - Latency~10msec
  - BDP ~ 10^8bits ~ 12.5GBytes

Packet Delay

Sending a 100B packet from A to B?



Time to transmit

Time to transmit
800 bits=800x1/10^6s

Time when that
bit reaches B
= 1/10^6+1/10^3s

The last bit
at
/10^3s

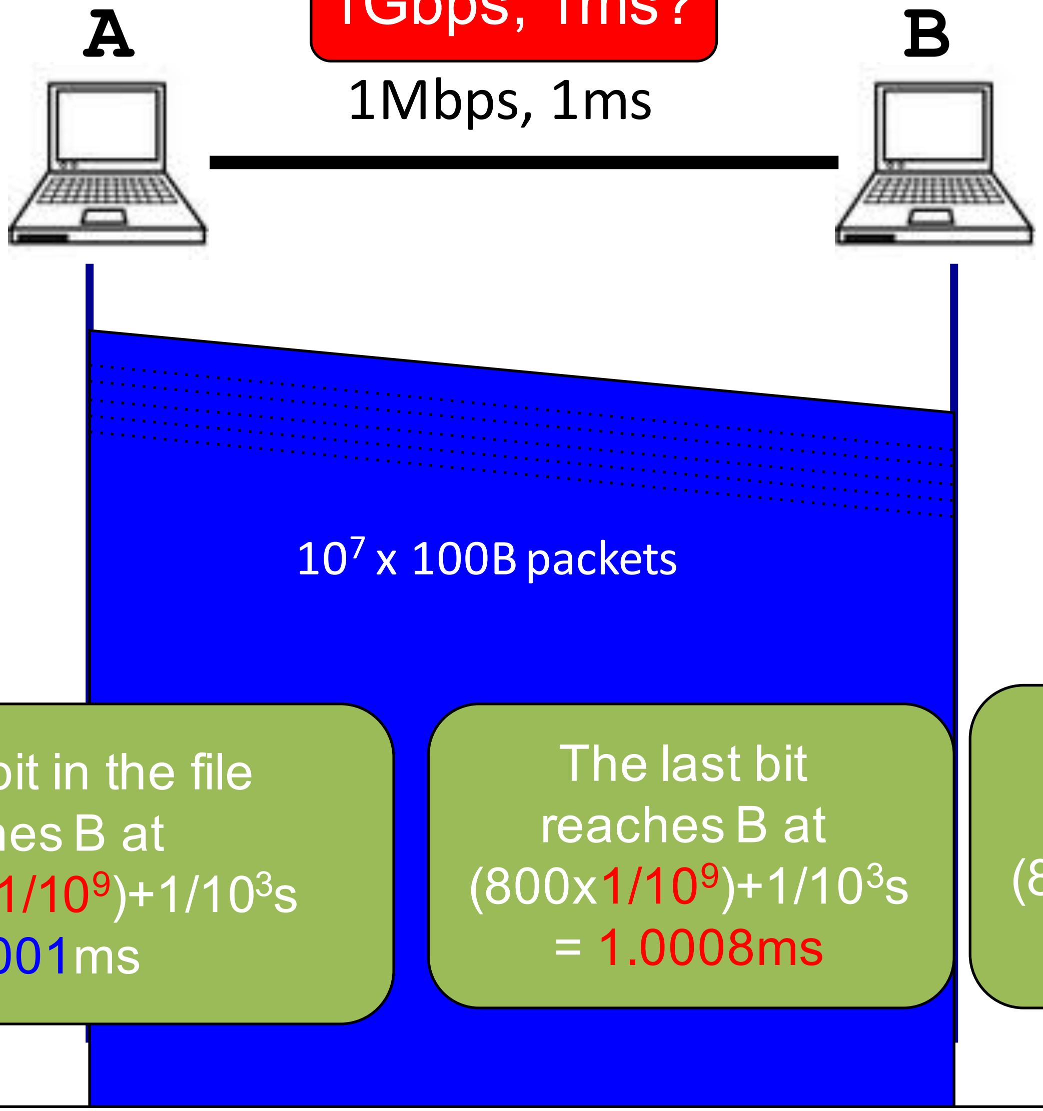
Packet Delay =
(Packet Size ÷ Link Bandwidth) + Link Latency

1GB file in 100B packets

ay

Sending a 100B packet from A to B?

1Gbps, 1ms?



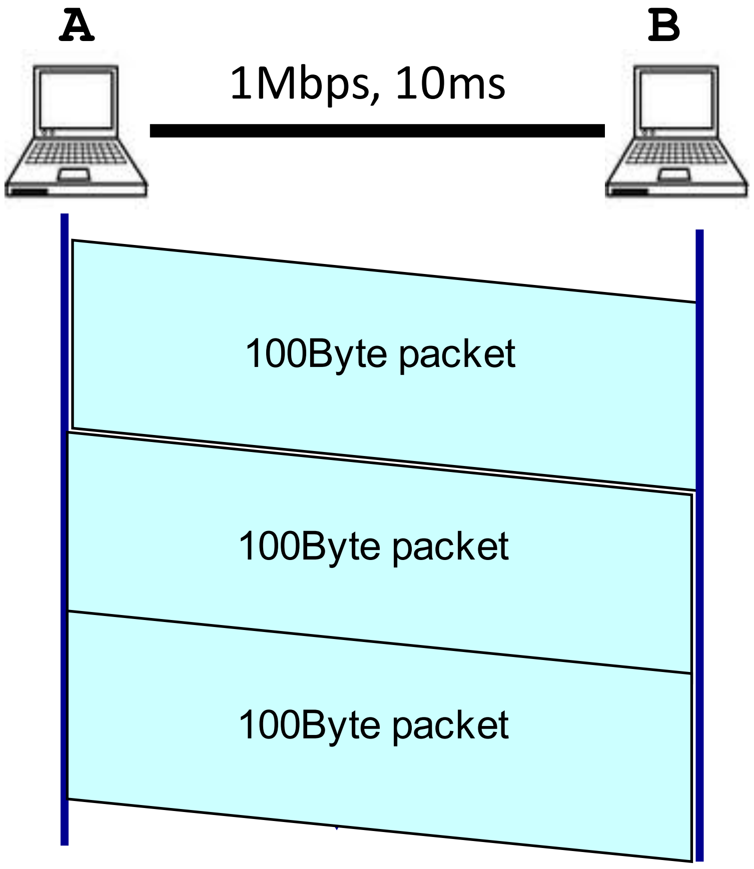
The last bit in the file reaches B at (10^{7}x800x1/10^{9})+1/10^{3}s = 8001ms

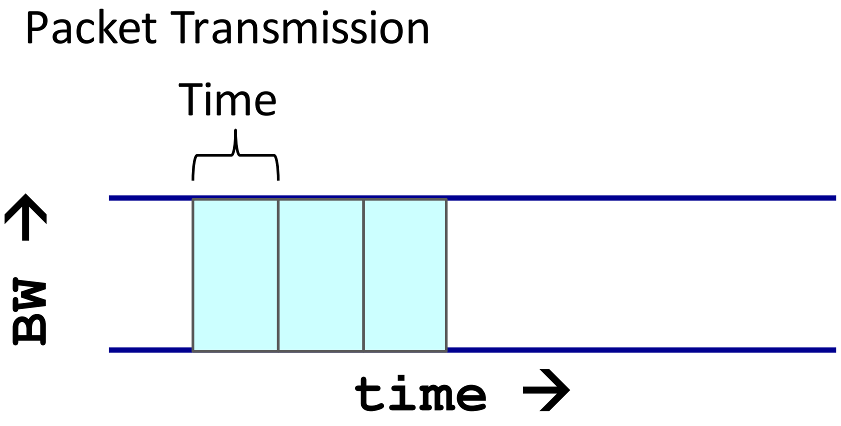
The last bit reaches B at (800x1/10^{9})+1/10^{3}s = 1.0008ms

The last bit reaches B at (800x1/10^{6})+1/10^{3}s = 1.8ms

Packet Delay: The “pipe” view

Sending 100B packets from A to B?

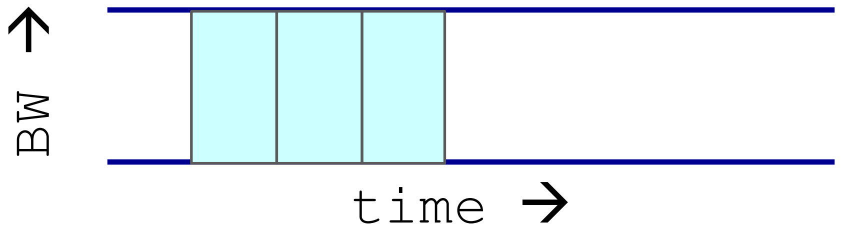




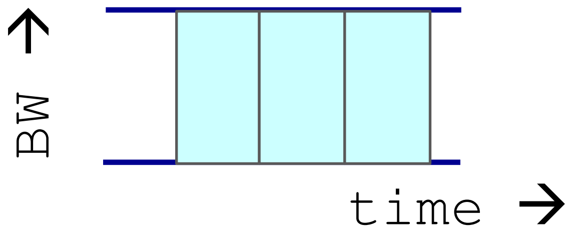
Packet Delay: The “pipe” view

Sending 100B packets from A to B?

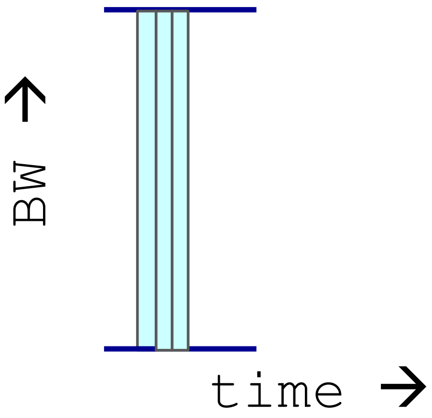
1Mbps, 10ms (BDP=10,000)



1Mbps, 5ms (BDP=5,000)



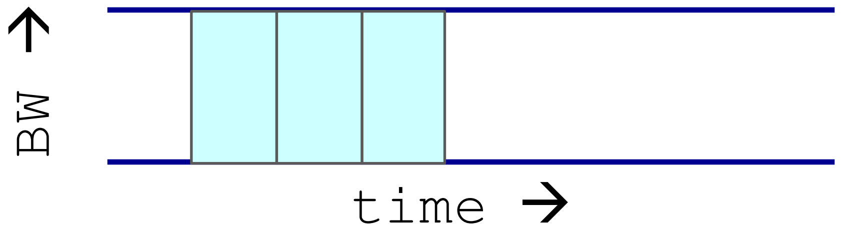
10Mbps, 1ms (BDP=10,000)



Packet Delay: The “pipe” view

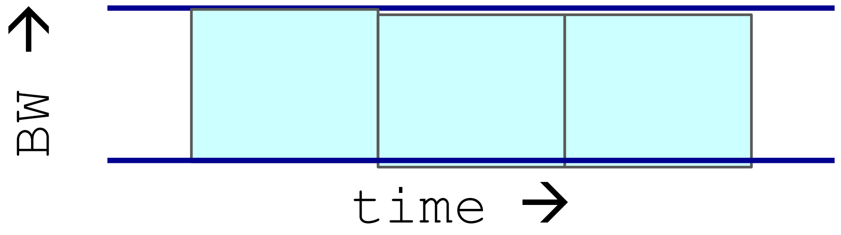
Sending 100B packets from A to B?

1Mbps, 10ms (BDP=10,000)

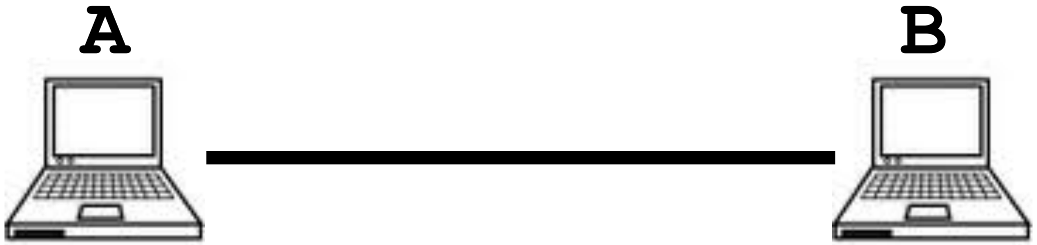


What if we used *200Byte packets??*

1Mbps, 10ms (BDP=10,000)

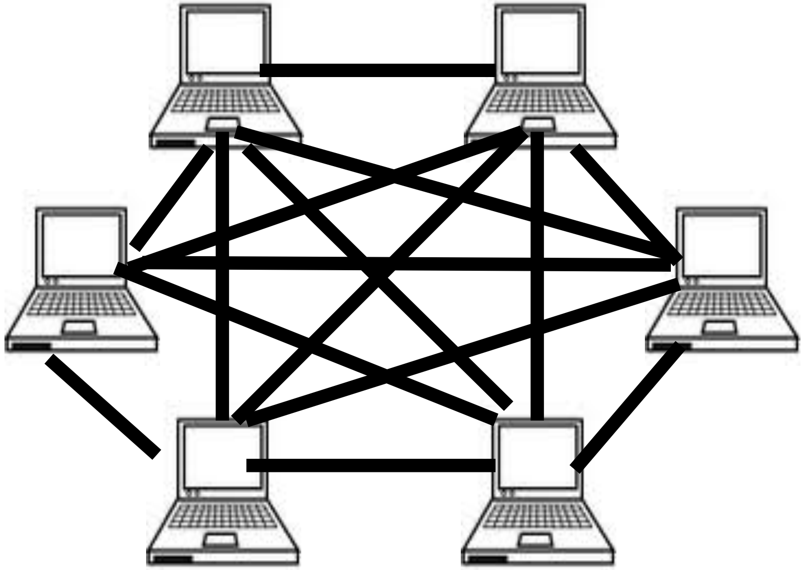


Recall Nodes and Links



What if we have more nodes?

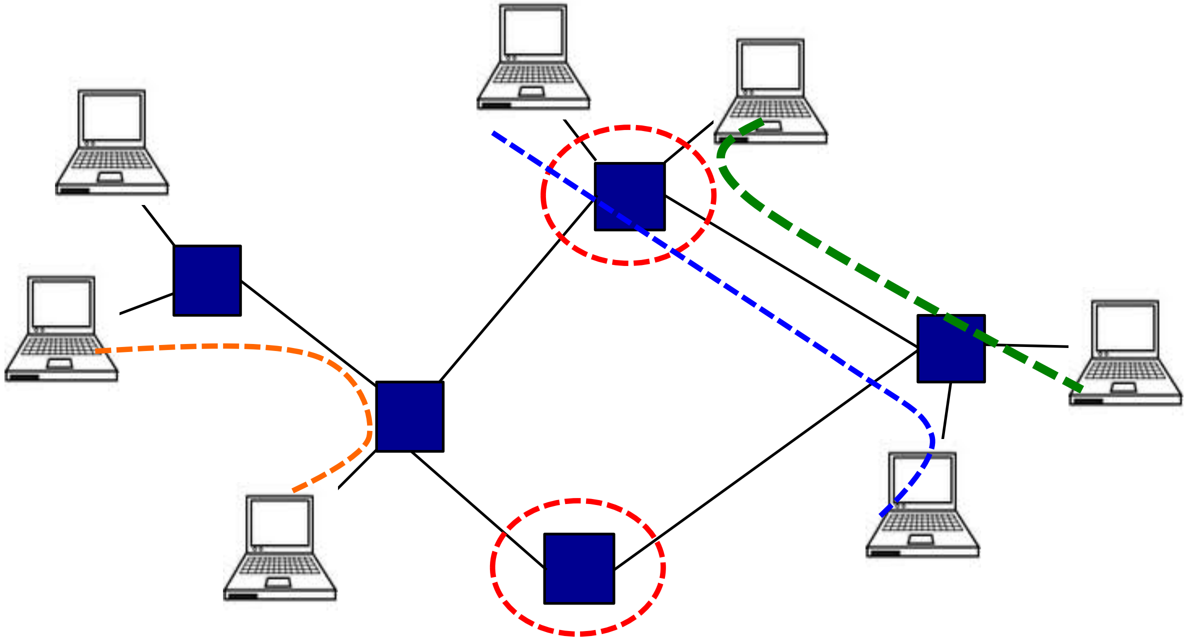
One link for every node?



Need a <u>scalable</u> way to interconnect nodes

Solution: A switched network

Nodes share network link resources



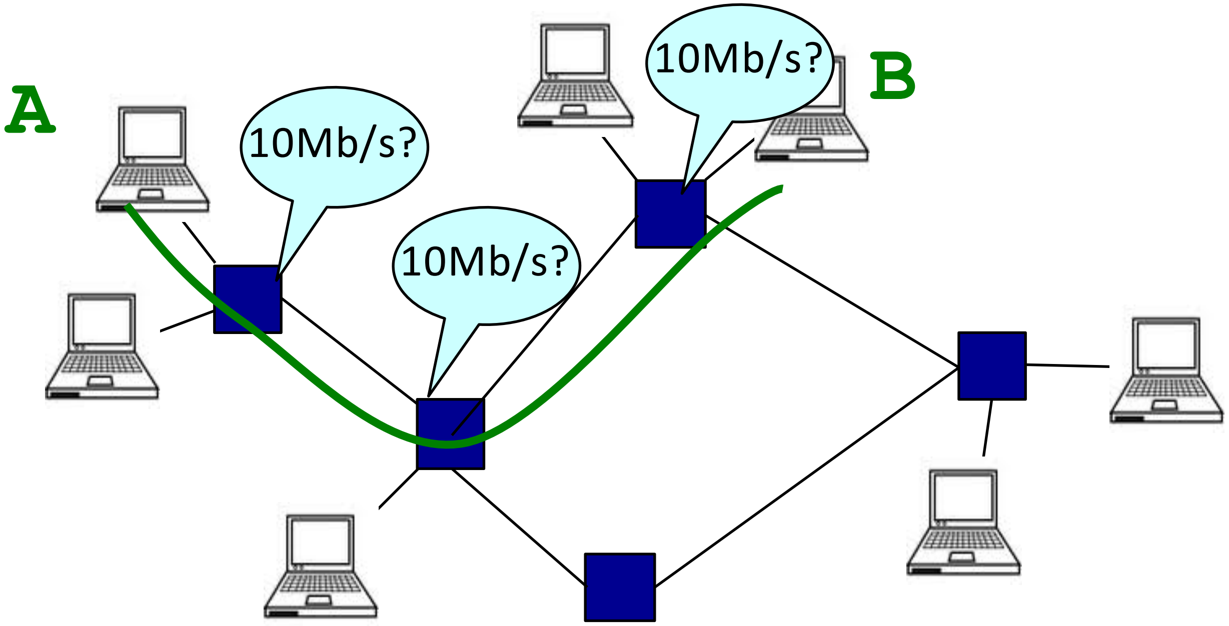
How is this sharing implemented?

Two forms of switched networks

- Circuit switching (used in the *POTS*: Plain Old Telephone system)
- Packet switching (used in the Internet)

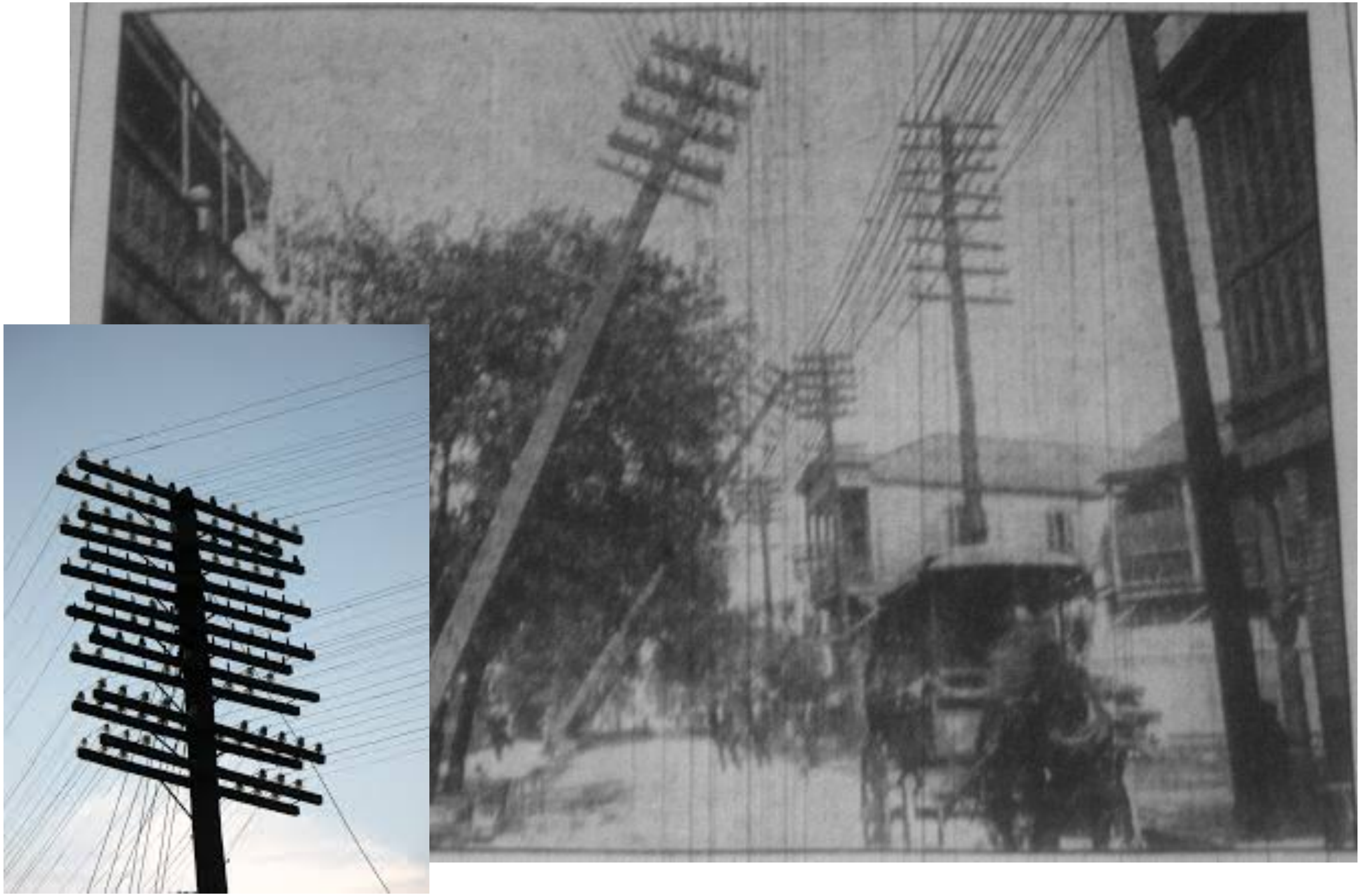
Circuit switching

Idea: source reserves network capacity along a path



(1) Node A sends a reservation request
(2) Interior switches establish a connection -- i.e., “circuit”
(3) A starts sending data
(4) A sends a “teardown circuit” message

Old Time Multiplexing

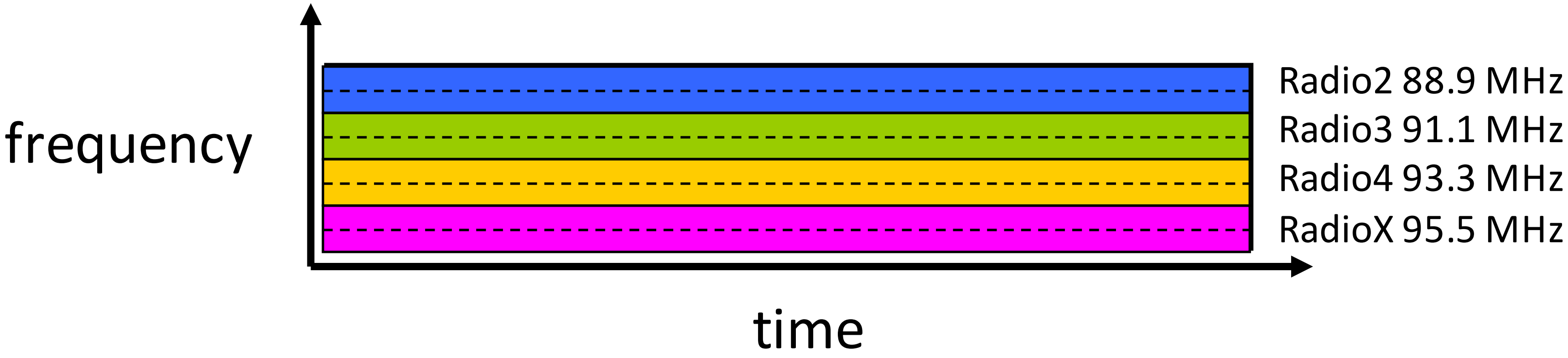


Circuit Switching: FDM and TDM

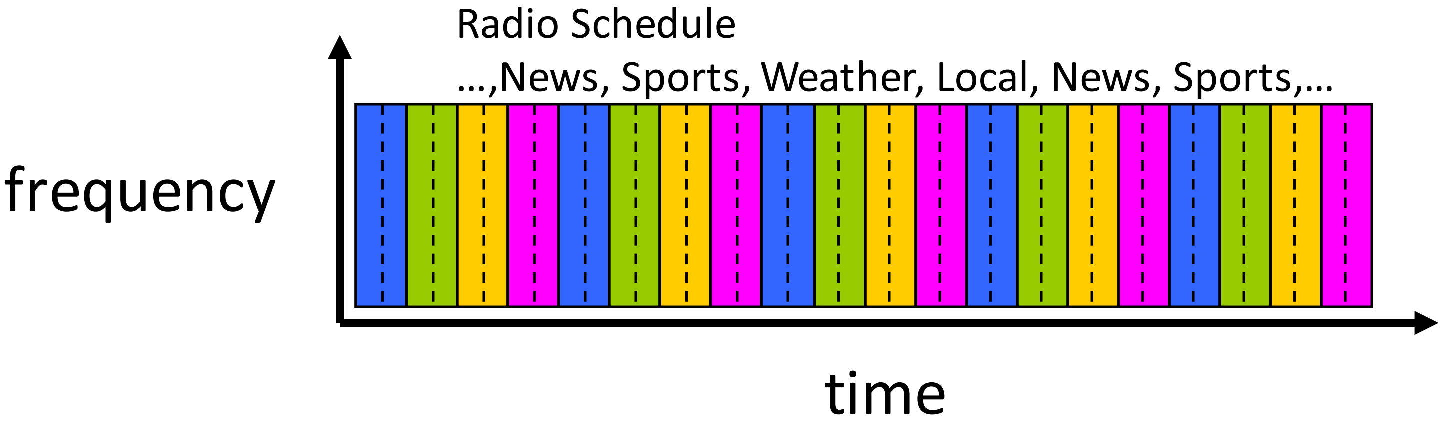
Example:

4 users 🟦🟩🟨🟪

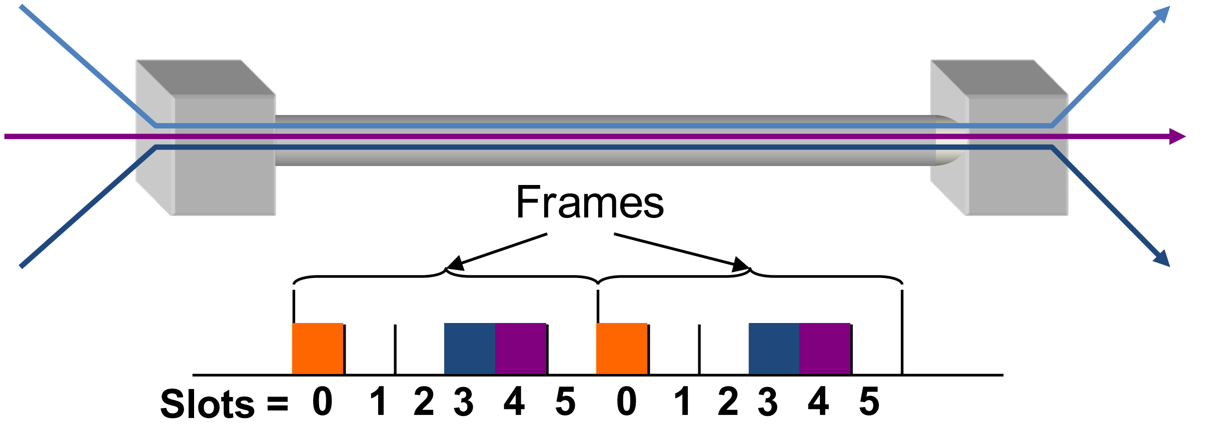
Frequency Division Multiplexing



Time Division Multiplexing

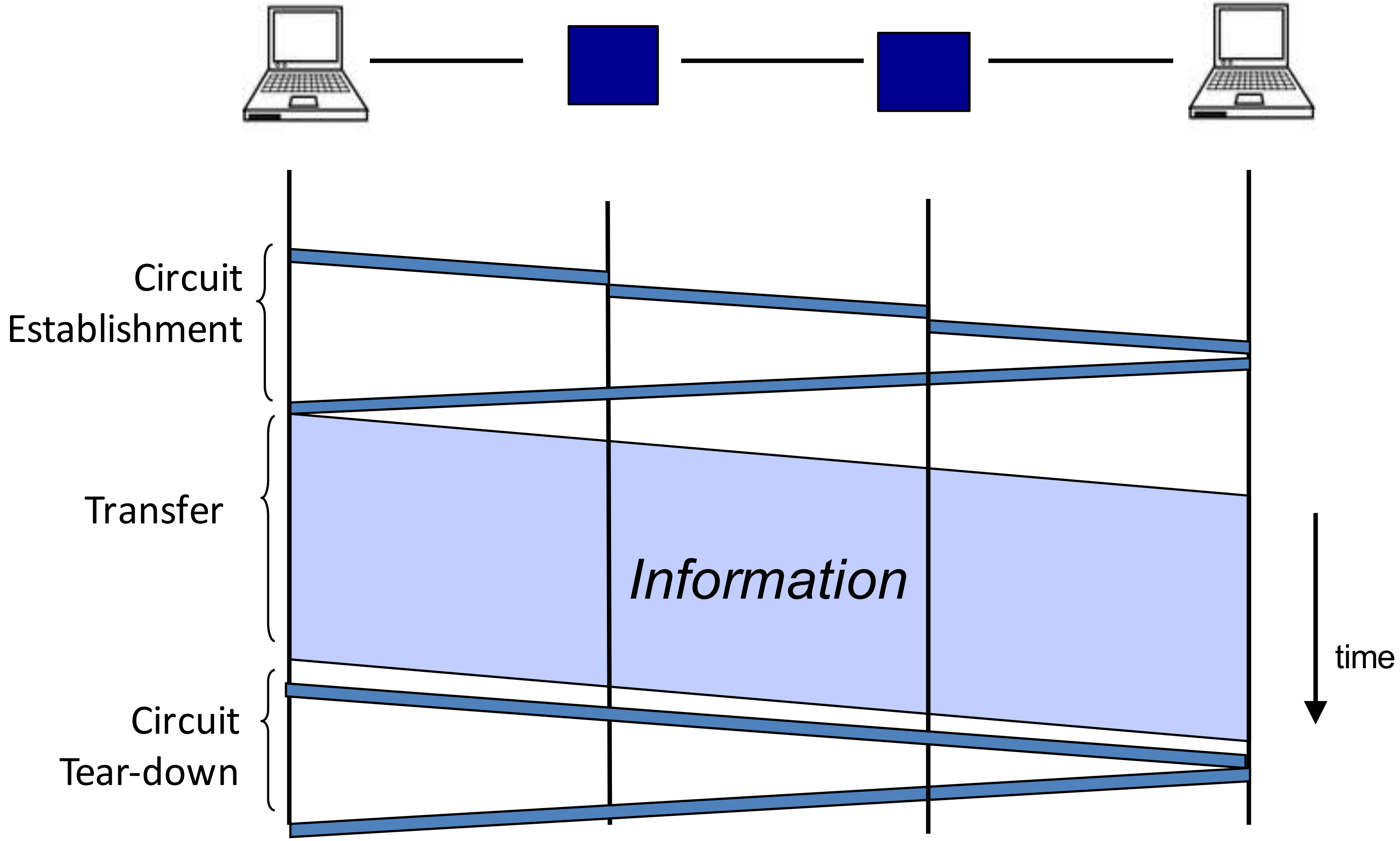


Time-Division Multiplexing/Demultiplexing



- Time divided into frames; frames into slots
- Relative slot position inside a frame determines to which conversation data belongs
  - e.g., slot 0 belongs to orange conversation
- Slots are reserved (released) during circuit setup (teardown)
- If a conversation does not use its circuit **capacity is lost!**

Timing in Circuit Switching

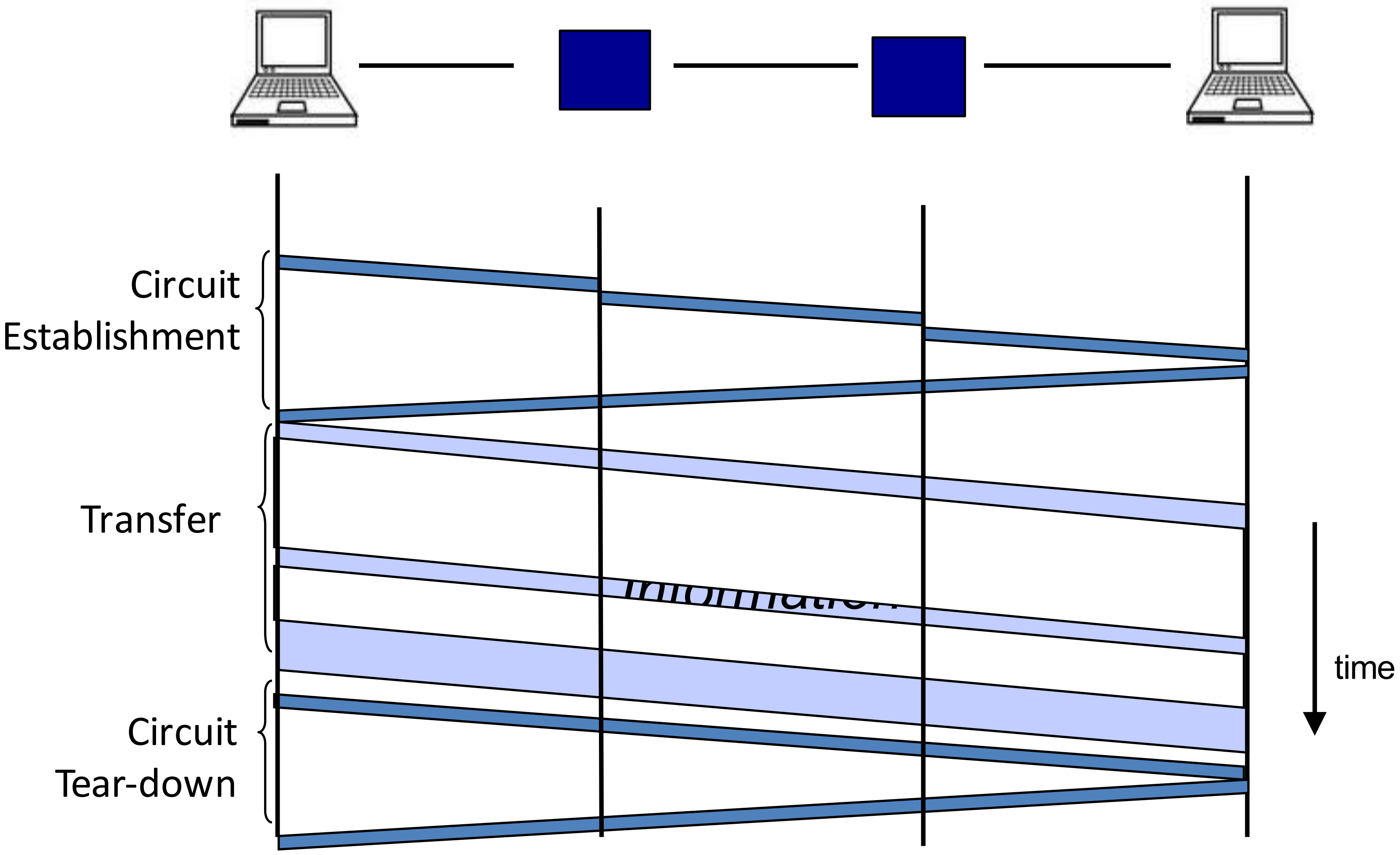


Circuit switching: pros and cons

- Pros
  - guaranteed performance
  - fast transfer (once circuit is established)

- Cons

Timing in Circuit Switching

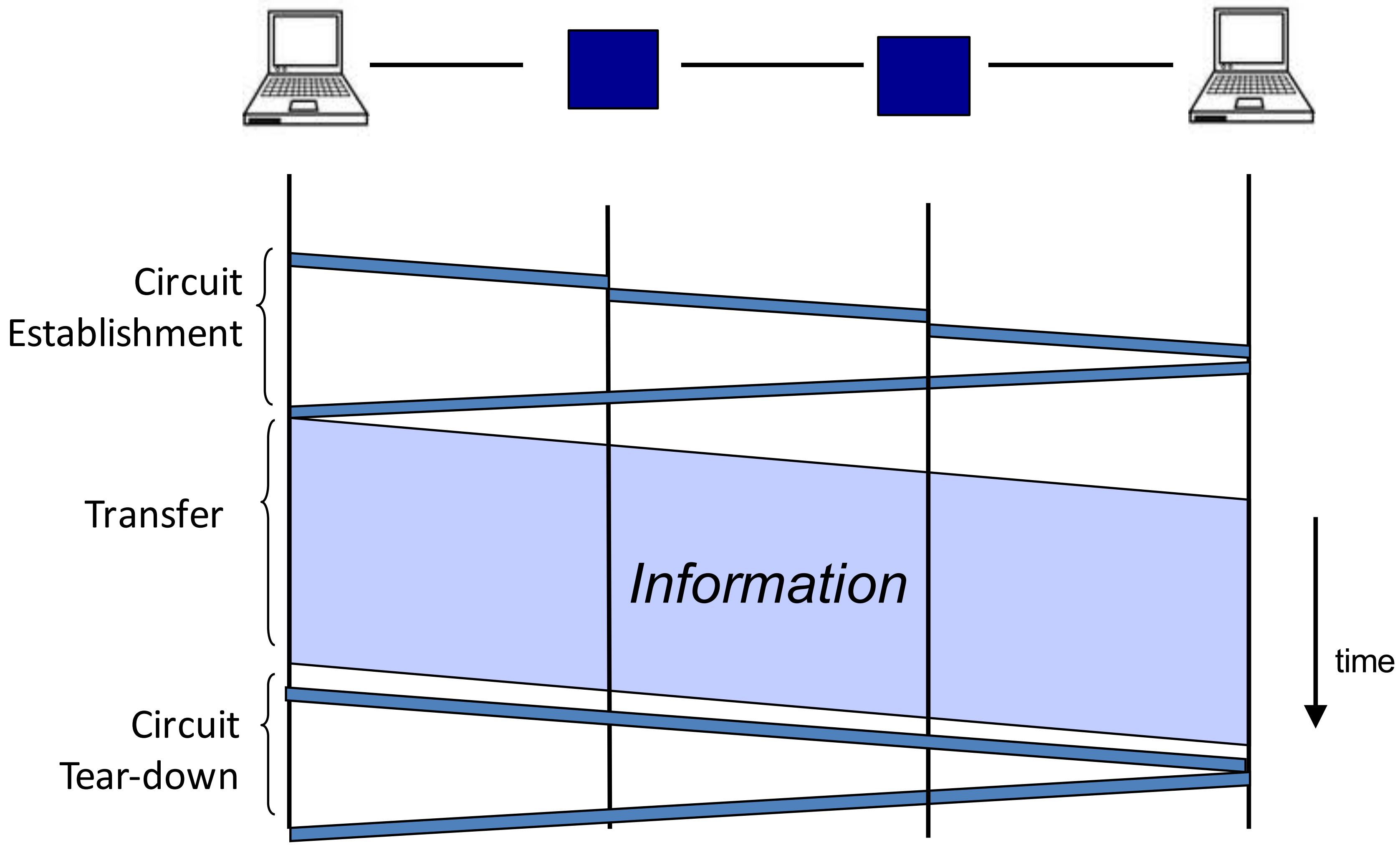


Circuit switching: pros and cons

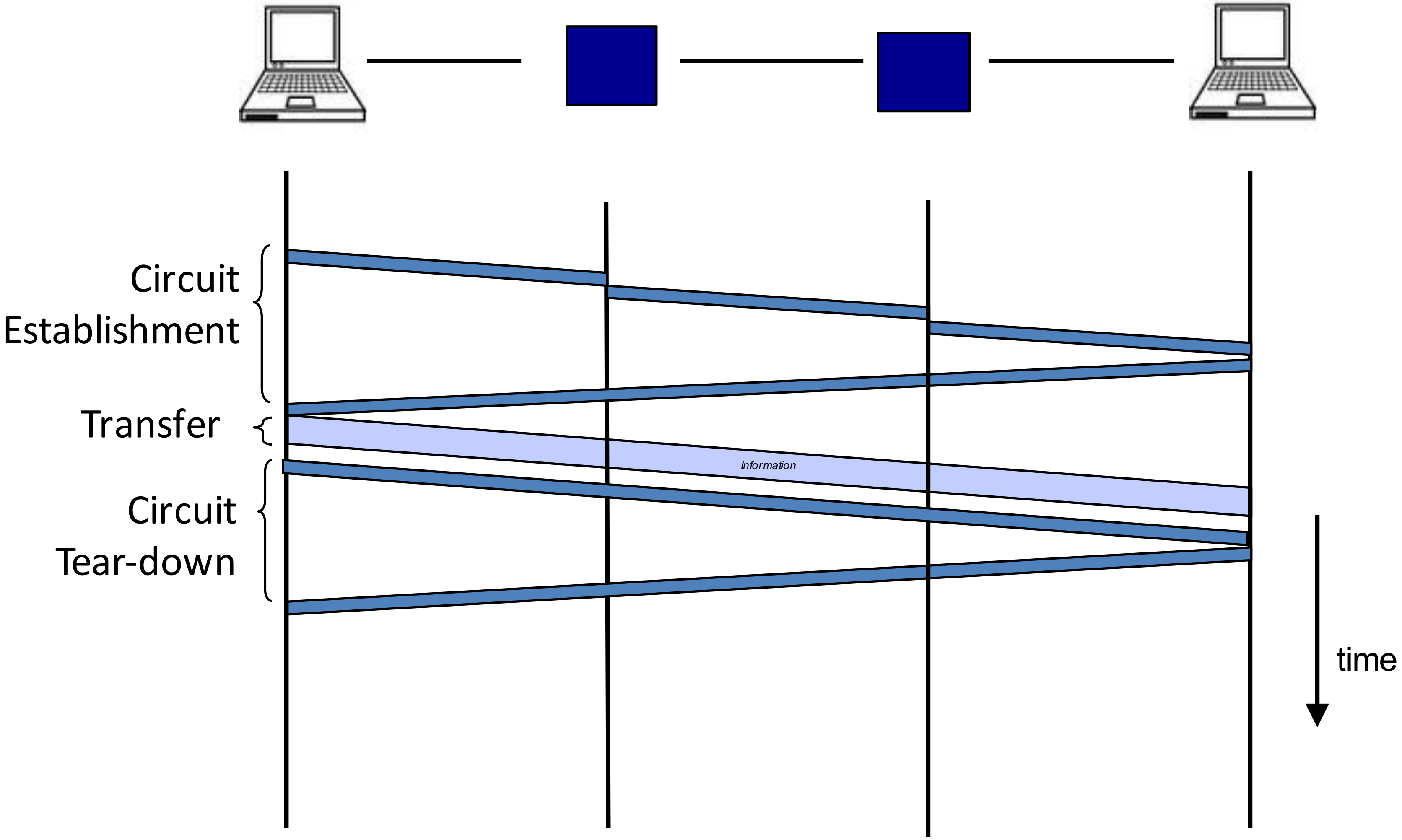
- Pros
  - guaranteed performance
  - fast transfer (once circuit is established)

- Cons
  - **wastes bandwidth if traffic is “bursty”**

Timing in Circuit Switching



Timing in Circuit Switching

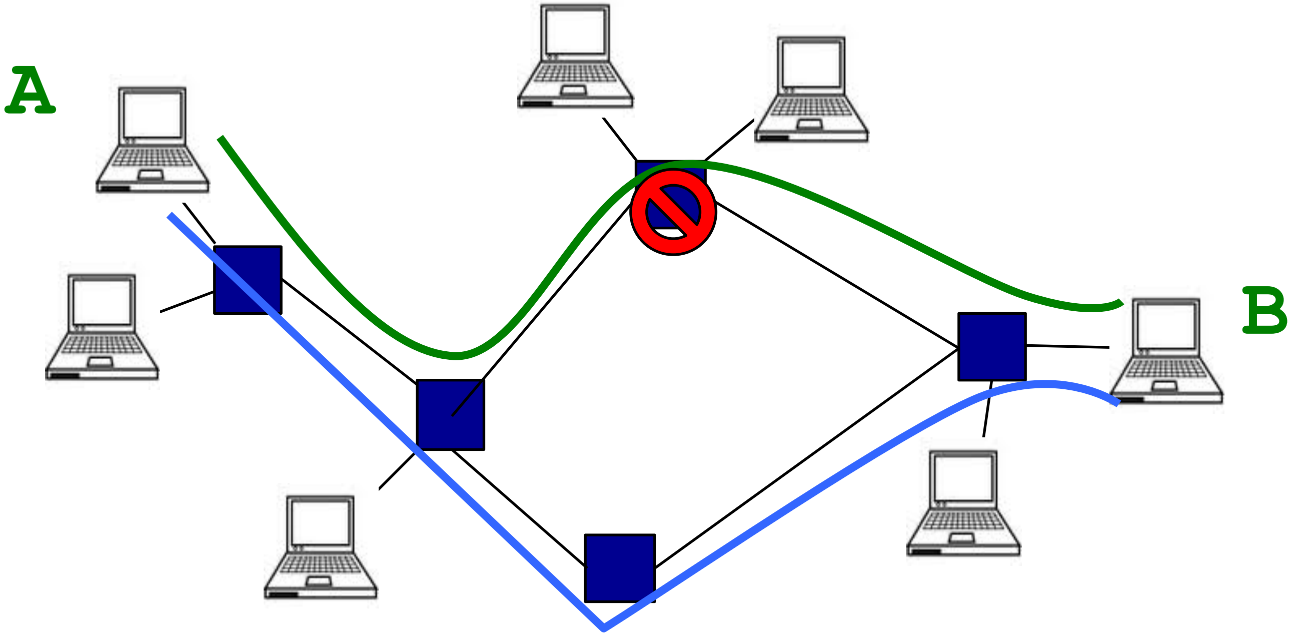


Circuit switching: pros and cons

- Pros
  - guaranteed performance
  - fast transfers (once circuit is established)

- Cons
  - wastes bandwidth if traffic is "bursty"
  - **connection setup time is overhead**

Circuit switching



Circuit switching doesn't "route around failure"

Circuit switching: pros and cons

- Pros
  - guaranteed performance
  - fast transfers (once circuit is established)

- Cons
  - wastes bandwidth if traffic is “bursty”
  - connection setup time is overhead
  - **recovery from failure is slow**

Numerical example

- How long does it take to send a file of 640,000 bits from host A to host B over a circuit-switched network?
  - All links are 1.536 Mbps
  - Each link uses TDM with 24 slots/sec
  - 500 msec to establish end-to-end circuit

Let’s work it out!

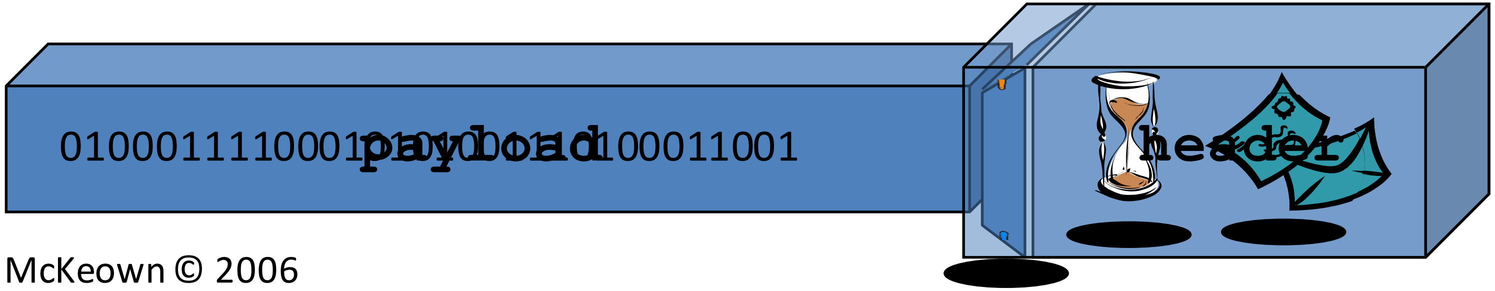
Two forms of switched networks

- Circuit switching (e.g., telephone network)
- Packet switching (e.g., Internet)

Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a “header” and “payload”*

1. Internet Address
2. Age (TTL)
3. Checksum to protect header



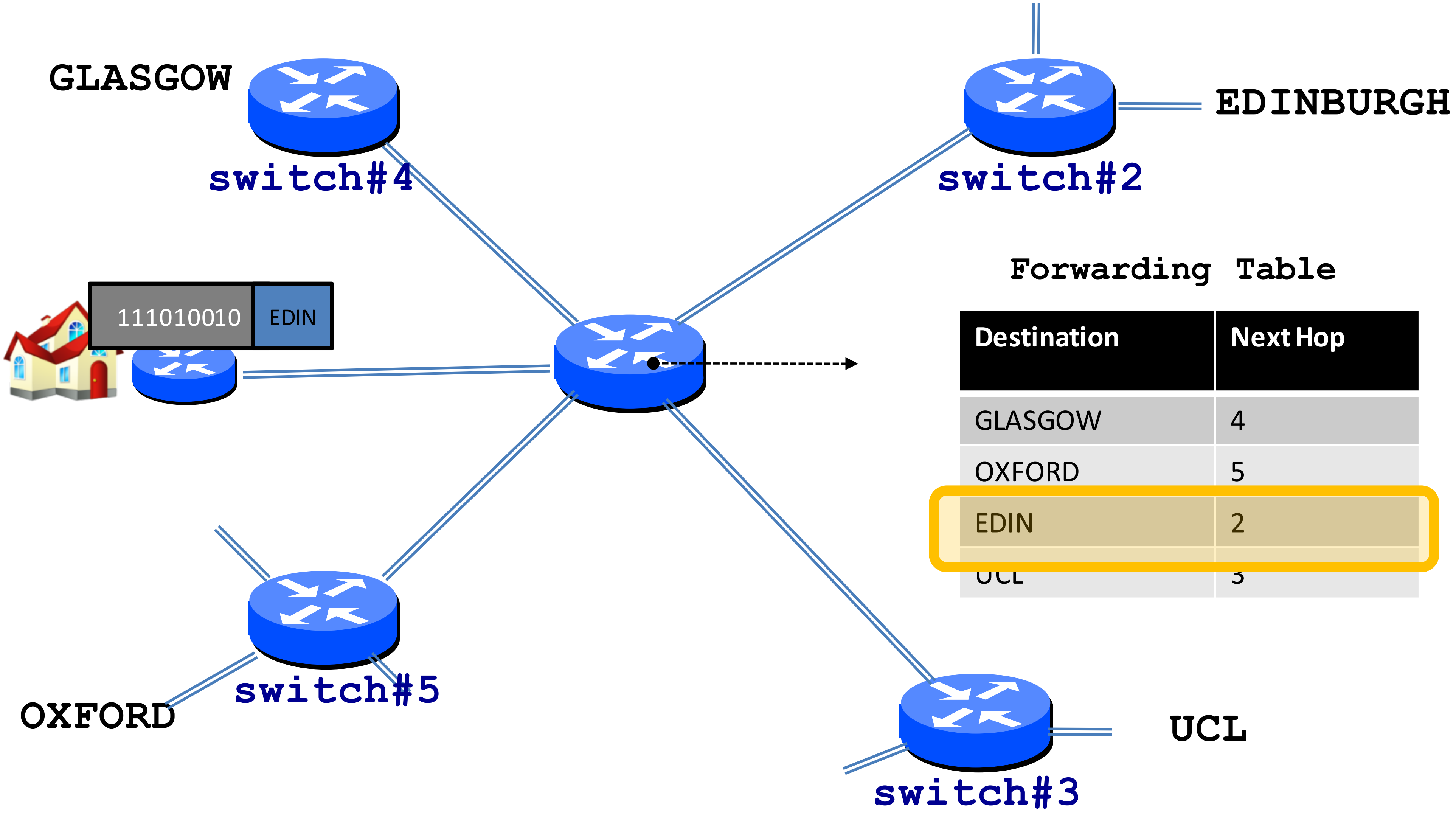
Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a “header” and “payload”*
  - payload is the data being carried
  - header holds instructions to the network for how to handle packet (think of the header as an API)

Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a “header” and “payload”
- Switches “forward” packets based on their headers

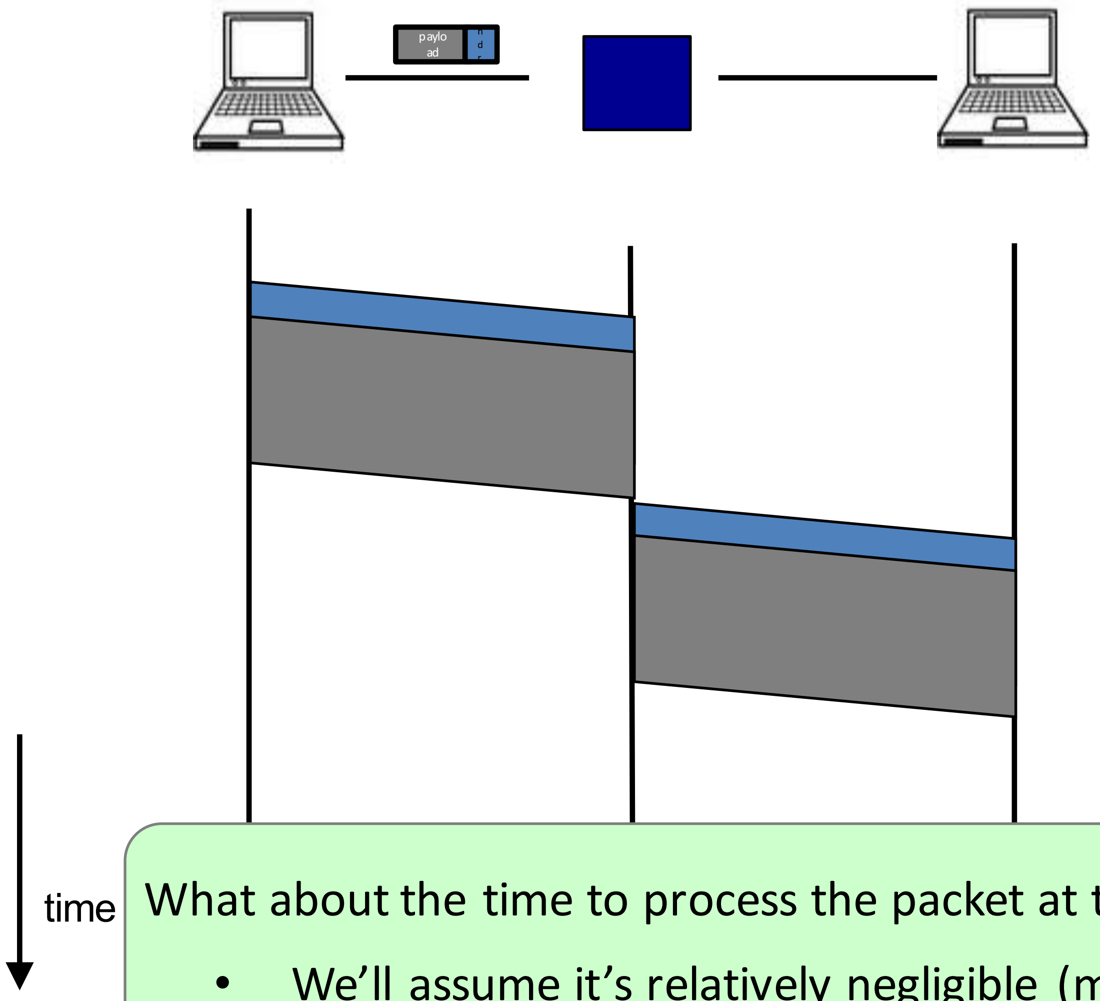
Switches forward packets



Forwarding Table

Destination	Next Hop
GLASGOW	4
OXFORD	5
EDIN	2
UCL	3

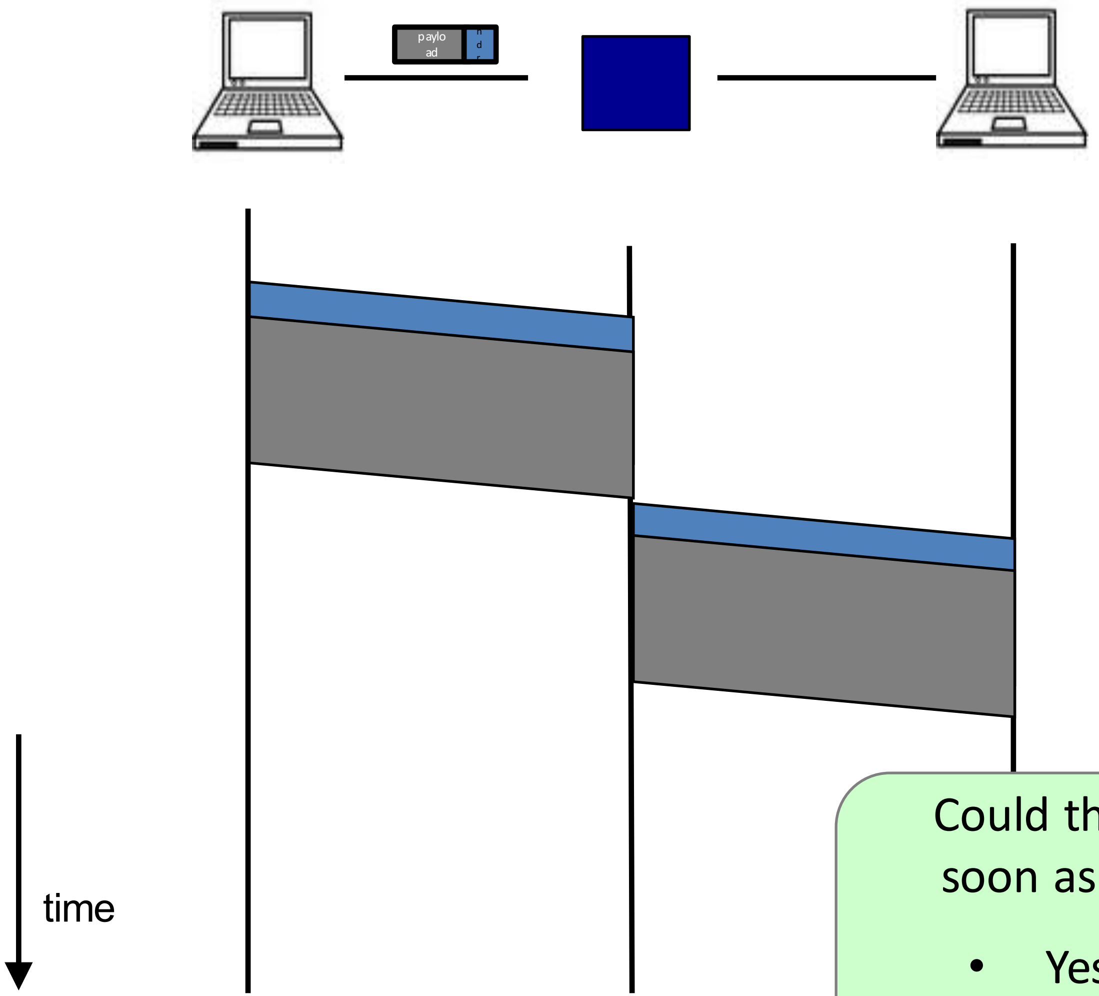
Timing in Packet Switching



What about the time to process the packet at the switch?

- We'll assume it's relatively negligible (mostly true)

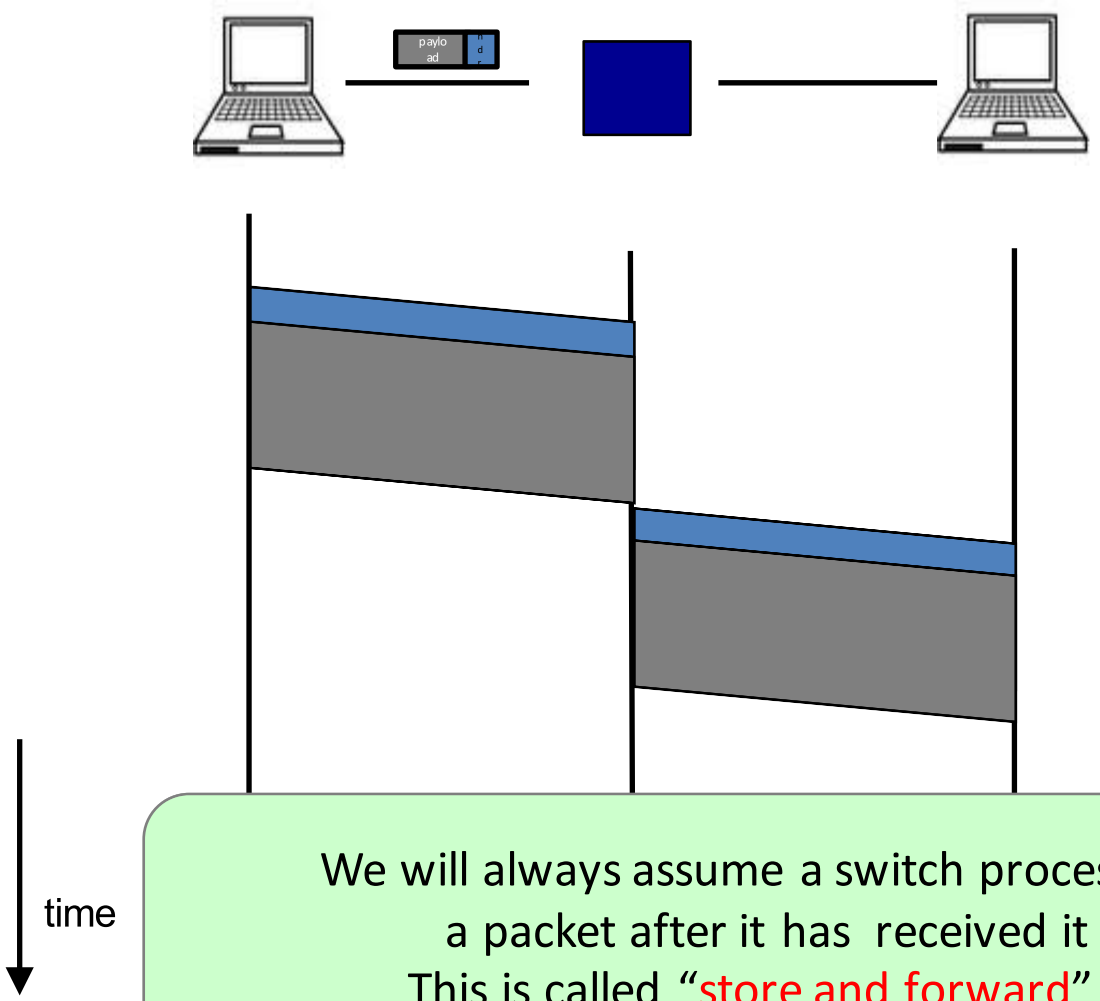
Timing in Packet Switching



Could the switch start transmitting as soon as it has processed the header?

- Yes! This would be called a "cut through" switch

Timing in Packet Switching



We will always assume a switch processes/forwards
a packet after it has received it entirely.
This is called "store and forward" switching

Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a “header” and “payload”
- Switches “forward” packets based on their headers

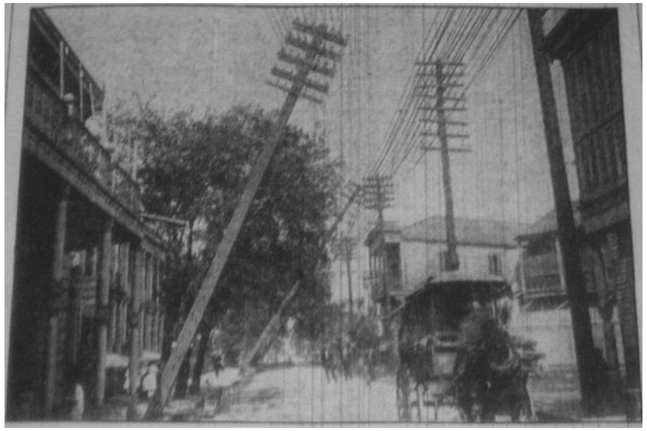
Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a “header” and “payload”
- Switches “forward” packets based on their headers
- Each packet travels independently
  - no notion of packets belonging to a “circuit”

Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a “header” and “payload”
- Switches “forward” packets based on their headers
- Each packet travels independently
- No link resources are reserved in advance. Instead packet switching leverages statistical multiplexing (stat muxing)

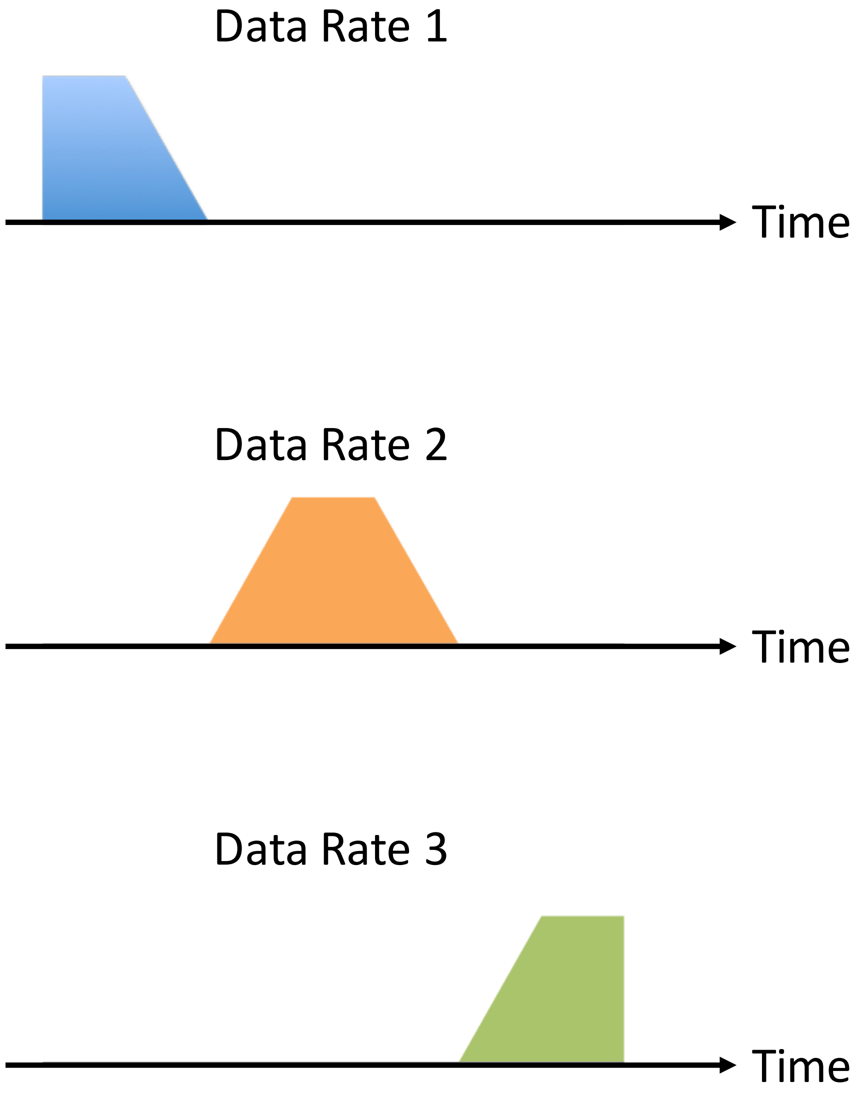
Multiplexing

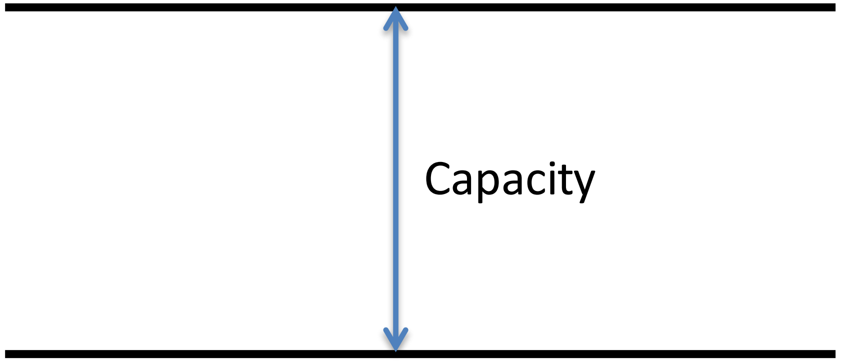


Sharing makes things efficient (cost less)

- One airplane/train for 100 people
- One telephone for many calls
- One lecture theatre for many classes
- One computer for many tasks
- One network for many computers
- One datacenter many applications

Three Flows with Bursty Traffic

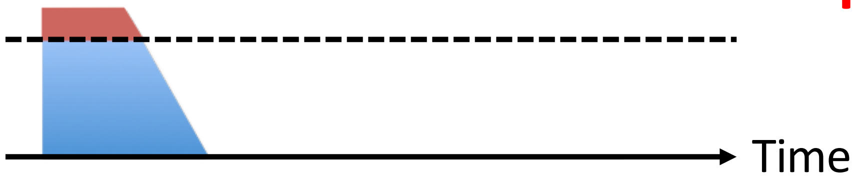




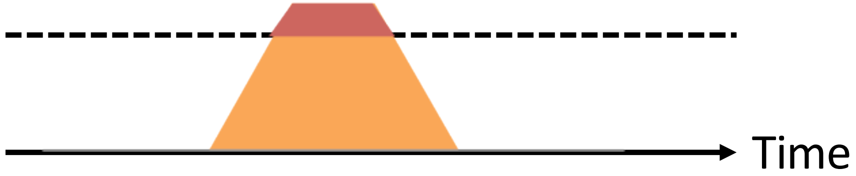
When Each Flow Gets 1/3rd of Capacity

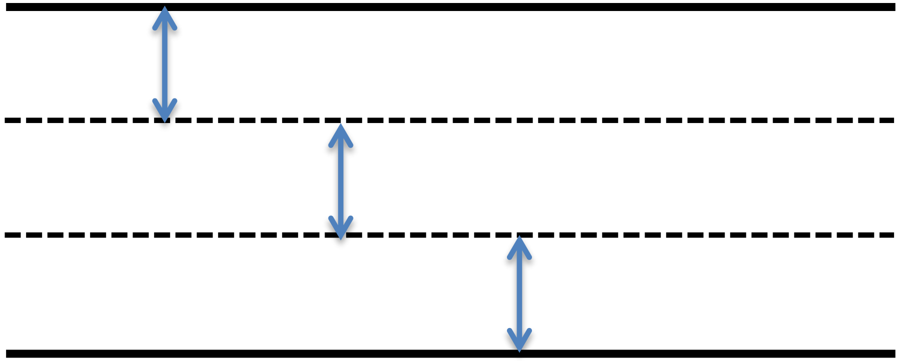
Frequent Overloading

Data Rate 1

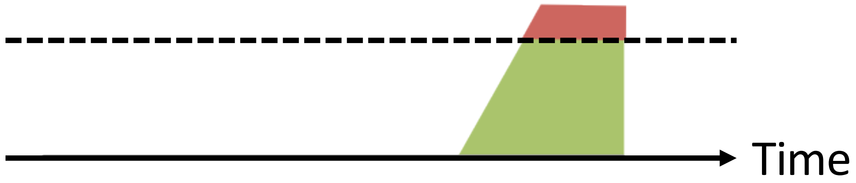


Data Rate 2

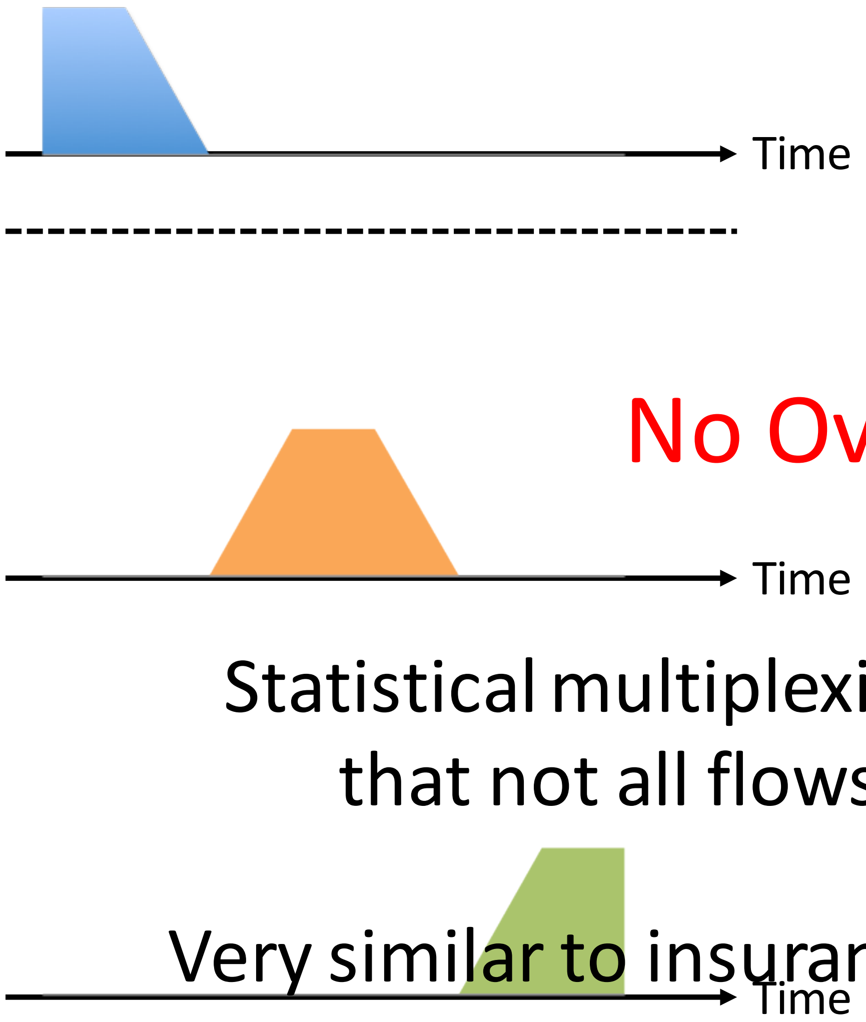




Data Rate 3



When Flows Share Total Capacity

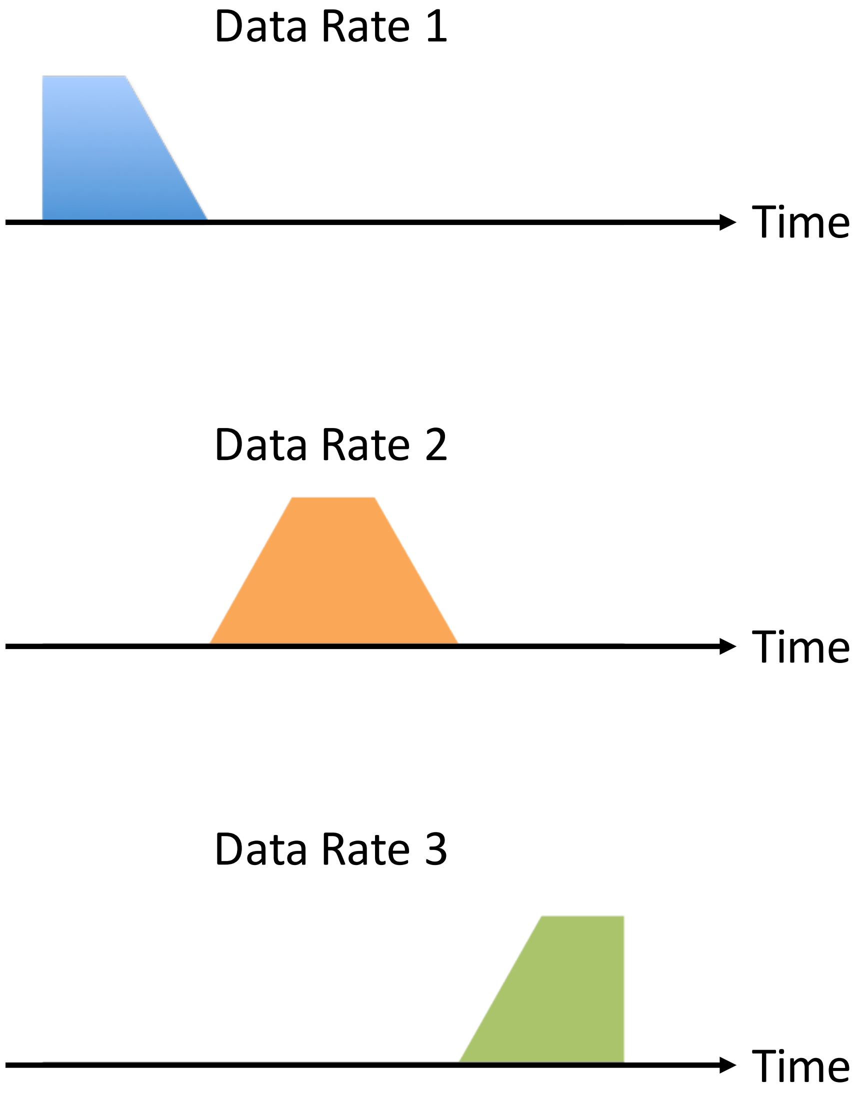


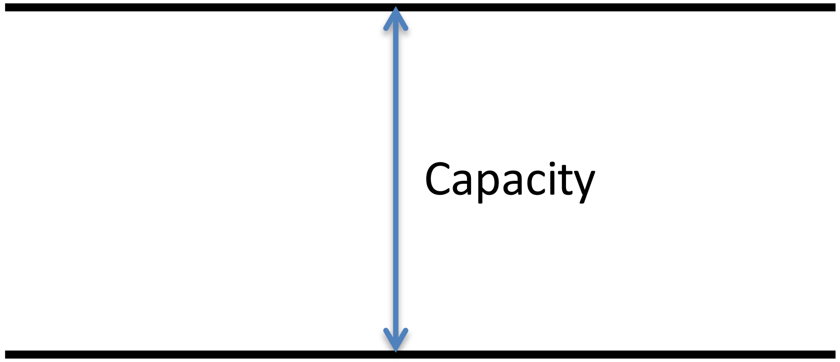
No Overloading

Statistical multiplexing relies on the assumption that not all flows burst at the same time.

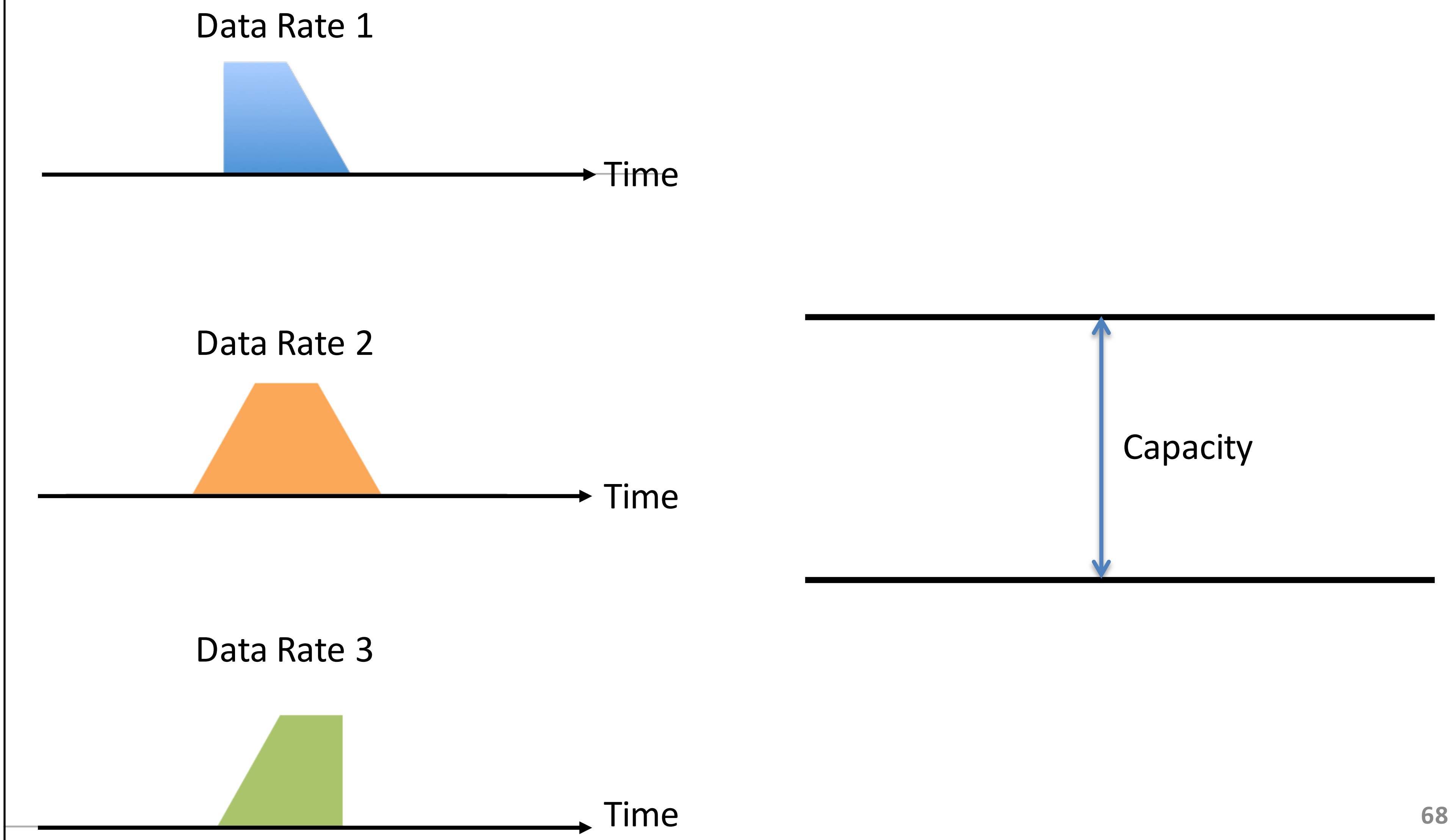
Very similar to insurance, and has same failure case

Three Flows with Bursty Traffic

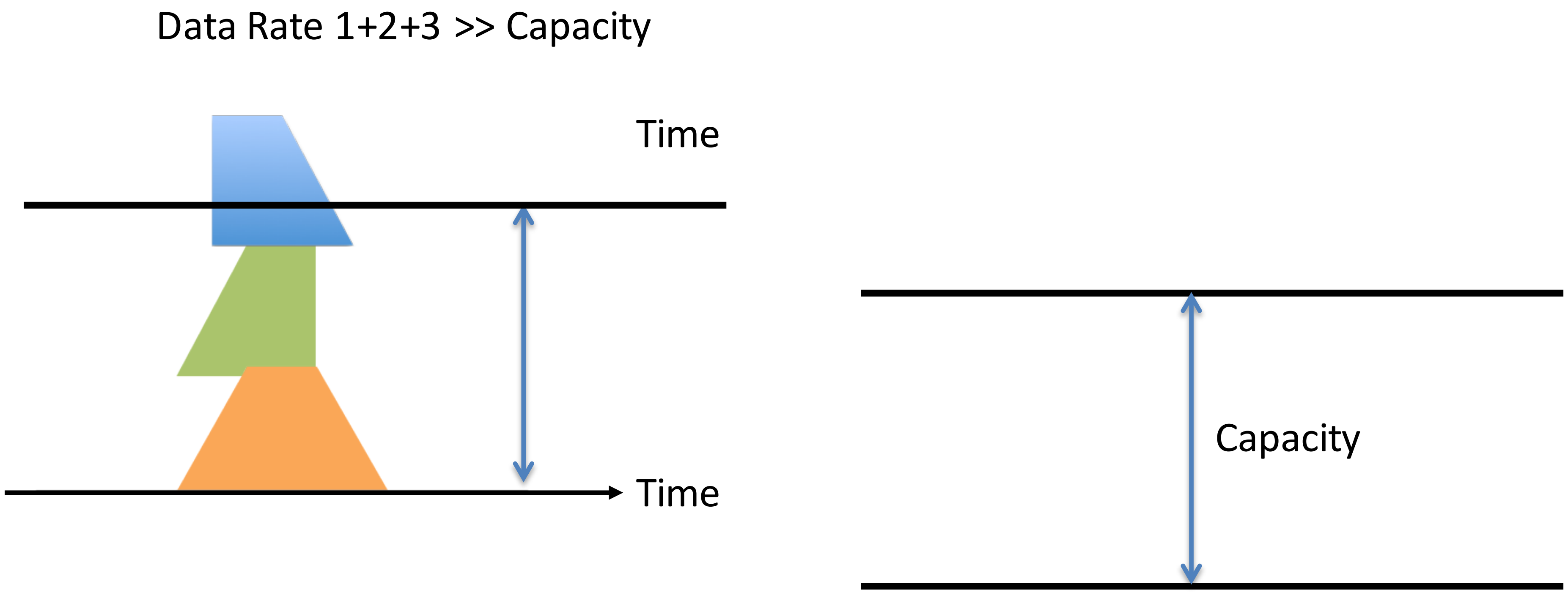




Three Flows with Bursty Traffic



Three Flows with Bursty Traffic



Data Rate 1+2+3 >> Capacity

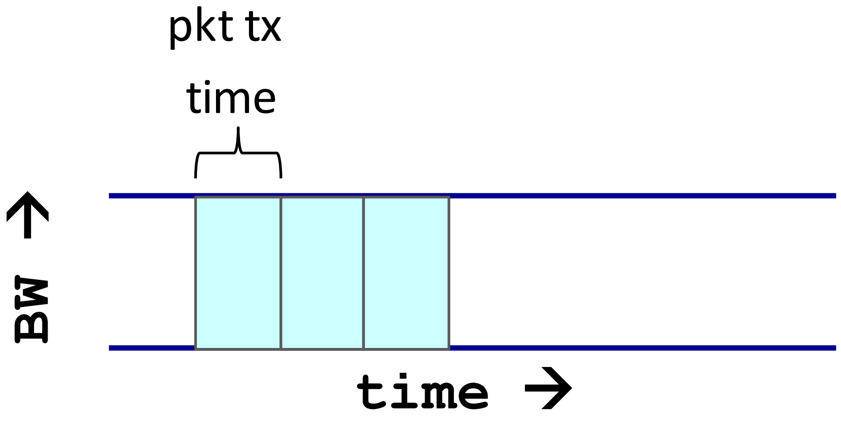
Time

Time

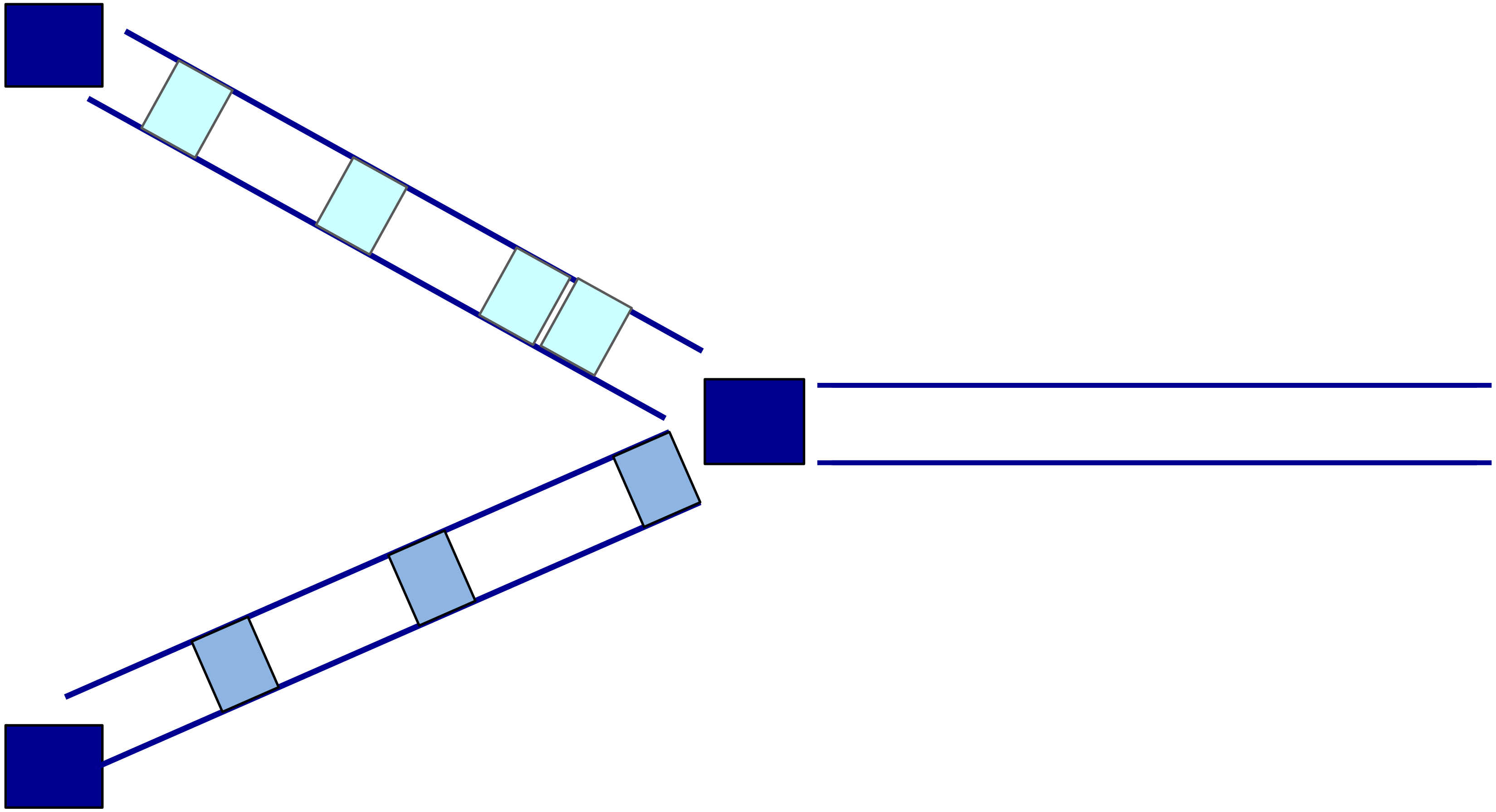
Capacity

What do we do under overload?

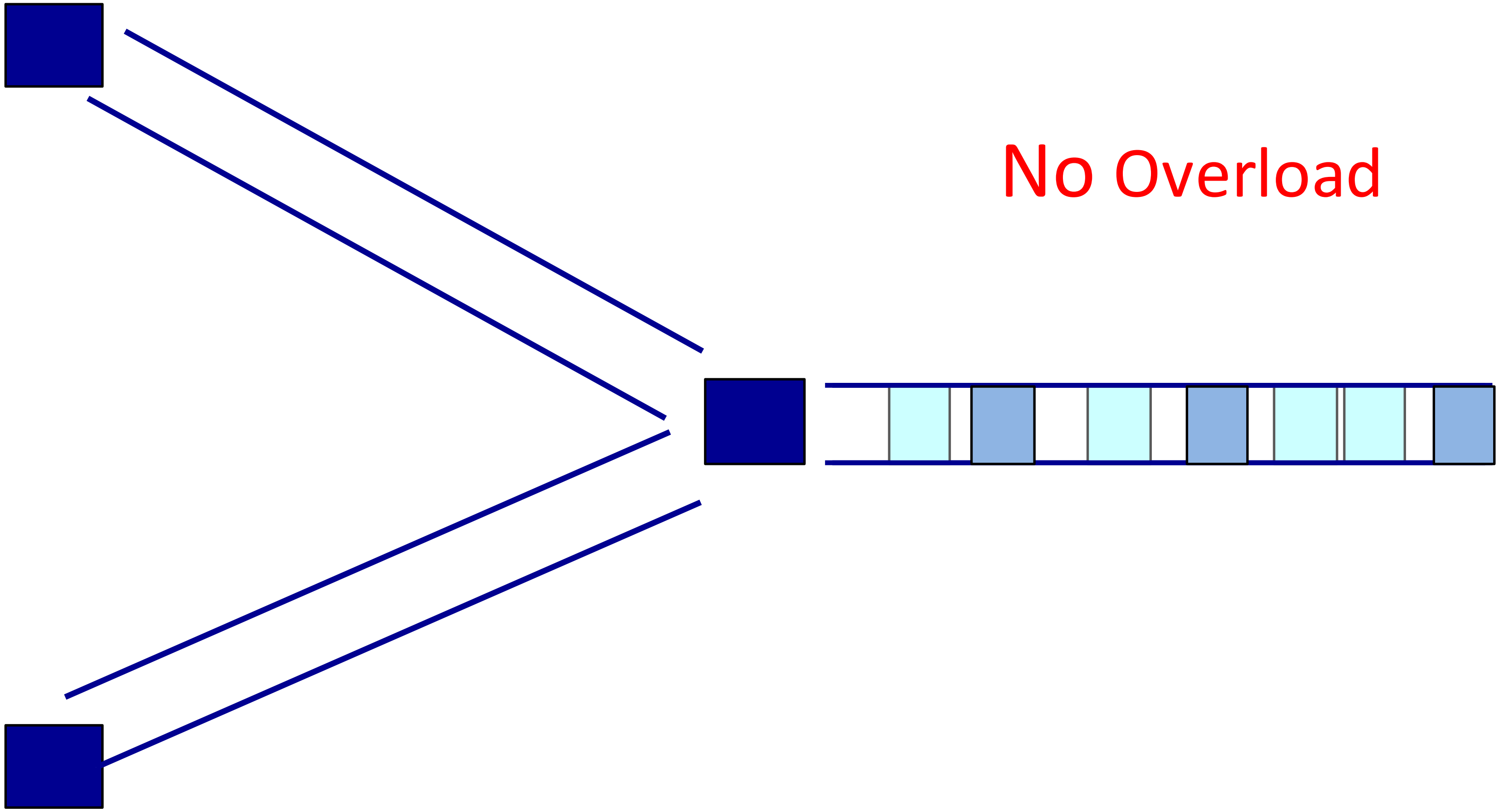
Statistical multiplexing: pipe view



Statistical multiplexing: pipe view

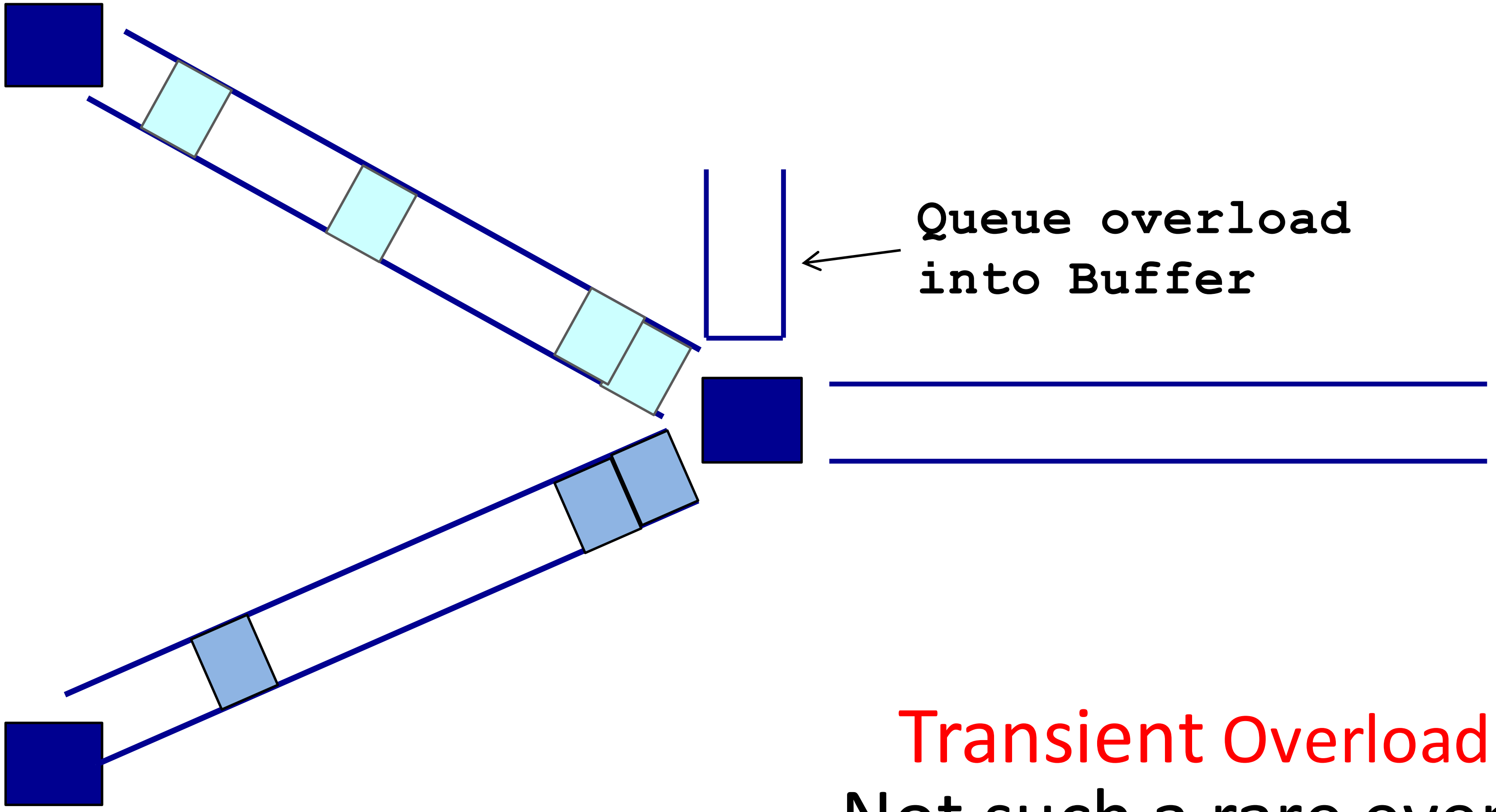


Statistical multiplexing: pipe view



No Overload

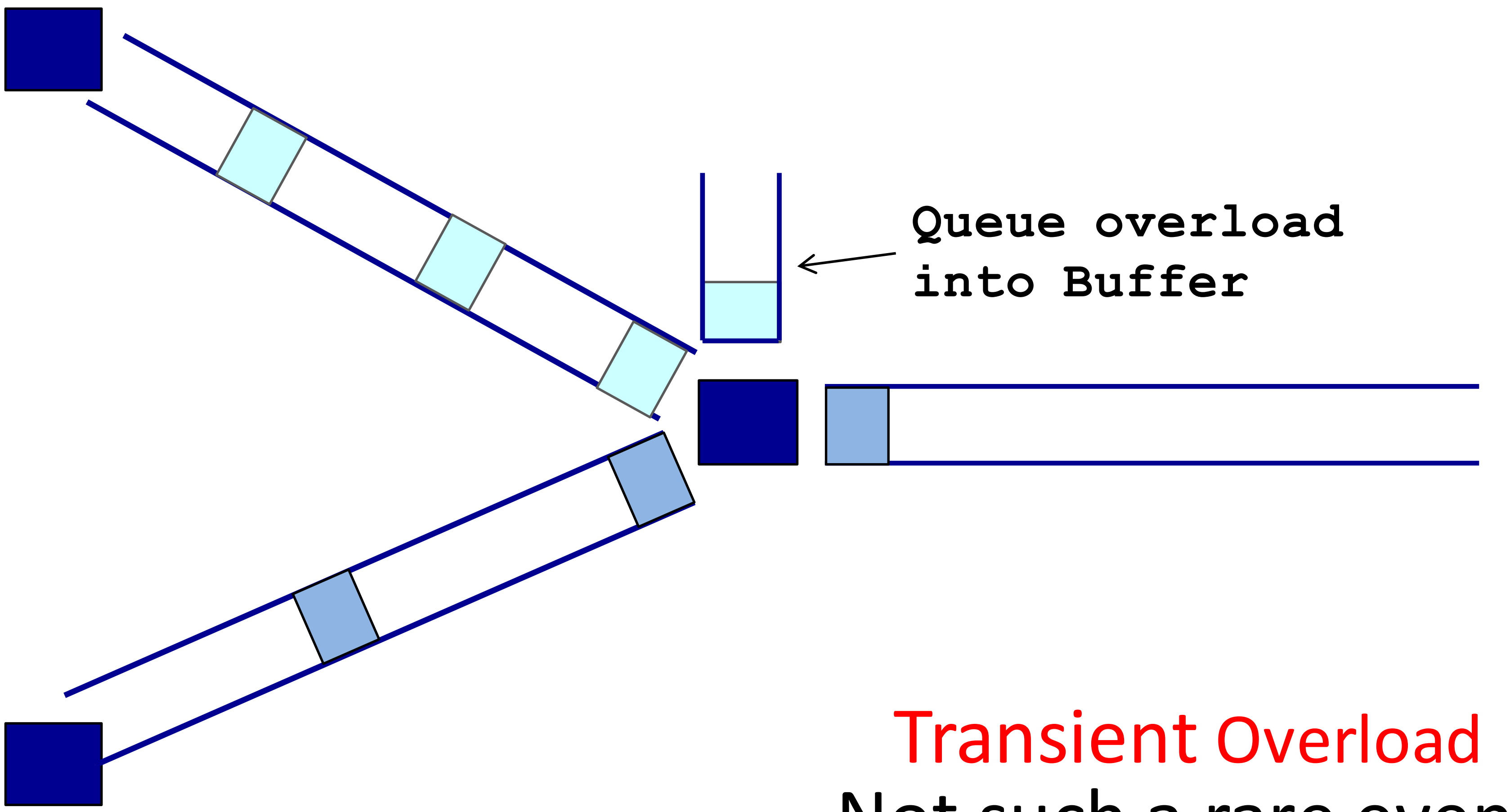
Statistical multiplexing: pipe view



Transient Overload

Not such a rare event

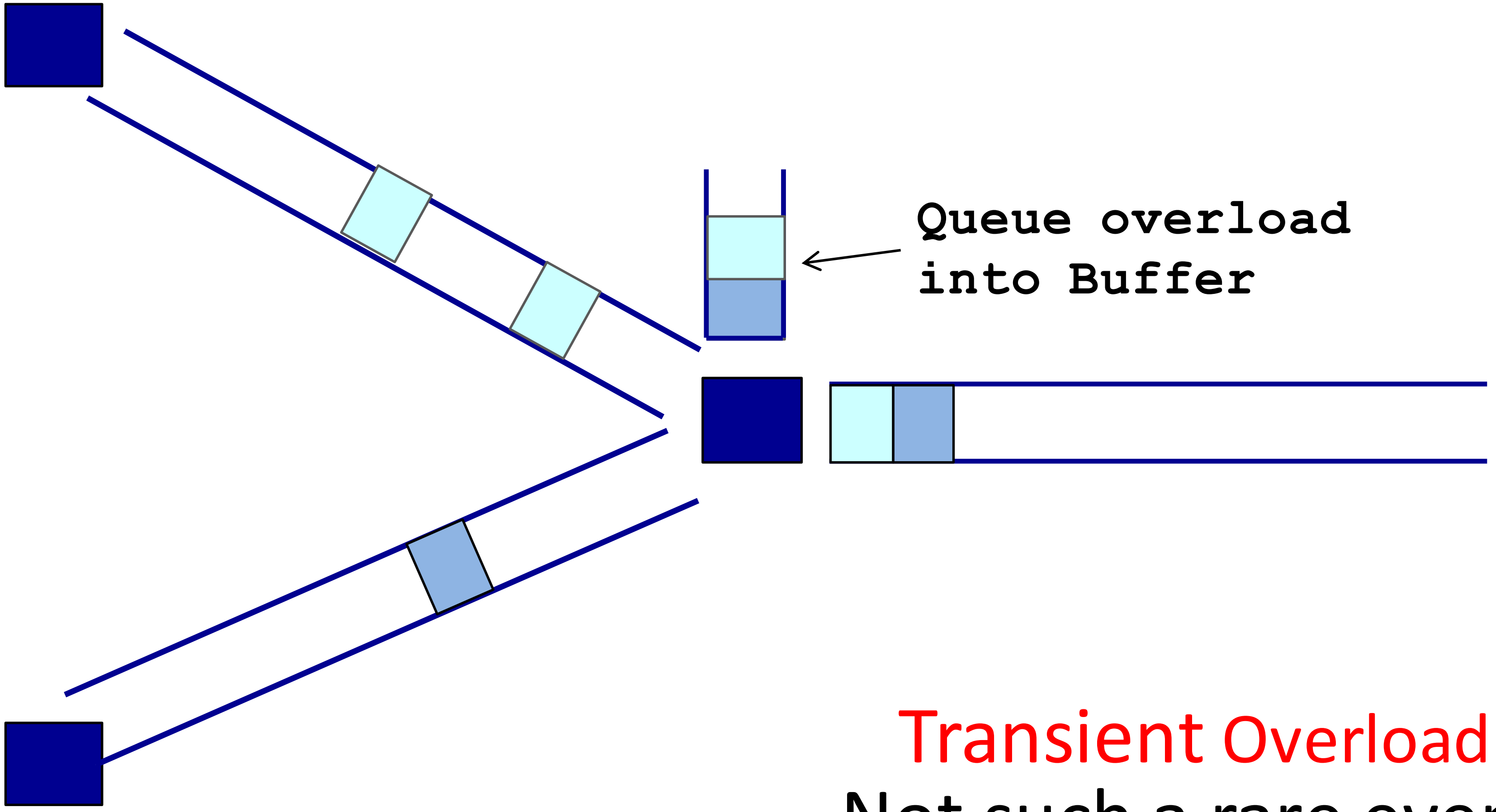
Statistical multiplexing: pipe view



Transient Overload

Not such a rare event

Statistical multiplexing: pipe view

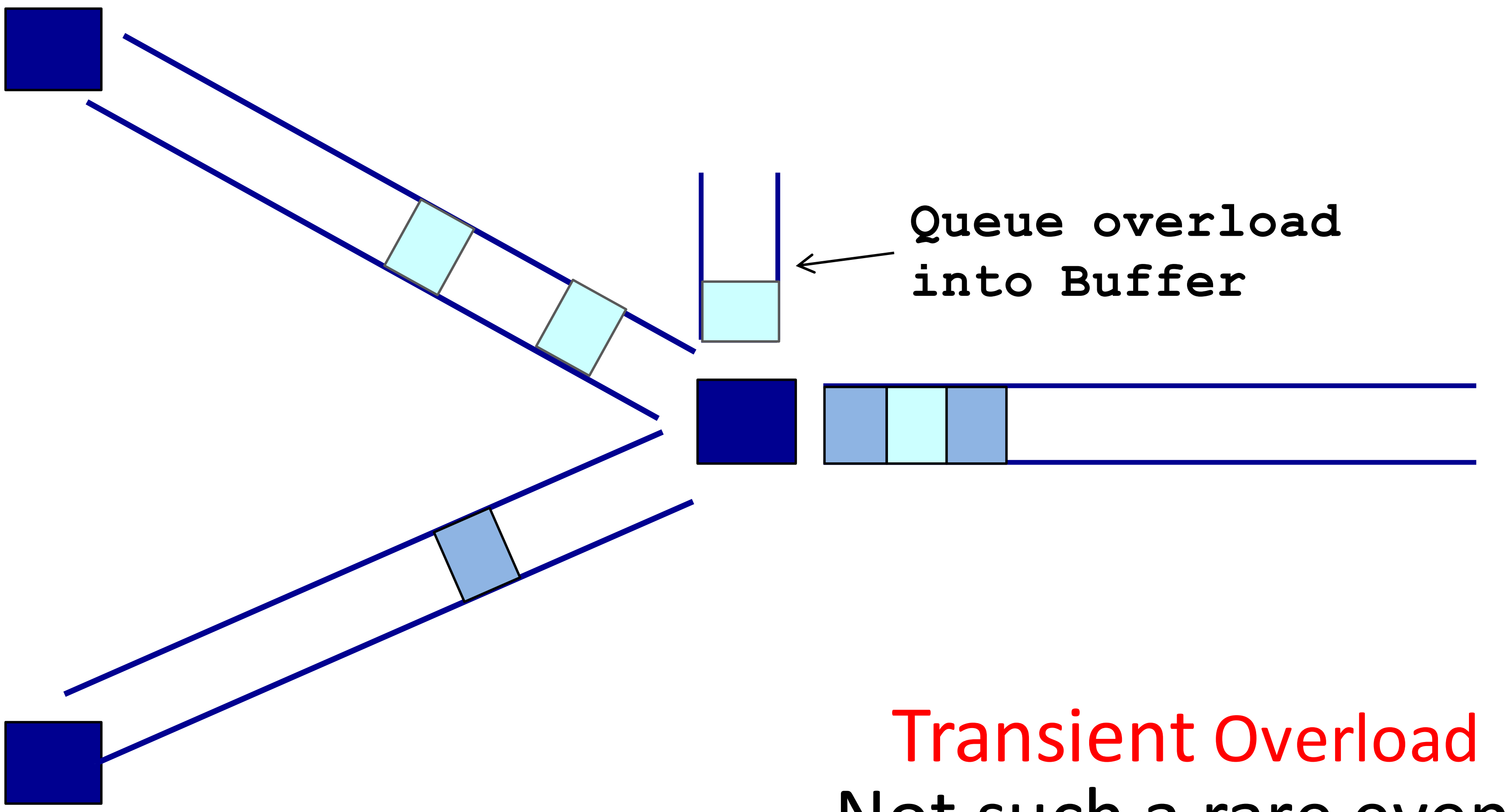


Queue overload into Buffer

<span style="color:red">Transient Overload</span>

Not such a rare event

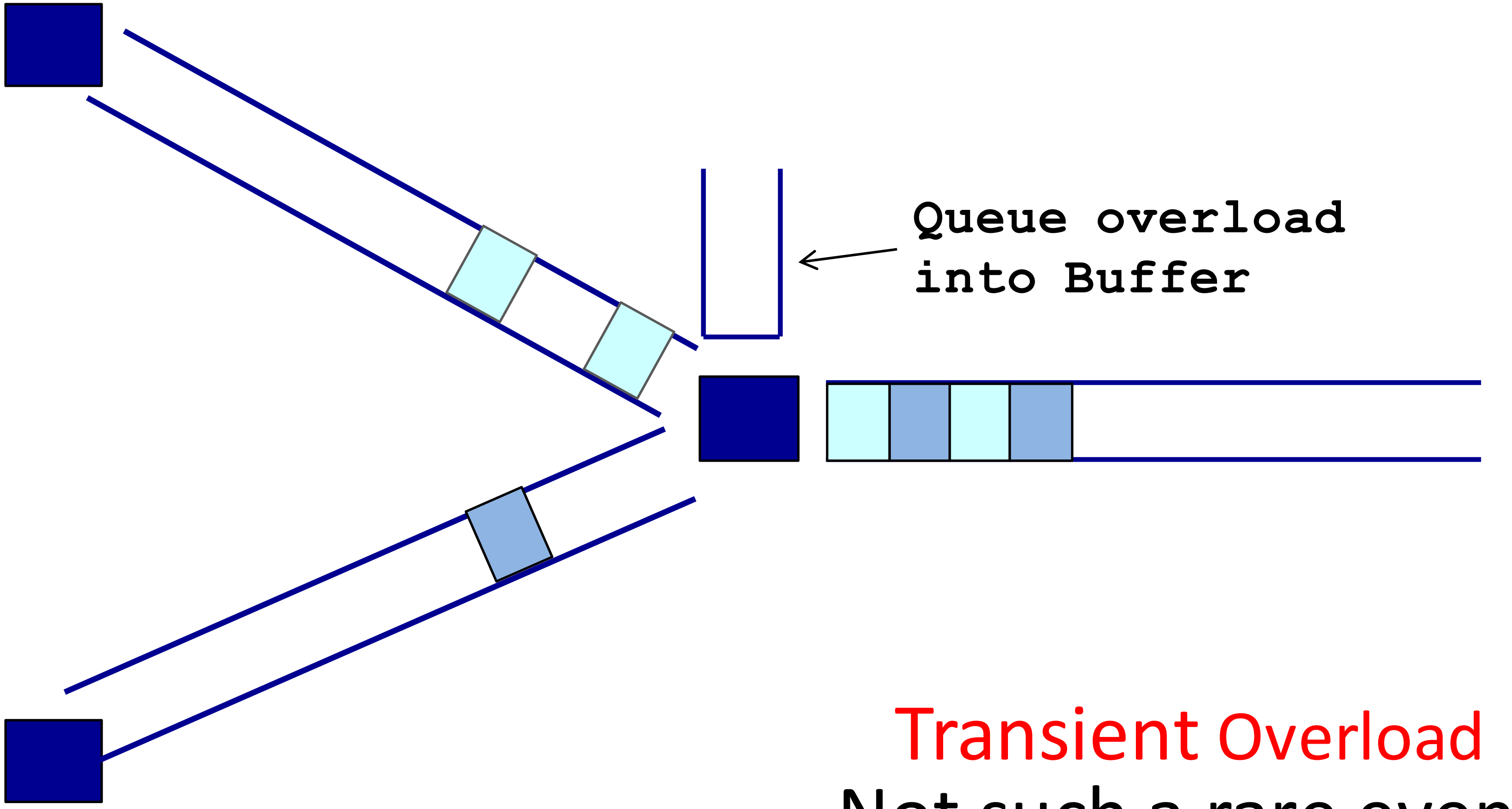
Statistical multiplexing: pipe view



Queue overload into Buffer

Transient Overload
Not such a rare event

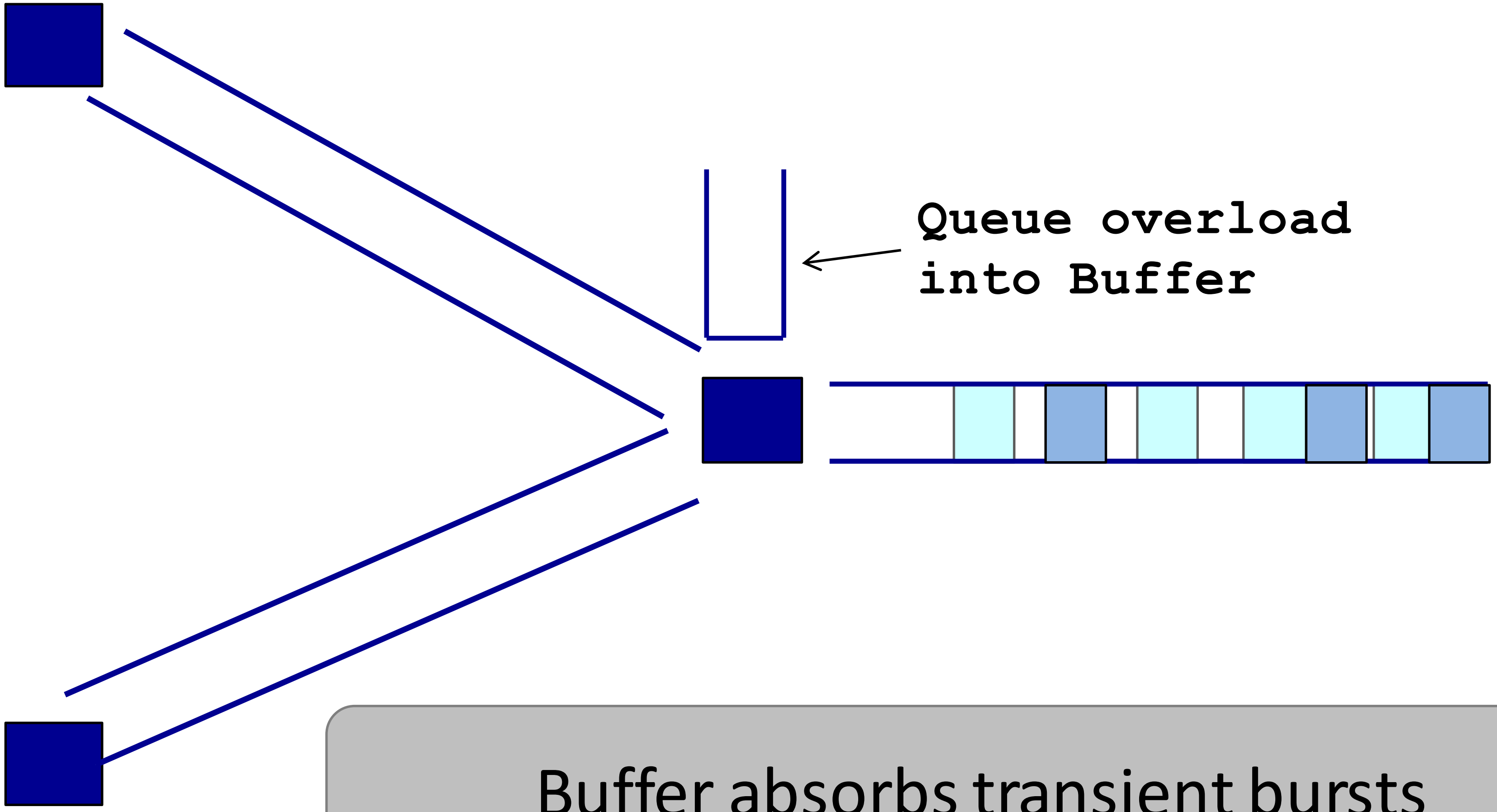
Statistical multiplexing: pipe view



Transient Overload

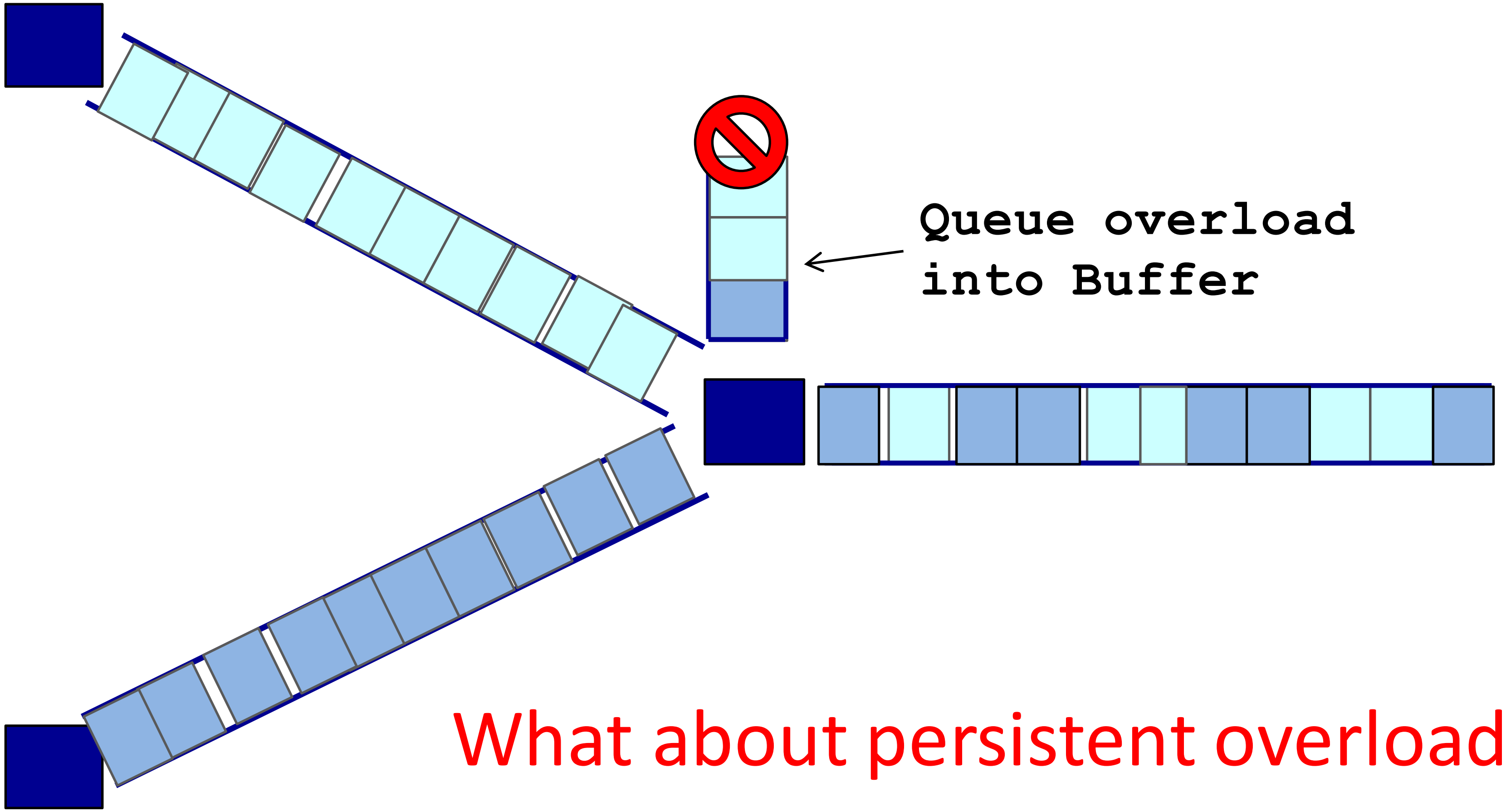
Not such a rare event

Statistical multiplexing: pipe view



Buffer absorbs transient bursts

Statistical multiplexing: pipe view



What about persistent overload?

Will eventually drop packets

Queues introduce queuing delays

- Recall,

  packet delay = transmission delay + propagation delay (*)

- With queues (statistical multiplexing)

  packet delay = transmission delay + propagation delay + queuing delay (*)

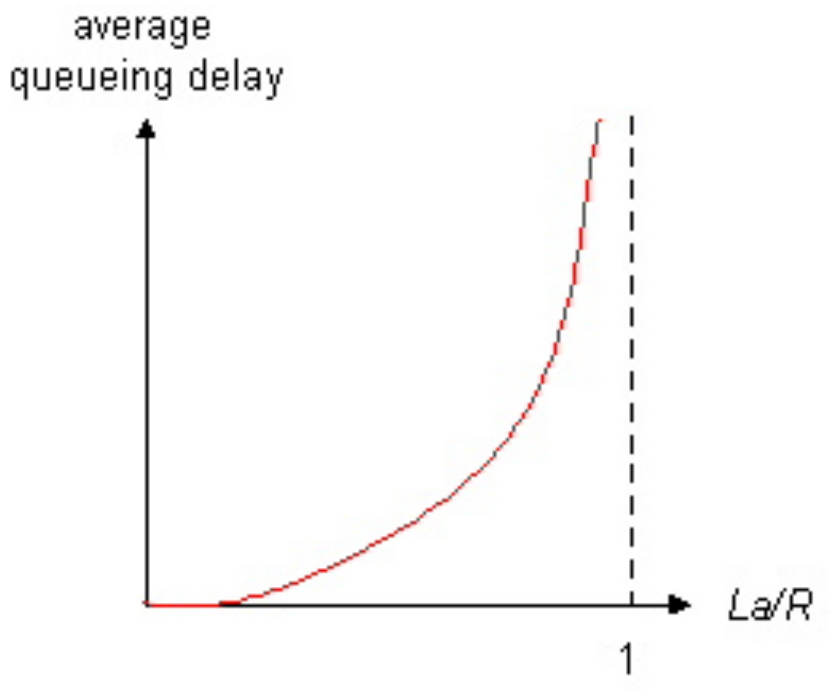
- Queuing delay caused by “packet interference”

- Made worse at high load
  - less “idle time” to absorb bursts
  - think about traffic jams at rush hour
    or rail network failure

(* plus per-hop *processing* delay that we define as negligible*)*

Queuing delay

- R=link bandwidth (bps)
- L=packet length (bits)
- a=average packet arrival rate

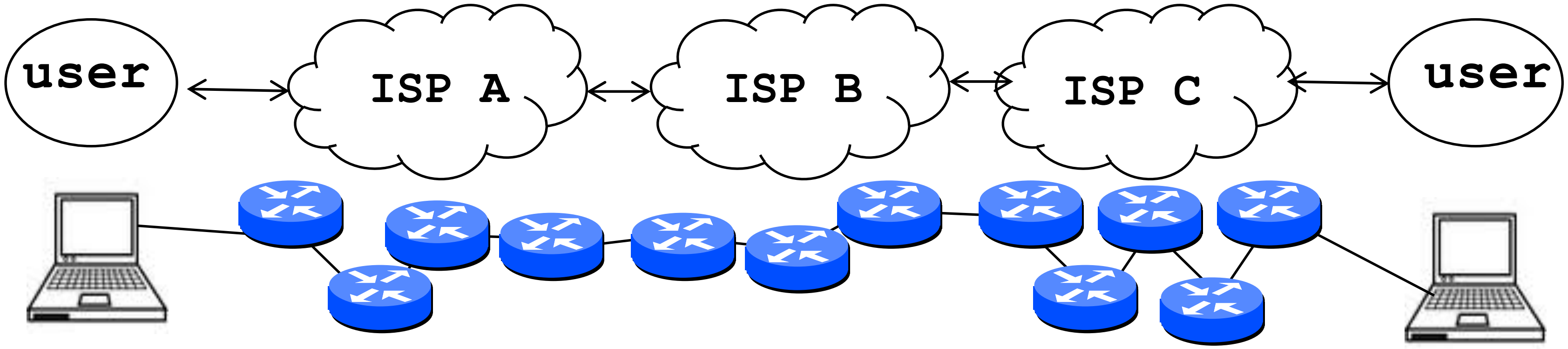


traffic intensity = La/R

- ❑ La/R ~ 0: average queuing delay small
- ❑ La/R -> 1: delays become large
- ❑ La/R > 1: more "work" arriving than can be serviced, average delay infinite – or data is lost (*dropped)*.

Recall the Internet *federation*

- The Internet ties together different networks
  - >18,000 ISP networks



We can see (hints) of the nodes and links using traceroute…

“Real” Internet delays and routes

traceroute: rio.cl.cam.ac.uk to munnari.oz.au

(tracepath on pwf is similar)

Three delay measurements from rio.cl.cam.ac.uk to gatwick.net.cl.cam.ac.uk

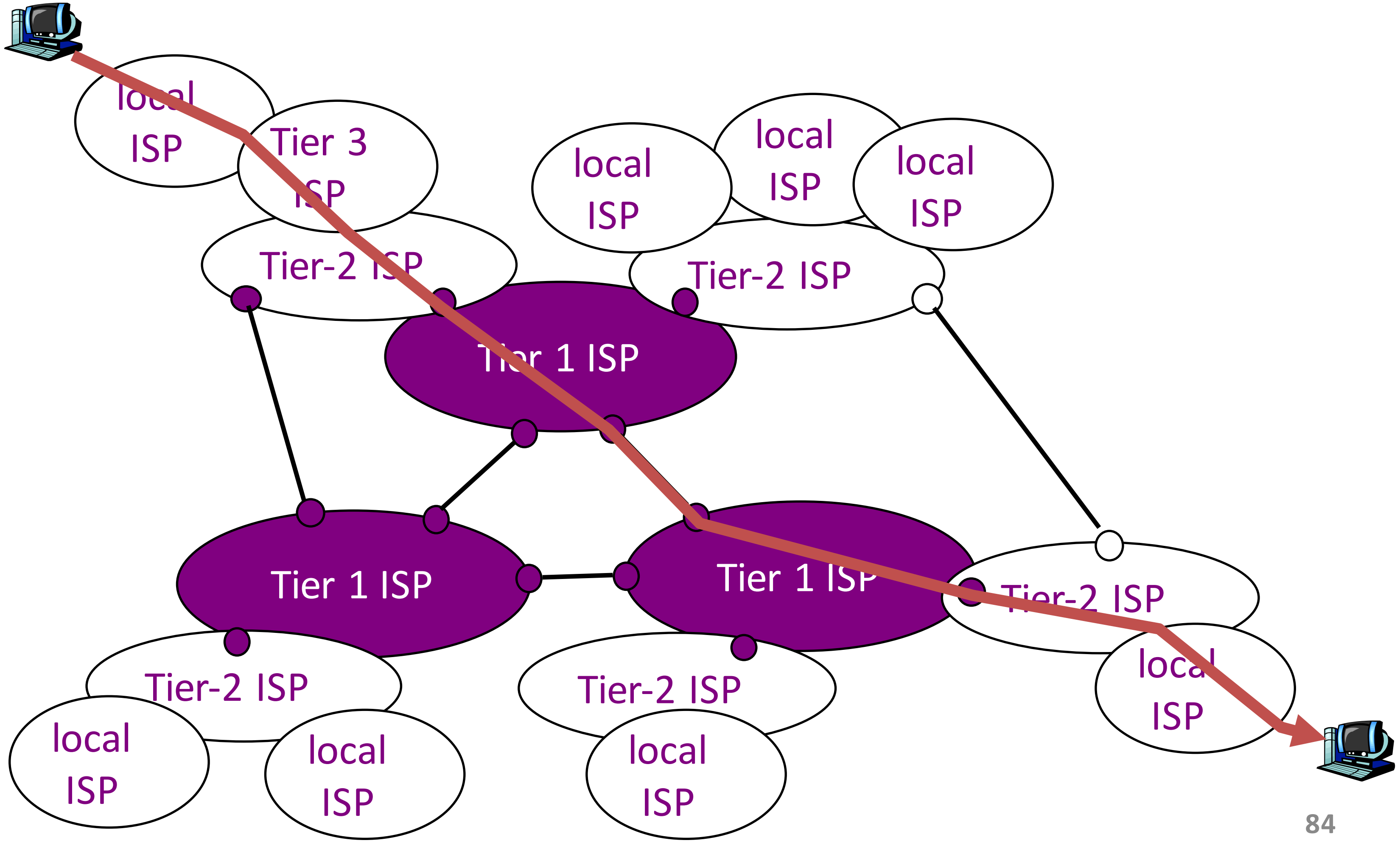
```
traceroute munnari.oz.au
traceroute to munnari.oz.au (202.29.151.3), 30 hops max, 60 byte packets
 1  gatwick.net.cl.cam.ac.uk (128.232.32.2)  0.416 ms  0.384 ms  0.427 ms
 2  cl-sby.route-nwest.net.cam.ac.uk (193.60.89.9)  0.393 ms  0.440 ms  0.494 ms
 3  route-nwest.route-mill.net.cam.ac.uk (192.84.5.137)  0.407 ms  0.448 ms  0.501 ms
 4  route-mill.route-enet.net.cam.ac.uk (192.84.5.94)  1.006 ms  1.091 ms  1.163 ms
 5  xe-11-3-0.camb-rbr1.eastern.ja.net (146.97.130.1)  0.300 ms  0.313 ms  0.350 ms
 6  ae24.lowdss-sbr1.ja.net (146.97.37.185)  2.679 ms  2.664 ms  2.712 ms
 7  ae28.londhx-sbr1.ja.net (146.97.33.17)  5.955 ms  5.953 ms  5.901 ms
 8  janet.mx1.lon.uk.geant.net (62.40.124.197)  6.059 ms  6.066 ms  6.052 ms
 9  ae0.mx1.par.fr.geant.net (62.40.98.77)  11.742 ms  11.779 ms  11.724 ms
10  ae1.mx1.mad.es.geant.net (62.40.98.64)  27.751 ms  27.734 ms  27.704 ms
    ------------------------------------------------------------------------
11  mb-so-02-v4.bb.tein3.net (202.179.249.117)  138.296 ms  138.314 ms  138.282 ms
12  sg-so-04-v4.bb.tein3.net (202.179.249.53)  196.303 ms  196.293 ms  196.264 ms
13  th-pr-v4.bb.tein3.net (202.179.249.66)  225.153 ms  225.178 ms  225.196 ms
14  pyt-thairen-to-02-bdr-pyt.uni.net.th (202.29.12.10)  225.163 ms  223.343 ms  223.363 ms
15  202.28.227.126 (202.28.227.126)  241.038 ms  240.941 ms  240.834 ms
16  202.28.221.46 (202.28.221.46)  287.252 ms  287.306 ms  287.282 ms
17  * * *
18  * * *
19  * * *
20  coe-gw.psu.ac.th (202.29.149.70)  241.681 ms  241.715 ms  241.680 ms
21  munnari.OZ.AU (202.29.151.3)  241.610 ms  241.636 ms  241.537 ms
```

trans-continent link

* means no response (probe lost, router not replying)

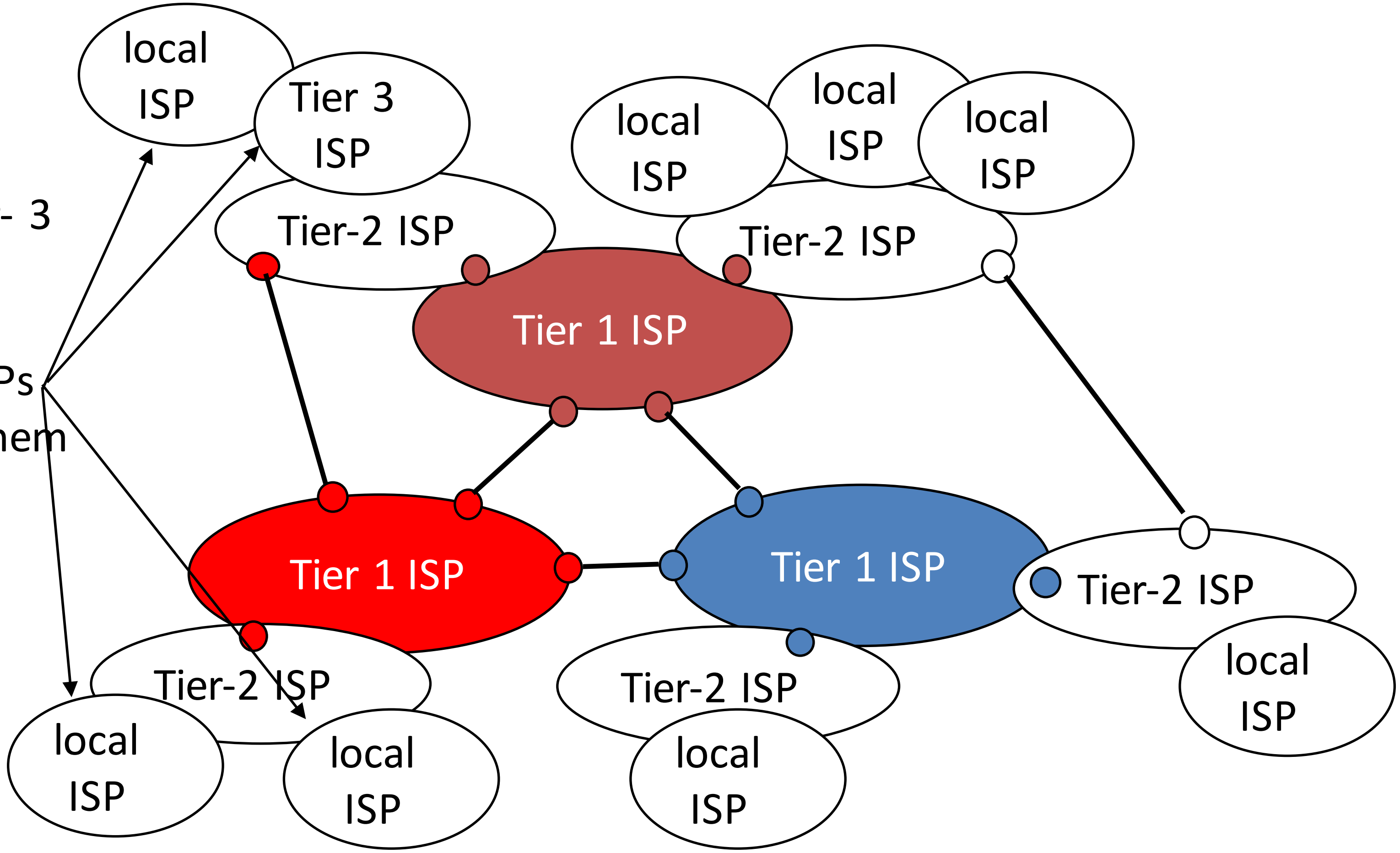
Internet structure: network of networks

- a packet passes through many networks!



Internet structure: network of networks

- "Tier-3" ISPs and local ISPs
  - last hop ("access") network (closest to end systems)



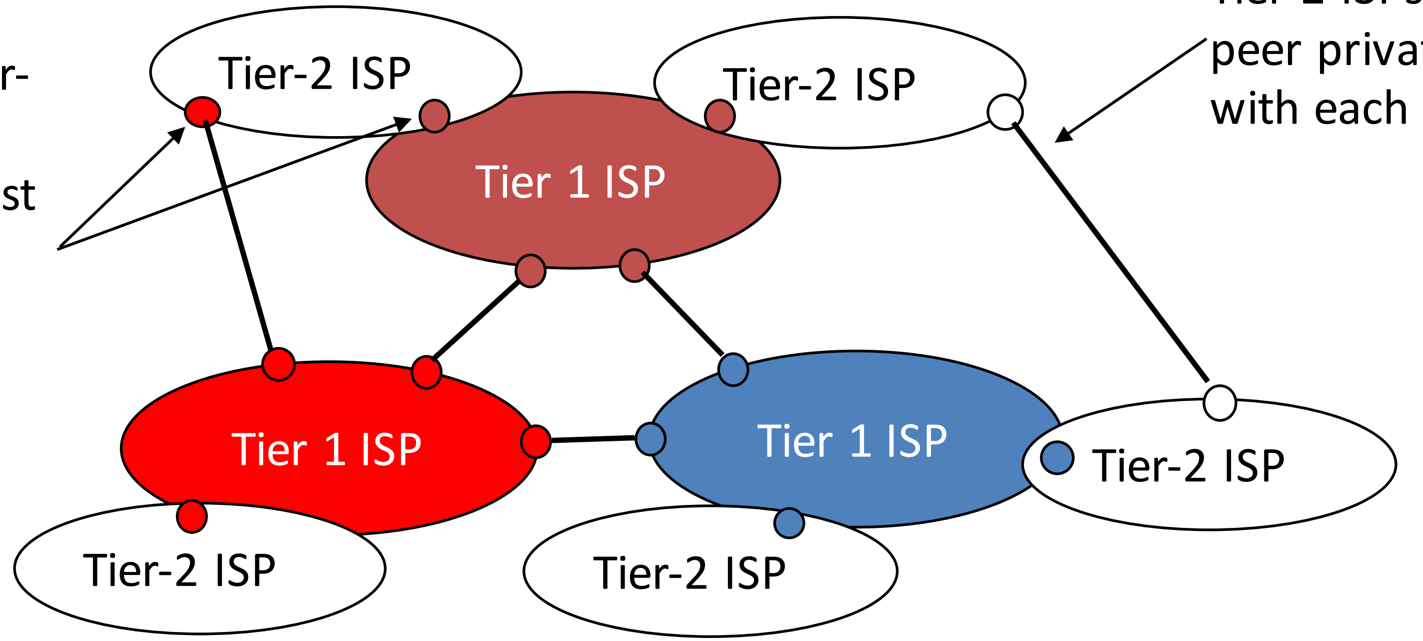
Local and tier- 3 ISPs are *customers* of higher tier ISPs connecting them to rest of Internet

Internet structure: network of networks

- "Tier-2" ISPs: smaller (often regional) ISPs
  - Connect to one or more tier-1 ISPs, possibly other tier-2 ISPs

Tier-2 ISP pays tier-1 ISP for connectivity to rest of Internet

❑ tier-2 ISP is *customer* of tier-1 provider

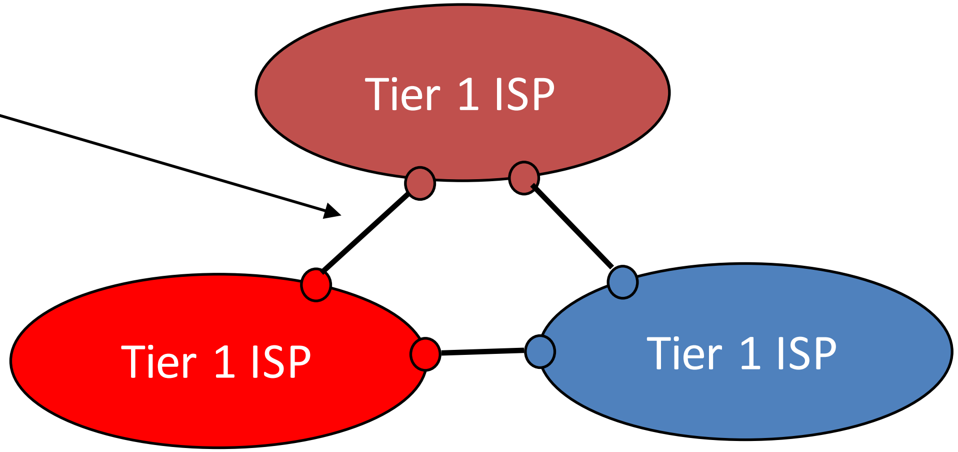


Tier-2 ISPs also peer privately with each other.

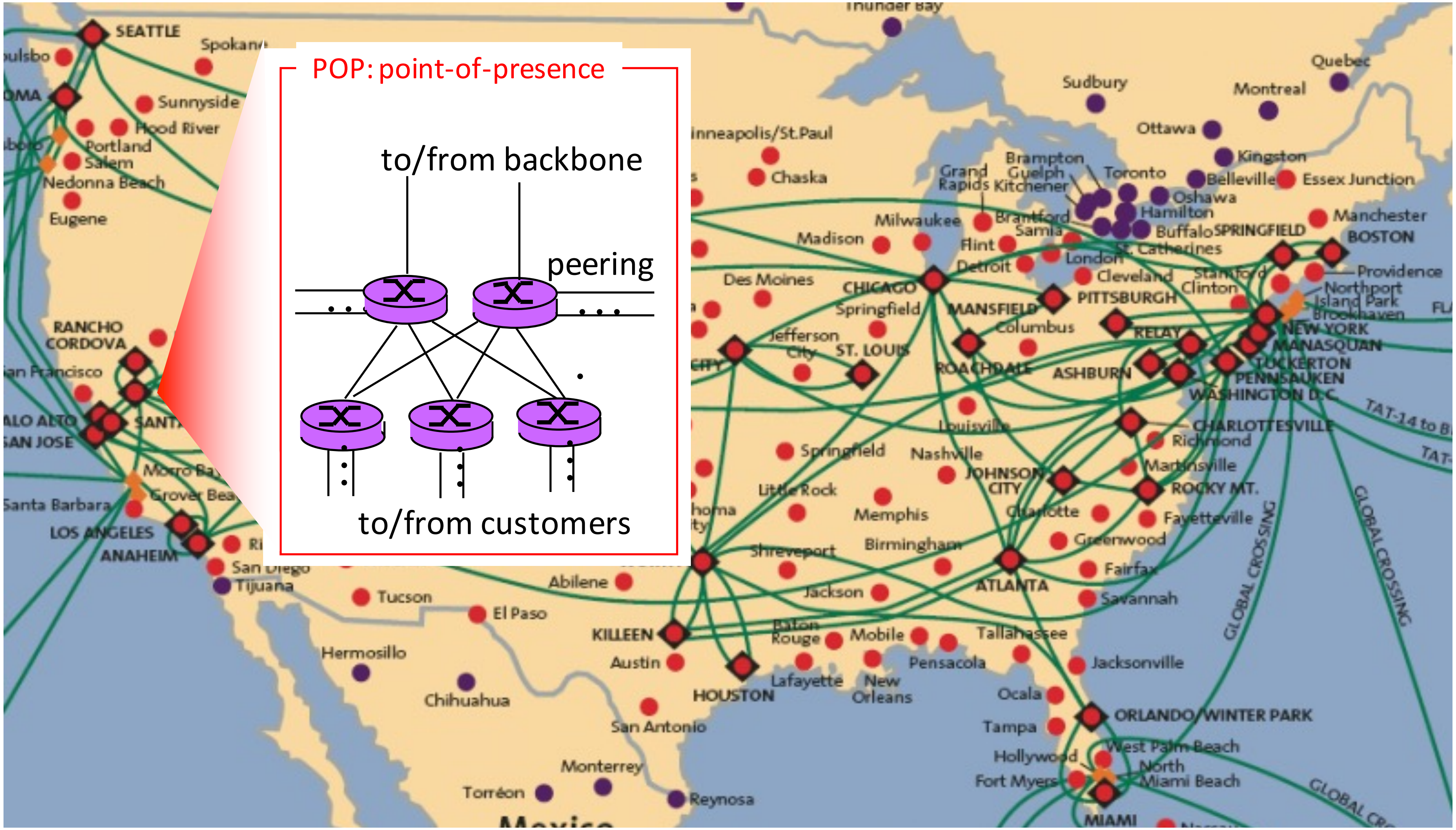
Internet structure: network of networks

- roughly hierarchical
- at center: “tier-1” ISPs (e.g., Verizon, Sprint, AT&T, Cable and Wireless), national/international coverage
  - treat each other as equals

Tier-1 providers interconnect (peer) privately



Tier-1 ISP: e.g., Sprint



POP: point-of-presence

to/from backbone

peering

to/from customers

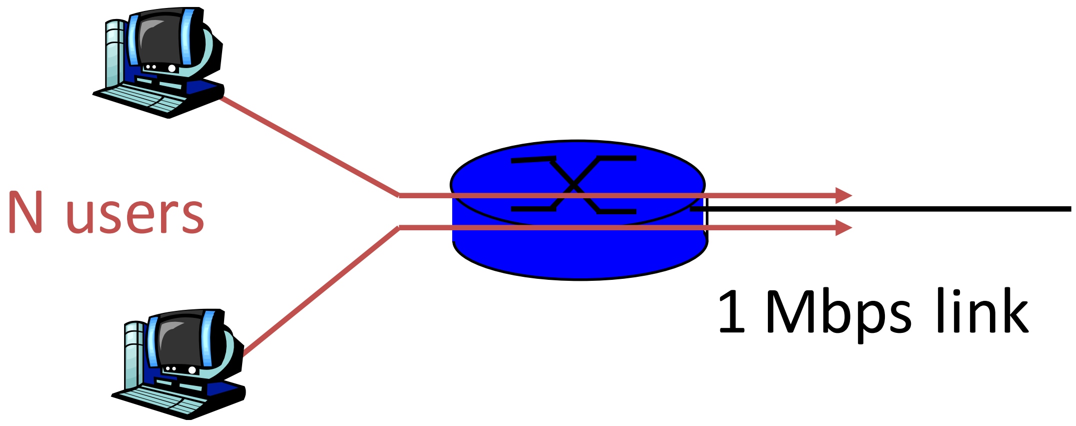
Packet Switching

- Data is sent as chunks of formatted bits (Packets)
- Packets consist of a “header” and “payload”
- Switches “forward” packets based on their headers
- Each packet travels independently
- No link resources are reserved in advance. Instead packet switching leverages statistical multiplexing
  - allows efficient use of resources
  - but introduces queues and queuing delays

Packet switching versus circuit switching

Packet switching may (does!) allow more users to use network

- 1 Mb/s link
- each user:
  - 100 kb/s when "active"
  - active 10% of time



- *circuit-switching:*
  - 10 users
- *packet switching:*
  - with 35 users, probability > 10 active at same time is less than .0004

Q: how did we get value 0.0004?

Packet switching versus circuit switching

Q: how did we get value 0.0004?

- 1 Mb/s link
- each user:
  - 100 kb/s when "active"
  - active 10% of time

HINT: Binomial Distribution

- *circuit-switching:*
  - 10 users
- *packet switching:*
  - with 35 users, probability > 10 active at same time is less than .0004

Circuit switching: pros and cons

- Pros
  - guaranteed performance
  - fast transfers (once circuit is established)

- Cons
  - wastes bandwidth if traffic is “bursty”
  - connection setup adds delay
  - recovery from failure is slow

Packet switching: pros and cons

- Cons
  - no guaranteed performance
  - header overhead per packet
  - queues and queuing delays

- Pros
  - efficient use of bandwidth (stat. muxing)
  - no overhead due to connection setup
  - resilient -- can `route around trouble'

Summary

- A sense of how the basic `plumbing' works
  - links and switches
  - packet delays= transmission + propagation + queuing + (negligible) per-switch processing
  - statistical multiplexing and queues
  - circuit vs. packet switching

Topic 2 – Architecture and Philosophy

- Abstraction
- Layering
- Layers and Communications
- Entities and Peers
- What is a protocol?
- Protocol Standardization
- The architects process
  - How to break system into modules
  - Where modules are implemented
  - Where is state stored
- Internet Philosophy and Tensions

Abstraction Concept

A mechanism for breaking down a problem

what not *how*

- eg Specification *versus* implementation
- eg Modules in programs

Allows replacement of implementations without affecting system behavior

Vertical versus *Horizontal*

"Vertical" what happens in a box "How does it attach to the network?"

"Horizontal" the communications paths running through the system

Hint: paths are build on top of ("layered over") other paths

Computer System Modularity

Partition system into modules & abstractions:

- Well-defined interfaces give flexibility
  - ***Hides*** implementation - can be freely changed
  - Extend functionality of system by adding new modules

- E.g., libraries encapsulating set of functionality

- E.g., programming language + compiler abstracts away how the particular CPU works …

Computer System Modularity (cnt’d)

- Well-defined interfaces hide information
  - Isolate assumptions
  - Present high-level abstractions
- **But can impair performance!**
- Ease of implementation vs worse performance

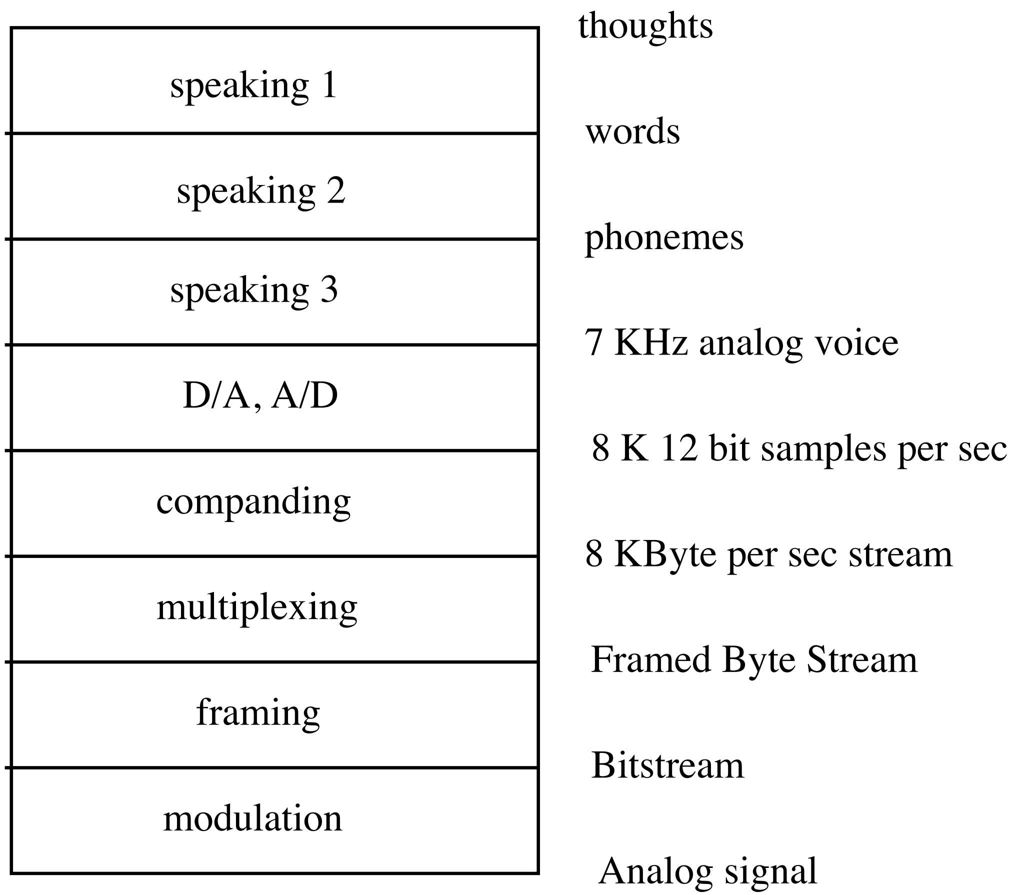
Network System Modularity

Like software modularity, but:

- Implementation is distributed across many machines (routers and hosts)
- Must decide:
  - How to break system into modules
    - **Layering**
  - Where modules are implemented
    - **End-to-End Principle**
  - Where state is stored
    - **Fate-sharing**

Layering Concept

- A restricted form of abstraction: system functions are divided into layers, one built upon another
- Often called a *stack*; but **not** a data structure!



thoughts

speaking 1

words

speaking 2

phonemes

speaking 3

7 KHz analog voice

D/A, A/D

8 K 12 bit samples per sec

companding

8 KByte per sec stream

multiplexing

Framed Byte Stream

framing

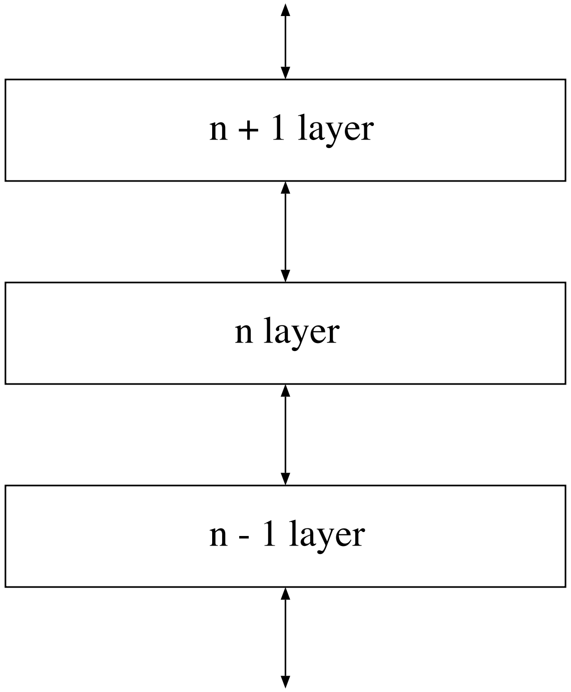
Bitstream

modulation

Analog signal

Layers and Communications

- Interaction only between adjacent layers
- *layer n* uses services provided by *layer n-1*
- *layer n* provides service to *layer n+1*
- Bottom layer is physical media
- Top layer is application



Entities and Peers

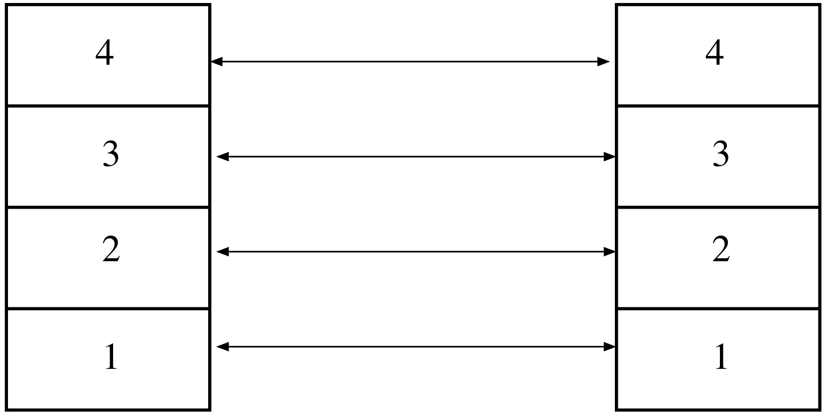
Entity – a *thing* (an independent existence)

Entities *interact* with the layers above and below

Entities *communicate* with *peer* entities

- same level but different place (eg different person, different box, different host)

Communications between peers is supported by entities at the lower layers



Entities and Peers

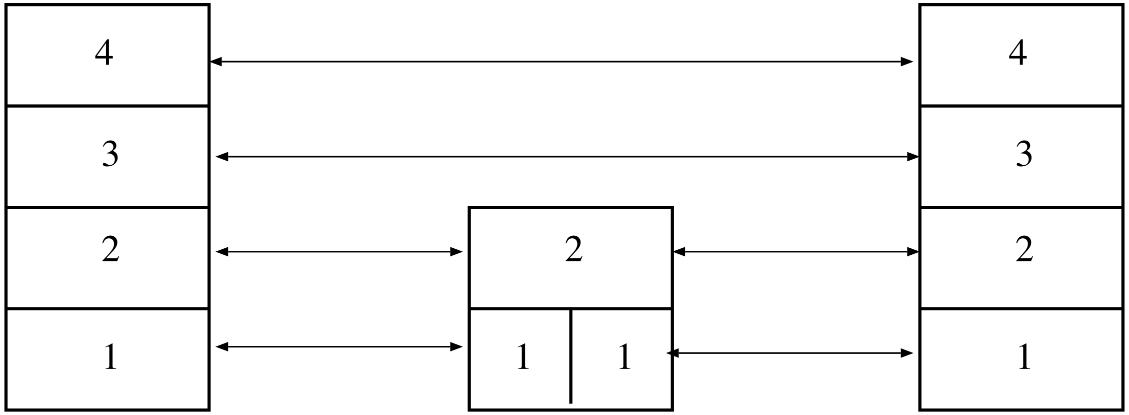
Entities usually do something useful

- Encryption – Error correction – Reliable Delivery
- Nothing at all is also reasonable

Not all communications is end-to-end

Examples for things in the middle

- IP Router – Mobile Phone Cell Tower
- Person translating French to English



Layering and Embedding

In Computer Networks we often see higher-layer information embedded within lower-layer information

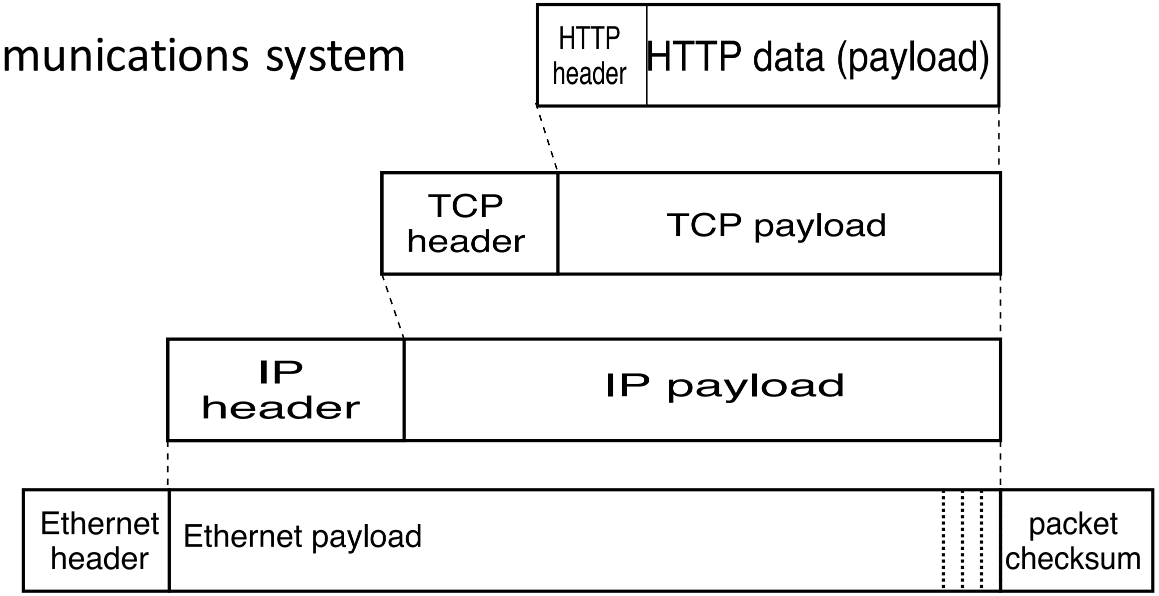
- Such embedding can be considered a form of layering
- Higher layer information is generated by stripping off headers and trailers of the current layer
- eg an IP entity only looks at the IP headers

BUT embedding is not the only form of layering

Layering is to help understand a communications system

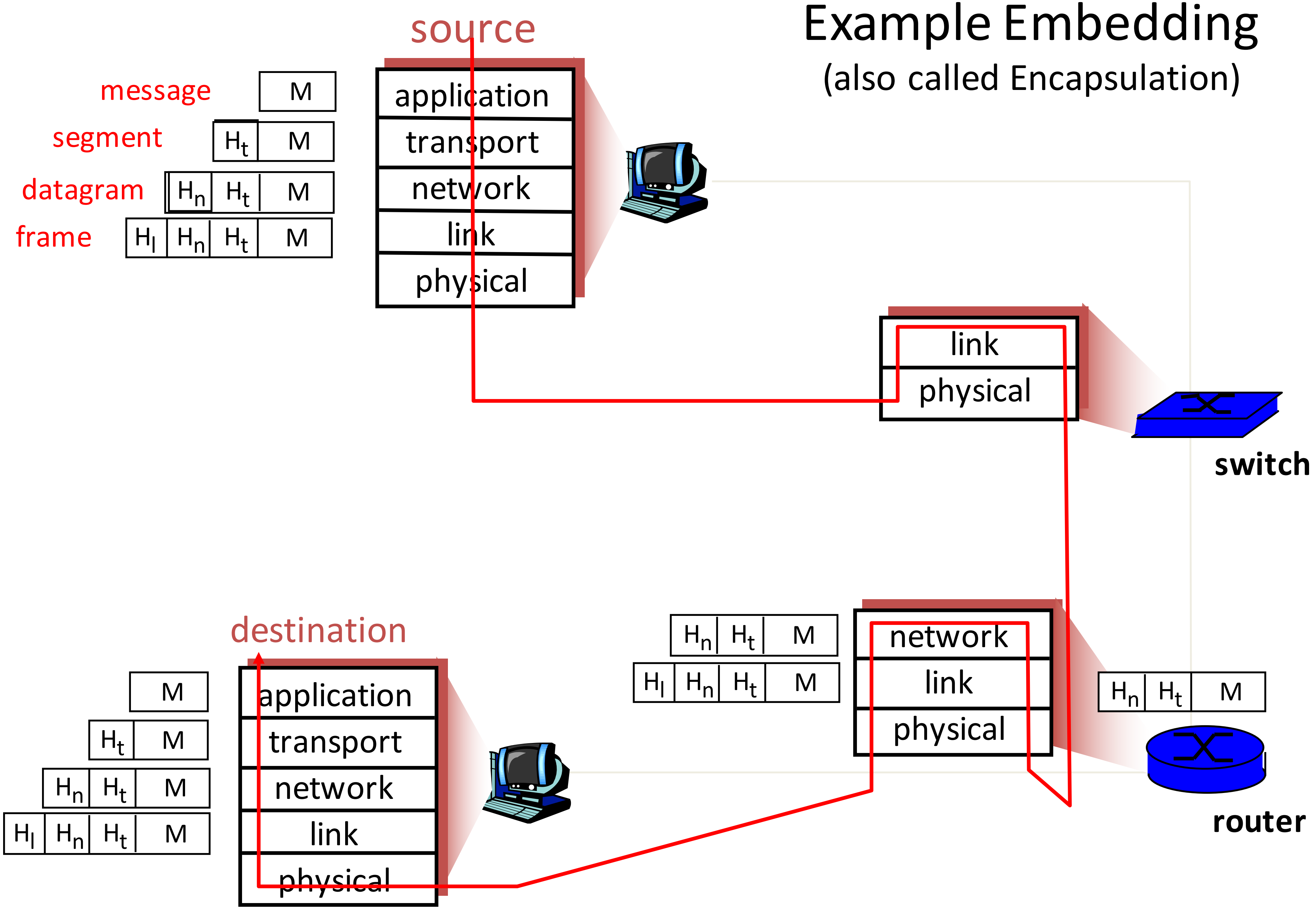
NOT

determine implementation strategy



Example Embedding

(also called Encapsulation)



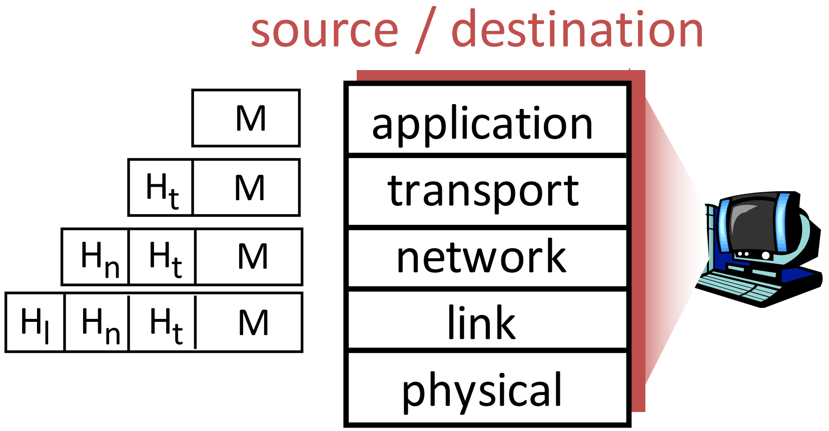
Distributing Layers Across Network

- Layers are simple if only on a single machine
  - Just stack of modules interacting with those above/below
- But we need to implement layers across machines
  - Hosts
  - Routers (switches)
- What gets implemented where?

What Gets Implemented on Host?

- Bits arrive on wire, must make it up to application

- Therefore, all layers must exist at the host



source / destination

M

H_t M

H_n H_t M

H_l H_n H_t M

application

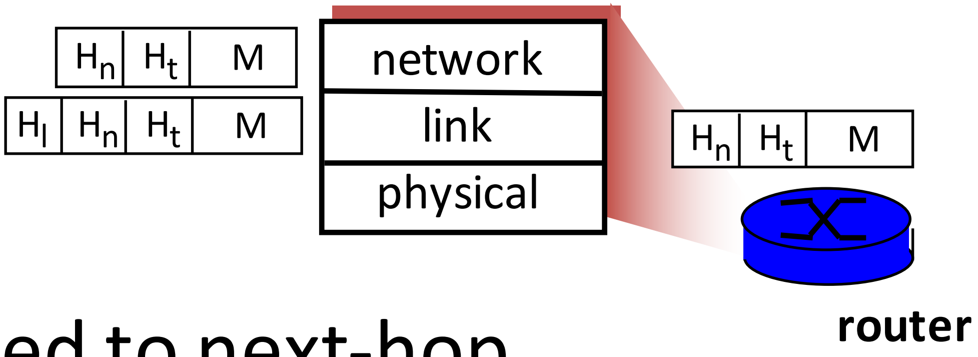
transport

network

link

physical

What Gets Implemented on a Router?

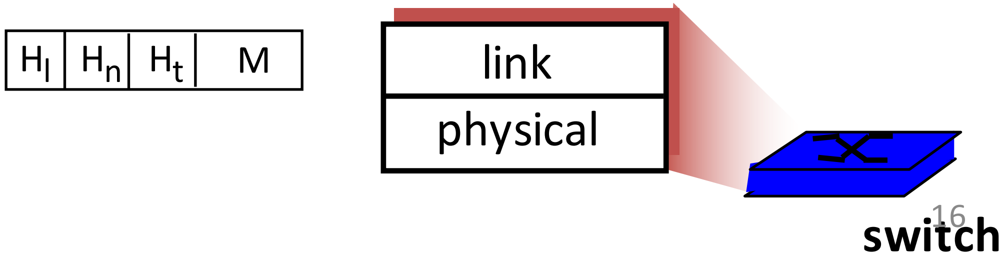


- Bits arrive on wire
  - Physical layer necessary
- Packets must be delivered to next-hop
  - Datalink layer necessary
- Routers participate in global delivery
  - Network layer necessary
- Routers don’t support reliable delivery
  - Transport layer (and above) ***not*** supported

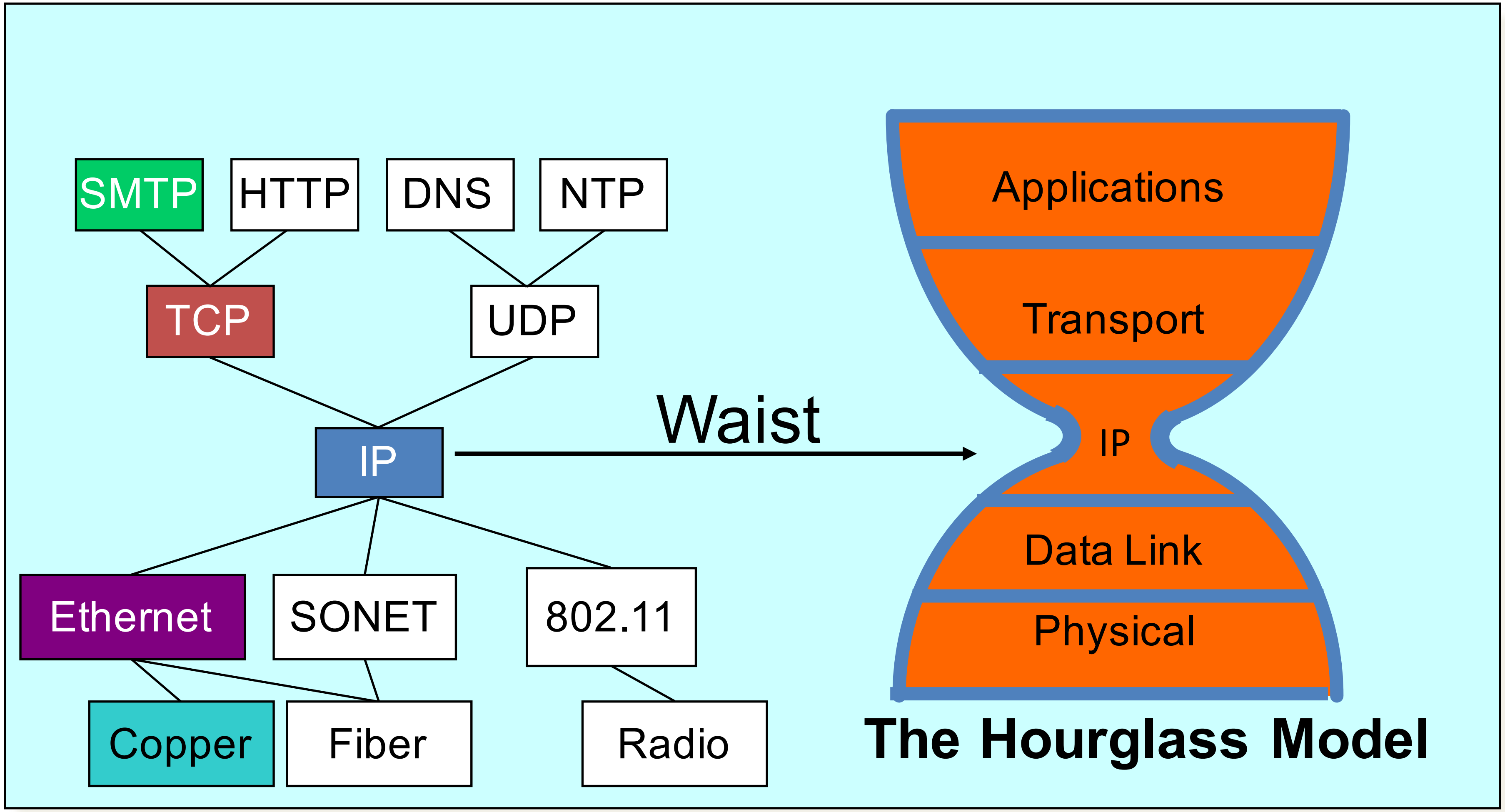
What Gets Implemented on Switches?

- Switches do what routers do, except they don’t participate in global delivery, just local delivery
- They only need to support Physical and Datalink
  - Don’t need to support Network layer
- Won’t focus on the router/switch distinction
  - When I say switch, I almost always mean router
  - Almost all boxes support network layer these days

  Routers have switches but switches do not have routers

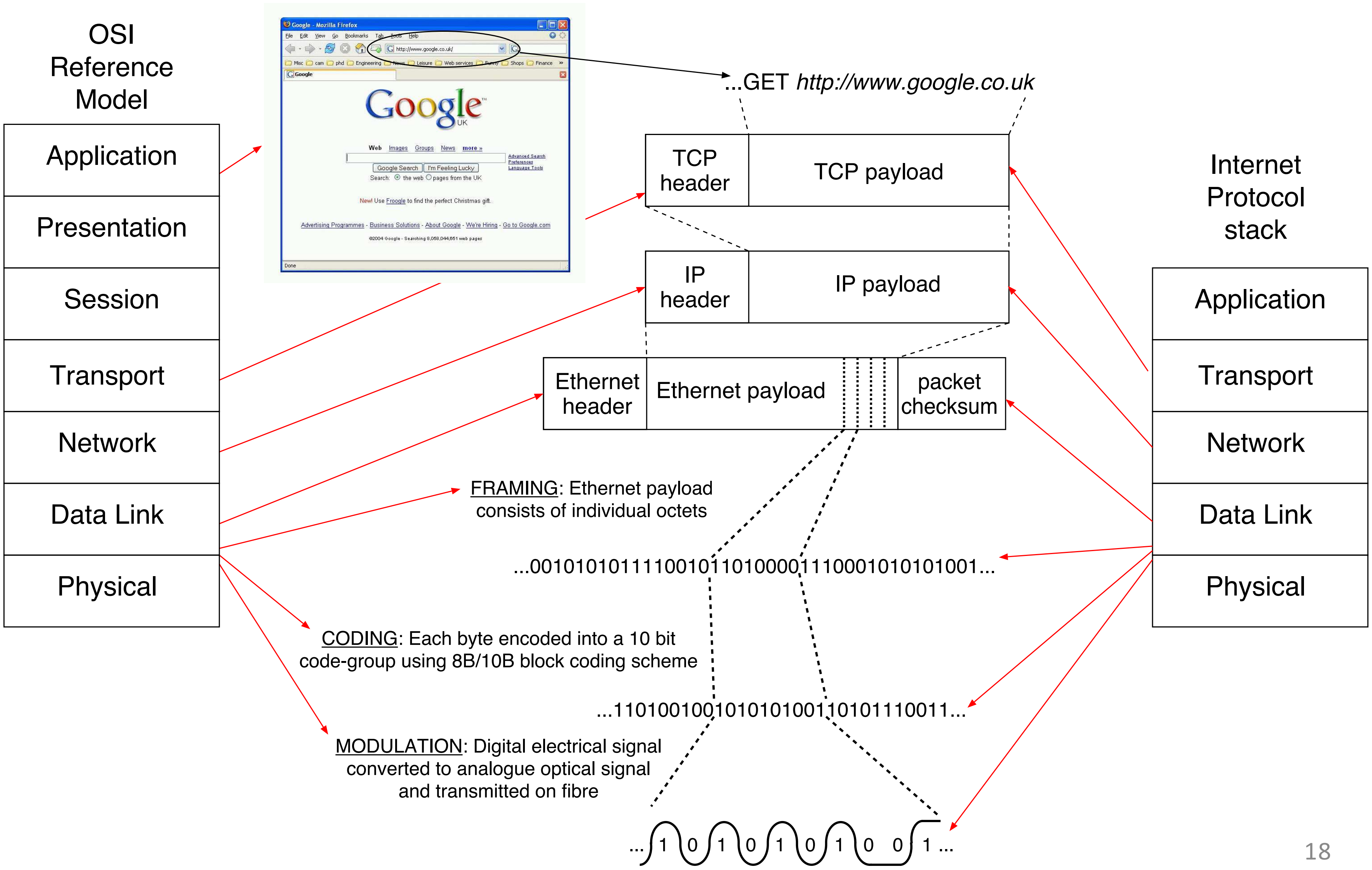


The Internet *Hourglass*



There is just one network-layer protocol, **IP**.
The “narrow waist” facilitates interoperability.

Internet protocol stack *versus* OSI Reference Model



New! Use Froogle to find the perfect Christmas gift.

©2004 Google - Searching 8,058,044,651 web pages

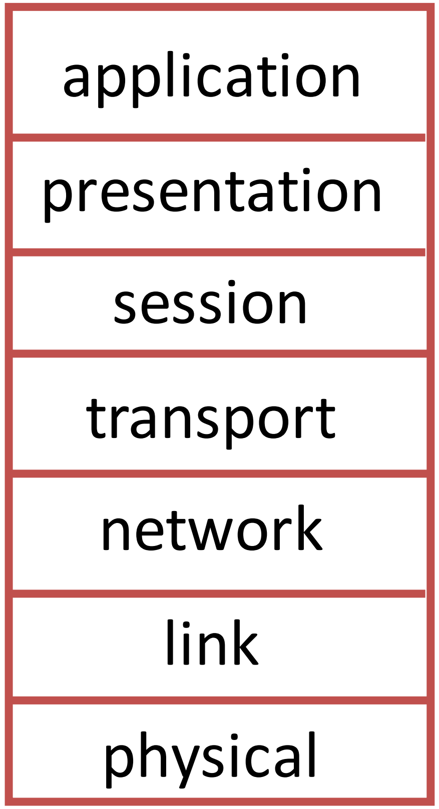
FRAMING: Ethernet payload consists of individual octets

CODING: Each byte encoded into a 10 bit code-group using 8B/10B block coding scheme

MODULATION: Digital electrical signal converted to analogue optical signal and transmitted on fibre

ISO/OSI reference model

- presentation: allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- *session:* synchronization, checkpointing, recovery of data exchange
- Internet stack "missing" these layers!
  - these services, *if needed,* must be implemented in application
  - needed?



application

presentation

session

transport

network

link

physical

What is a protocol?

human protocols:

- “what’s the time?”
- “I have a question”
- introductions

… specific msgs sent

… specific actions taken when msgs received, or other events

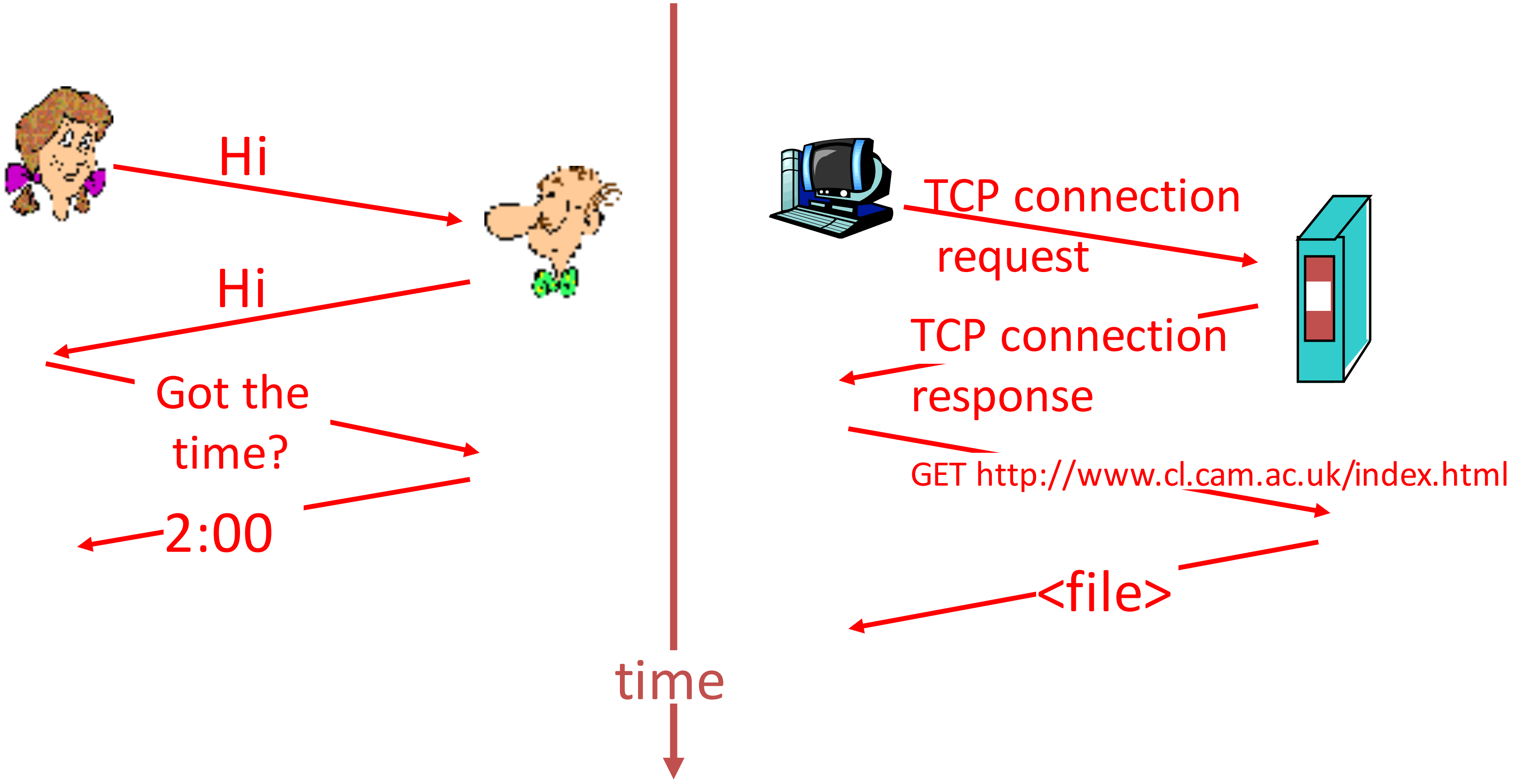
network protocols:

- machines rather than humans
- all communication activity in Internet governed by protocols

protocols define format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt

What is a protocol?

a human protocol and a computer network protocol:



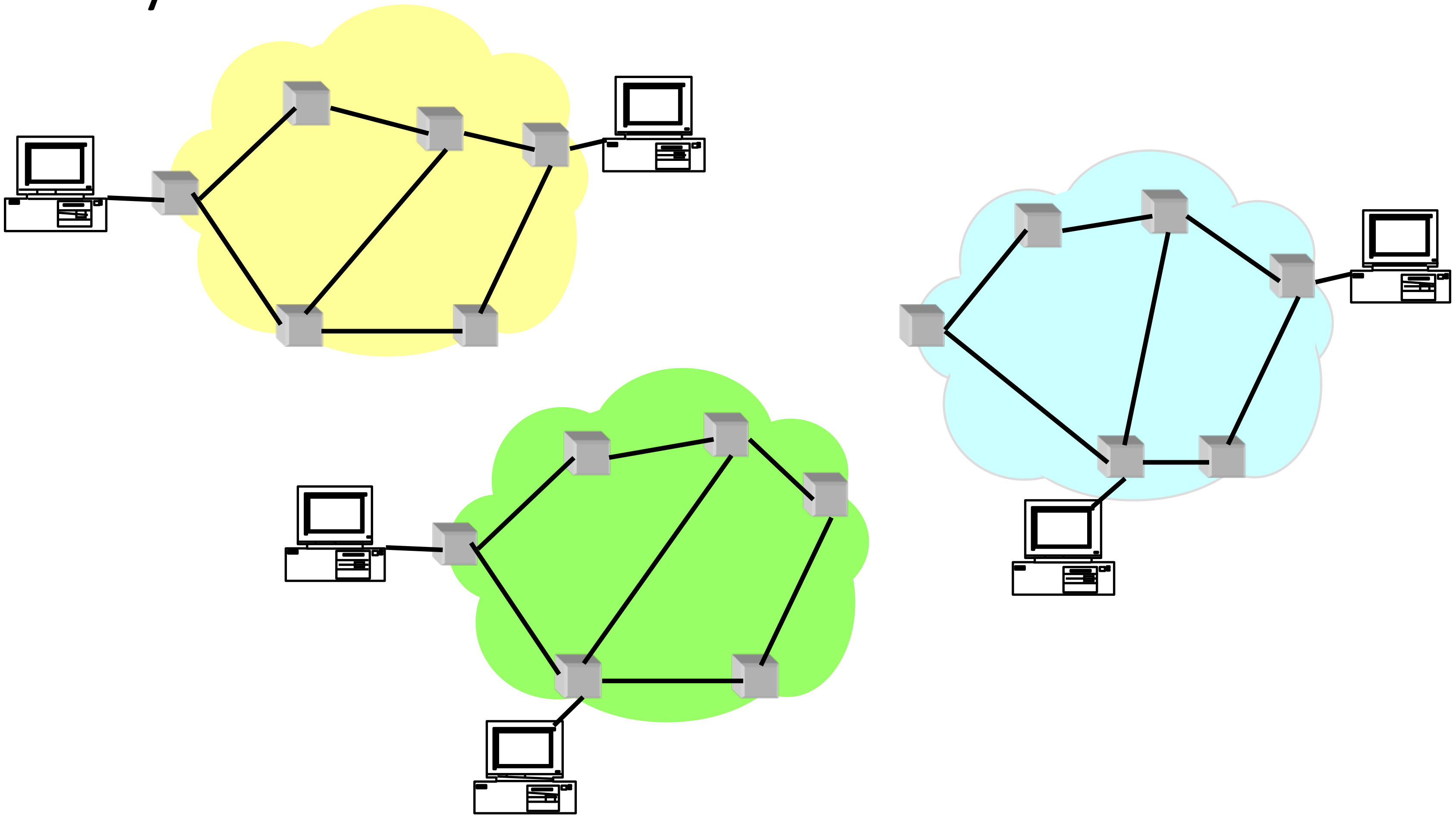
Q: Other human protocols?

Protocol Standardization

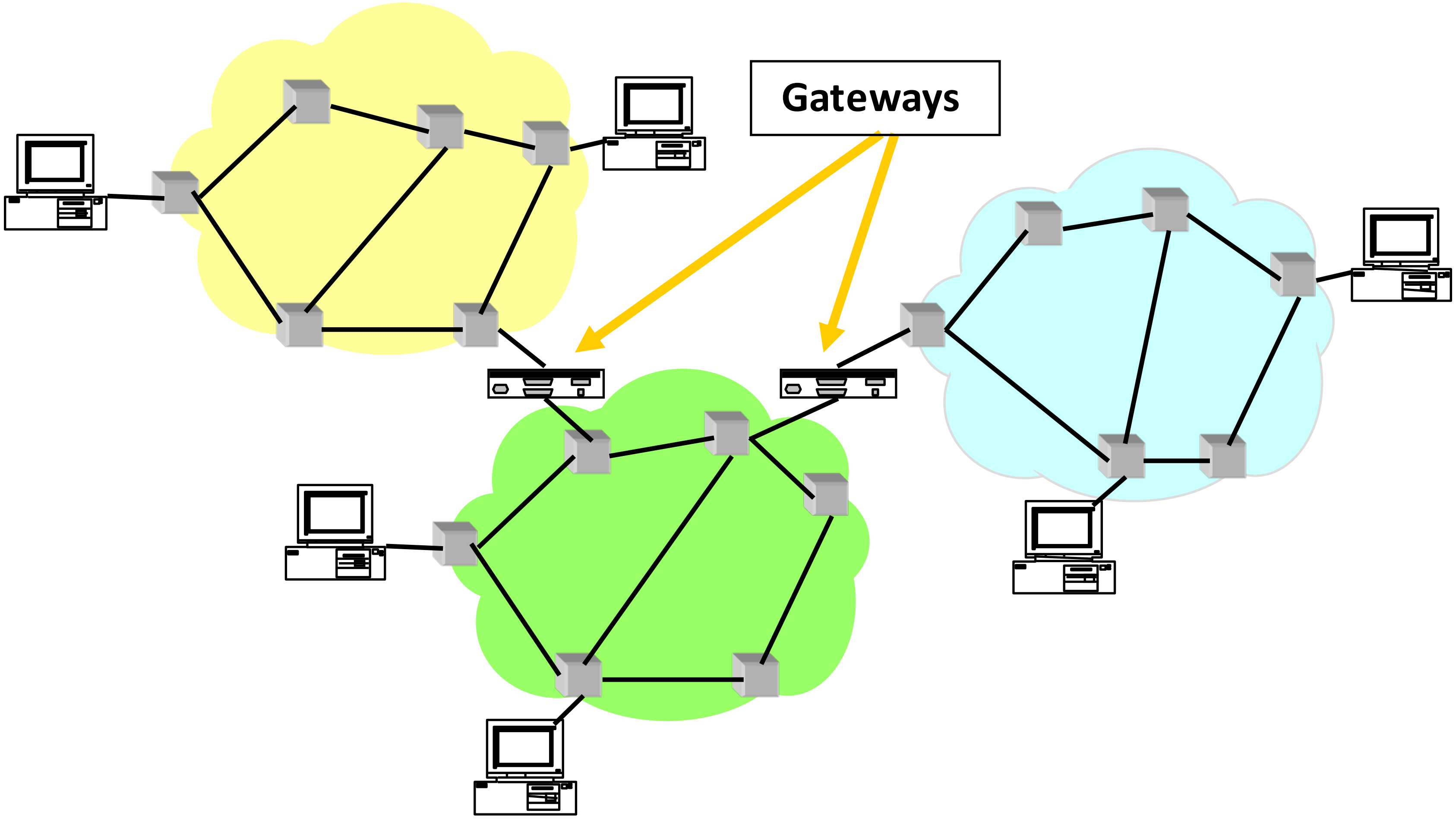
- All hosts must follow same protocol
  - Very small modifications can make a big difference
  - Or prevent it from working altogether
  - Cisco bug compatible!
- This is why we have standards
  - Can have multiple implementations of protocol
- Internet Engineering Task Force
  - Based on working groups that focus on specific issues
  - Produces “Request For Comments” (RFCs)
  - IETF Web site is ***http://www.ietf.org***
  - RFCs archived at ***http://www.rfc-editor.org***

So many Standards Problem

- Many different packet-switching networks
- Each with its own Protocol
- Only nodes on the same network could communicate



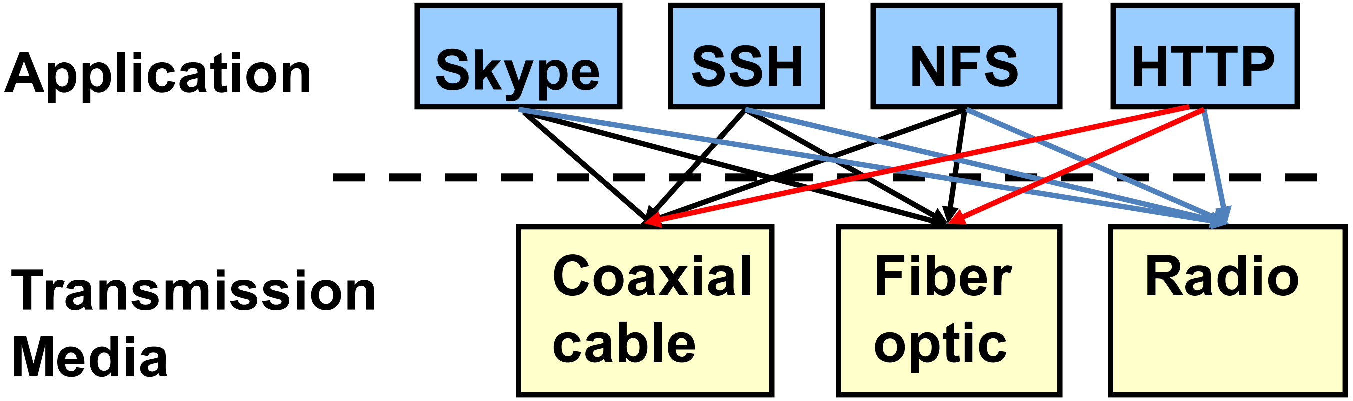
INTERnet Solution



Alternative to Standardization?

- Have one implementation used by everyone
- Open-source projects
  - Which has had more impact, Linux or POSIX?
- Or just sole-sourced implementation
  - Skype, many P2P implementations, etc.

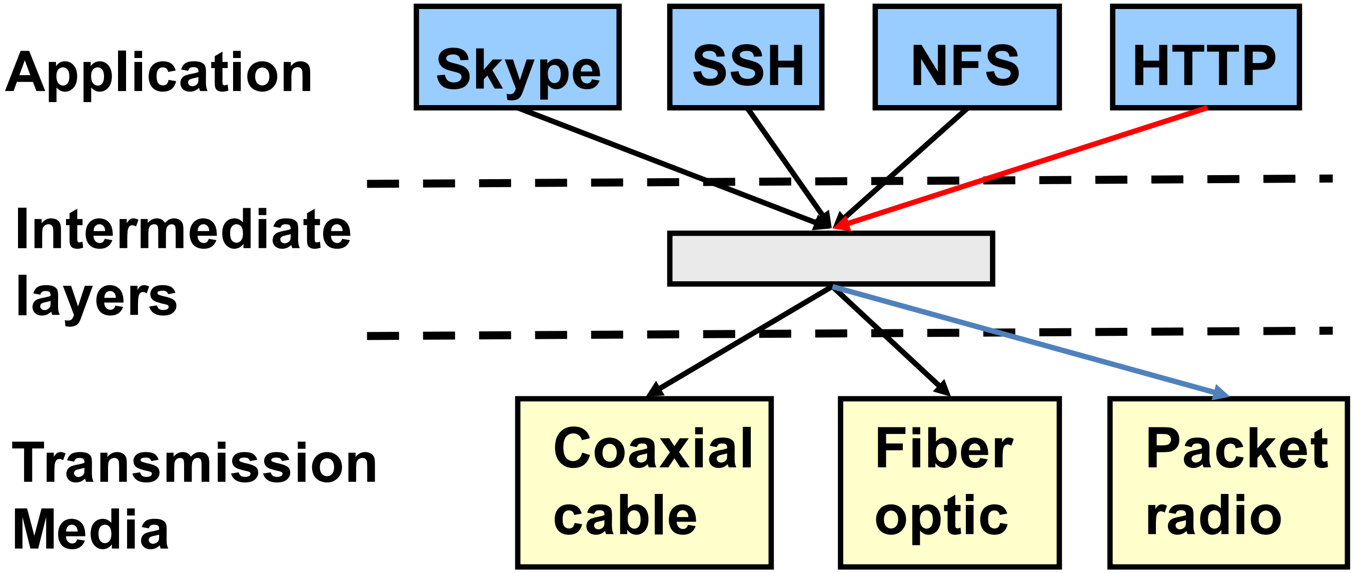
A Multitude of Apps Problem



- Re-implement every application for every technology?
- No! But how does the Internet design avoid this?

Solution: Intermediate Layers

- Introduce intermediate layers that provide set of abstractions for various network functionality and technologies
  - A new app/media implemented only once
  - Variation on "add another level of indirection"



Remember that slide!

- The relationship between architectural principles and architectural decisions is crucial to understand

Internet Design Goals (Clark ‘88)

- **Connect existing networks**
- Robust in face of failures
- Support multiple types of delivery services
- Accommodate a variety of networks
- Allow distributed management
- Easy host attachment
- Cost effective
- Allow resource accountability

Real Goals

Internet Motto

We reject kings , presidents, and voting. We believe in rough consensus and running code." – David Clark

- **Build something that works!**
- Connect existing networks
- Robust in face of failures
- Support multiple types of delivery services
- Accommodate a variety of networks
- Allow distributed management
- Easy host attachment
- Cost effective
- ~~Allow resource accountability~~

In the context of the Internet

Applications

…built on…

Reliable (or unreliable) transport

…built on…

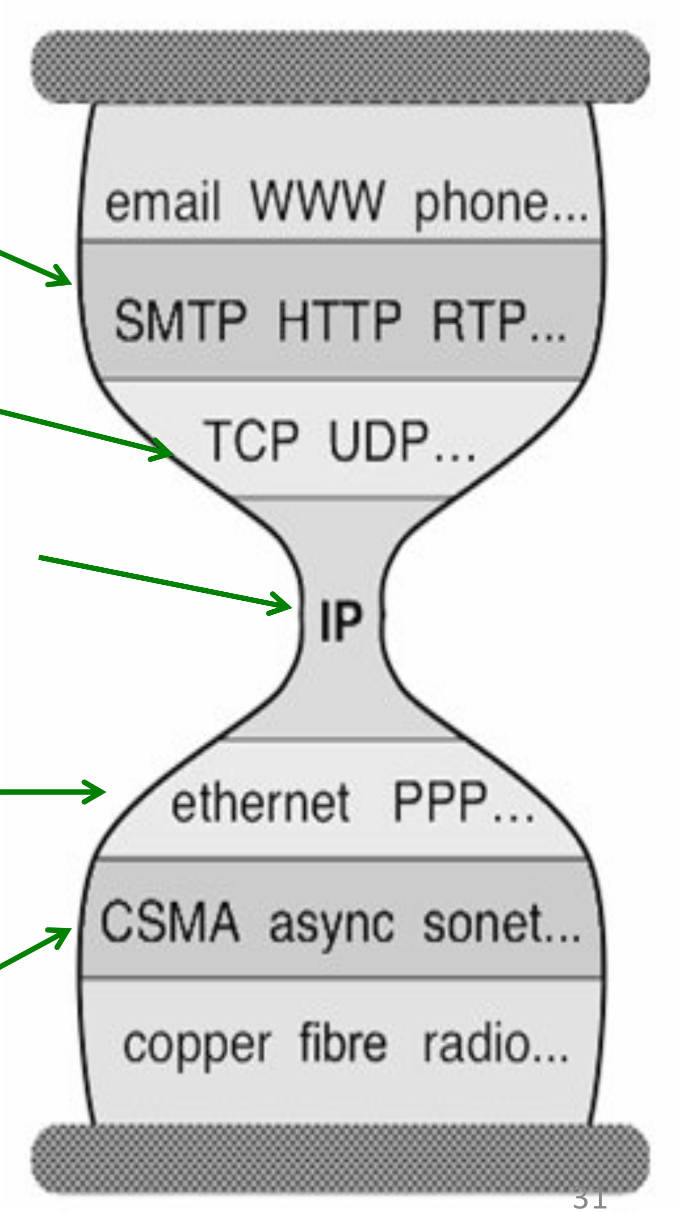
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

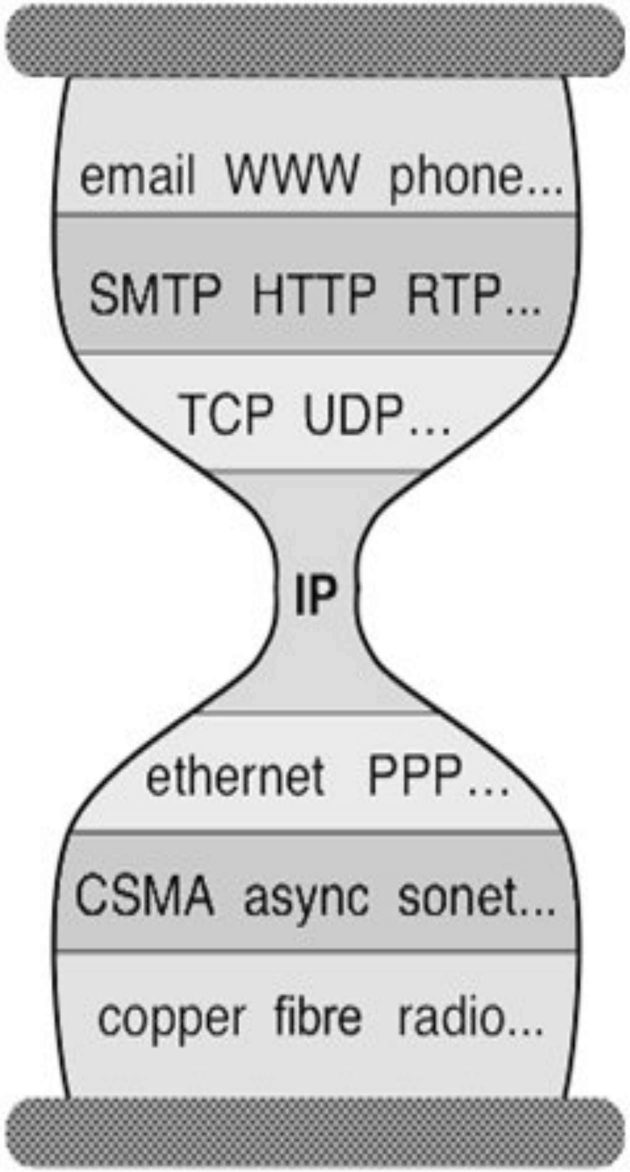
…built on…

Physical transfer of bits



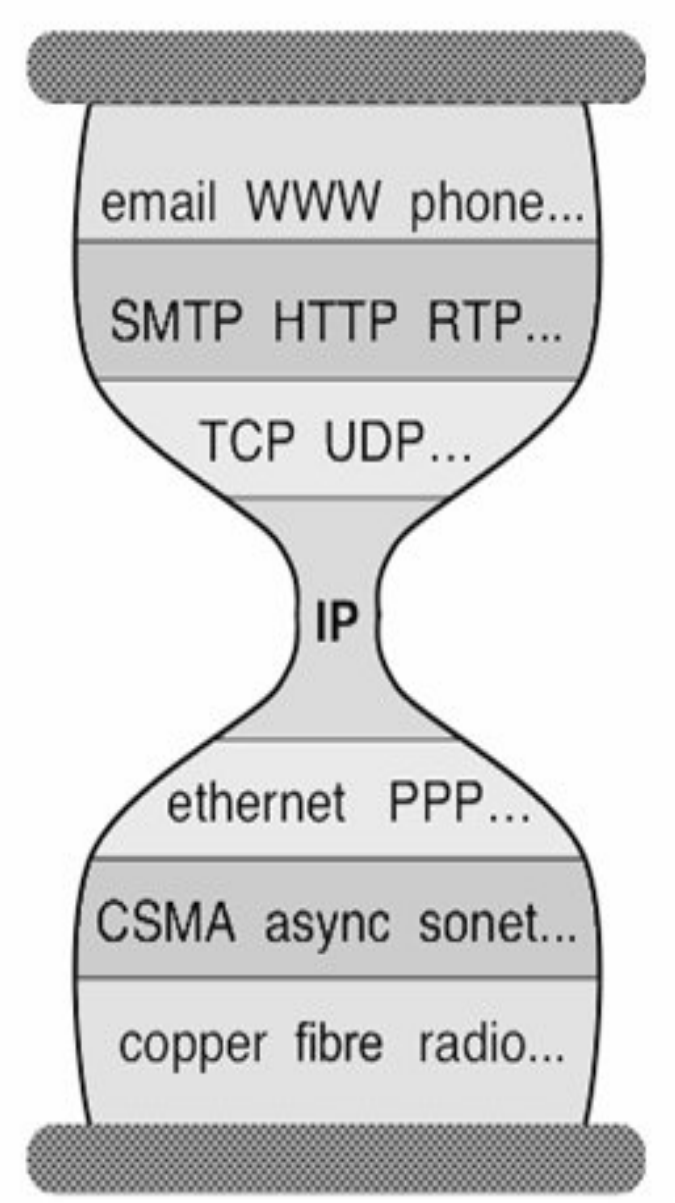
Three Observations

- Each layer:
  - Depends on layer below
  - Supports layer above
  - Independent of others
- Multiple versions in layer
  - Interfaces differ somewhat
  - Components pick which lower-level protocol to use
- But only one IP layer
  - Unifying protocol



Layering Crucial to Internet's Success

- Reuse
- Hides underlying detail
- Innovation at each level can proceed in parallel
- Pursued by very different communities



What are some of the drawbacks of protocols and layering?

Drawbacks of Layering

- Layer N may duplicate lower layer functionality
  - e.g., error recovery to retransmit lost data
- Information hiding may hurt performance
  - e.g., packet loss due to corruption vs. congestion
- Headers start to get really big
  - e.g., typical TCP+IP+Ethernet is 54 bytes
- Layer violations when the gains too great to resist
  - e.g., TCP-over-wireless
- Layer violations when network doesn’t trust ends
  - e.g., firewalls

Placing Network Functionality

- Hugely influential paper: “End-to-End Arguments in System Design” by Saltzer, Reed, and Clark (‘84)
  - articulated as the “End-to-End Principle” (E2E)

- Endless debate over what it means

- Everyone cites it as supporting their position

  (regardless of the position!)

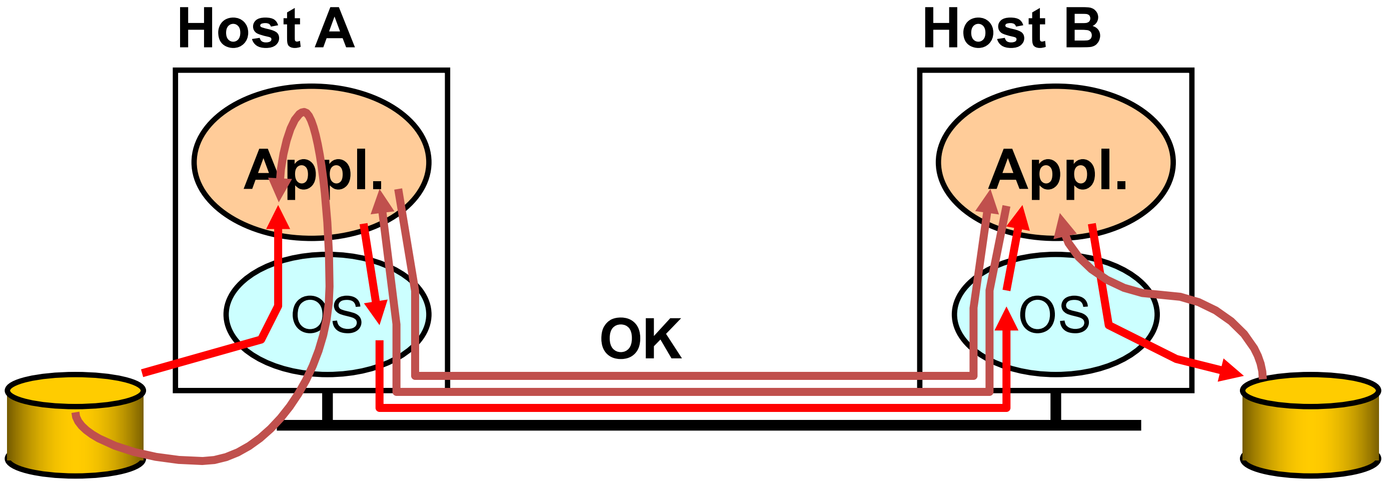
Basic Observation

- Some application requirements can only be correctly implemented **end-to-end**
  - reliability, security, *etc.*

- Implementing these in the network is hard
  - every step along the way must be fail proof

- Hosts
  - **Can** satisfy the requirement without network's help
  - **Will/must** do so, since they can't rely on the network

Example: Reliable File Transfer



- Solution 1: make each step reliable, and string them together to make reliable end-to-end process
- Solution 2: end-to-end **check** and retry

Discussion

- Solution 1 is incomplete
  - What happens if any network element misbehaves?
  - Receiver has to do the check anyway!

- Solution 2 is complete
  - Full functionality can be entirely implemented at application layer with no need for reliability from lower layers

- Is there any need to implement reliability at lower layers?

Summary of End-to-End Principle

- Implementing functionality (e.g., reliability) in the network
  - Doesn't reduce host implementation complexity
  - Does increase network complexity
  - Probably increases delay and overhead on all applications even if they don't need the functionality (e.g. VoIP)

- However, implementing in the network can improve performance in some cases
  - e.g., consider a very lossy link

“Only-if-Sufficient” Interpretation

- Don’t implement a function at the lower levels of the system unless it can be completely implemented at this level
- *Unless you can relieve the burden from hosts, don’t bother*

“Only-if-Necessary” Interpretation

- Don’t implement *anything* in the network that can be implemented correctly by the hosts
- Make network layer absolutely minimal
  - This E2E interpretation trumps performance issues
  - Increases flexibility, since lower layers stay **simple**

“Only-if-Useful” Interpretation

- If hosts can implement functionality correctly, implement it in a lower layer only as a performance enhancement
- But do so only if it does not impose burden on applications that do not require that functionality

We have some tools:

- Abstraction
- Layering
- Layers and Communications
- Entities and Peers
- Protocol as motivation
- Examples of the architects process
- Internet Philosophy and Tensions

Topic 3: The Data Link Layer

Our goals:

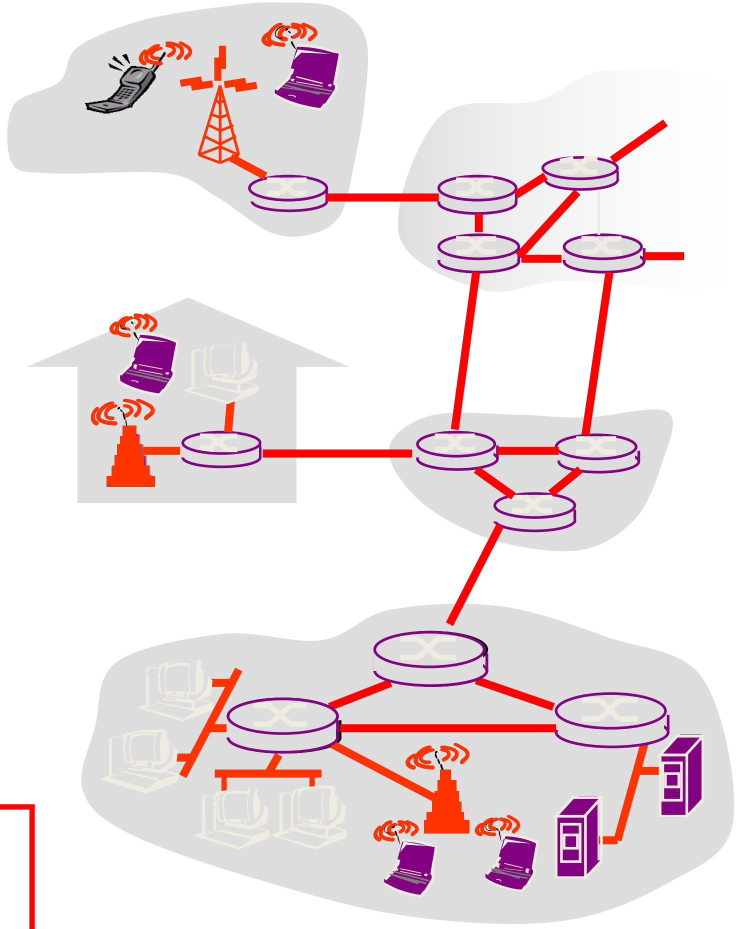
- understand principles behind data link layer services:

  (these are methods & mechanisms in your networking toolbox)
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
  - reliable data transfer, flow control:
- instantiation and implementation of various link layer technologies
  - Wired Ethernet (aka 802.3)
  - Wireless Ethernet (aka 802.11 WiFi)
- Algorithms
  - Binary Exponential Backoff
  - Spanning Tree

Link Layer: Introduction

Some terminology:

- hosts and routers are **nodes**
- communication channels that connect adjacent nodes along communication path are **links**
  - wired links
  - wireless links
  - LANs
- layer-2 packet is a **frame**, encapsulates datagram



data-link layer has responsibility of transferring datagram from one node to adjacent node over a link

Link Layer (Channel) Services

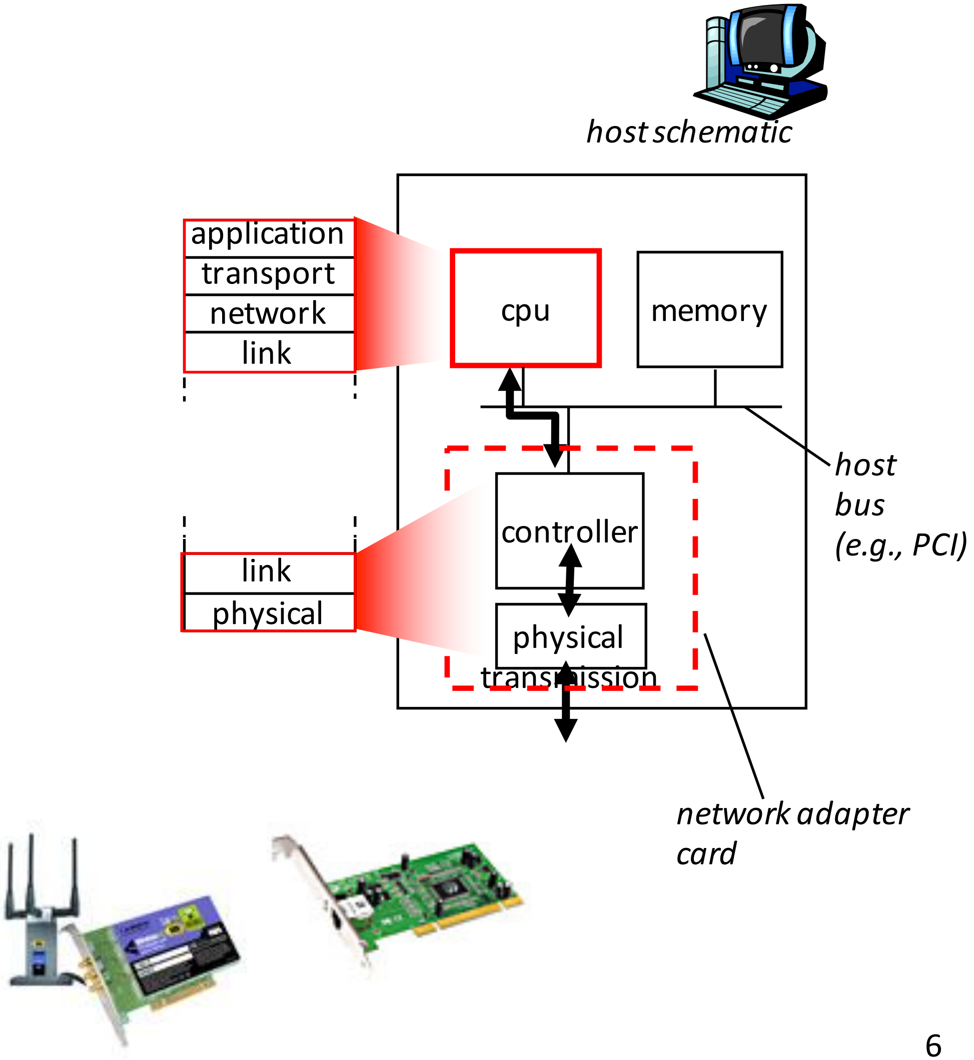
- *framing, link access:*
  - encapsulate datagram into frame, adding header, trailer
  - channel access if shared medium
  - "MAC" addresses used in frame headers to identify source, dest
    - different from IP address!
- *reliable delivery between adjacent nodes*
  - we see some of this again in the Transport Topic
  - seldom used on low bit-error link (fiber, some twisted pair)
  - wireless links: high error rates
    - Q: why both link-level and end-end reliability?

Link Layer (Channel) Services - 2

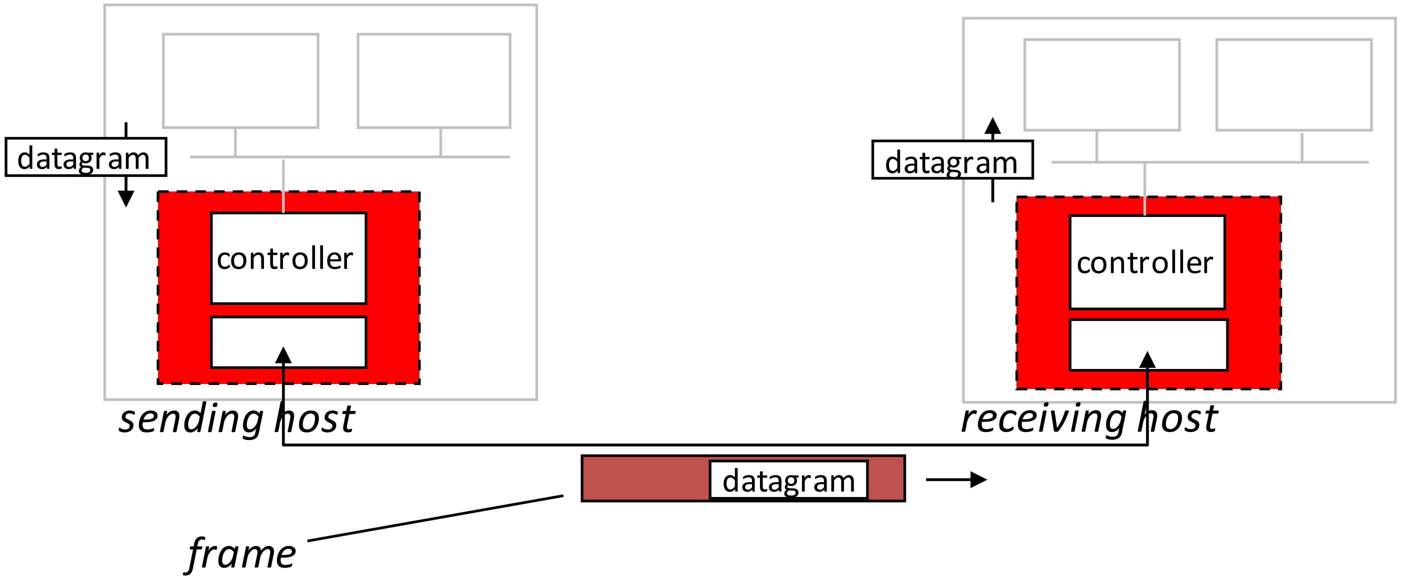
- *flow control:*
  - pacing between adjacent sending and receiving nodes
- *error detection*:
  - errors caused by signal attenuation, noise.
  - receiver detects presence of errors:
    - signals sender for retransmission or drops frame
- error correction:
  - receiver identifies *and corrects* bit error(s) without resorting to retransmission
- *half-duplex and full-duplex*
  - with half duplex, nodes at both ends of link can transmit, but not at same time

Where is the link layer implemented?

- in each and every host
- link layer implemented in "adaptor" (aka *network interface card* NIC)
  - Ethernet card, PCMCI card, 802.11 card
  - implements link, physical layer
- attaches into host's system buses
- combination of hardware, software, firmware



Adaptors Communicating

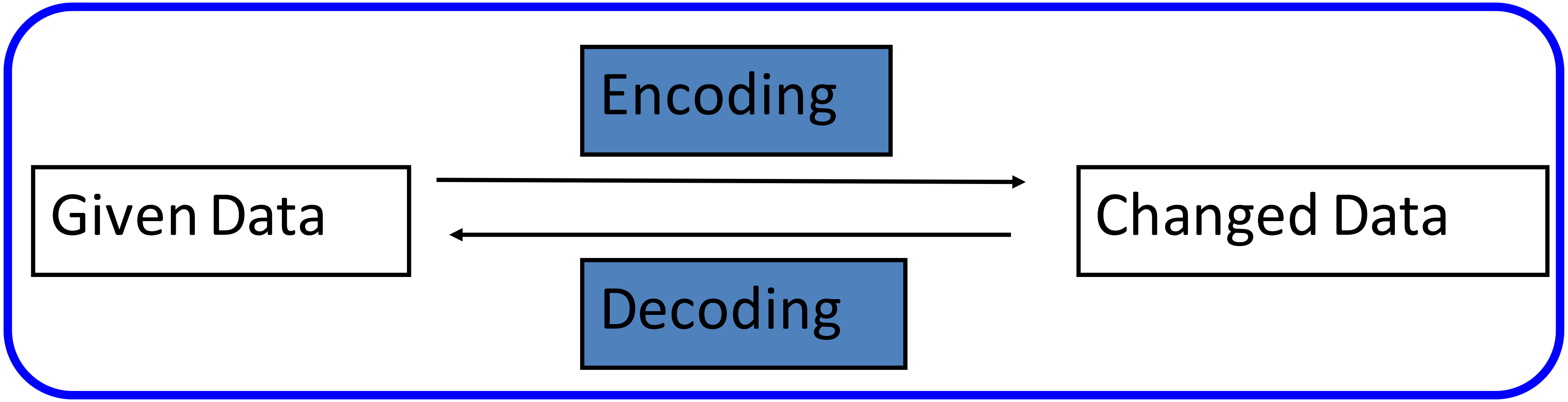


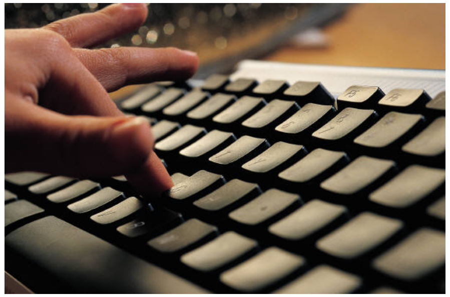
- sending side:
  - encapsulates datagram in frame
  - encodes data for the physical layer
  - adds error checking bits, provide reliability, flow control, etc.

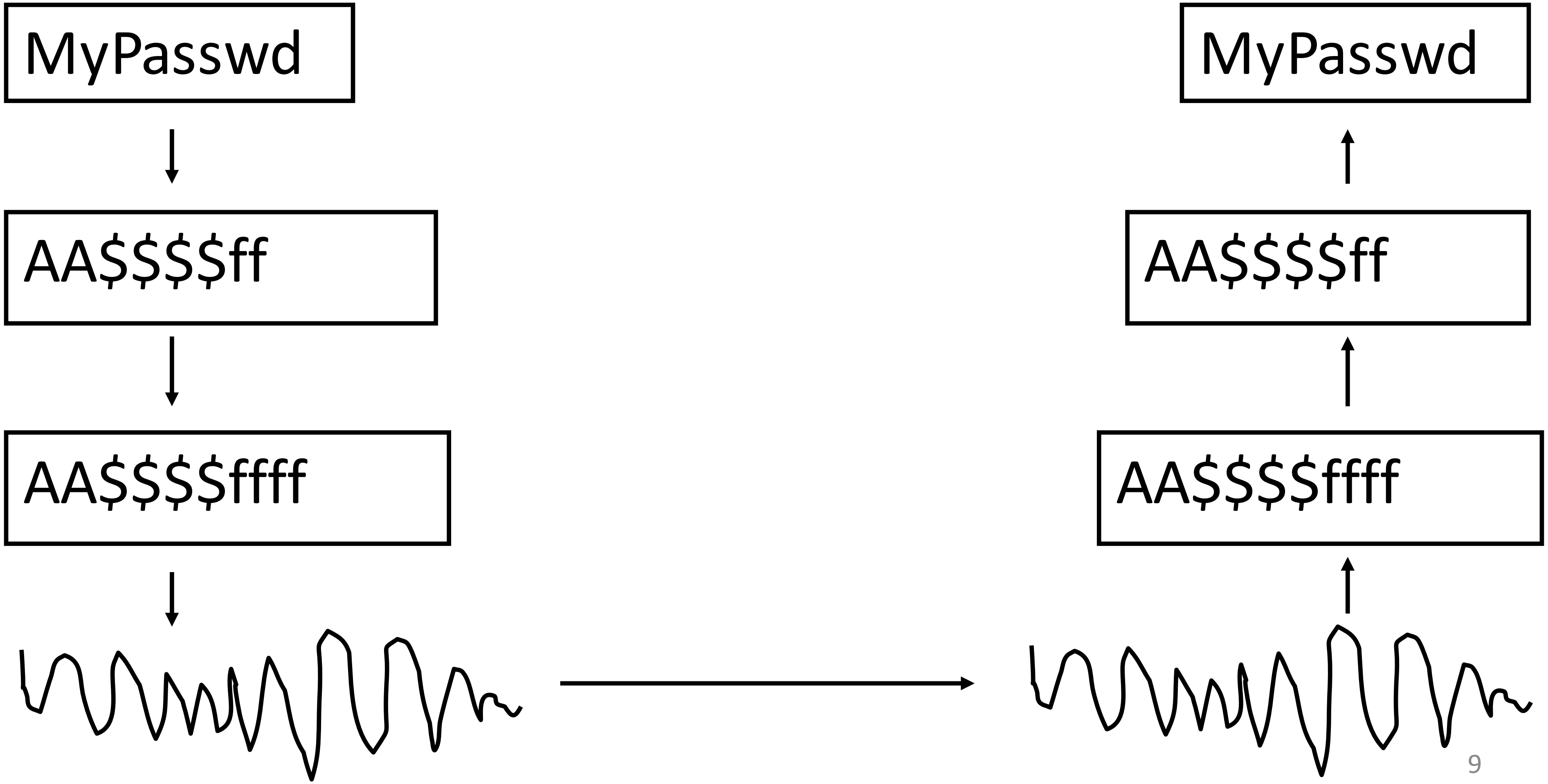
- receiving side
  - decodes data from the physical layer
  - looks for errors, provide reliability, flow control, etc
  - extracts datagram, passes to upper layer at receiving side

Coding – a channel function

Change the representation of data.







MyPasswd

AA$$$$ff

AA$$$$ffff

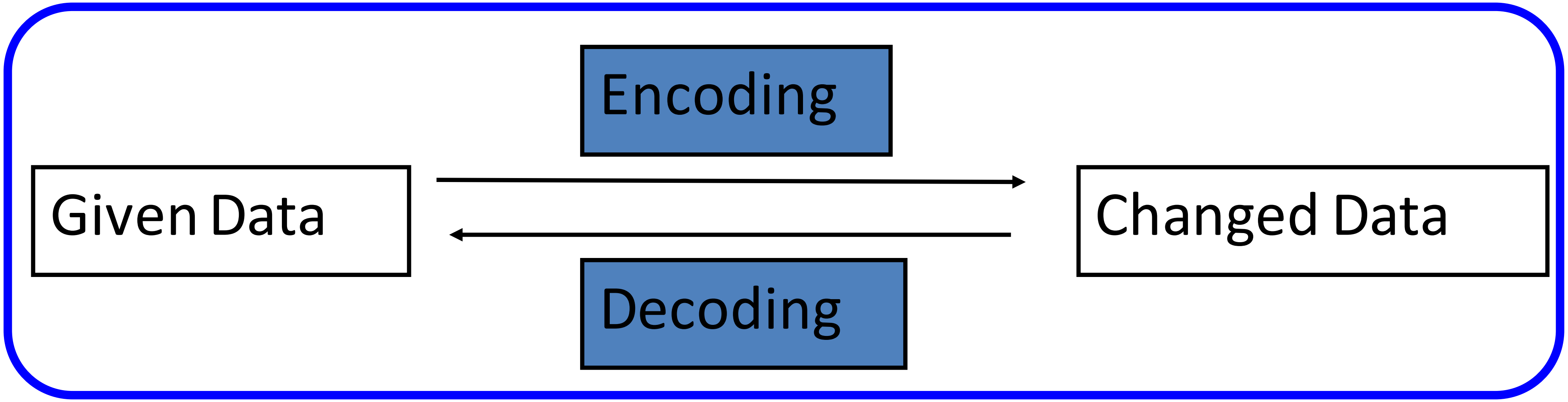
MyPasswd

AA$$$$ff

AA$$$$ffff

Coding

Change the representation of data.



Encoding

Given Data

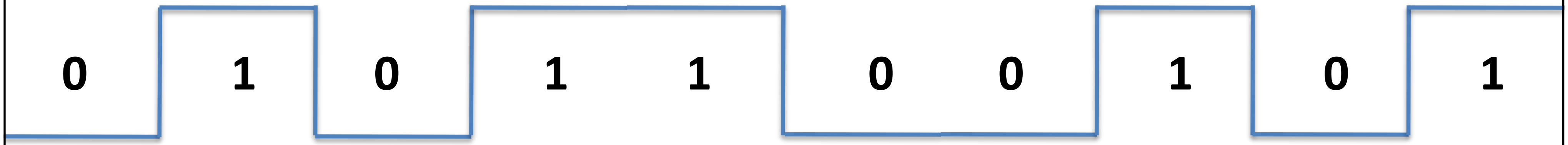
Changed Data

Decoding

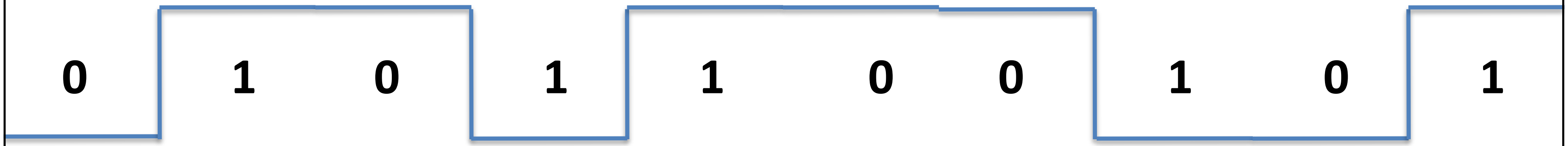
1. Encryption: MyPasswd <-> AA$$$$ff
2. Error Detection: AA$$$$ff <-> AA$$$$ffff
3. Compression: AA$$$$ffff <-> A2$4f4
4. Analog: A2$4f4 <->

Line Coding Examples
where Baud=bit-rate

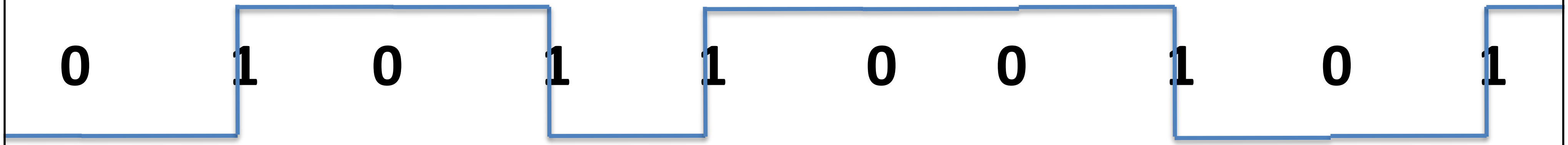
Non-Return-to-Zero (NRZ)



Non-Return-to-Zero-Mark (NRZM) 1 = transition 0 = no transition

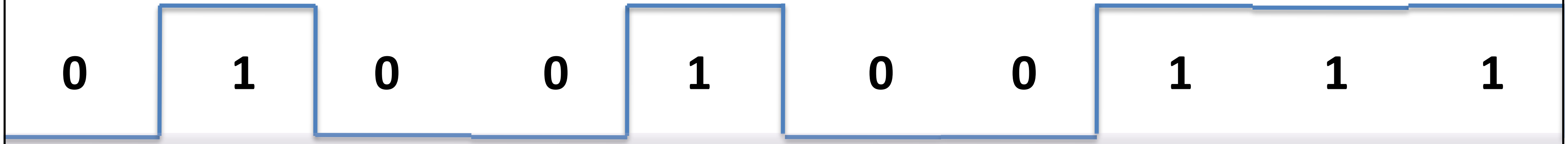


Non-Return-to-Zero Inverted (NRZI) (note transitions on the 1)

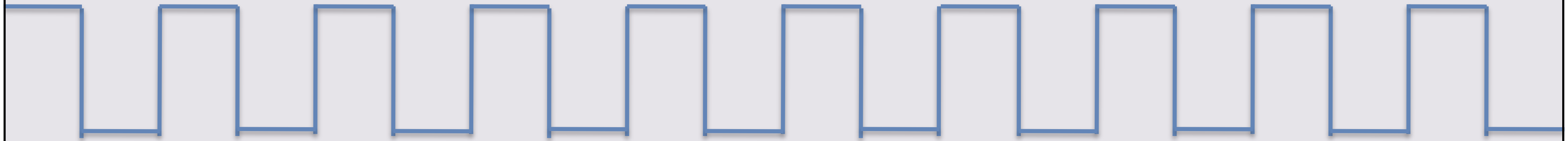


Line Coding Examples

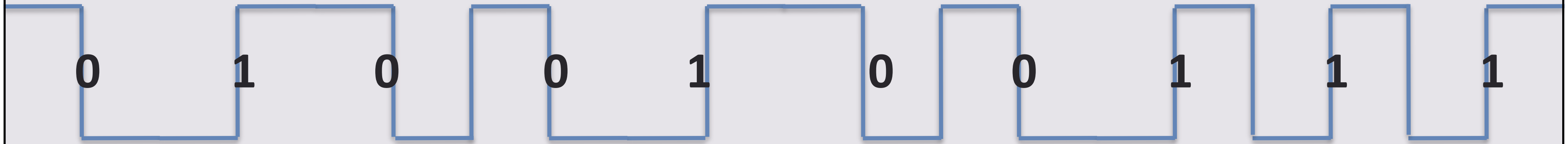
Non-Return-to-Zero (NRZ) (Baud = bit-rate)



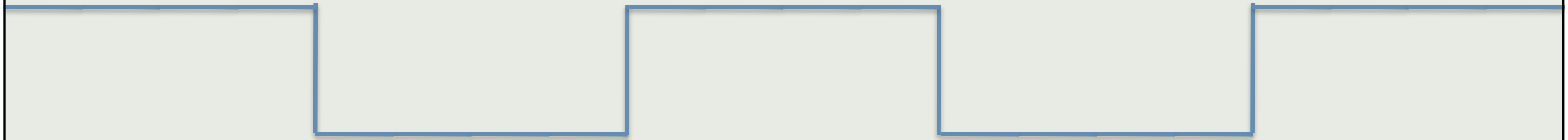
Clock



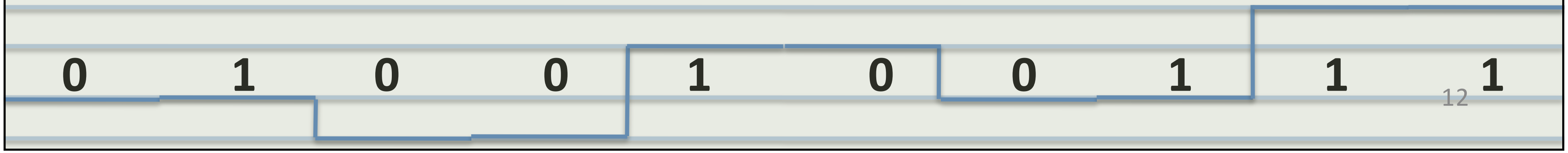
Manchester example (Baud = 2 x bit-rate)



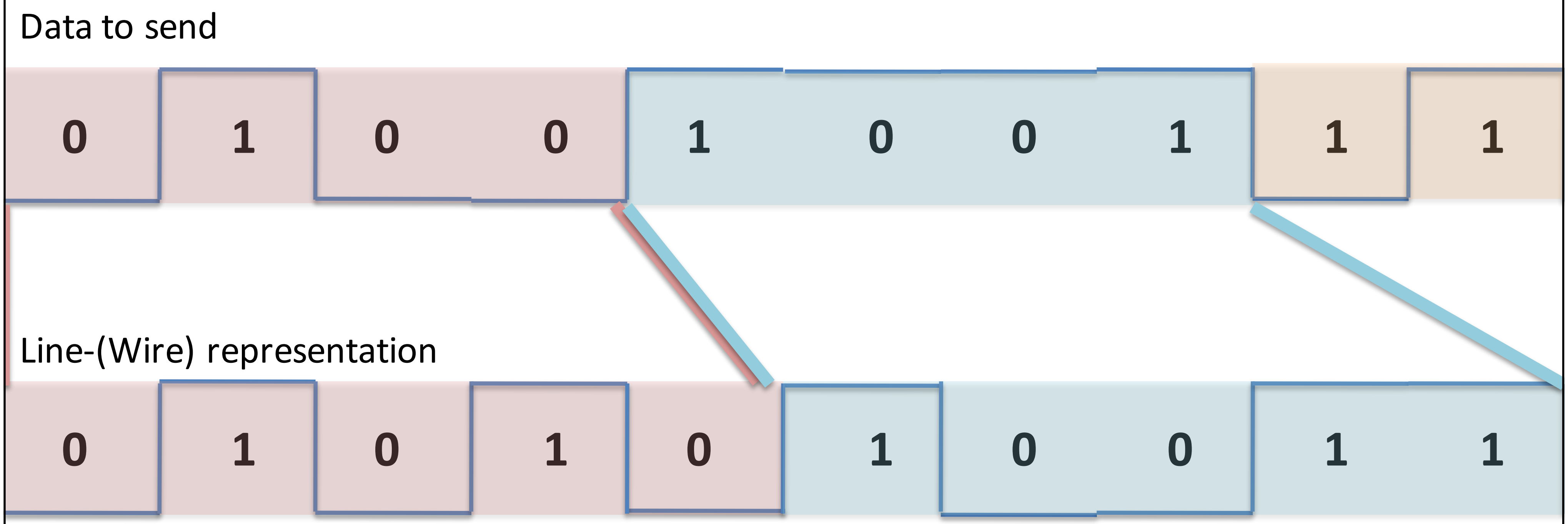
Clock



Quad-level code (2 x Baud = bit-rate)



Line Coding – Block Code example



Name	4b	5b	Description
0	0000	11110	hex data 0
1	0001	01001	hex data 1
2	0010	10100	hex data 2
3	0011	10101	hex data 3
4	0100	01010	hex data 4
5	0101	01011	hex data 5
6	0110	01110	hex data 6
7	0111	01111	hex data 7
8		1000	10010 hex data 8
9		1001	10011 hex data 9
A	1010	10110	hex data A
B	1011	10111	hex data B
C	1100	11010	hex data C
D	1101	11011	hex data D
E	1110	11100	hex data E
F	1111	11101	hex data F

Name	4b	5b	Description
Q	-NONE-	00000	Quiet
I	-NONE-	11111	Idle
J	-NONE-	11000	SSD #1
K	-NONE-	10001	SSD #2
T	-NONE-	01101	ESD #1
R	-NONE-	00111	ESD #2
H	-NONE-	00100	Halt

Block coding transfers data with a fixed overhead: 20% less information per Baud in the case of 4B/5B

So to send data at 100Mbps; the line rate (the Baud rate) must be 125Mbps.

1Gbps uses an 8b/10b codec; encoding entire bytes at a time but with 25% overhead

Line Coding Scrambling – with secrecy

Step 1

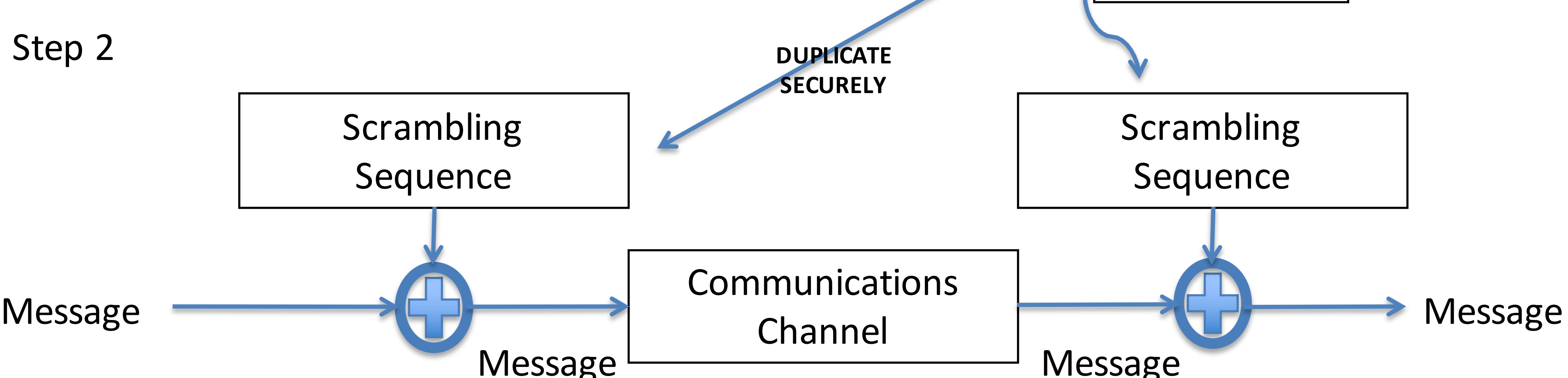
….G8wDFrB
EAFDSWbzQ7
BW2fbdTqeT
ImrukTYwQY
ndYdKb4….

**REPLICAT
E
SECURELY**

Scrambling Sequence

Scrambling Sequence

Step 2



**DUPLICATE
SECURELY**

Scrambling Sequence

Scrambling Sequence

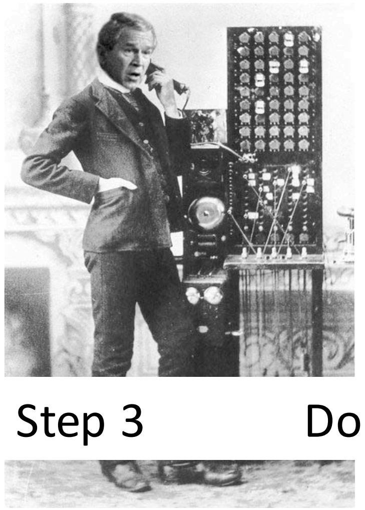
Message

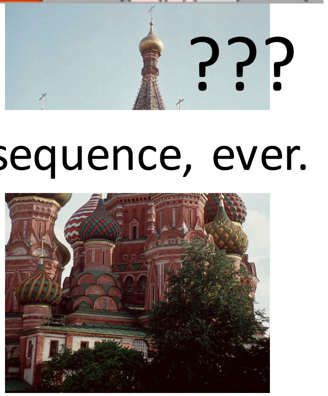
Communications Channel

Message

Message XOR Sequence

Message XOR Sequence

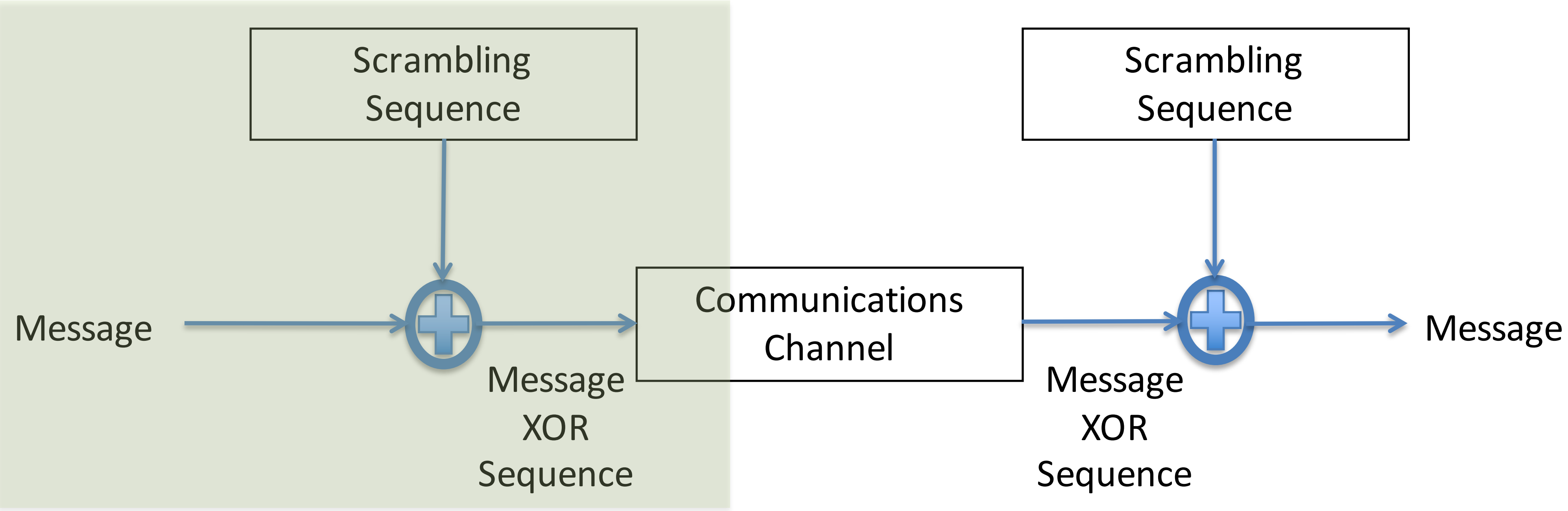




???

Step 3 Don't ever reuse Scrambling sequence, ever. <<< **this is quite important**

Line Coding Scrambling– no secrecy



Scrambling Sequence

Message

Message XOR Sequence

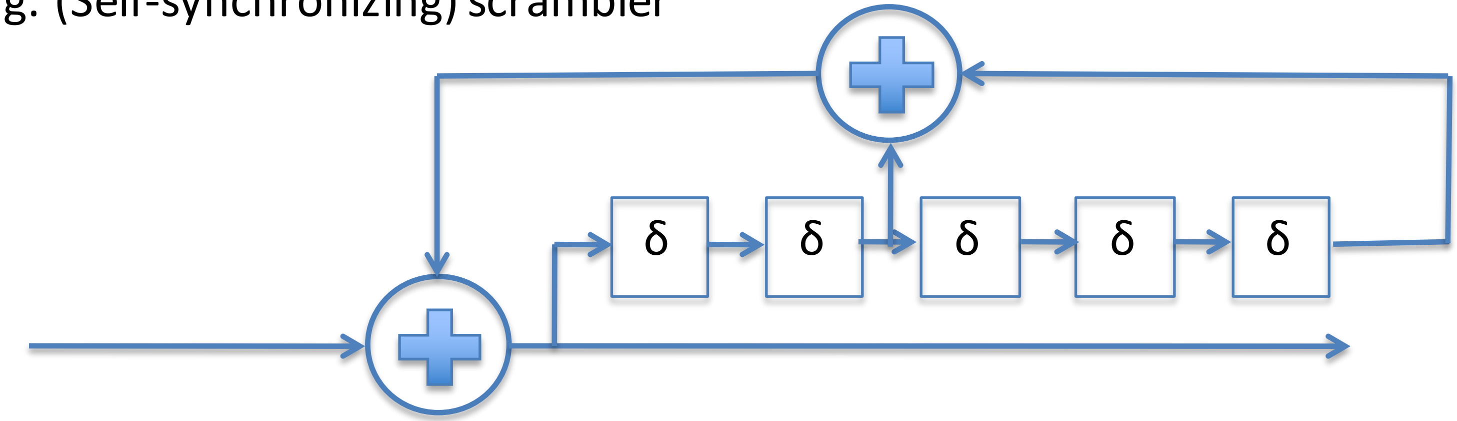
Communications Channel

Scrambling Sequence

Message XOR Sequence

Message

e.g. (Self-synchronizing) scrambler



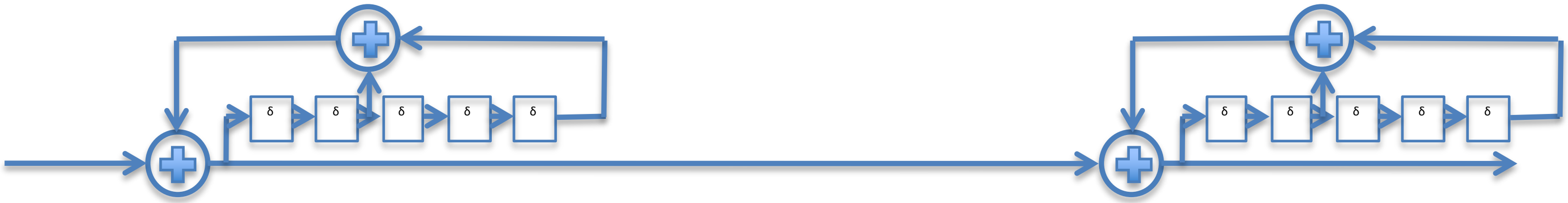
Line Coding Examples (Hybrid)

…100111101101010001000101100111010001010010110101001001110101110100…

…10011110110101000101000101100111010001010010110101001001110101110100…

Inserted bits marking “start of frame/block/sequence”

Scramble / Transmit / Unscramble

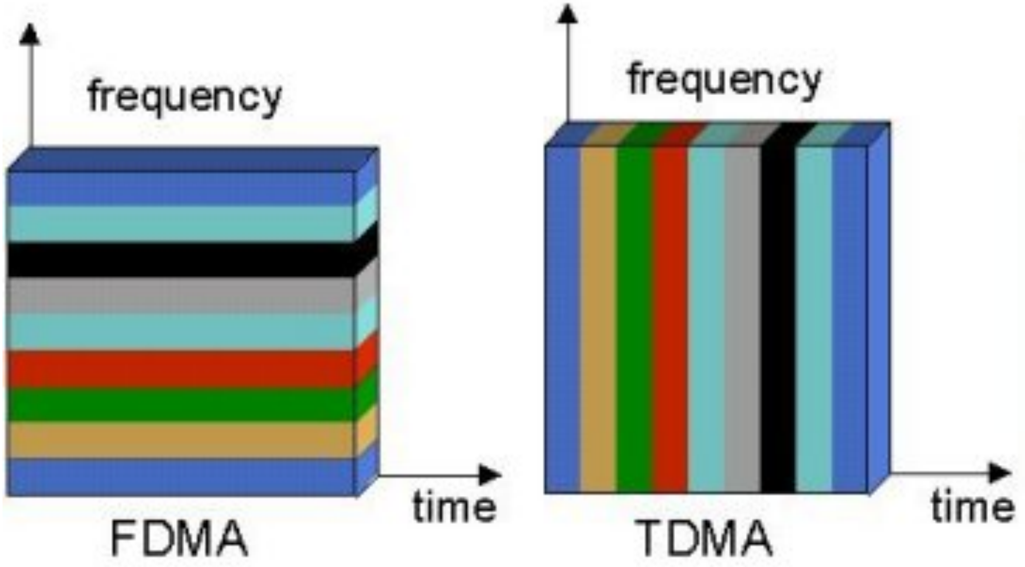


…0100010110011101000101001011010100100111010111010010010111011101111000…

Identify (and remove) “start of frame/block/sequence”
This gives you the Byte-delineations for *free*

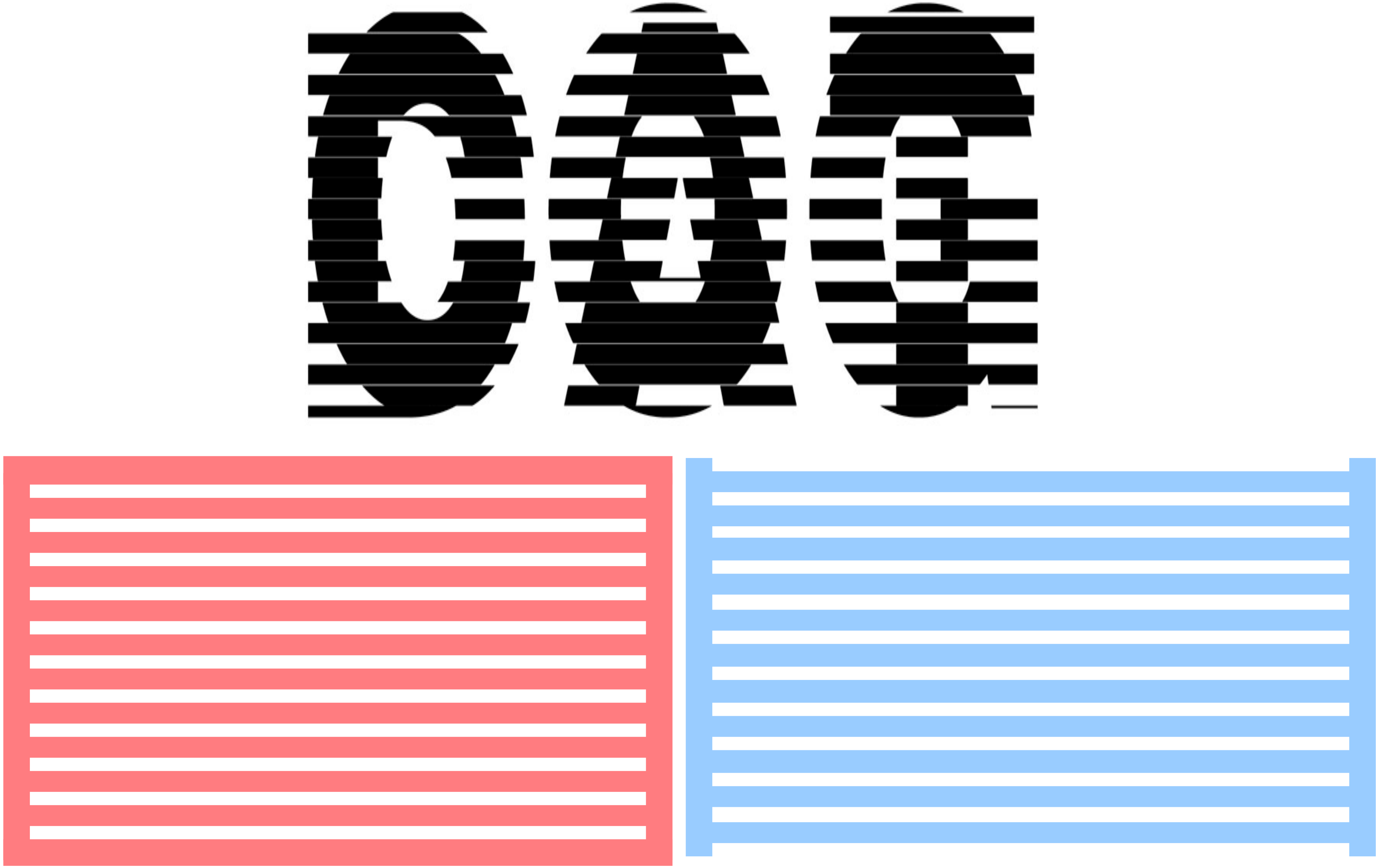
64b/66b combines a scrambler and a framer. The start of frame is a pair of bits 01 or 10: 01 means “this frame is data” 10 means “this frame contains data and control” – control could be configuration information, length of encoded data or simply “this line is idle” (no data at all)

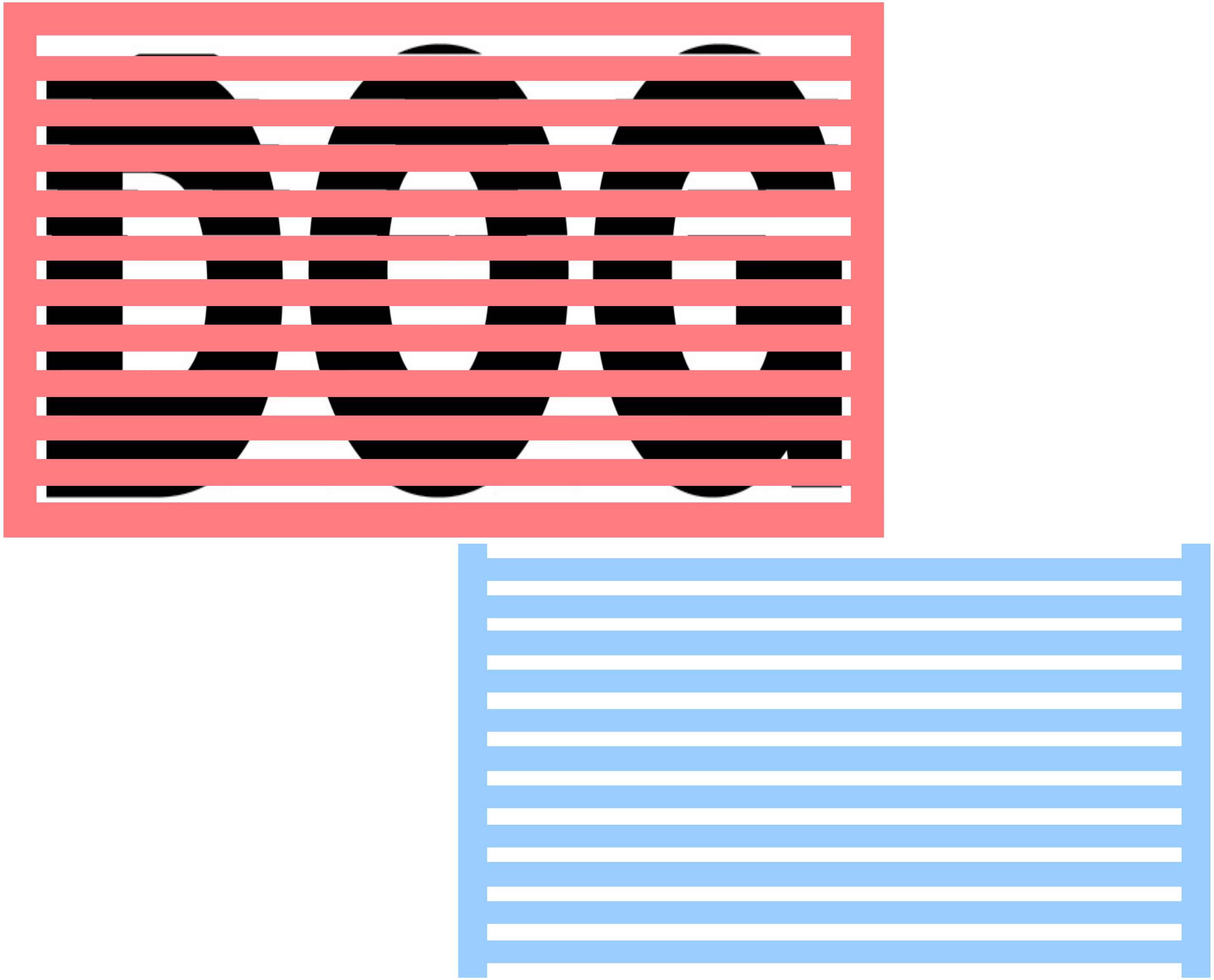
Multiple Access Mechanisms

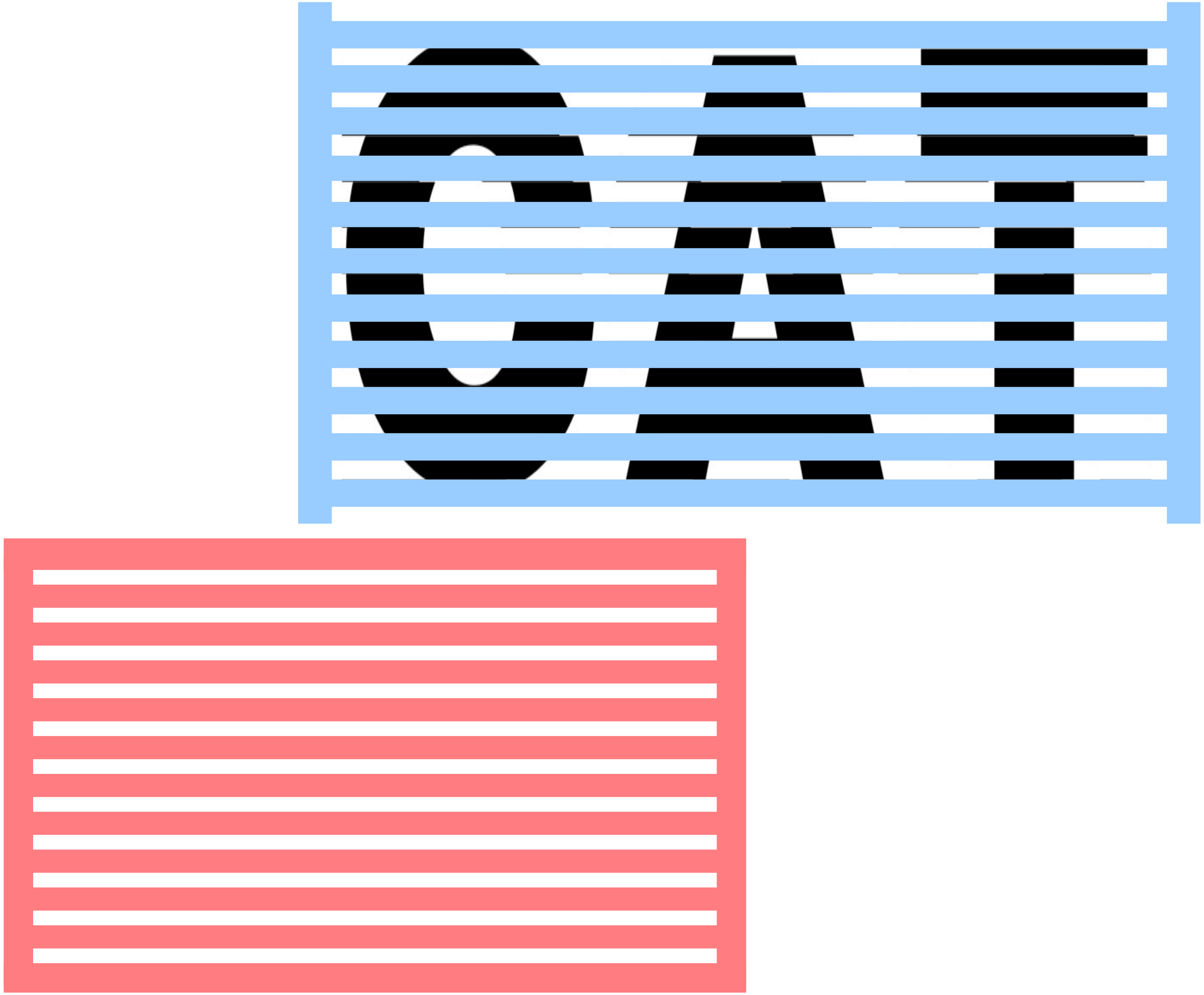


Each dimension is orthogonal (so may be trivially combined)

There are other dimensions too; can you think of them?




DOG

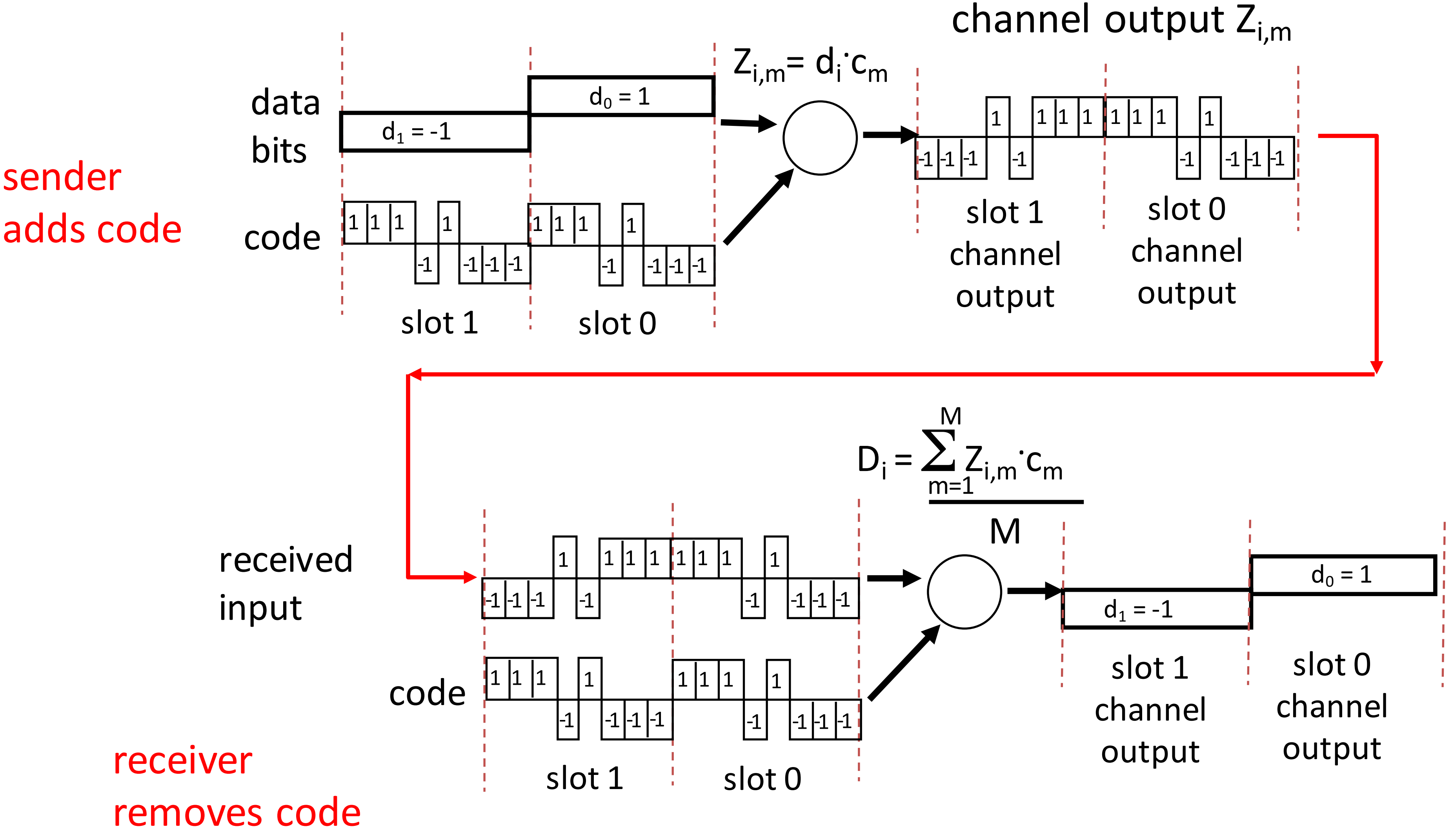


CAT

Code Division Multiple Access (CDMA) (not to be confused with CSMA!)

- used in several wireless broadcast channels (cellular, satellite, etc) standards
- unique "code" assigned to each user; i.e., code set partitioning
- all users share same frequency, but each user has own "chipping" sequence (i.e., code) to encode data
- *encoded signal* = (original data) XOR (chipping sequence)
- *decoding:* inner-product of encoded signal and chipping sequence
- allows multiple users to "coexist" and transmit simultaneously with minimal interference (if codes are "orthogonal")

CDMA Encode/Decode



sender adds code

data bits

$d_1 = -1$

$d_0 = 1$

code

slot 1

slot 0

$Z_{i,m} = d_i \cdot c_m$

channel output $Z_{i,m}$

slot 1 channel output

slot 0 channel output

receiver removes code

received input

code

slot 1

slot 0

$$D_i = \frac{\sum_{m=1}^{M} Z_{i,m} \cdot c_m}{M}$$

$d_1 = -1$

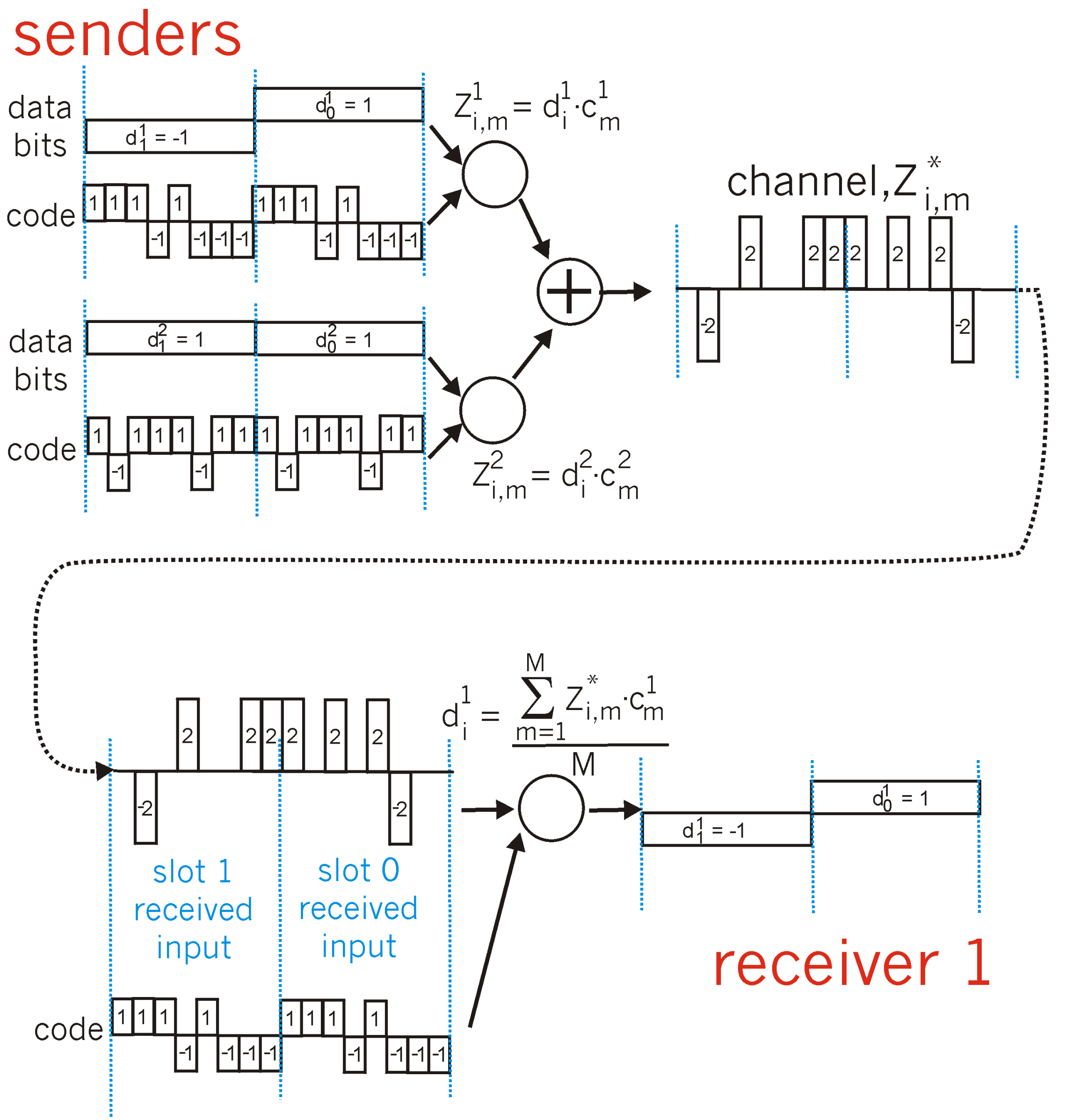
$d_0 = 1$

slot 1 channel output

slot 0 channel output

CDMA: two-sender interference

Each sender adds a ***unique*** code



$$d^1_i = \frac{\sum_{m=1}^{M} Z^*_{i,m} \cdot c^1_m}{M}$$

sender removes its ***unique*** code

Coding Examples summary

- Common Wired coding
  - Block codecs: table-lookups
    - fixed overhead, inline control signals
  - Scramblers: shift registers
    - overhead free

Like earlier coding schemes and error correction/detection; you can combine these

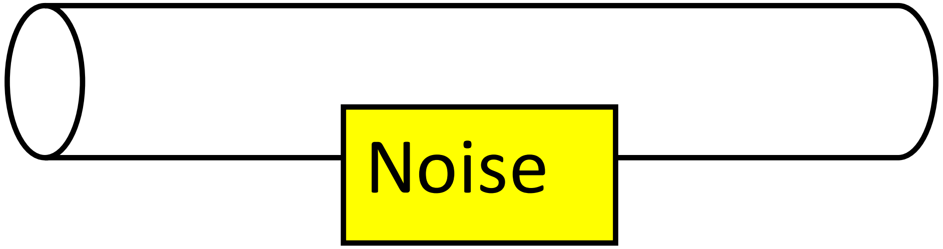
- e.g, 10Gb/s Ethernet may use a hybrid

CDMA (Code Division Multiple Access)

- coping intelligently with competing sources
- Mobile phones

Error Detection and Correction

How to use coding to deal with errors in data communication?



0000 0000

0001 0000

Basic Idea :

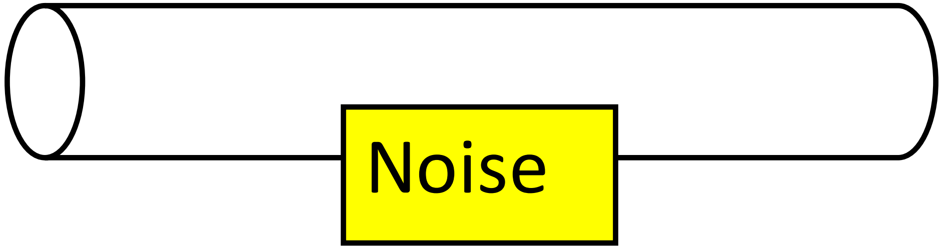
1. Add additional information to a message.
2. Detect an error and re-send a message.

   Or, fix an error in the received message.

Error Detection and Correction

How to use coding to deal with errors in data communication?

0000	0000



0000	0000

Basic Idea :

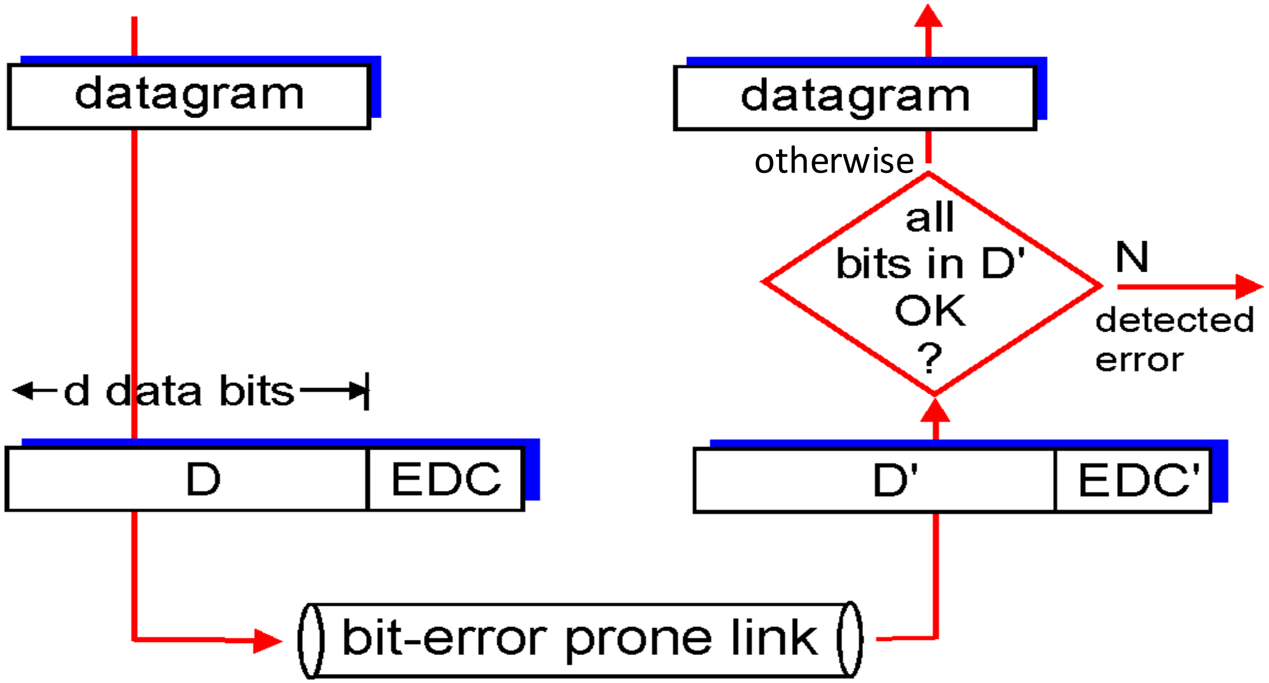
1. Add additional information to a message.
2. Detect an error and re-send a message.

   Or, fix an error in the received message.

Error Detection

EDC= Error Detection and Correction bits (redundancy = overhead)
D = Data protected by error checking, may include header fields

- Error detection not 100% reliable!
  - protocol may miss some errors, but rarely
  - larger EDC field yields better detection and correction



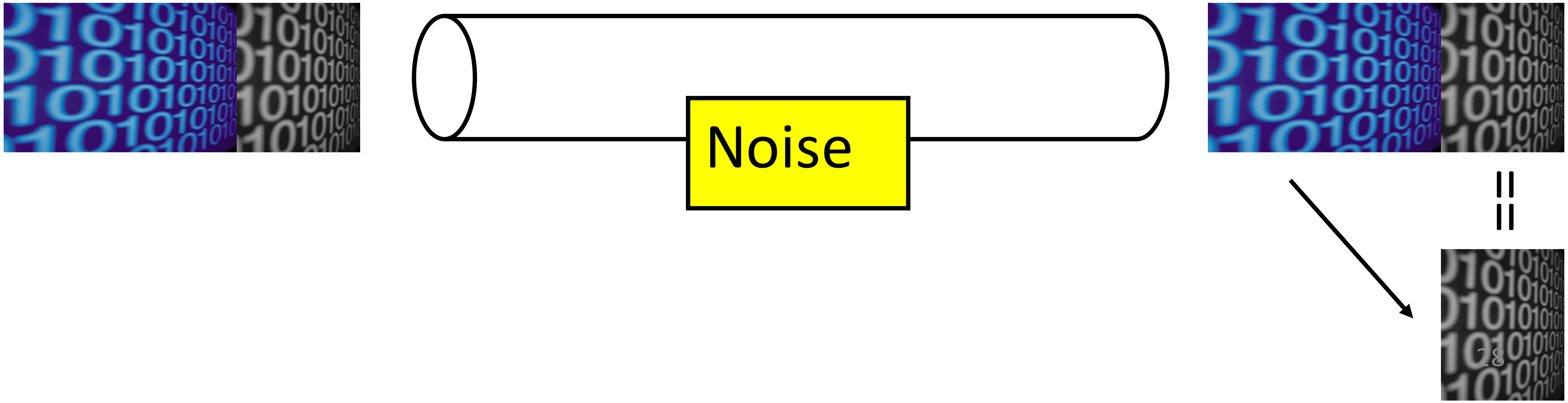
Error Detection Code

Sender:

```
Y = generateCheckBit(X);
send(XY);
```

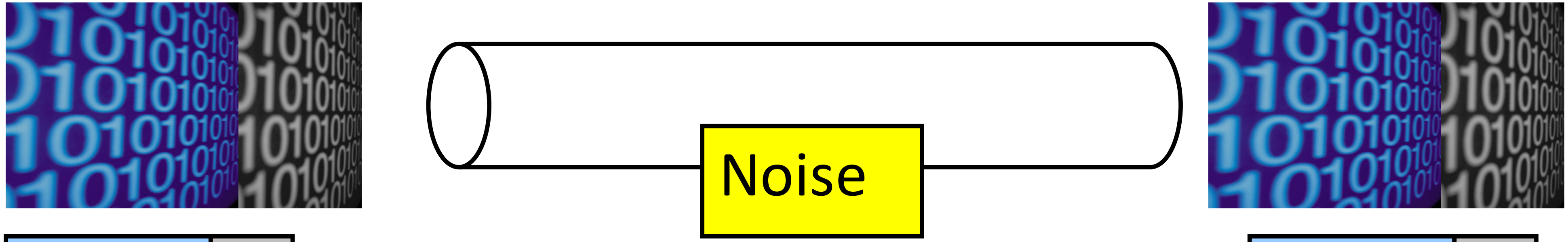
Receiver:

```
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) ERROR;
else NOERROR
```



Error Detection Code: Parity

Add one bit, such that the number of 1’s is even.



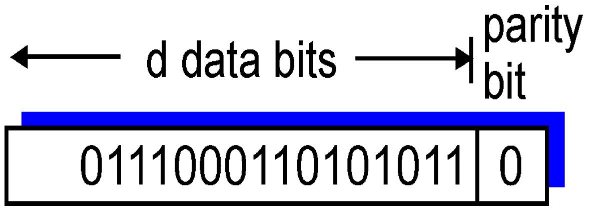
0000	0	✗	0001	0
0001	1	✓	0001	1
1001	0	✓	1111	0

Problem: This simple parity cannot detect two-bit errors.

Parity Checking

Single Bit Parity:

Detect single bit errors



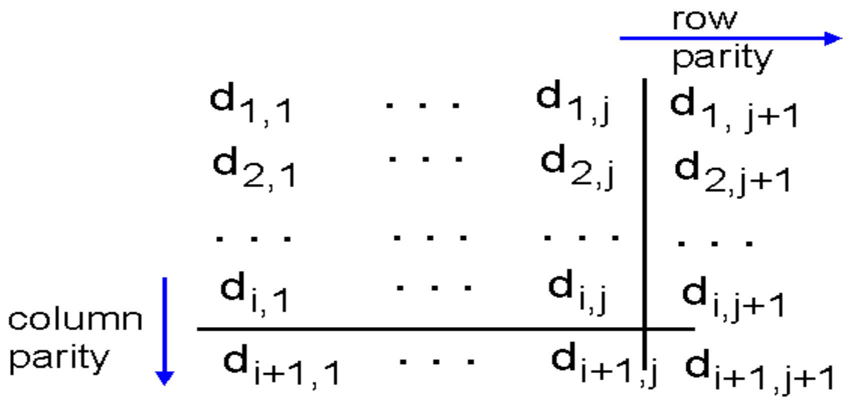
d data bits

parity bit

0111000110101011 | 0

Two Dimensional Bit Parity:

Detect *and correct* single bit errors



row parity

column parity

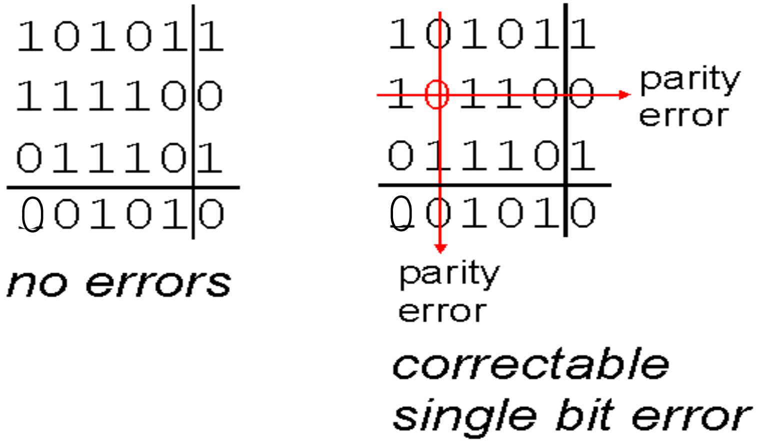
$d_{1,1}$ . . . $d_{1,j}$ | $d_{1,\ j+1}$

$d_{2,1}$ . . . $d_{2,j}$ | $d_{2,j+1}$

. . . . | . . .

$d_{i,1}$ . . . $d_{i,j}$ | $d_{i,j+1}$

$d_{i+1,1}$ . . . $d_{i+1,j}$ | $d_{i+1,j+1}$



```
10101|1        10101|1
11110|0        10110|0   parity error
01110|1        01110|1
------         ------
00101|0        00101|0
```

no errors

parity error

correctable single bit error

Internet checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted packet (note: used at transport layer only)

Sender:

- treat segment contents as sequence of 1bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?*

Error Detection Code: CRC

- CRC means “Cyclic Redundancy Check”.
- More powerful than parity.
  - It can detect various kinds of errors, including 2-bit errors.
- More complex: multiplication, binary division.
- Parameterized by n-bit divisor P.
  - Example: 3-bit divisor 101.
  - Choosing good P is crucial.

CRC with 3-bit Divisor 101

1111

1001

00

11

CRC

0

0

Parity

111

100

same check bits from Parity,

but different ones from CRC

Multiplication by 2^3

$D2 = D * 2^3$

Add three 0’ s at the end

Binary Division by 101

CheckBit = (D2) rem (101)

Kurose p478 §5.2.3
Peterson p97 §2.4.3

The divisor (P) – Secret sauce of CRC

- If the divisor were 100, instead of 101, data 1111 and 1001 would give the same check bit 00.
- Mathematical analysis about the divisor:
  - Last bit should be 1.
  - Should contain at least two 1's.
  - Should be divisible by 11.
- ATM, HDLC, Ethernet each use a CRC with well-chosen fixed divisors

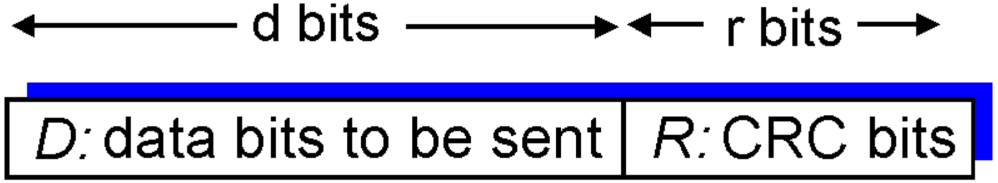
Divisor analysis keeps mathematicians in jobs

(a branch of *pure* math: combinatorial mathematics)

FYI: *in K&R P is called the Generator: G*

Checksumming: Cyclic Redundancy Check recap

- view data bits, D, as a binary number
- choose r+1 bit pattern (generator), P
- goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - can detect all burst errors less than r+1 bits
- widely used in practice (Ethernet, 802.11 WiFi, ATM)



bit pattern

$D * 2^r$ XOR R

mathematical formula

FYI: *in K&R P is called the Generator: G*

CRC Another Example – this time with long division

Want:

$D \cdot 2^r$ XOR R = nP

equivalently:

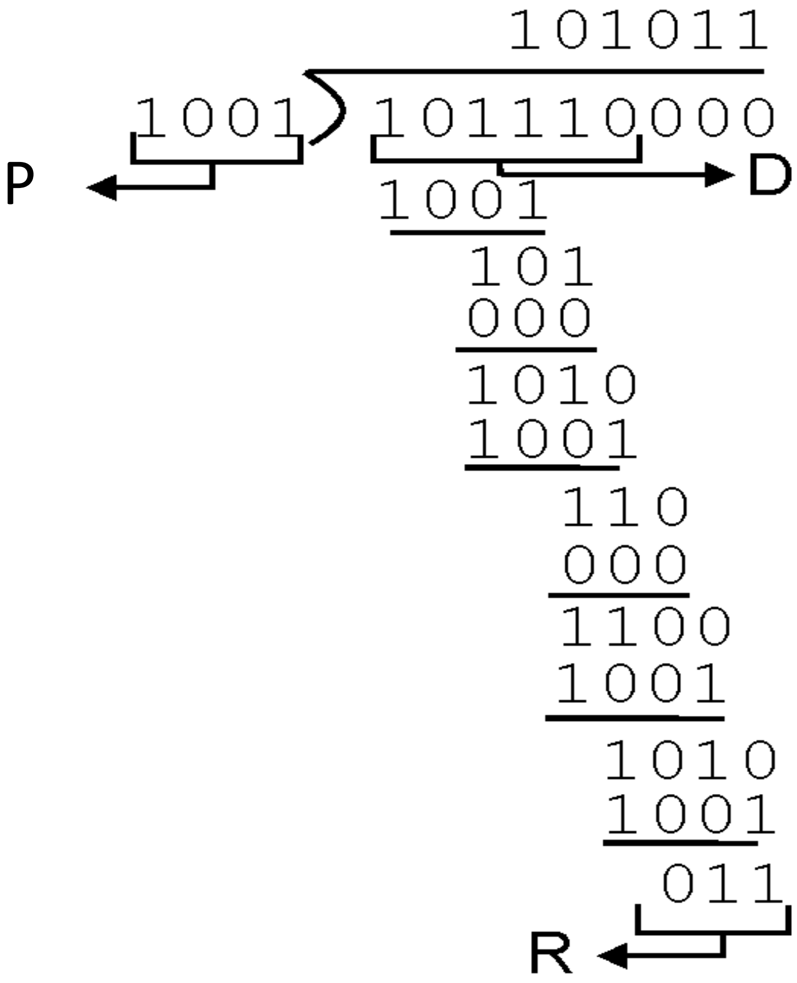
$D \cdot 2^r$ = nP XOR R

equivalently:

if we divide $D \cdot 2^r$ by P,

want remainder R

$$R = \text{remainder}\left[\frac{D \cdot 2^r}{P}\right]$$



```
               101011
      1001 ) 101110000
             1001
              101
              000
              1010
              1001
               110
               000
               1100
               1001
                1010
                1001
                 011
```

FYI: *in K&R P is called the Generator: G*

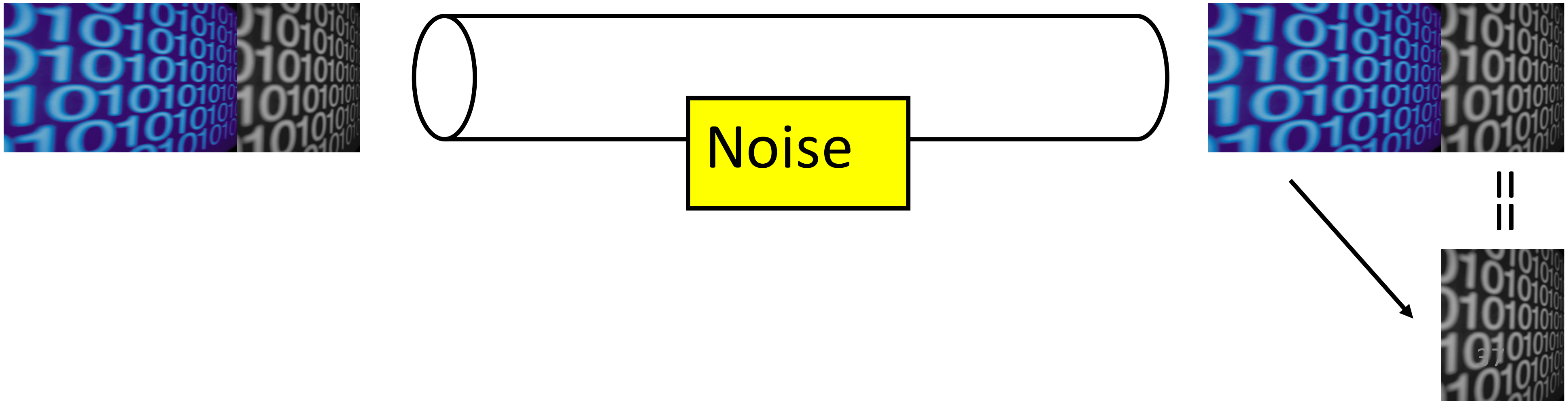
Error Detection Code becomes….

Sender:

```
Y = generateCheckBit(X);
send(XY);
```

Receiver:

```
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) ERROR;
else NOERROR
```



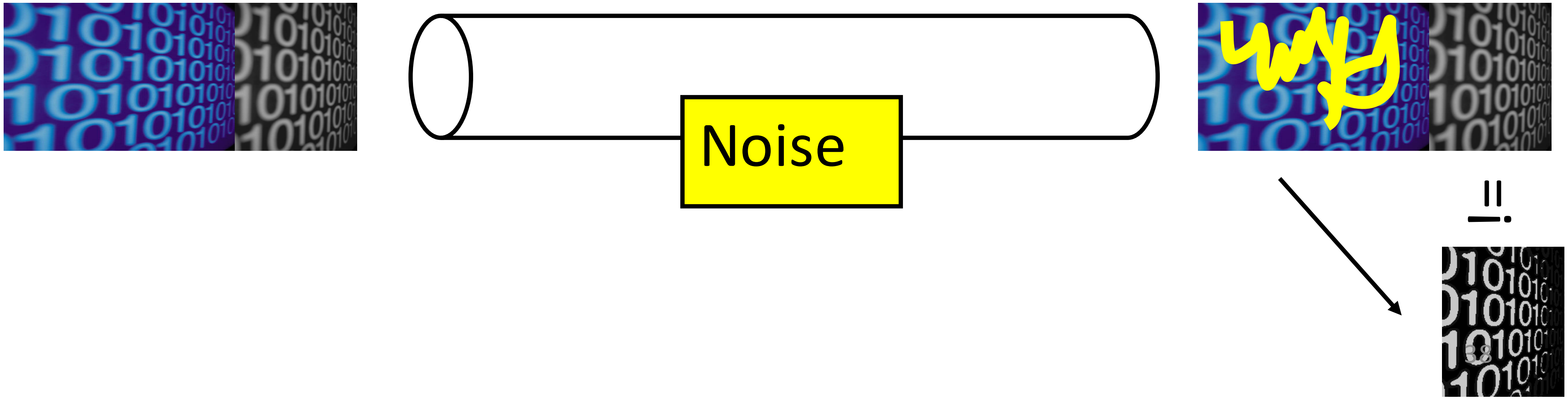
Forward Error Correction (FEC)

Sender:

```
Y = generateCheckBit(X);
send(XY);
```

Receiver:

```
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) FIXERROR(X1Y1);
else NOERROR
```



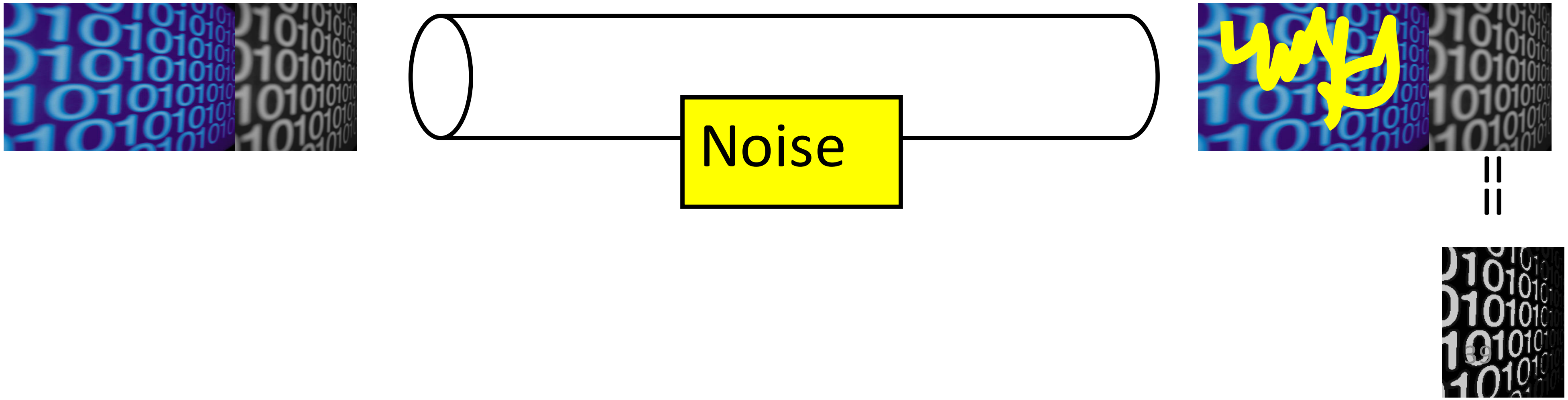
Forward Error Correction (FEC)

Sender:

```
Y = generateCheckBit(X);
send(XY);
```

Receiver:

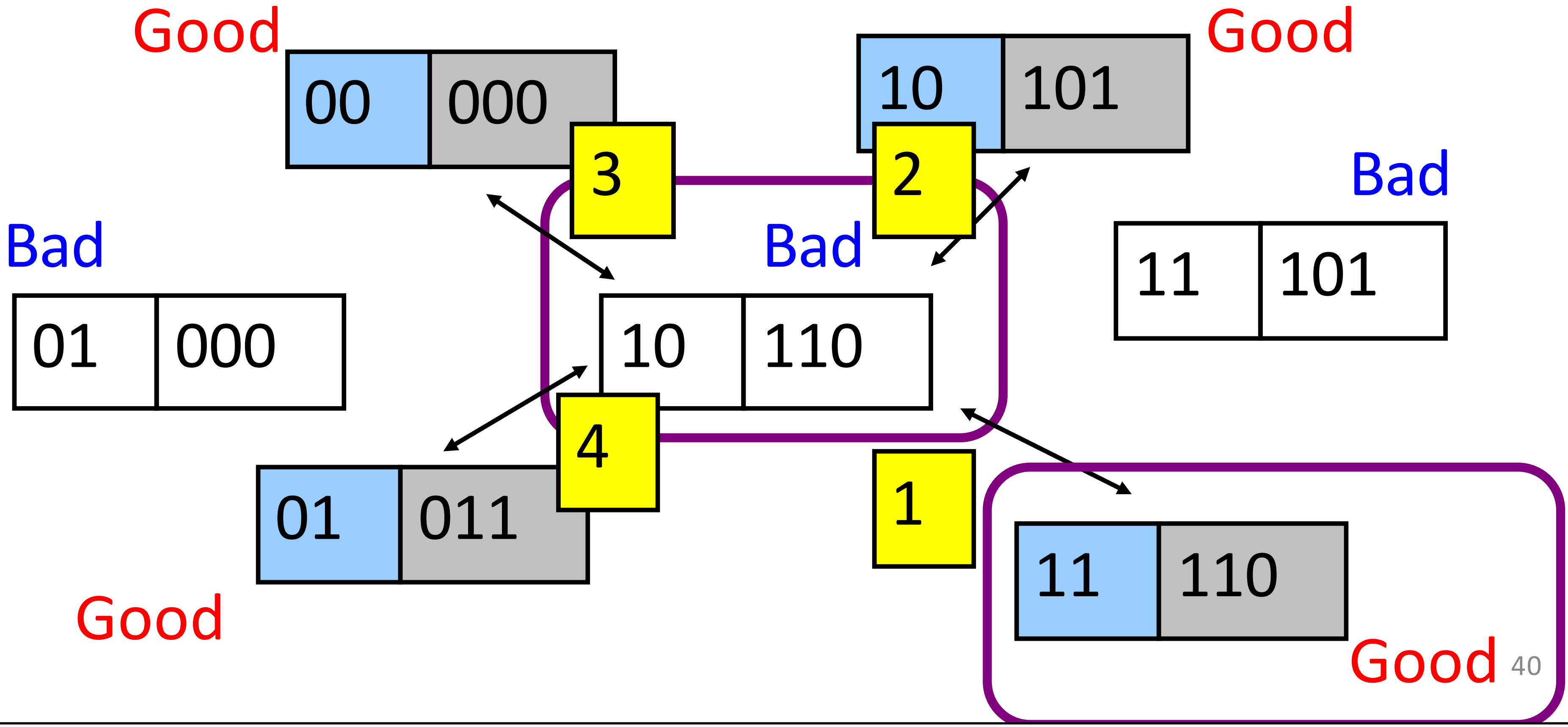
```
receive(X1Y1);
Y2=generateCheckBit(X1);
if (Y1 != Y2) FIXERROR(X1Y1);
else NOERROR
```



Basic Idea of Forward Error Correction

Replace erroneous data

by its "closest" error-free data.



Error Detection vs Correction

Error Correction:

- Cons: More check bits. False recovery.
- Pros: No need to re-send.

Error Detection:

- Cons: Need to re-send.
- Pros: Less check bits.

Usage:

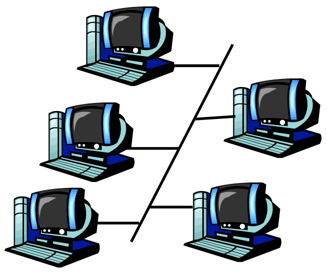
- Correction: A lot of noise. Expensive to re-send.
- Detection: Less noise. Easy to re-send.
- Can be used together.

Multiple Access Links and Protocols

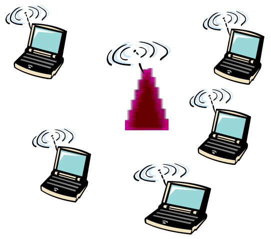
Two types of "links":

- point-to-point
  - point-to-point link between Ethernet switch and host

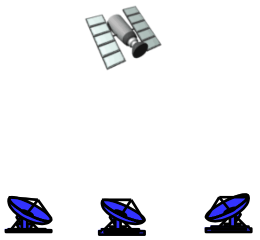
- broadcast (shared wire or medium)
  - old-fashioned wired Ethernet (*here be dinosaurs* – extinct)
  - upstream HFC (Hybrid Fiber-Coax – the Coax may be broadcast)
  - Home plug / Powerline networking
  - 802.11 wireless LAN



shared wire (e.g., cabled Ethernet)



shared RF (e.g., 802.11 WiFi)



shared RF (satellite)

humans at a cocktail party (shared air, acoustical)

Multiple Access protocols

- single shared broadcast channel
- two or more simultaneous transmissions by nodes: interference
    - collision if node receives two or more signals at the same time

multiple access protocol

- distributed algorithm that determines how nodes share channel, i.e., determine when node can transmit
- communication about channel sharing must use channel itself!
    - no out-of-band channel for coordination

Ideal Multiple Access Protocol

Broadcast channel of rate *R* bps

1. when one node wants to transmit, it can send at rate *R*
2. when *M* nodes want to transmit, each can send at average rate *R/M*
3. fully decentralized:
   - no special node to coordinate transmissions
   - no synchronization of clocks, slots
4. simple

MAC Protocols: a taxonomy

Three broad classes:

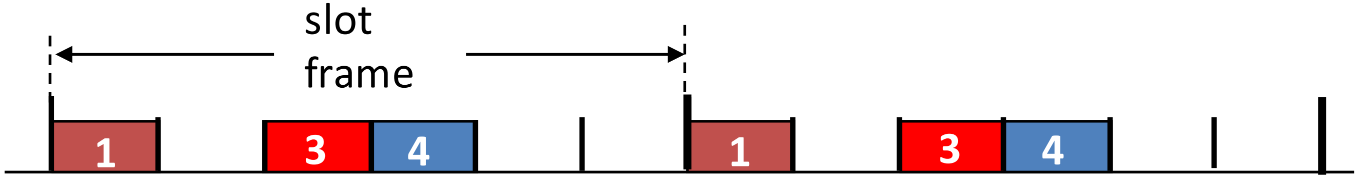
- Channel Partitioning
  - divide channel into smaller “pieces” (time slots, frequency, code)
  - allocate piece to node for exclusive use
- Random Access
  - channel not divided, allow collisions
  - “recover” from collisions
- “Taking turns”
  - nodes take turns, but nodes with more to send can take longer turns

Channel Partitioning MAC protocols: TDMA

(time travel warning – we mentioned this earlier)

TDMA: time division multiple access

- access to channel in "rounds"
- each station gets fixed length slot (length = pkt trans time) in each round
- unused slots go idle
- example: station LAN, 1,3,4 have pkt, slots 2,5,6 idle

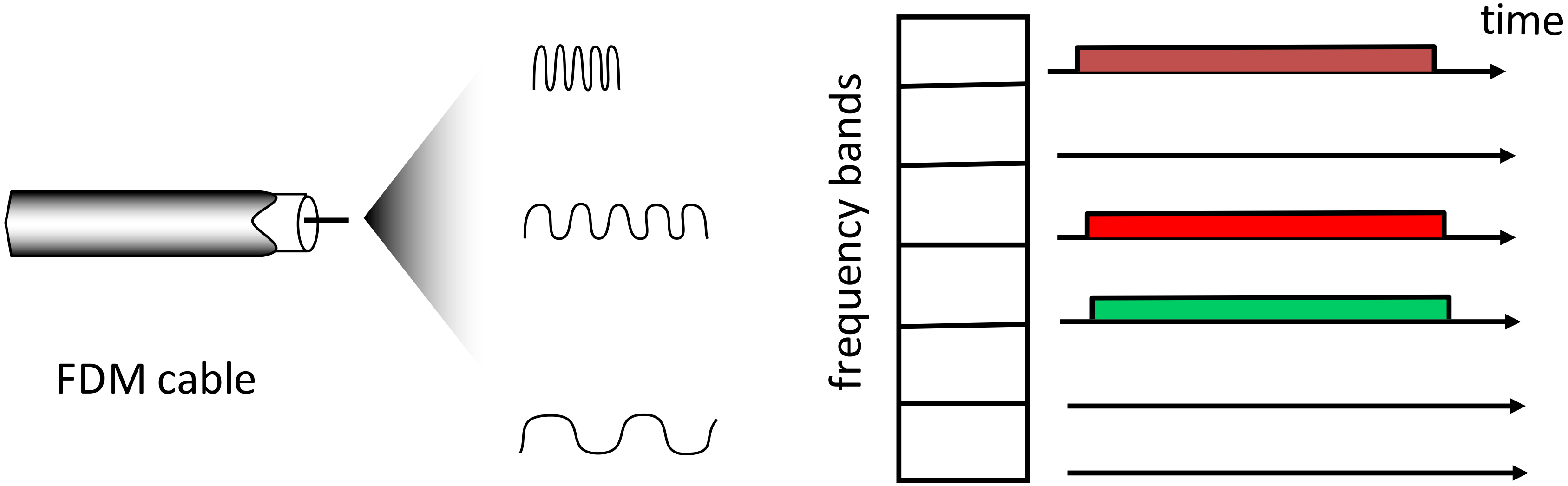


Channel Partitioning MAC protocols: FDMA

(time travel warning – we mentioned this earlier)

FDMA: frequency division multiple access

- channel spectrum divided into frequency bands
- each station assigned fixed frequency band
- unused transmission time in frequency bands go idle
- example: station LAN, 1,3,4 have pkt, frequency bands 2,5,6 idle



“Taking Turns” MAC protocols

channel partitioning MAC protocols:

- share channel *efficiently* and *fairly* at high load
- inefficient at low load: delay in channel access, 1/N bandwidth allocated even if only 1 active node!

Random access MAC protocols

- efficient at low load: single node can fully utilize channel
- high load: collision overhead

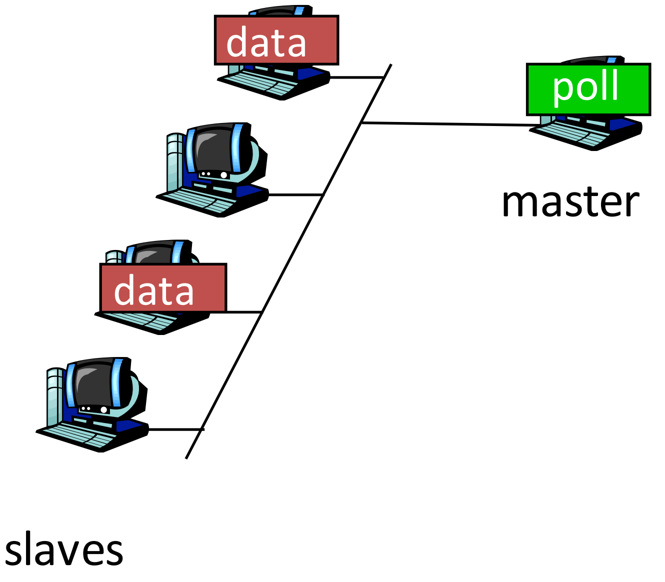
“taking turns” protocols

look for best of both worlds!

“Taking Turns” MAC protocols

Polling:

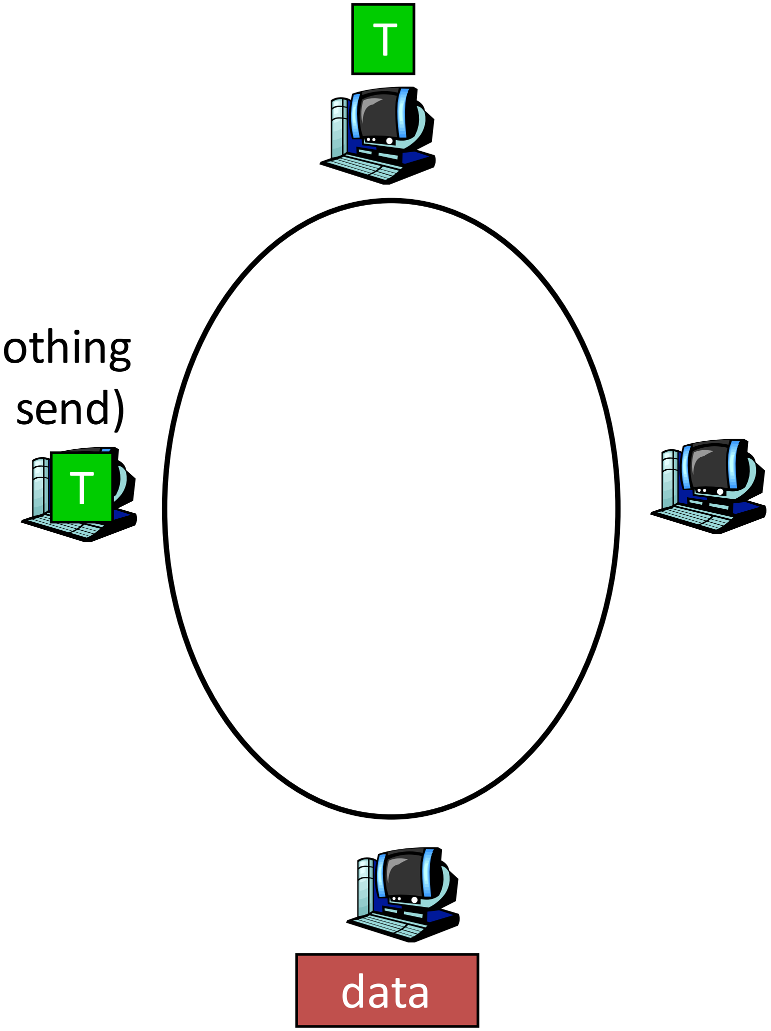
- master node “invites” slave nodes to transmit in turn
- typically used with “dumb” slave devices
- concerns:
  - polling overhead
  - latency
  - single point of failure (master)



"Taking Turns" MAC protocols

Token passing:

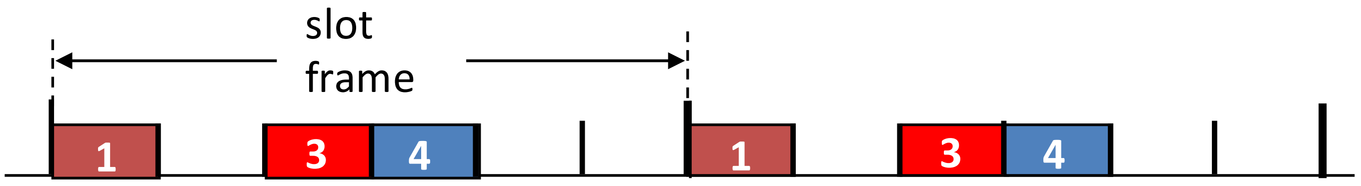
- control **token** passed from one node to next sequentially.
- token message
- concerns:
  - token overhead
  - latency
  - single point of failure (token)
- concerns fixed in part by a slotted ring (many simultaneous *tokens)*



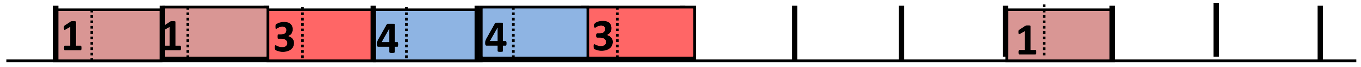
Cambridge students – this is YOUR heritage
Cambridge RING, Cambridge Fast RING,
Cambridge Backbone RING, these things gave us ATDM (and ATM)

ATM

In TDM a sender may only use a pre-allocated slot



In ATM a sender transmits labeled cells whenever necessary



ATM = Asynchronous Transfer Mode – an ugly expression
think of it as ATDM – Asynchronous Time Division Multiplexing

That’s a variant of **PACKET SWITCHING** to the rest of us – just like Ethernet
but using fixed length slots/packets/cells

Use the media when you need it, but
ATM had virtual circuits and these needed setup….
Worse ATM had an utterly irrational size

Random Access MAC Protocols

- When node has packet to send
  - Transmit at full channel data rate
  - No *a priori* coordination among nodes
- Two or more transmitting nodes ⇒ collision
  - Data lost
- Random access MAC protocol specifies:
  - How to detect collisions
  - How to recover from collisions
- Examples
  - ALOHA and Slotted ALOHA
  - CSMA, CSMA/CD, CSMA/CA (wireless)

Key Ideas of Random Access

- Carrier sense
  - *Listen before speaking, and don't interrupt*
  - Checking if someone else is already sending data
  - … and waiting till the other node is done
- Collision detection
  - *If someone else starts talking at the same time, stop*
  - Realizing when two nodes are transmitting at once
  - …by detecting that the data on the wire is garbled
- Randomness
  - *Don't start talking again right away*
  - Waiting for a random time before trying again

CSMA (Carrier Sense Multiple Access)

- CSMA: listen before transmit
  - If channel sensed idle: transmit entire frame
  - If channel sensed busy, defer transmission
- Human analogy: don't interrupt others!
- Does this eliminate all collisions?
  - No, because of nonzero propagation delay

CSMA Collisions

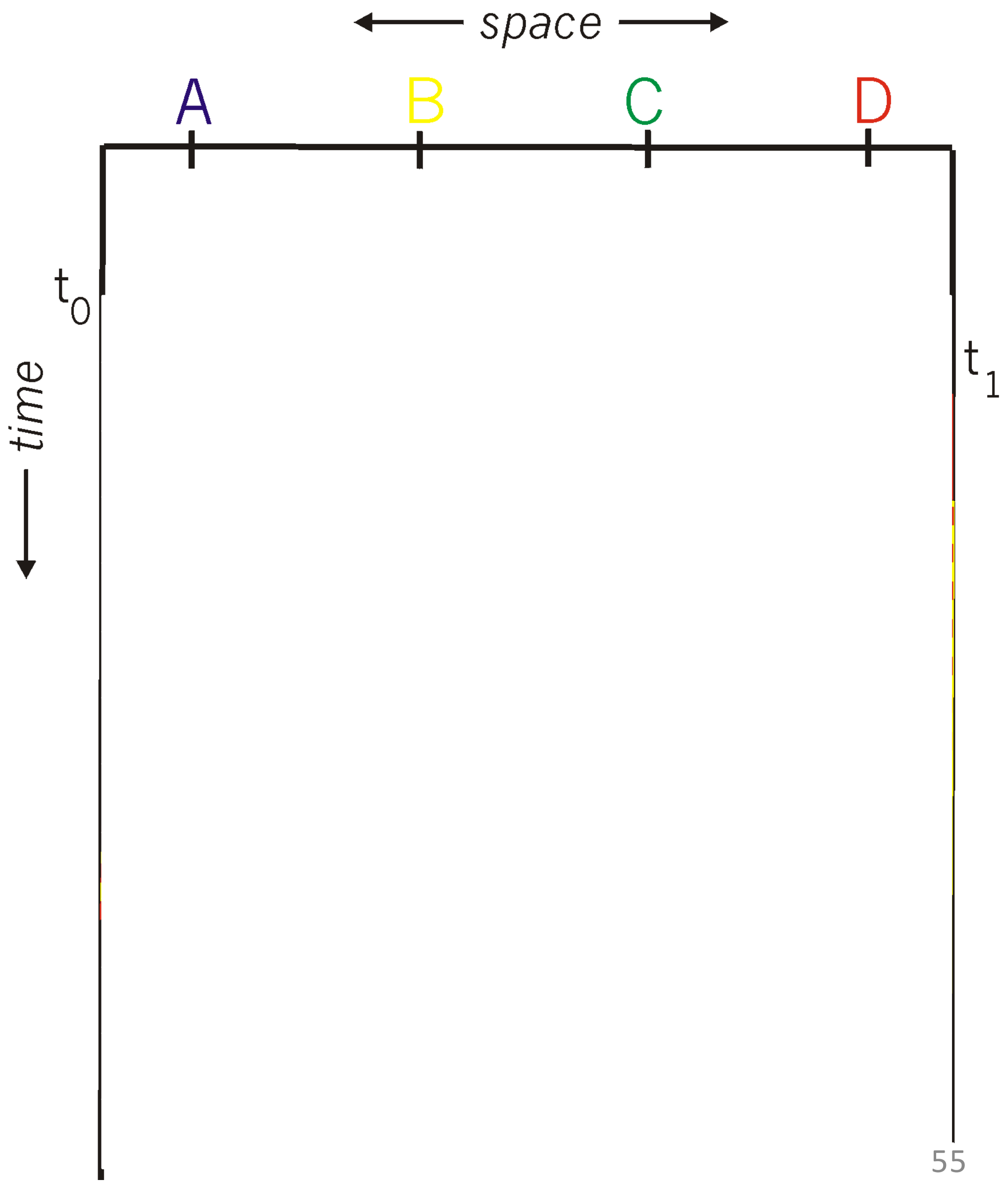
Propagation delay: two nodes may not hear each other's before sending.

Would slots hurt or help?

CSMA reduces but does not eliminate collisions

Biggest remaining problem?

Collisions still take full slot! How do you fix that?



CSMA/CD (Collision Detection)

- CSMA/CD: carrier sensing, deferral as in CSMA
  - **Collisions detected within short time**
  - Colliding transmissions aborted, reducing wastage

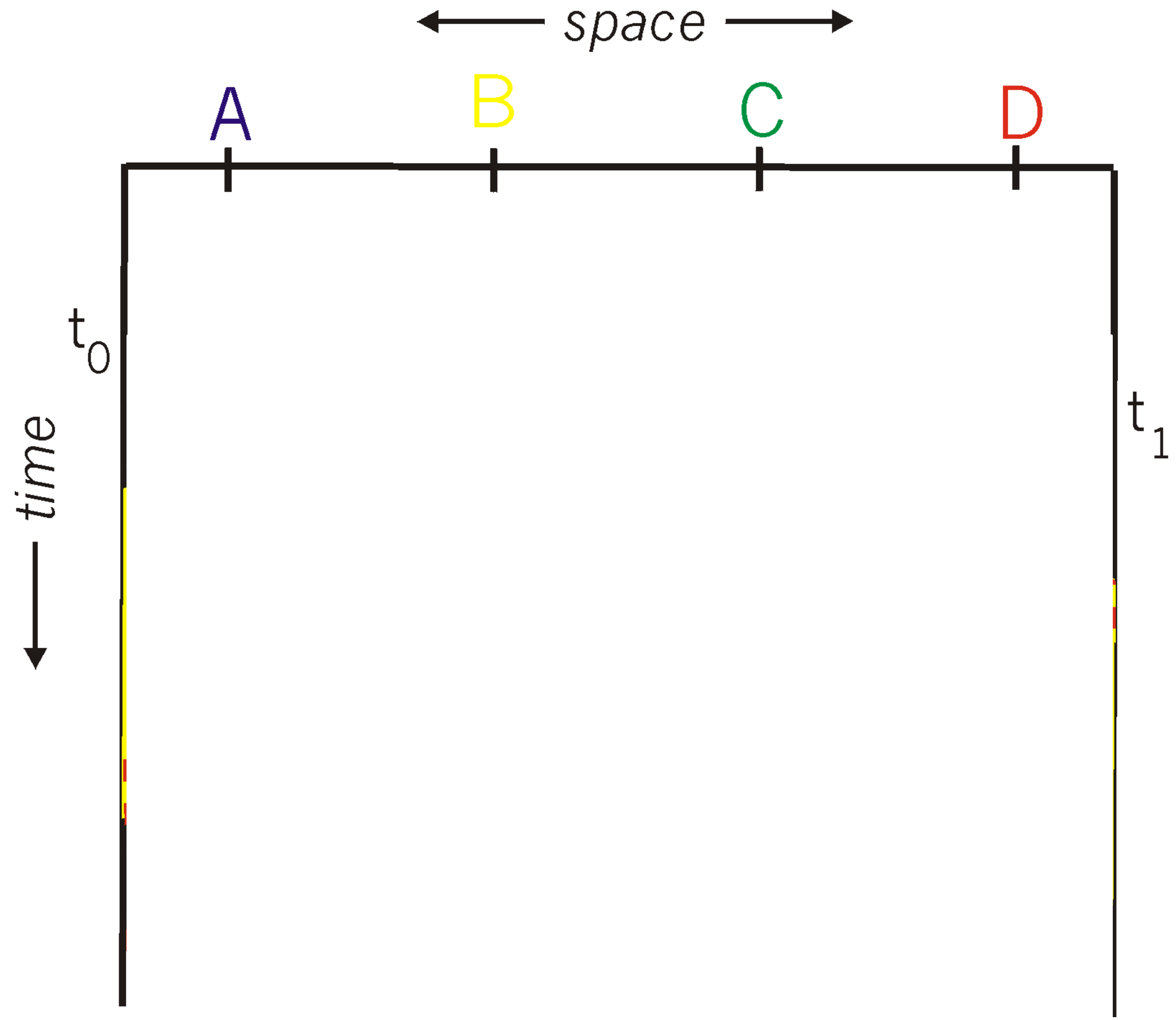
- Collision detection easy in wired LANs:
  - Compare transmitted, received signals

- Collision detection difficult in wireless LANs:
  - Reception shut off while transmitting (well, perhaps not)
  - Not perfect broadcast (limited range) so collisions local
  - Leads to use of *collision avoidance* instead (later)

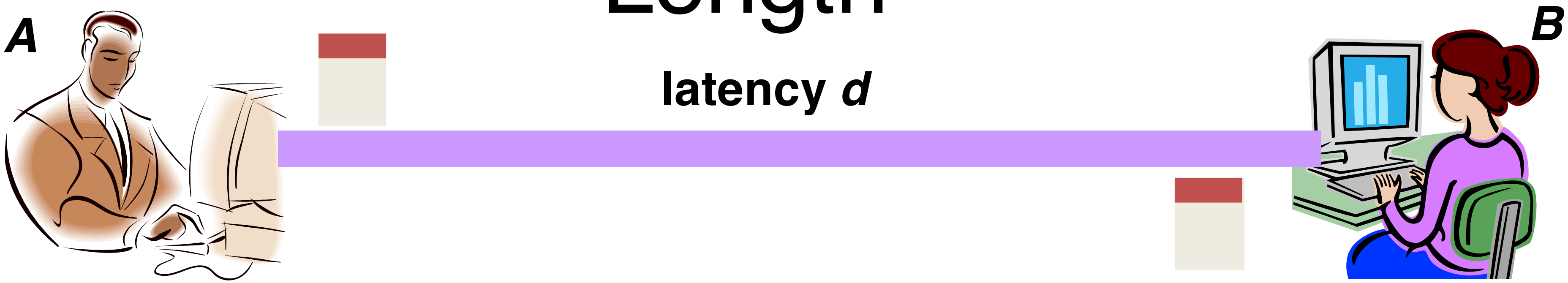
CSMA/CD Collision Detection

B and D can tell that collision occurred.

Note: for this to work, need restrictions on minimum frame size and maximum distance. Why?



Limits on CSMA/CD Network Length



- Latency depends on physical length of link
  - Time to propagate a packet from one end to the other
- Suppose *A* sends a packet at time ***t***
  - And *B* sees an idle line at a time just before ***t+d***
  - … so *B* happily starts transmitting a packet
- *B* detects a collision, and sends jamming signal
  - But *A* can't see collision until ***t+2d***

Performance of CSMA/CD

- Time wasted in collisions
  - Proportional to distance d
- Time spend transmitting a packet
  - Packet length p divided by bandwidth b
- Rough estimate for efficiency (K some constant)

$$E \sim \frac{\frac{p}{b}}{\frac{p}{b} + Kd}$$

- Note:
  - For large packets, small distances, E ~ 1
  - As bandwidth increases, E decreases
  - That is why high-speed LANs are all switched

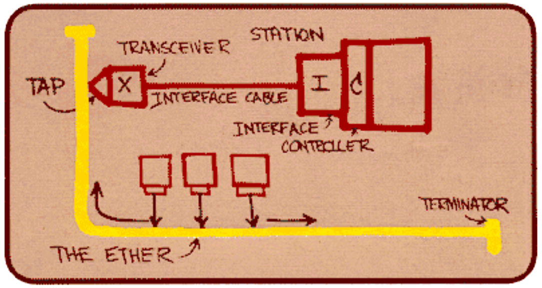
Benefits of Ethernet

- Easy to administer and maintain
- Inexpensive
- Increasingly higher speed
- Evolvable!

Evolution of Ethernet

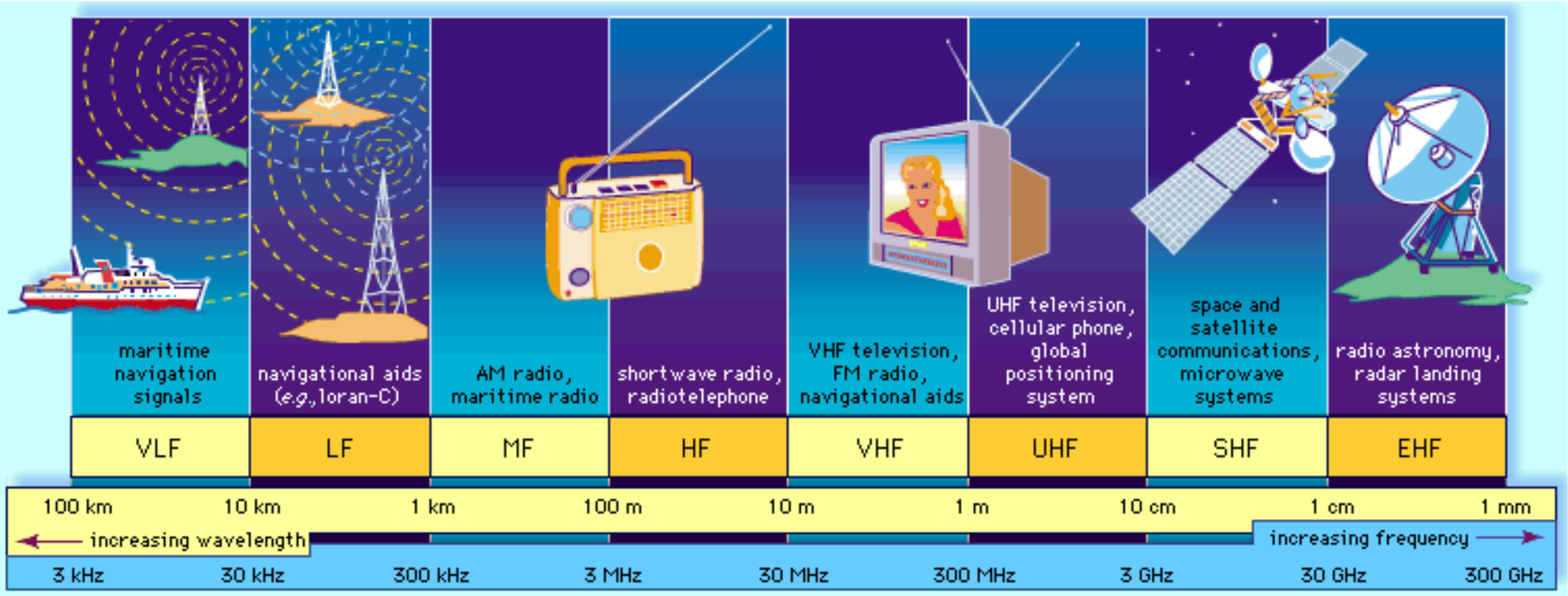
- Changed everything except the frame format
  - From single coaxial cable to hub-based star
  - From shared media to switches
  - From electrical signaling to optical
- Lesson #1
  - The right interface can accommodate many changes
  - Implementation is hidden behind interface
- Lesson #2
  - Really hard to displace the dominant technology
  - Slight performance improvements are not enough

Ethernet: CSMA/CD Protocol

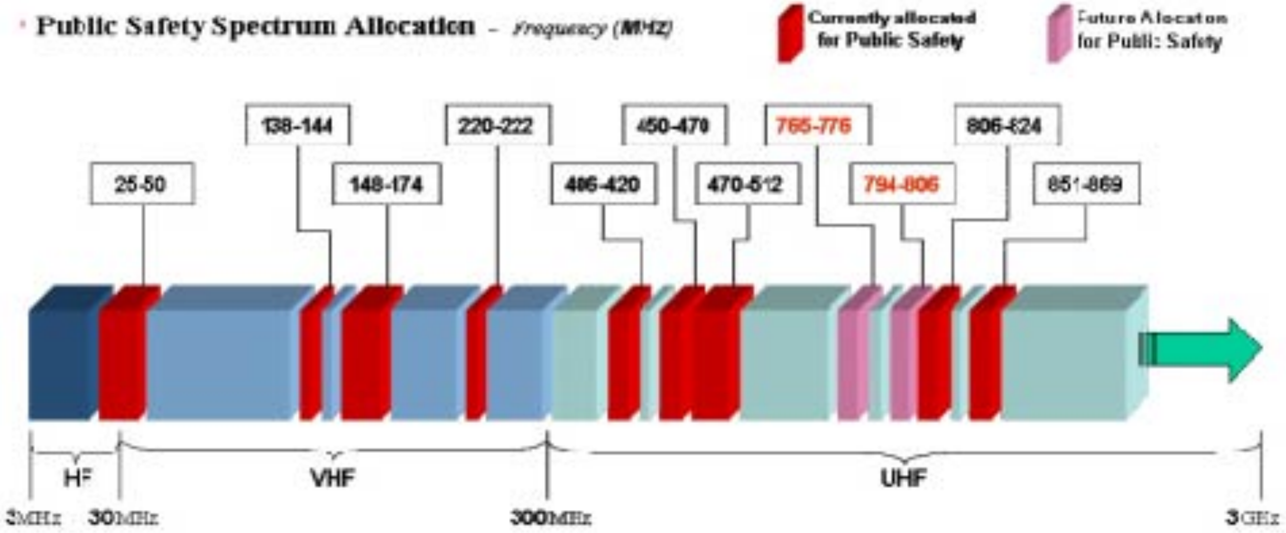


- **Carrier sense**: wait for link to be idle
- **Collision detection**: listen while transmitting
  - No collision: transmission is complete
  - Collision: abort transmission & send **jam** signal
- **Random access**: binary exponential back-off
  - After collision, wait a random time before trying again
  - After mth collision, choose K randomly from {0, …, 2^m-1}
  - … and wait for K*512 bit times before trying again
    - Using min packet size as "slot"
    - **If transmission occurring when ready to send, wait until end of transmission (CSMA)**

The Wireless Spectrum


maritime navigation signals
navigational aids (e.g., loran-C)
AM radio, maritime radio
shortwave radio, radiotelephone
VHF television, FM radio, navigational aids
UHF television, cellular phone, global positioning system
space and satellite communications, microwave systems
radio astronomy, radar landing systems
VLF
LF
MF
HF
VHF
UHF
SHF
EHF
100 km
10 km
1 km
100 m
10 m
1 m
10 cm
1 cm
1 mm
increasing wavelength
increasing frequency
3 kHz
30 kHz
300 kHz
3 MHz
30 MHz
300 MHz
3 GHz
30 GHz
300 GHz

© 1999 Encyclopædia Britannica, Inc.


Public Safety Spectrum Allocation - Frequency (MHZ)
Currently allocated for Public Safety
Future Allocation for Public Safety
25-50
138-144
148-174
220-222
406-420
450-470
470-512
765-776
794-806
806-824
851-869
HF
VHF
UHF
2MHz
30MHz
300MHz
3GHz

Metrics for evaluation / comparison of wireless technologies

- Bitrate or Bandwidth
- Range - PAN, LAN, MAN, WAN
- Two-way / One-way
- Multi-Access / Point-to-Point
- Digital / Analog
- Applications and industries
- Frequency – Affects most physical properties:

  Distance (free-space loss)

  Penetration, Reflection, Absorption

  Energy proportionality

  Policy: Licensed / Deregulated

  Line of Sight (Fresnel zone)

  Size of antenna

➢ Determined by wavelength – $\lambda = \frac{v}{f},$)

Wireless Communication Standards

- Cellular (800/900/*1700*/1800/1900Mhz):
  - 2G: GSM / CDMA / GPRS /EDGE
  - 3G: CDMA2000/UMTS/HSDPA/EVDO
  - 4G: LTE, WiMax
- IEEE 802.11 (aka WiFi):
  - b: 2.4Ghz band, 11Mbps (~*4.5 Mbps operating rate*)
  - g: 2.4Ghz, 54-108Mbps (~*19 Mbps operating rate*)
  - a: 5.0Ghz band, 54-108Mbps (~*25 Mbps operating rate*)
  - n: 2.4/5Ghz, 150-600Mbps (4x4 mimo).
- IEEE 802.15 – lower power wireless:
  - 802.15.1: 2.4Ghz, 2.1 Mbps (Bluetooth)
  - 802.15.4: 2.4Ghz, 250 Kbps (Sensor Networks)

What Makes Wireless Different?

- Broadcast and multi-access medium…
  - err, so….
- BUT, Signals sent by sender don’t always end up at receiver intact
  - Complicated physics involved, which we won’t discuss
  - But what can go wrong?

Path Loss / Path Attenuation

- Free Space Path Loss:

  d = distance

  λ = wave length

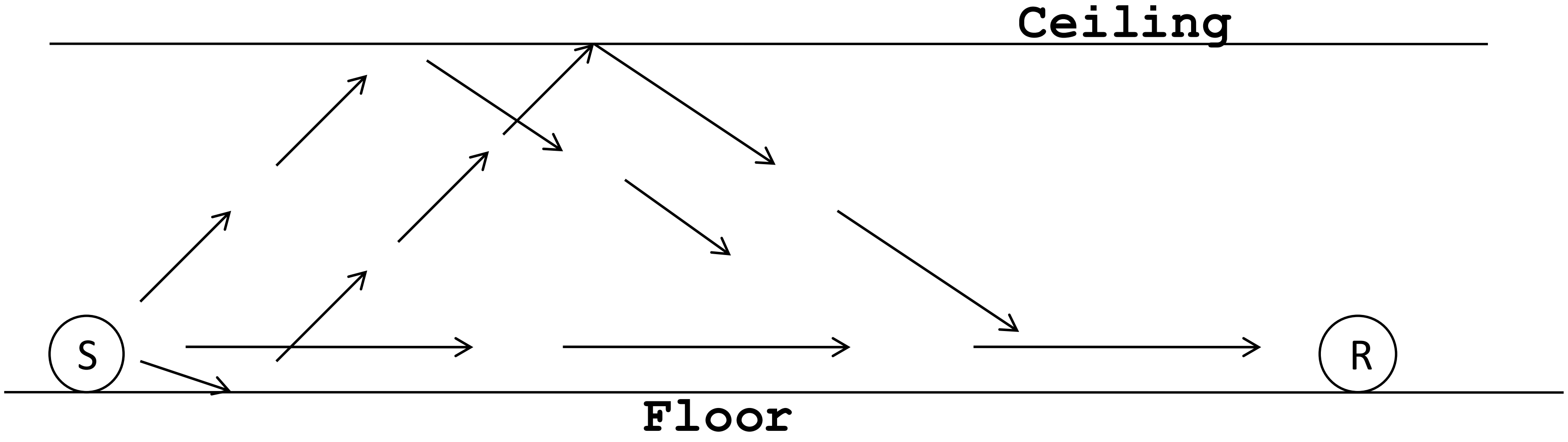
  f = frequency

  c = speed of light

$$\mathrm{FSPL} = \left(\frac{4\pi d}{\lambda}\right)^2 = \left(\frac{4\pi df}{c}\right)^2$$

- Reflection, Diffraction, Absorption
- Terrain contours (Urban, Rural, Vegetation).
- Humidity

Multipath Effects



- Signals bounce off surface and interfere with one another
- Self-interference

Interference from Other Sources

- External Interference
  - Microwave is turned on and blocks your signal
  - Would that affect the sender or the receiver?
- Internal Interference
  - Hosts within range of each other collide with one another’s transmission

- We have to tolerate path loss, multipath, etc., but we can try to avoid internal interference

Wireless Bit Errors

- The lower the SNR (Signal/Noise) the higher the Bit Error Rate (BER)
- We could make the signal stronger…
- Why is this not always a good idea?
  - Increased signal strength requires more power
  - Increases the interference range of the sender, so you interfere with more nodes around you
    - And then they increase their power…….
- Local link-layer Error Correction schemes can correct some problems

Lets focus on 802.11

aka - WiFi …
What makes it special?

Deregulation > Innovation > Adoption > Lower cost = Ubiquitous technology

JUST LIKE ETHERNET – not lovely but sufficient

802.11 Architecture

802.11 frames exchanges

802.3 (Ethernet) frames exchanged

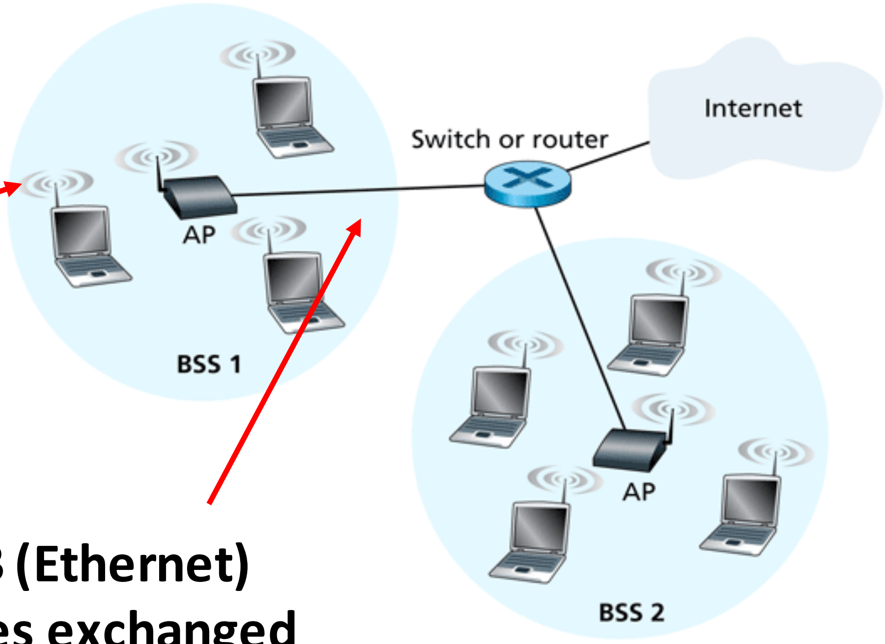


Figure 6.7 ♦ IEEE 802.11 LAN architecture

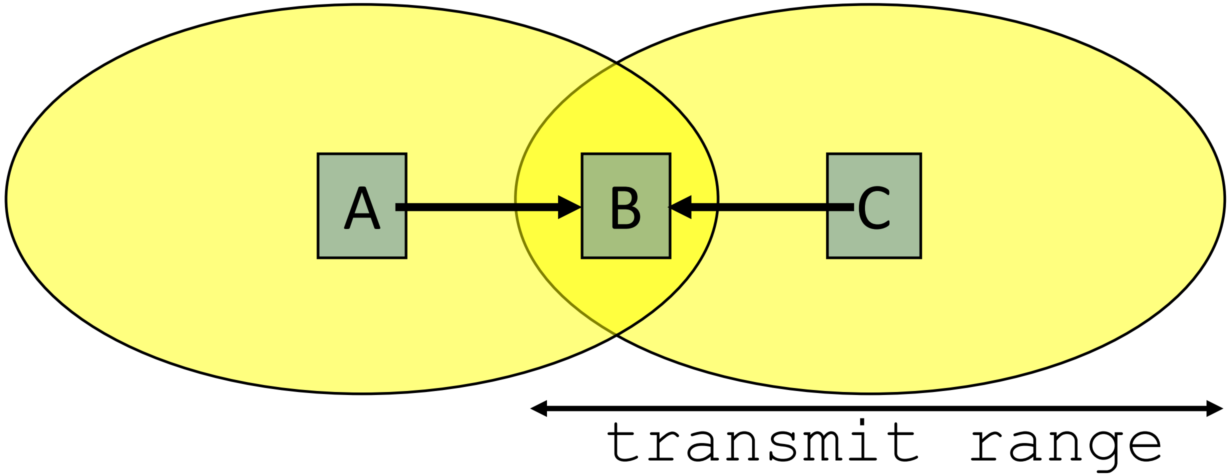
- Designed for limited area
- AP’s (Access Points) set to specific channel
- Broadcast beacon messages with SSID (Service Set Identifier) and MAC Address periodically
- Hosts scan all the channels to discover the AP’s
  - Host associates with AP

Wireless Multiple Access Technique?

- Carrier Sense?
  - Sender can listen before sending
  - What does that tell the sender?

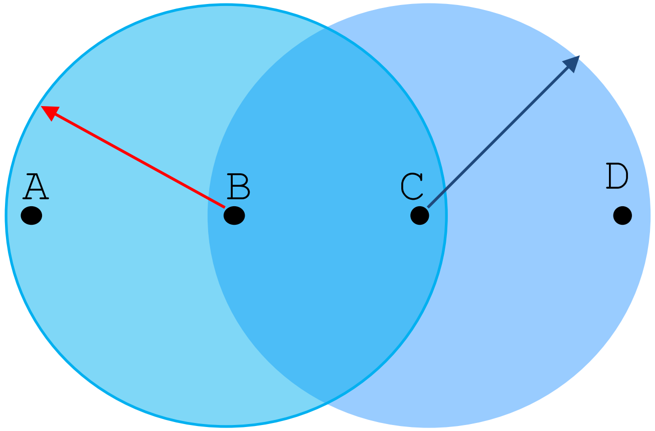
- Collision Detection?
  - Where do collisions occur?
  - How can you detect them?

Hidden Terminals



- A and C can both send to B but can’t hear each other
  - A is a *hidden terminal* for C and vice versa
- Carrier Sense will be ineffective

Exposed Terminals



- Exposed node: B sends a packet to A; C hears this and decides not to send a packet to D (despite the fact that this will not cause interference)!
- Carrier sense would prevent a successful transmission.

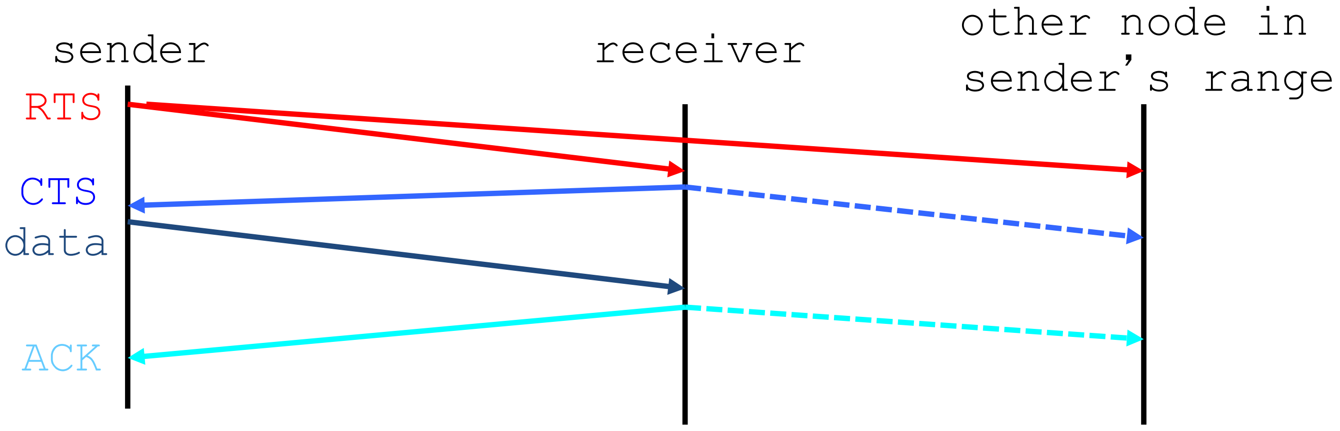
Key Points

- No concept of a global collision
  - Different receivers hear different signals
  - Different senders reach different receivers
- Collisions are at receiver, not sender
  - Only care if receiver can hear the sender clearly
  - It does not matter if sender can hear someone else
  - As long as that signal does not interfere with receiver
- Goal of protocol:
  - Detect if receiver can hear sender
  - Tell senders who might interfere with receiver to shut up

Basic Collision Avoidance

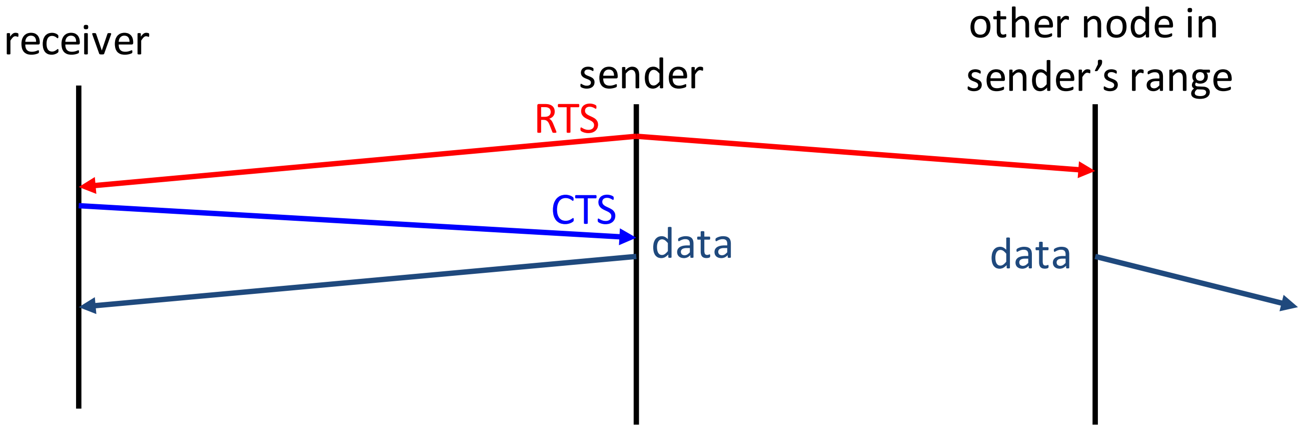
- Since can't detect collisions, we try to *avoid* them
- Carrier sense:
  - When medium busy, choose random interval
  - Wait that many **idle** timeslots to pass before sending
- When a collision is inferred, retransmit with binary exponential backoff (like Ethernet)
  - Use ACK from receiver to infer "no collision"
  - Use exponential backoff to adapt contention window

CSMA/CA -MA with Collision Avoidance



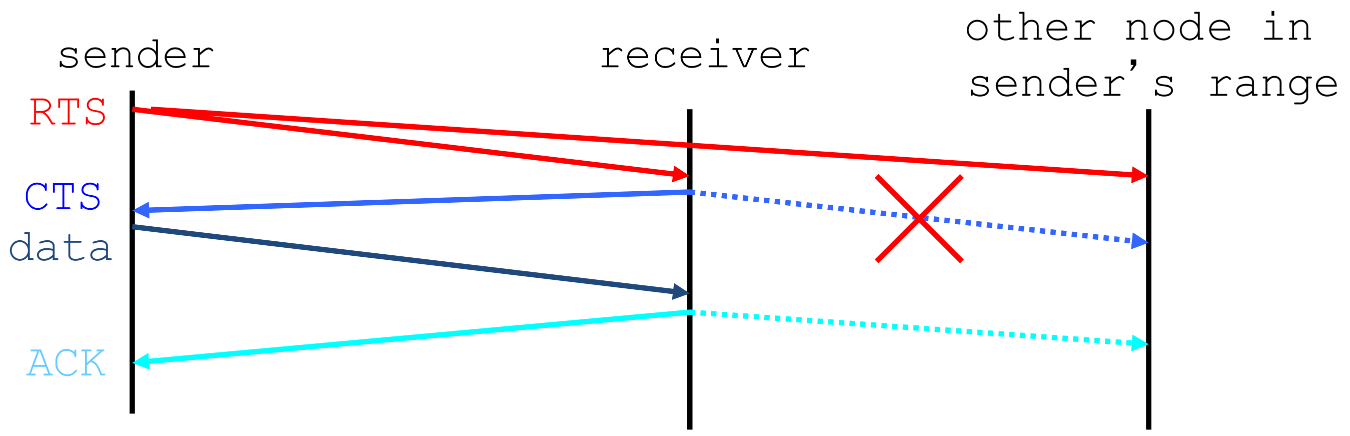
- Before every data transmission
  - Sender sends a Request to Send (RTS) frame containing the length of the transmission
  - Receiver respond with a Clear to Send (CTS) frame
  - Sender sends data
  - Receiver sends an ACK; now another sender can send data
- When sender doesn't get a CTS back, it assumes collision

CSMA/CA, con’t



- If other nodes hear RTS, but not CTS: send
  - Presumably, destination for first sender is out of node’s range …

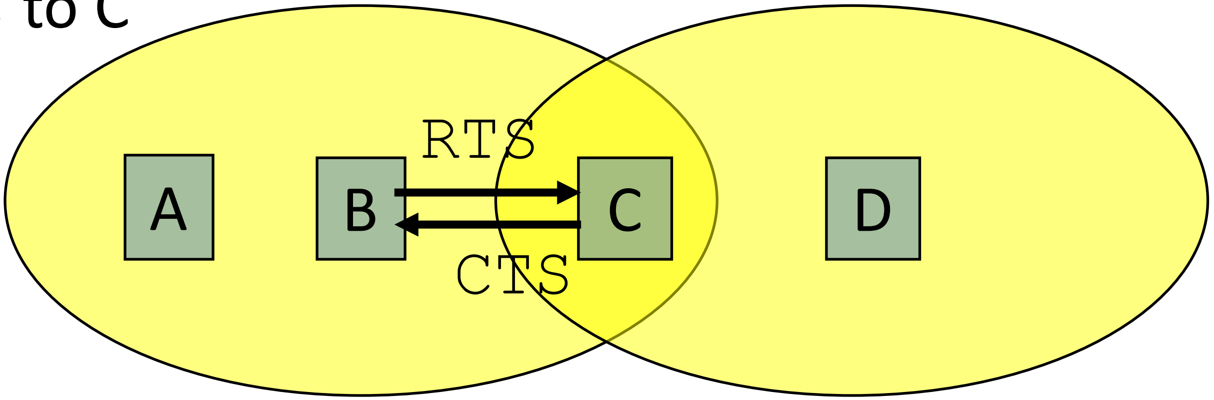
CSMA/CA, con’t



- If other nodes hear RTS, but not CTS: send
  - Presumably, destination for first sender is out of node’s range …
  - … Can cause problems when a CTS is lost
- When you hear a CTS, you keep quiet until scheduled transmission is over (hear ACK)

RTS / CTS Protocols (CSMA/CA)

B sends to C

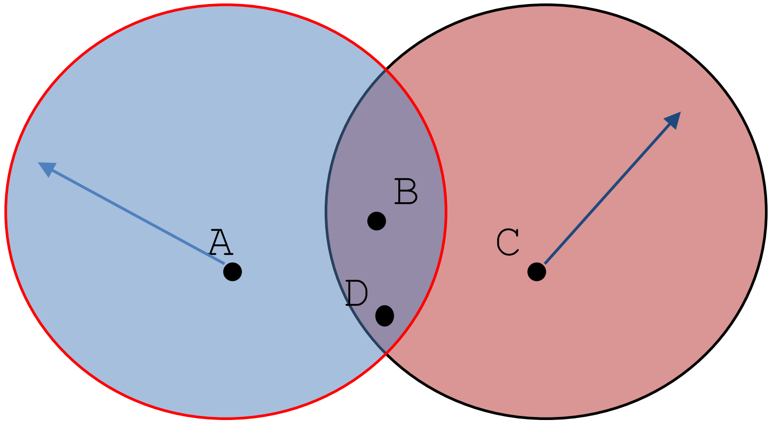


Overcome hidden terminal problems with contention-free protocol

1. B sends to C Request To Send (RTS)
2. A hears RTS and defers (to allow C to answer)
3. C replies to B with Clear To Send (CTS)
4. D hears CTS and defers to allow the data
5. B sends to C

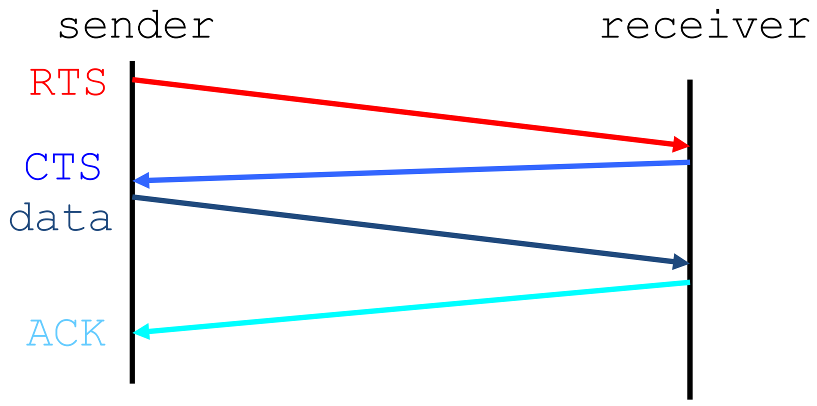
Preventing Collisions Altogether

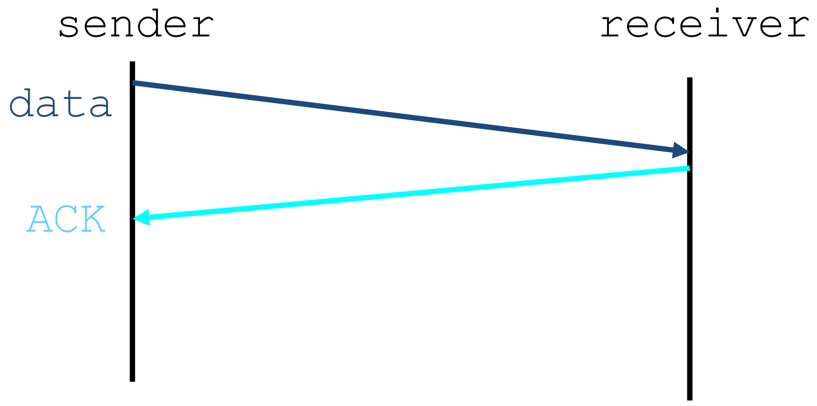
- Frequency Spectrum partitioned into several channels
  - Nodes within interference range can use separate channels



  - Now A and C can send without any interference!
- Most cards have only 1 transceiver
  - **Not Full Duplex: Cannot send and receive at the same time**
  - Aggregate Network throughput doubles

CSMA/CA and RTS/CTS





RTS/CTS

- helps with hidden terminal
- good for high-traffic Access Points
- often turned on/off dynamically

Without RTS/CTS

- lower latency -> faster!
- reduces wasted b/w

  if the *Pr(collision)* is low
- good for when net is small and not *weird*

  eg no hidden/exposed terminals

CSMA/CD vs CSMA/CA (without RTS/CTS)

CD Collision Detect

wired – listen and talk

1. Listen for others
2. Busy? goto 1.
3. Send message (and listen)
4. Collision?
   a. JAM
   b. increase your BEB
   c. sleep
   d. goto 1.

CA Collision Avoidance

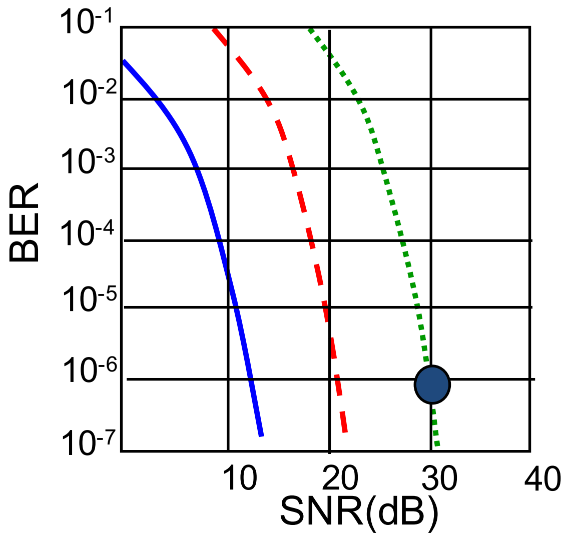
wireless – talk OR listen

1. Listen for others
2. Busy?
   a. increase your BEB
   b. sleep
   c. goto 1.
3. Send message
4. Wait for ACK (*MAC ACK*)
5. Got No ACK from MAC?
   a. increase your BEB
   b. sleep
   c. goto 1.

Changing the rules: an 802.11 feature

Rate Adaptation

- base station, mobile dynamically change transmission rate (physical layer modulation technique) as mobile moves, SNR varies



1. SNR decreases, BER increase as node moves away from base station

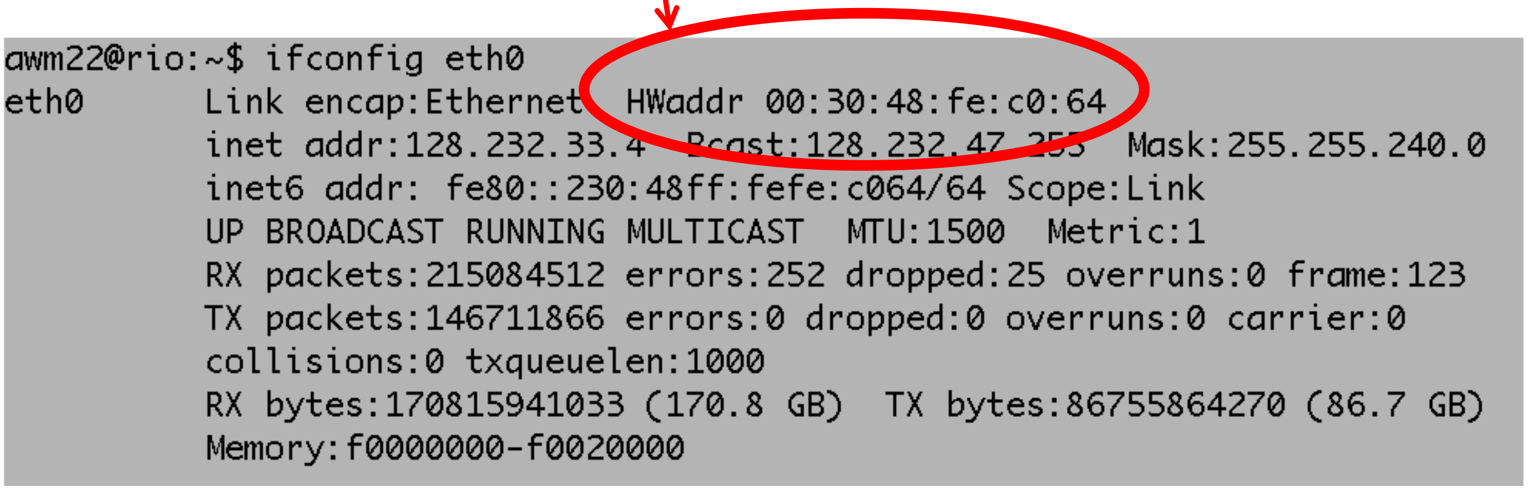
2. When BER becomes too high, switch to lower transmission rate but with lower BER

Summary of MAC protocols

- *channel partitioning,* by time, frequency or code
  - Time Division, Frequency Division
- *random access* (dynamic),
  - ALOHA, S-ALOHA, CSMA, CSMA/CD
  - carrier sensing: easy in some technologies (wire), hard in others (wireless)
  - CSMA/CD used in Ethernet
  - CSMA/CA used in 802.11
- *taking turns*
  - polling from central site, token passing
  - Bluetooth, FDDI, IBM Token Ring

MAC Addresses

- MAC (or LAN or physical or Ethernet) address:
  - function: *get frame from one interface to another physically-connected interface (same network)*
  - 48 bit MAC address (for most LANs)
    - *burned* in NIC ROM, nowadays usually software settable and set at boot time



```
awm22@rio:~$ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:30:48:fe:c0:64
          inet addr:128.232.33.4  Bcast:128.232.47.255  Mask:255.255.240.0
          inet6 addr: fe80::230:48ff:fefe:c064/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:215084512 errors:252 dropped:25 overruns:0 frame:123
          TX packets:146711866 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:170815941033 (170.8 GB)  TX bytes:86755864270 (86.7 GB)
          Memory:f0000000-f0020000
```

LAN Address (more)

- MAC address allocation administered by IEEE
- manufacturer buys portion of MAC address space (to assure uniqueness)
- analogy:

  (a) MAC address: like Social Security Number

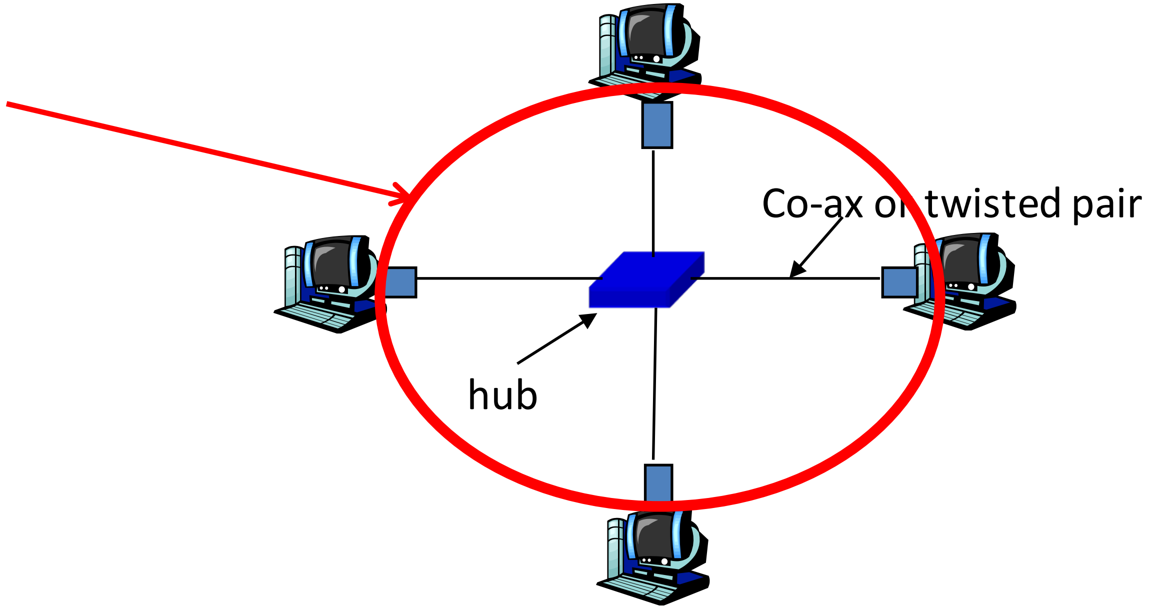
  (b) IP address: like postal address
- MAC flat address ➜ portability
  - can move LAN card from one LAN to another
- IP hierarchical address NOT portable
  - address depends on IP subnet to which node is attached

Hubs

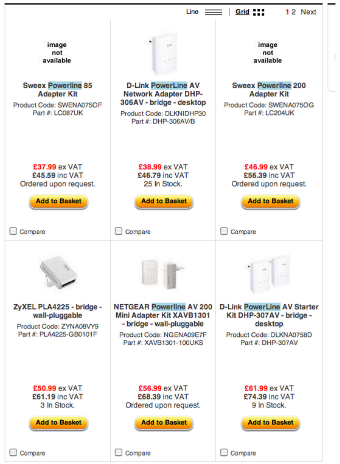
… physical-layer ("dumb") repeaters:

- bits coming in one link go out *all* other links at same rate
- all nodes connected to hub can collide with one another
- no frame buffering
- no CSMA/CD at hub: host NICs detect collisions

Collision Domain
in CSMA/CD *speak*



CSMA/CD Lives....



Line | Grid 1 2 Next

image not available

Sweex Powerline 85 Adapter Kit
Product Code: SWENA075OF
Part #: LC087UK
£37.99 ex VAT
£45.59 inc VAT
Ordered upon request.
Add to Basket
□ Compare

D-Link PowerLine AV Network Adapter DHP-306AV - bridge - desktop
Product Code: DLKNIDHP30
Part #: DHP-306AV/B
£38.99 ex VAT
£46.79 inc VAT
25 In Stock.
Add to Basket
□ Compare

image not available

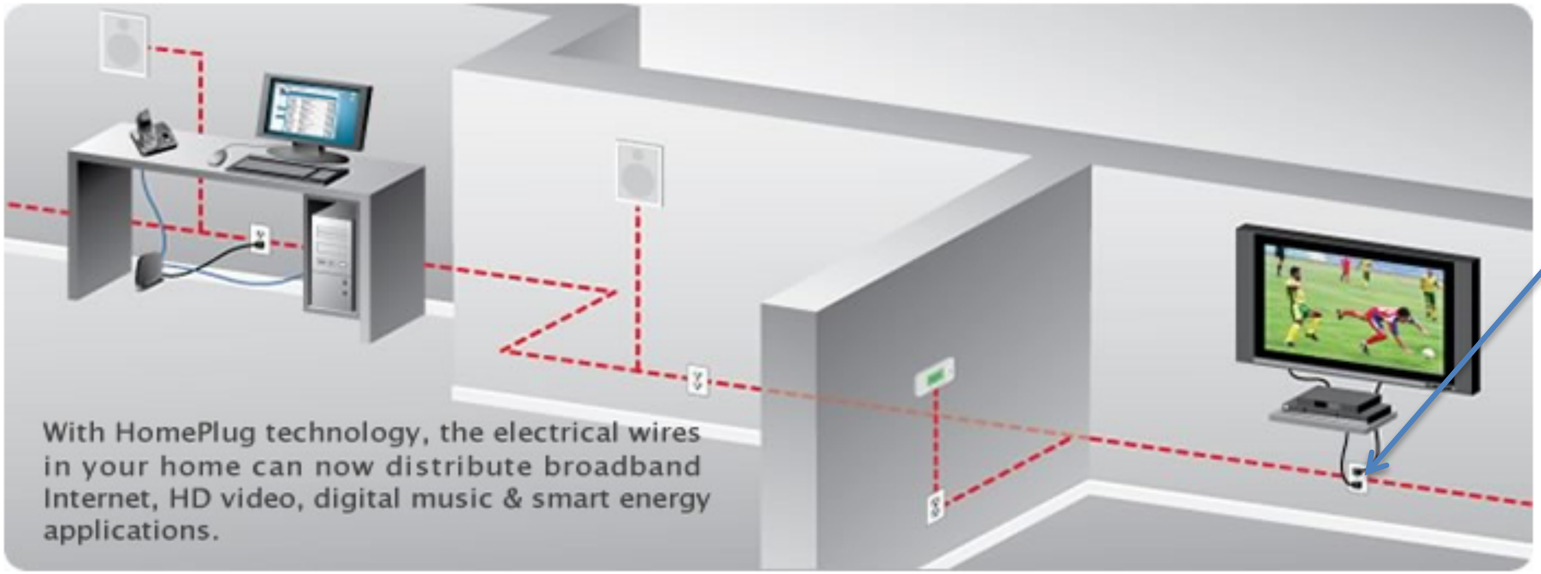
Sweex Powerline 200 Adapter Kit
Product Code: SWENA075OG
Part #: LC204UK
£46.99 ex VAT
£56.39 inc VAT
Ordered upon request.
Add to Basket
□ Compare

ZyXEL PLA4225 - bridge - wall-pluggable
Product Code: ZYNA08VY9
Part #: PLA4225-GB0101F
£50.99 ex VAT
£61.19 inc VAT
3 In Stock.
Add to Basket
□ Compare

NETGEAR Powerline AV 200 Mini Adapter Kit XAVB1301 - bridge - wall-pluggable
Product Code: NGENA09E7F
Part #: XAVB1301-100UKS
£56.99 ex VAT
£68.39 inc VAT
Ordered upon request.
Add to Basket
□ Compare

D-Link PowerLine AV Starter Kit DHP-307AV - bridge - desktop
Product Code: DLKNA0758D
Part #: DHP-307AV
£61.99 ex VAT
£74.39 inc VAT
9 In Stock.
Add to Basket
□ Compare

Home Plug and similar Powerline Networking....



With HomePlug technology, the electrical wires in your home can now distribute broadband Internet, HD video, digital music & smart energy applications.

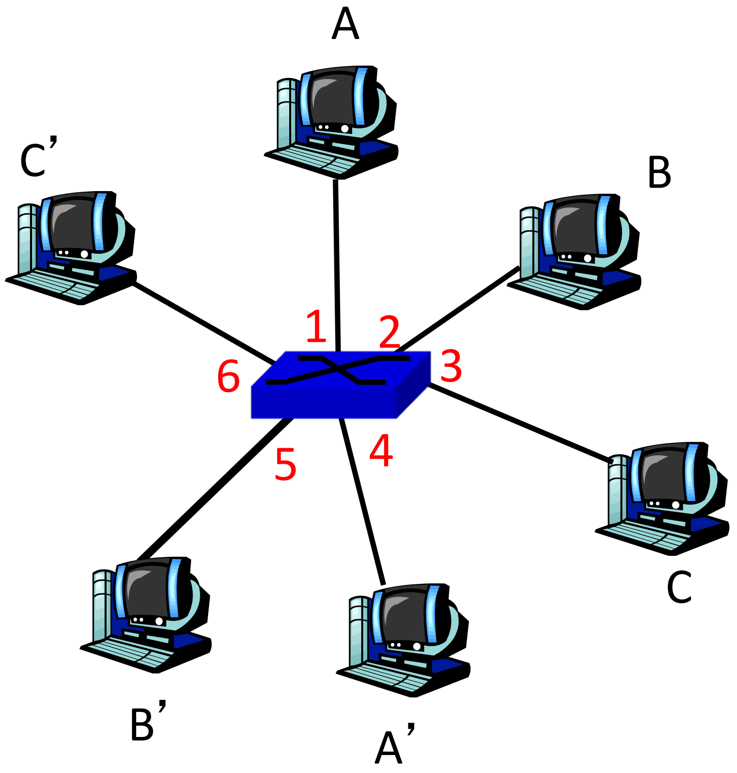
Switch

(like a Hub but smarter)

- link-layer device: smarter than hubs, take *active* role
  - store, forward Ethernet frames
  - examine incoming frame's MAC address, selectively forward frame to one-or-more outgoing links when frame is to be forwarded on segment, uses CSMA/CD to access segment
- *transparent*
  - hosts are unaware of presence of switches
- *plug-and-play, self-learning*
  - switches do not need to be configured

Switch: allows *multiple* simultaneous transmissions

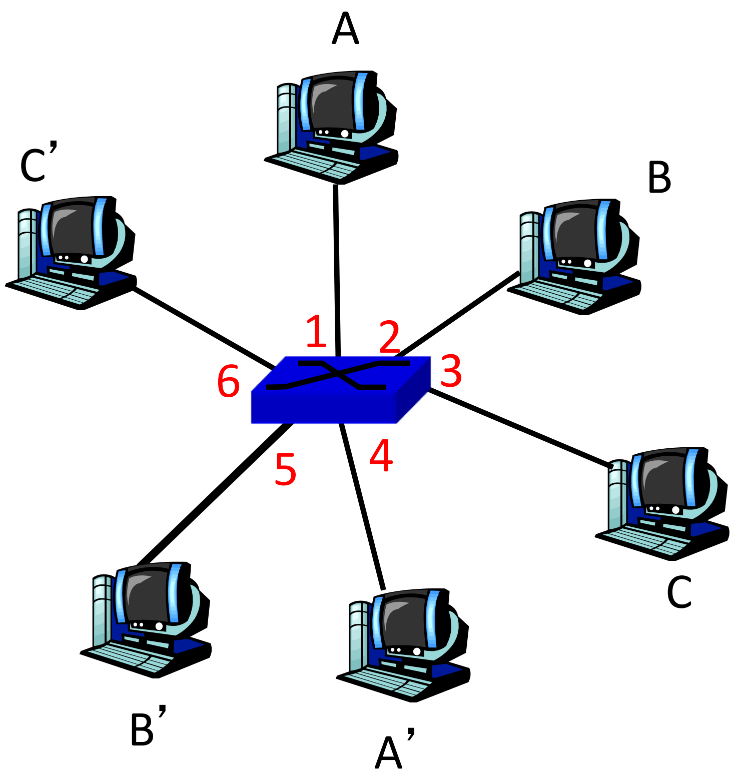
- hosts have dedicated, direct connection to switch
- switches buffer packets
- Ethernet protocol used on *each* incoming link, but no collisions; full duplex
  - each link is its own collision domain
- *switching:* A-to-A' and B-to-B' simultaneously, without collisions
  - not possible with dumb hub



switch with six interfaces (1,2,3,4,5,6)

Switch Table

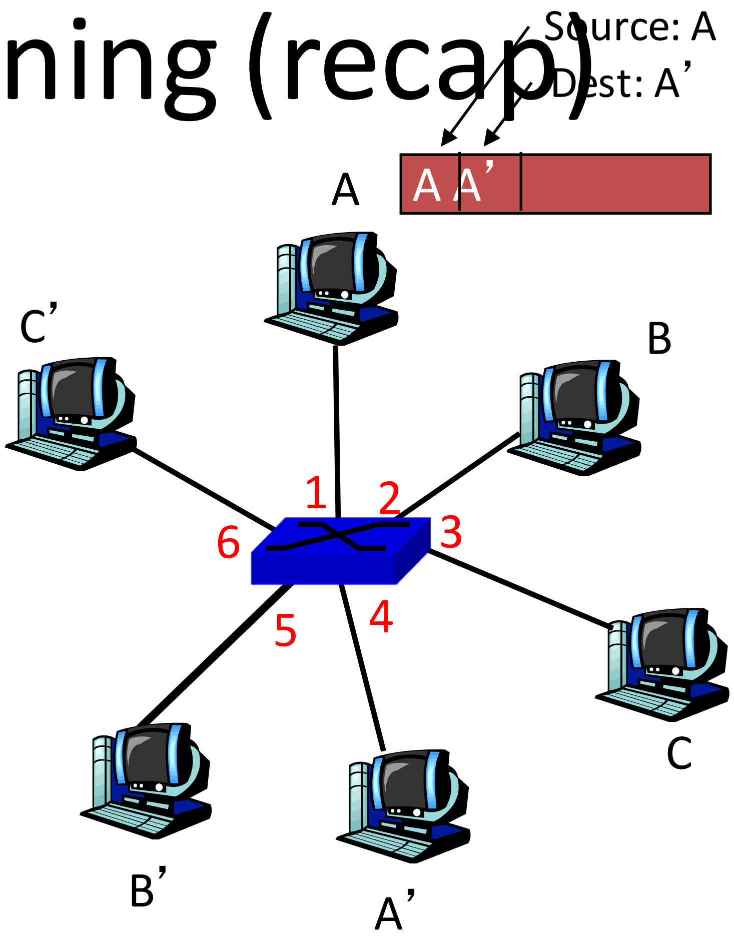
- *Q:* how does switch know that A' reachable via interface 4, B' reachable via interface 5?
- *A:* each switch has a switch table, each entry:
  - (MAC address of host, interface to reach host, time stamp)
- looks like a routing table!
- *Q:* how are entries created, maintained in switch table?
  - something like a routing protocol?



switch with six interfaces (1,2,3,4,5,6)

Switch: self-learning (recap)

- switch *learns* which hosts can be reached through which interfaces
  - when frame received, switch "learns" location of sender: incoming LAN segment
  - records sender/location pair in switch table



MAC addr	interface	TTL
A	*1*	*60*

Switch table (initially empty)

Switch: frame filtering/forwarding

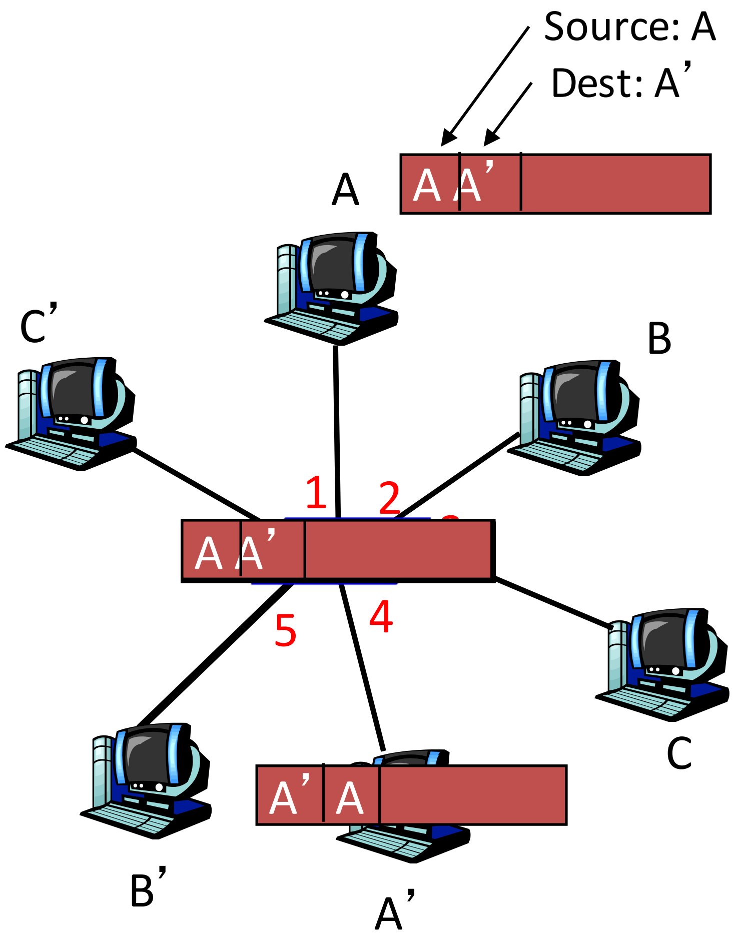
When frame received:

```
1. record link associated with sending host
2. index switch table using MAC dest address
3. if entry found for destination
      then {
     if dest on segment from which frame arrived
          then drop the frame
         else forward the frame on interface indicated
      }
     else flood
```

forward on all but the interface on which the frame arrived

Self-learning, forwarding: example

- frame destination unknown: *flood*
- destination A location known: selective send

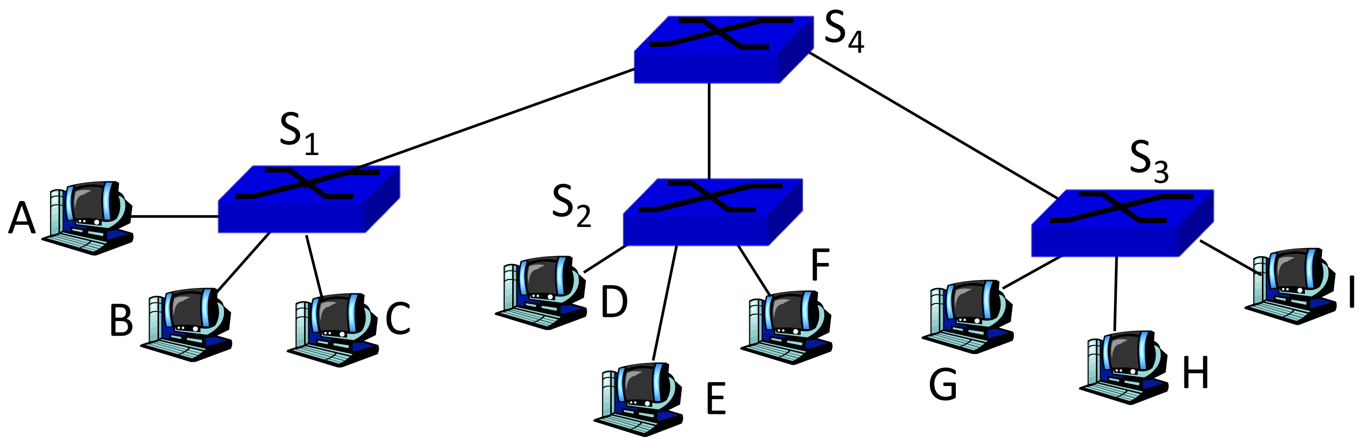


MAC addr	interface	TTL
A	*1*	*60*
A’	*4*	*60*

Switch table (initially empty)

Interconnecting switches

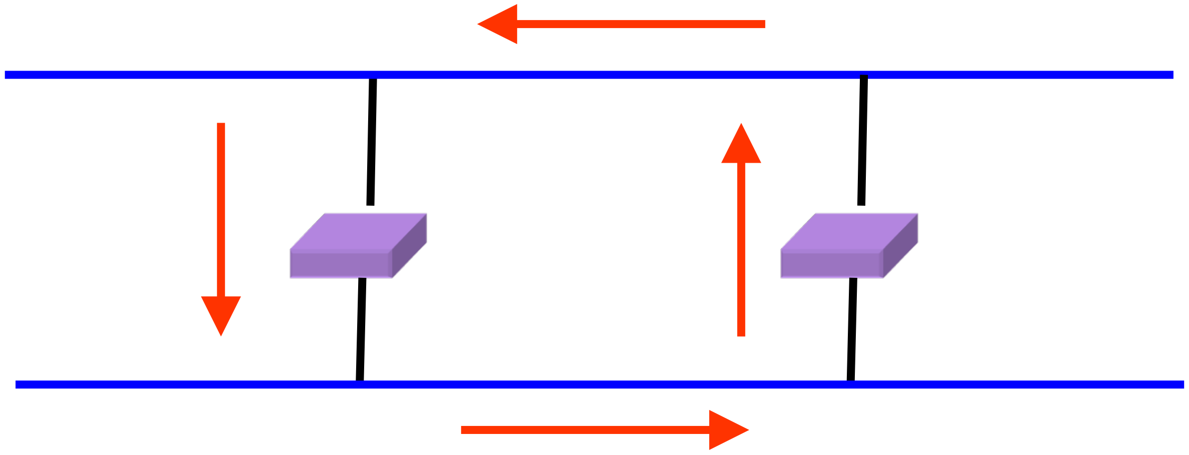
- switches can be connected together



- Q: sending from A to G - how does S_1 know to forward frame destined to F via S_4 and S_3?
- A: self learning! (works exactly the same as in single-switch case – flood/forward/drop)

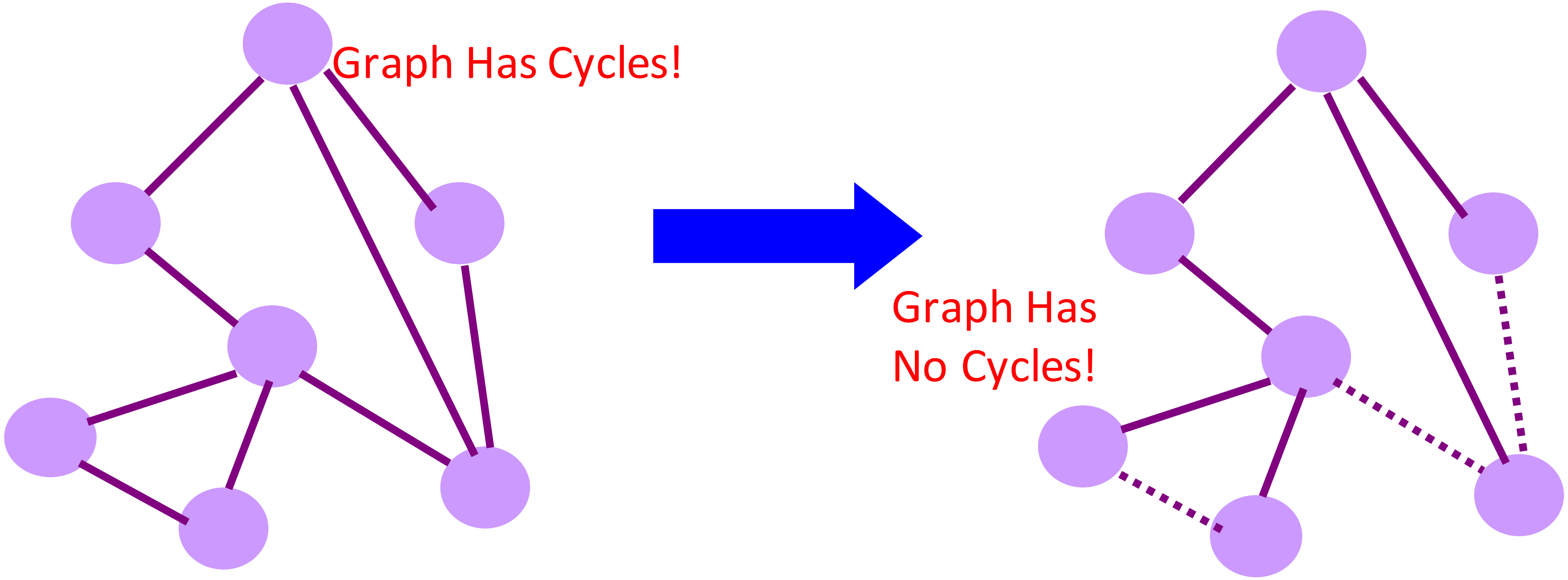
Flooding Can Lead to Loops

- Flooding can lead to forwarding loops
  - E.g., if the network contains a cycle of switches
  - “Broadcast storm”



Solution: Spanning Trees

- Ensure the forwarding topology has no loops
  - Avoid using some of the links when flooding
  - … to prevent loop from forming
- Spanning tree
  - Sub-graph that covers all vertices but *contains no cycles*
  - Links not in the spanning tree do not forward frames

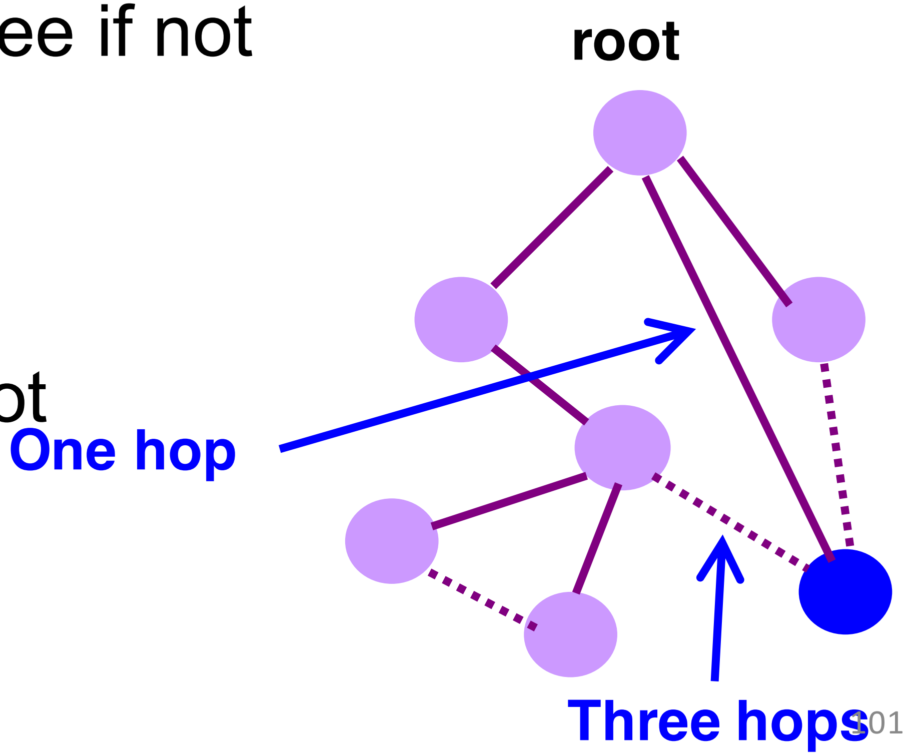


What Do We Know?

- Shortest paths to (or from) a node form a tree
- So, algorithm has two aspects :
  - Pick a root
  - Compute shortest paths to it
- Only keep the links on shortest-path

Constructing a Spanning Tree

- Switches need to elect a root
  - The switch w/ smallest identifier (MAC addr)
- Each switch determines if each interface is on the shortest path from the root
  - Excludes it from the tree if not
- Messages (Y, d, X)
  - From node X
  - Proposing Y as the root
  - And the distance is d

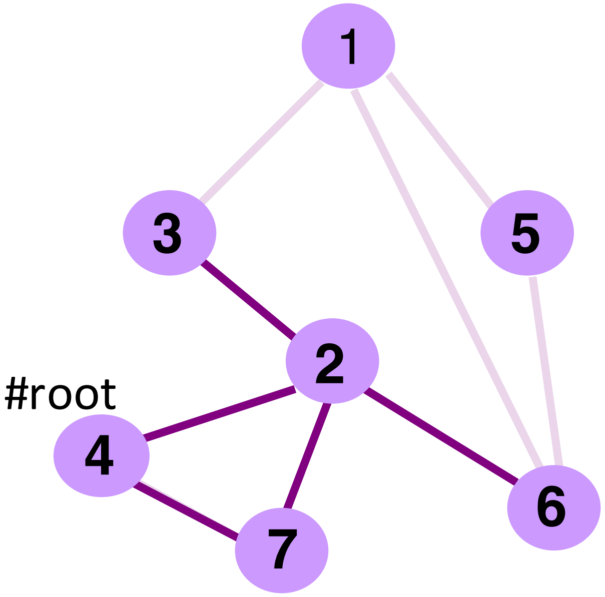


Steps in Spanning Tree Algorithm

- Initially, each switch proposes itself as the root
  - Switch sends a message out every interface
  - … proposing itself as the root with distance 0
  - Example: switch X announces (X, 0, X)
- Switches update their view of the root
  - Upon receiving message (Y, d, Z) from Z, check Y's id
  - If new id smaller, start viewing that switch as root
- Switches compute their distance from the root
  - Add 1 to the distance received from a neighbor
  - Identify interfaces not on shortest path to the root
  - … and exclude them from the spanning tree
- If root or shortest distance to it changed, "flood" updated message (Y, d+1, X)

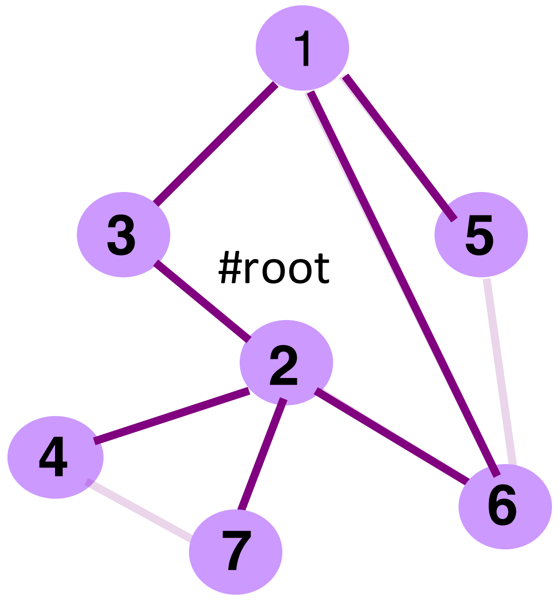
Example From Switch #4's Viewpoint

- Switch #4 thinks it is the root
  - Sends (4, 0, 4) message to 2 and 7
- Then, switch #4 hears from #2
  - Receives (2, 0, 2) message from 2
  - … and thinks that #2 is the root
  - And realizes it is just one hop away
- Then, switch #4 hears from #7
  - Receives (2, 1, 7) from 7
  - And realizes this is a longer path
  - So, prefers its own one-hop path
  - And removes 4-7 link from the tree



Example From Switch #4’s Viewpoint

- Switch #2 hears about switch #1
  - Switch 2 hears (1, 1, 3) from 3
  - Switch 2 starts treating 1 as root
  - And sends (1, 2, 2) to neighbors
- Switch #4 hears from switch #2
  - Switch 4 starts treating 1 as root
  - And sends (1, 3, 4) to neighbors
- Switch #4 hears from switch #7
  - Switch 4 receives (1, 3, 7) from 7
  - And realizes this is a longer path
  - So, prefers its own three-hop path
  - And removes 4-7 link from the tree



Robust Spanning Tree Algorithm

- Algorithm must react to failures
  - Failure of the root node
    - Need to elect a new root, with the next lowest identifier
  - Failure of other switches and links
    - Need to recompute the spanning tree
- Root switch continues sending messages
  - Periodically reannouncing itself as the root (1, 0, 1)
  - Other switches continue forwarding messages
- Detecting failures through timeout (soft state)
  - If no word from root, times out and claims to be the root
  - Delay in reestablishing spanning tree is ***major problem***
  - Work on rapid spanning tree algorithms…

Topic 3: Summary

- principles behind data link layer services:
  - error detection, correction
  - sharing a broadcast channel: multiple access
  - link layer addressing
- instantiation and implementation of various link layer technologies
  - Ethernet
  - switched LANS
  - WiFi
- algorithms
  - Binary Exponential Backoff
  - Spanning Tree

Topic 4: Network Layer

Our goals:

- understand principles behind network layer services:
  - network layer service models
  - forwarding versus routing (versus switching)
  - how a router works
  - routing (path selection)
  - IPv6
- For the most part, the Internet is our example – again.

Name: a *something*

Address: Where a *something* is

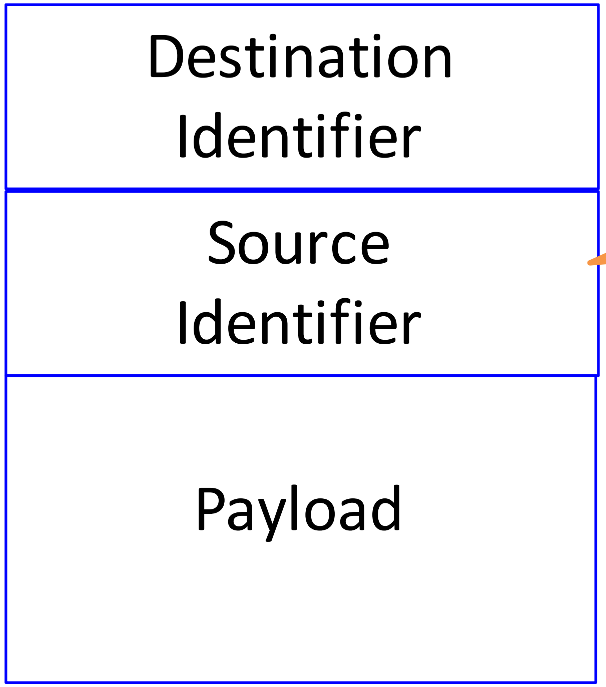
Routing: How do I get to the *something*

Addressing (at a conceptual level)

- Assume all hosts have unique IDs
- No particular structure to those IDs
- Later in topic I will talk about real IP addressing
- Do I route on location or identifier?
- If a host moves, should its address change?
  - If not, how can you build scalable Internet?
  - If so, then what good is an address for identification?

Packets (at a conceptual level)

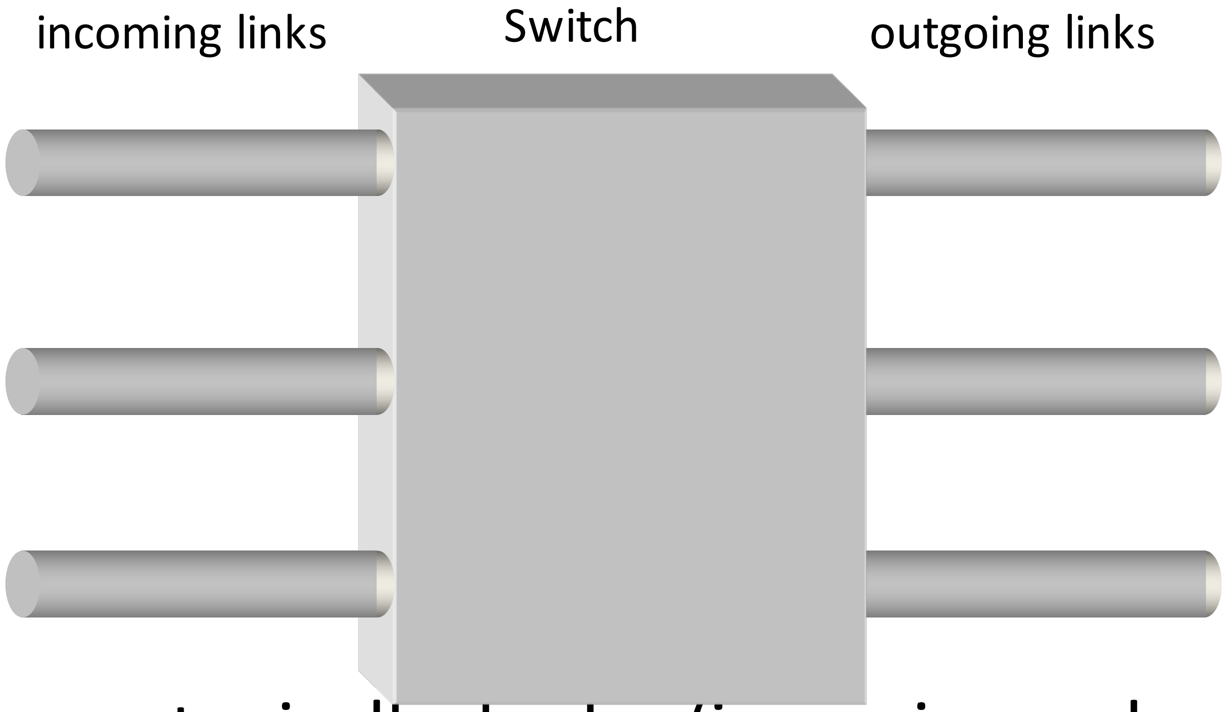
- Assume packet headers contain:
  - Source ID, Destination ID, and perhaps other information



Why include this?

Switches/Routers

- Multiple ports (attached to other switches or hosts)

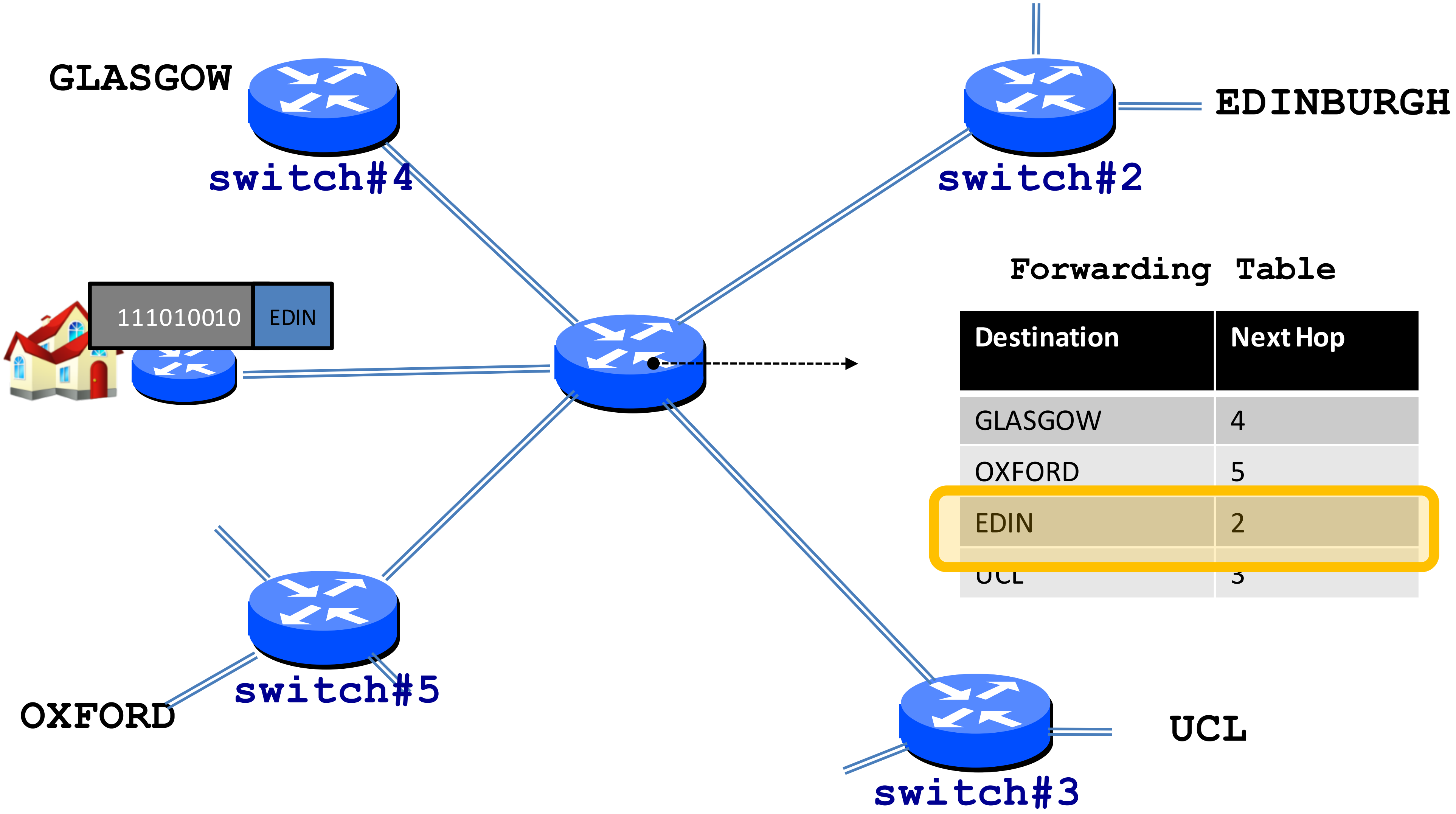


- Ports are typically duplex (incoming and outgoing)

A Variety of Networks

- ISPs: carriers
  - Backbone
  - Edge
  - Border (to other ISPs)
- Enterprises: companies, universities
  - Core
  - Edge
  - Border (to outside)
- Datacenters: massive collections of machines
  - Top-of-Rack
  - Aggregation and Core
  - Border (to outside)

Switches forward packets



Forwarding Table

Destination	Next Hop
GLASGOW	4
OXFORD	5
EDIN	2
UCL	3

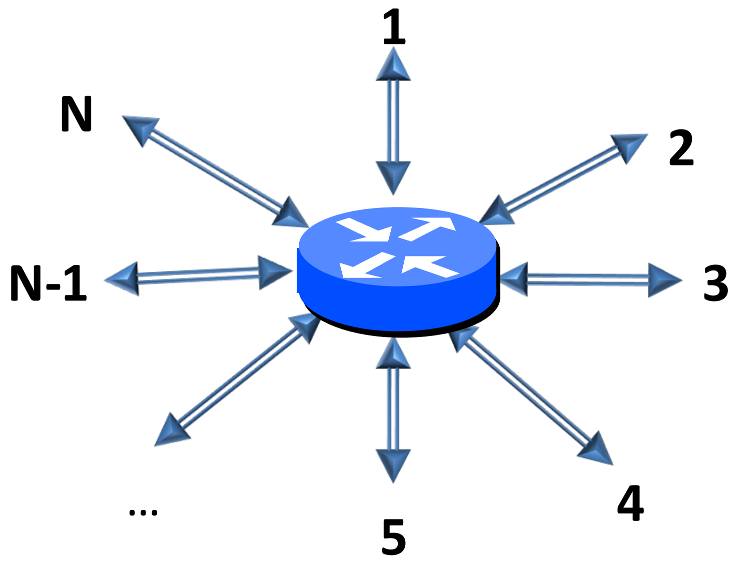
Forwarding Decisions

- When packet arrives..
  - Must decide which outgoing port to use
  - In single transmission time
  - Forwarding decisions must be ***simple***
- Routing state dictates where to forward packets
  - Assume decisions are **deterministic**
- *Global routing state* means collection of routing state in each of the routers
  - Will focus on where this routing state comes from
  - But first, a few preliminaries….

Forwarding vs Routing

- Forwarding: “data plane”
  - Directing a data packet to an outgoing link
  - Individual router using routing state
- Routing: “control plane”
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Jointly creating the routing state
- Two very different timescales….

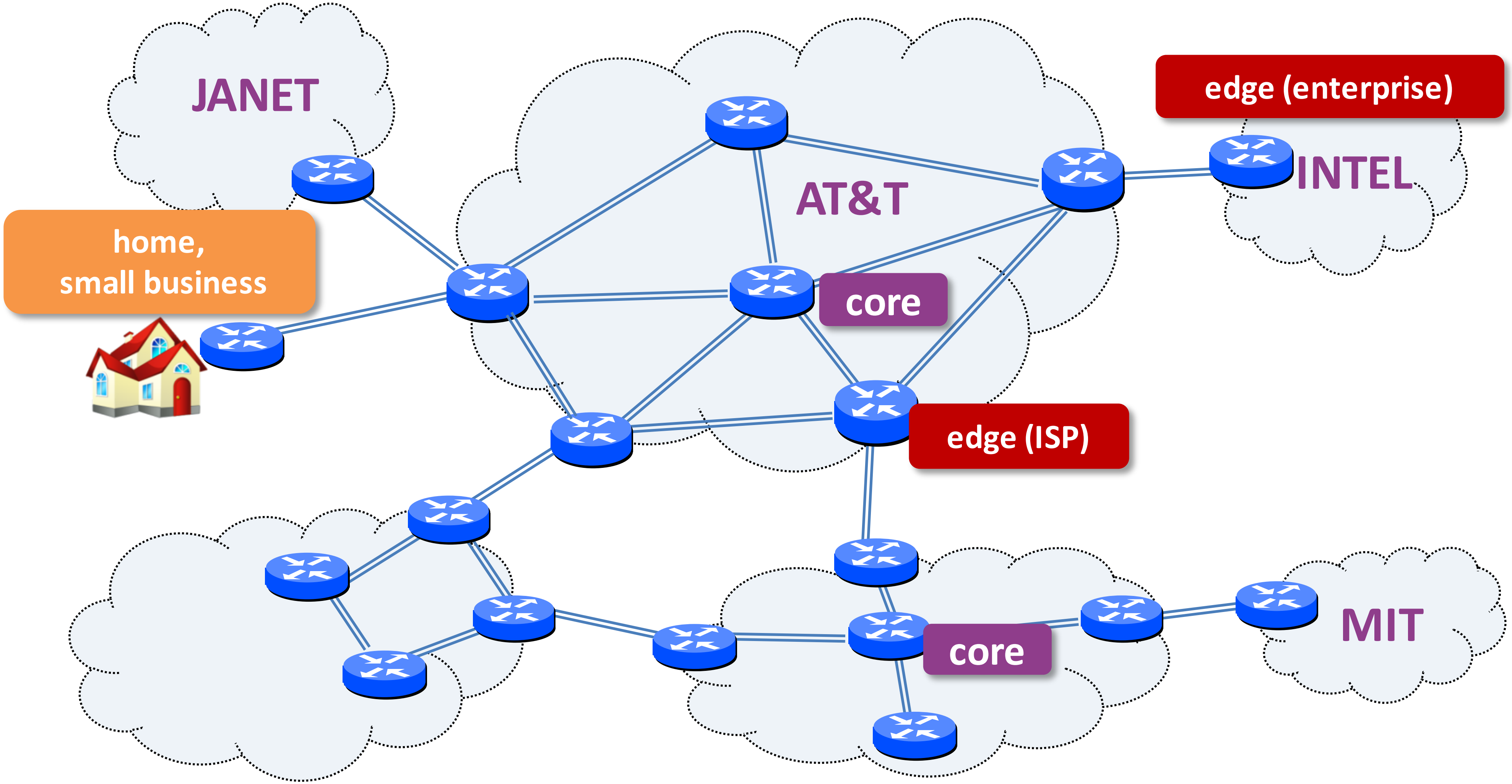
Router definitions



R *bits/sec*

- N = number of external router "ports"
- R = speed ("line rate") of a port
- Router capacity = N x R

Networks and routers



JANET

AT&T

edge (enterprise)

INTEL

home,
small business

core

edge (ISP)

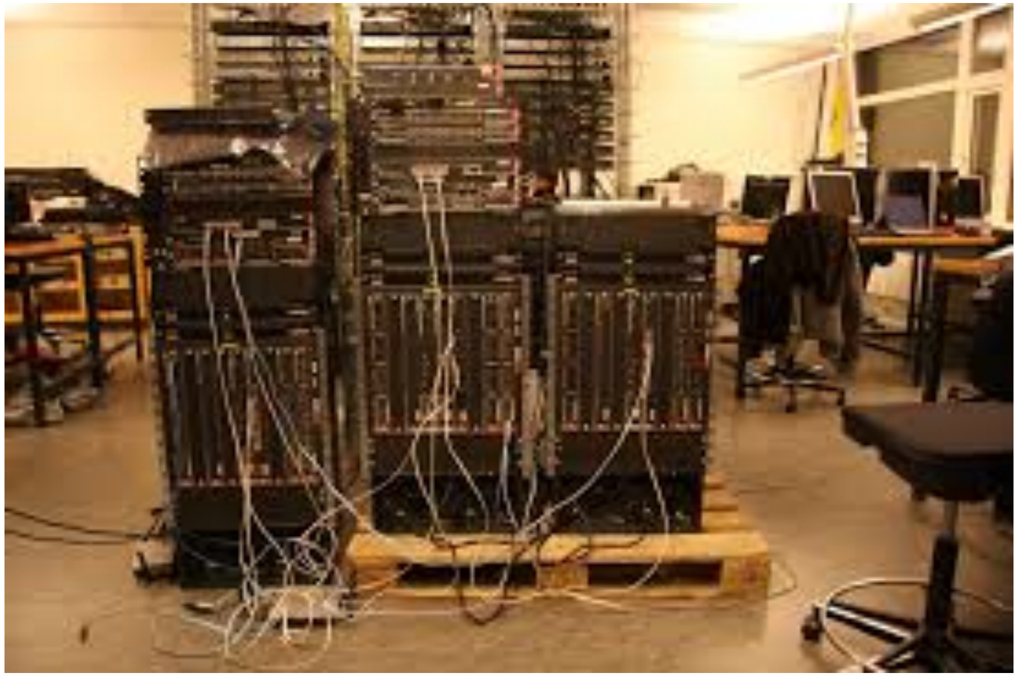
core

MIT

Examples of routers (core)

Cisco CRS

- R=10/40/100 Gbps
- NR = 922 Tbps
- Netflix: 0.7GB per hour (1.5Mb/s)
- ~600 million concurrent Netflix users



72 racks, >1MW

Examples of routers (edge)

Cisco ASR

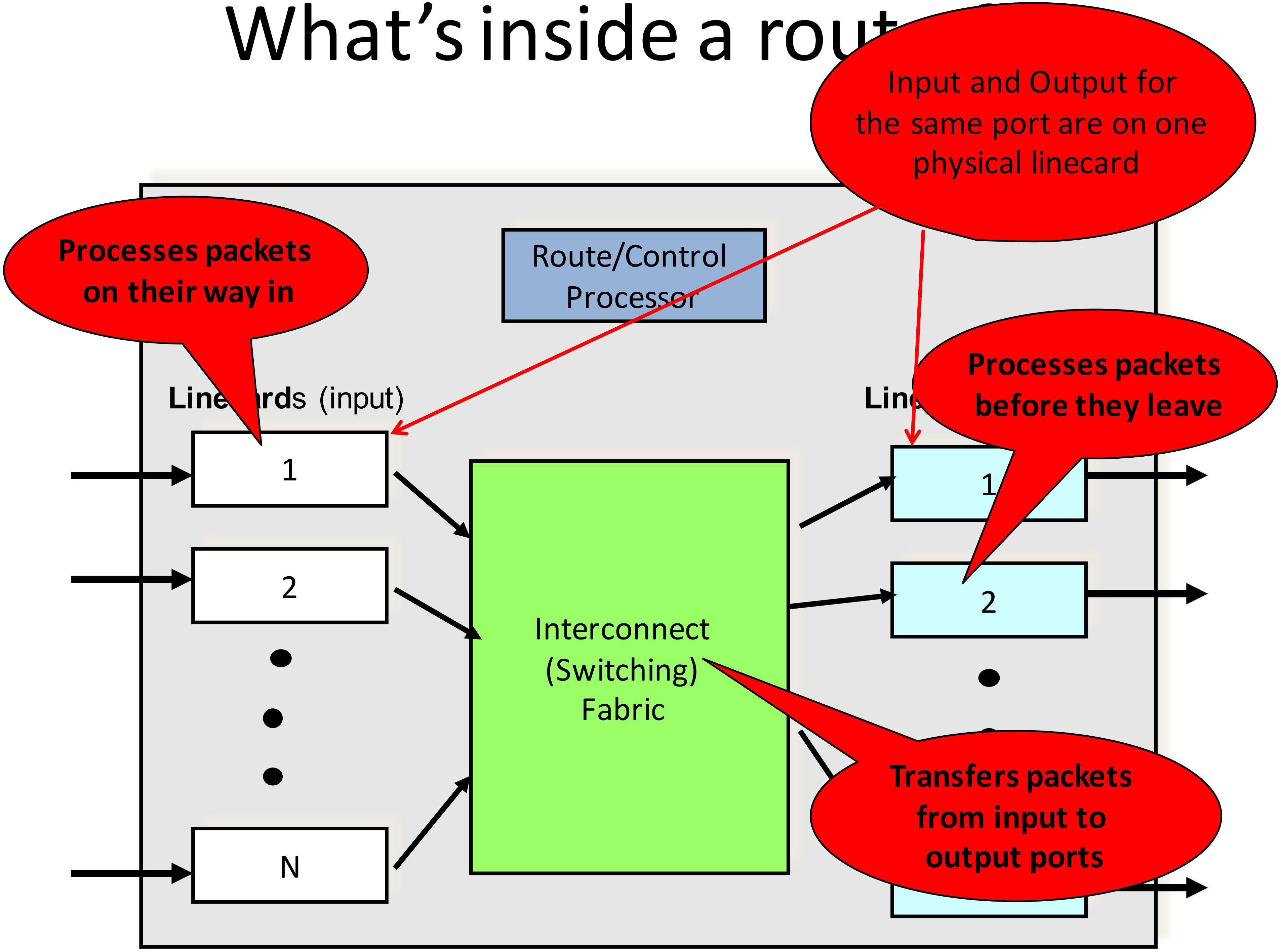
- R=1/10/40 Gbps
- NR = 120 Gbps

Examples of routers (small business)

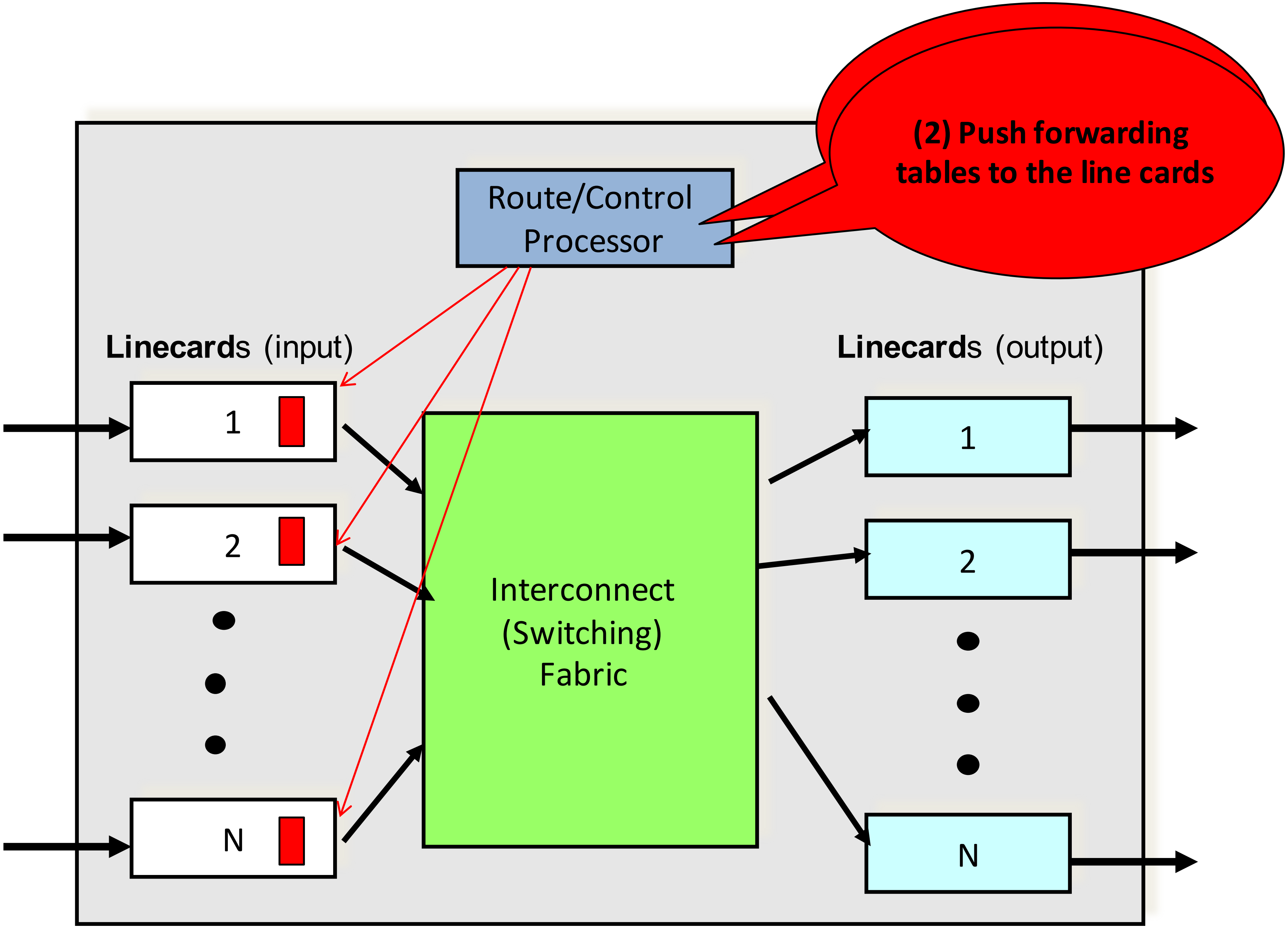
Cisco 3945E

- R = 10/100/1000 Mbps
- NR < 10 Gbps

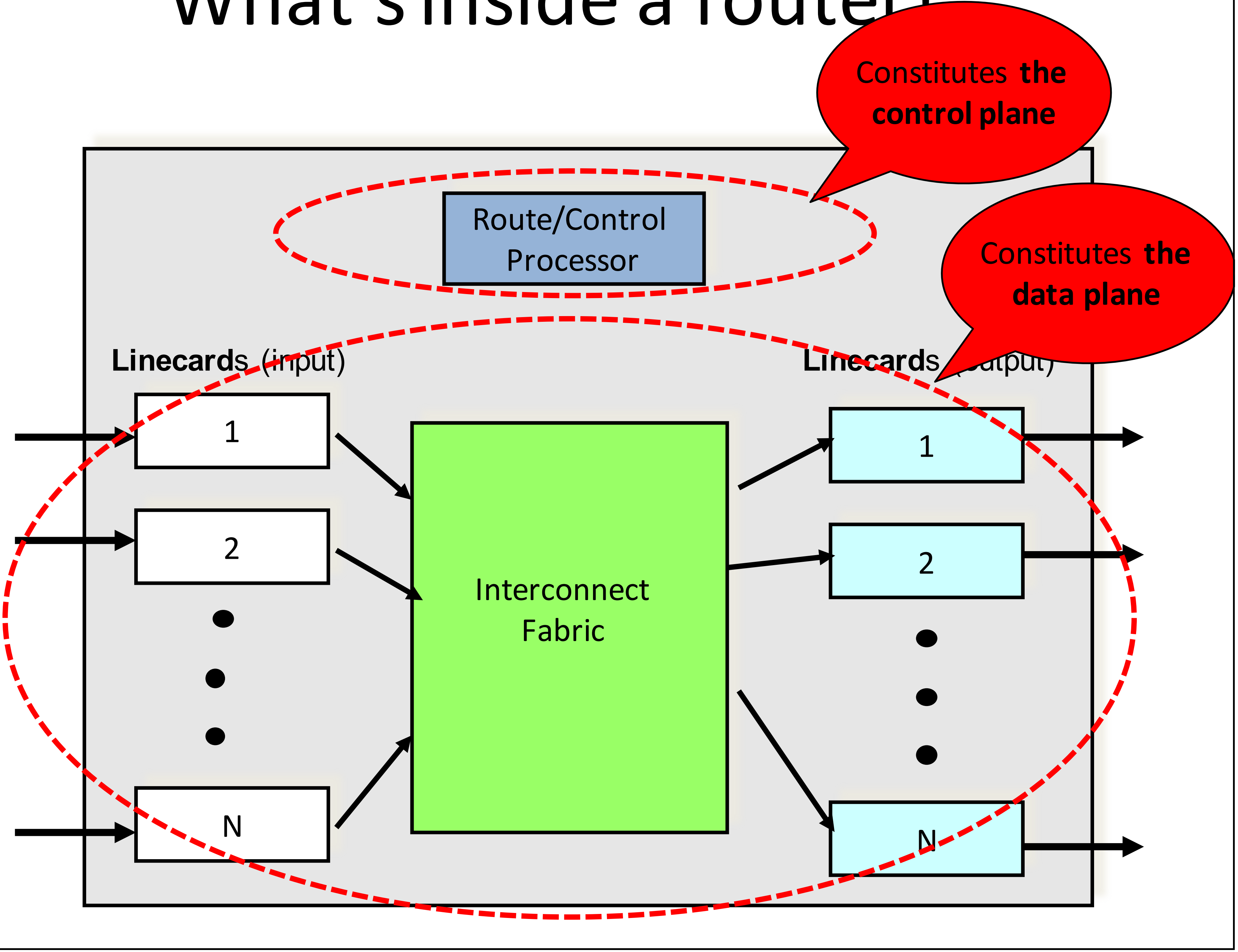
What's inside a rout


Input and Output for the same port are on one physical linecard
Processes packets on their way in
Route/Control Processor
Line rds (input)
Lin
Processes packets before they leave
1
2
N
1
2
Interconnect (Switching) Fabric
Transfers packets from input to output ports

What's inside a router?

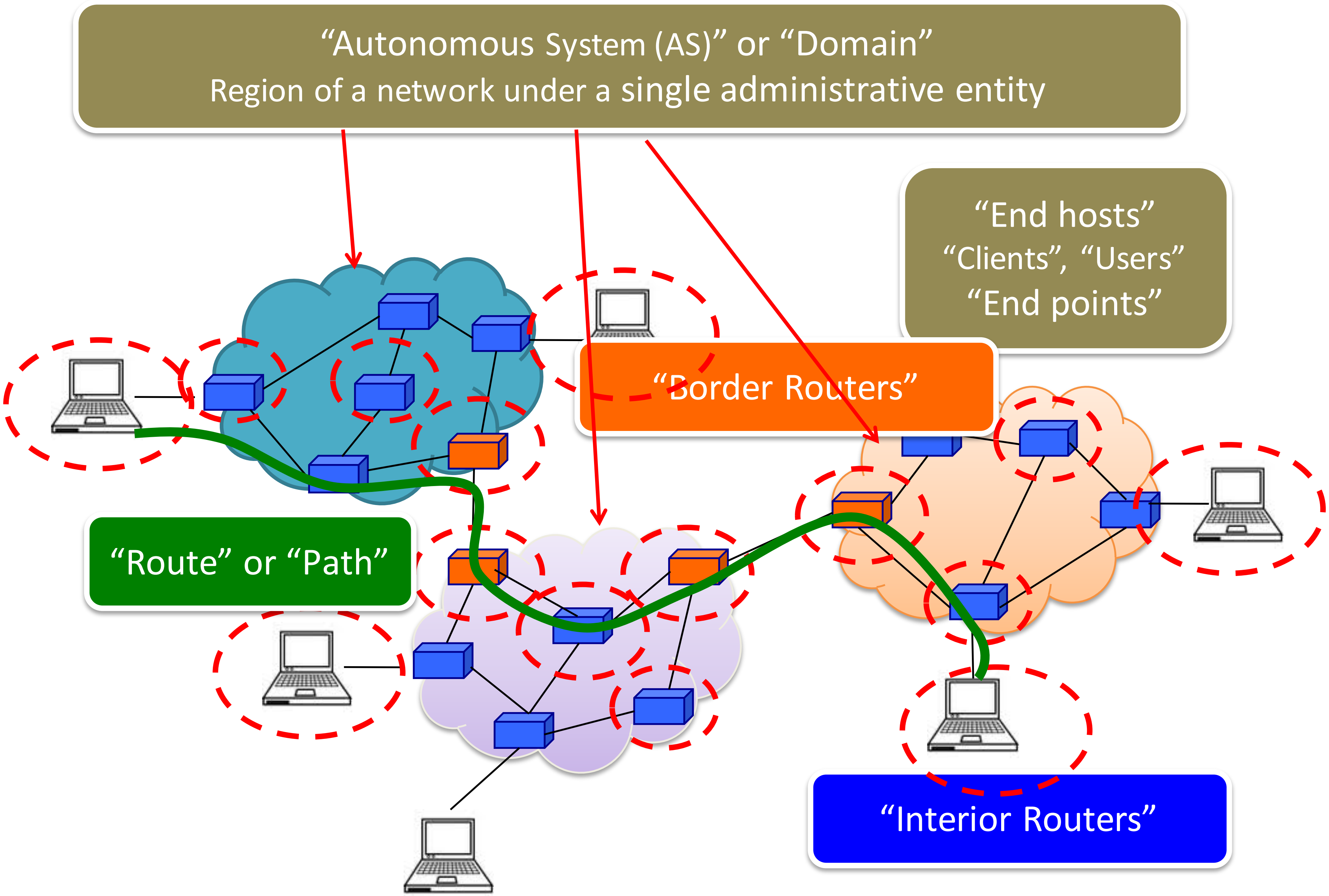

(2) Push forwarding tables to the line cards
Route/Control Processor
Linecards (input)
Linecards (output)
1
2
N
Interconnect (Switching) Fabric
1
2
N

What’s inside a router?



Constitutes **the control plane**

Constitutes **the data plane**



"Autonomous System (AS)" or "Domain"

Region of a network under a single administrative entity

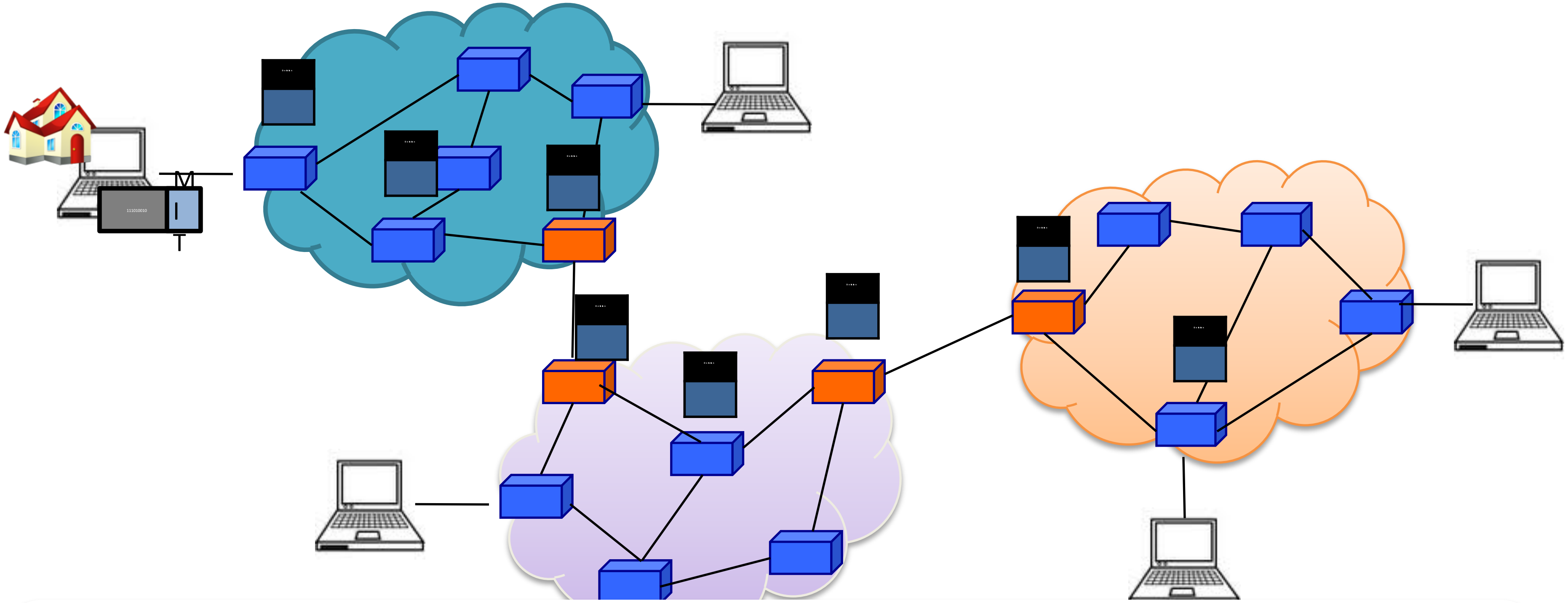
"End hosts"
"Clients", "Users"
"End points"

"Border Routers"

"Route" or "Path"

"Interior Routers"

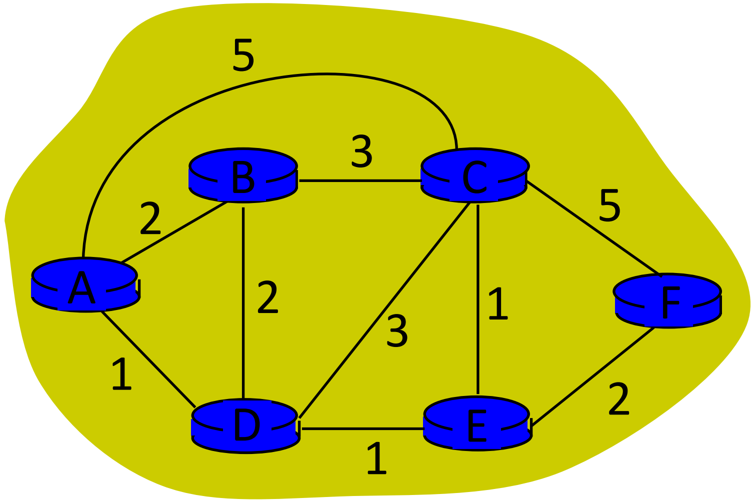
Context and Terminology



Internet routing protocols are responsible for constructing and updating the forwarding tables at routers

Routing Protocols

- Routing protocols implement the core function of a network
  - Establish paths between nodes
  - Part of the network's "control plane"
- Network modeled as a graph
  - Routers are graph vertices
  - Links are edges
  - Edges have an associated "cost"
    - e.g., distance, loss



- Goal: compute a "good" path from source to destination
  - "good" usually means the shortest (least cost) path

Internet Routing

- Internet Routing works at two levels

- Each AS runs an intra-domain routing protocol that establishes routes within its domain
  - (AS -- region of network under a single administrative entity)
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Distance Vector, e.g., Routing Information Protocol (RIP)

- ASes participate in an inter-domain routing protocol that establishes routes between domains
  - Path Vector, e.g., Border Gateway Protocol (BGP)

Addressing (for now)

- Assume each host has a unique ID (address)

- No particular structure to those IDs

- Later in course will talk about real IP addressing

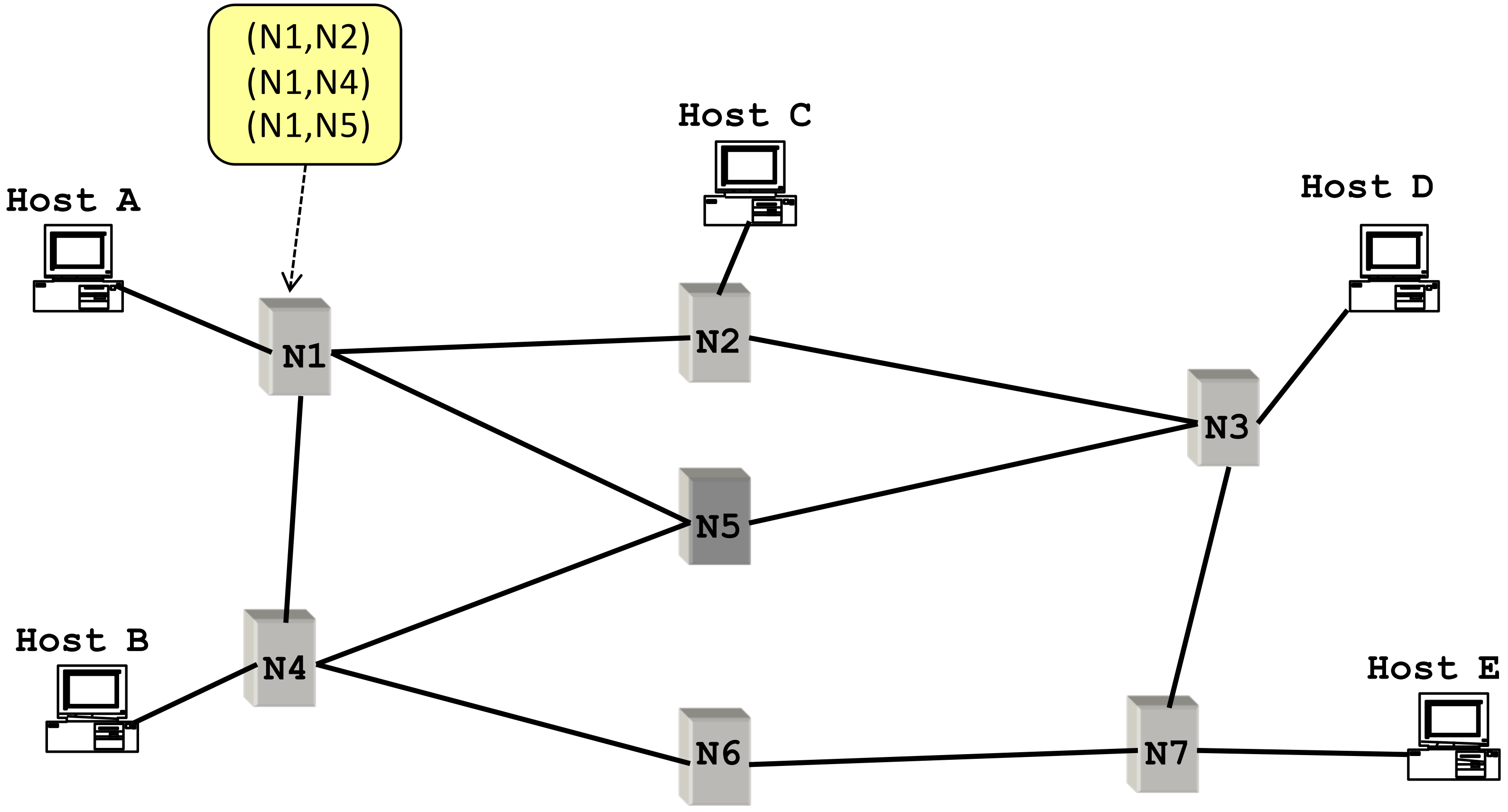
Outline

- Link State
- Distance Vector
- Routing: goals and metrics (if time)

Link-State

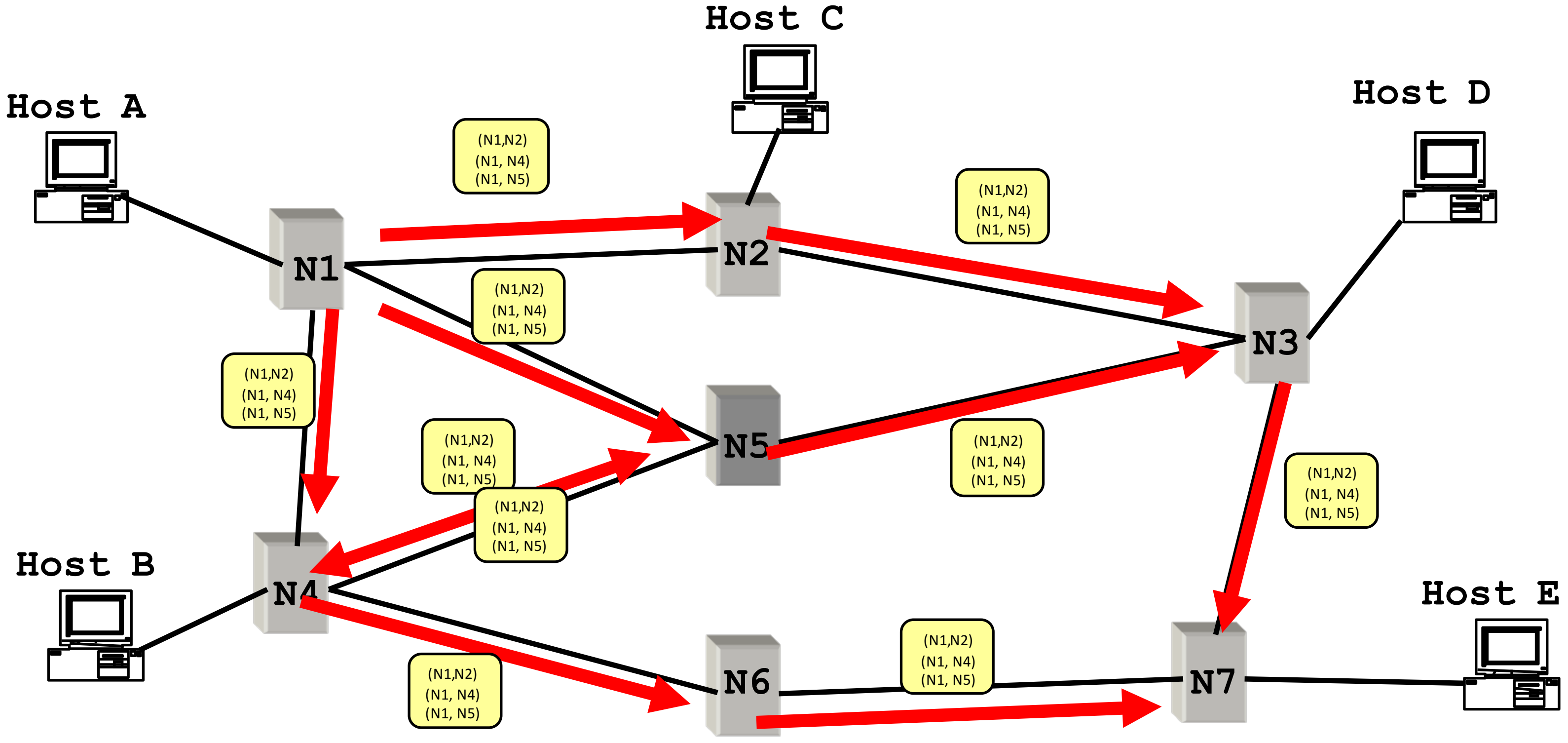
Link State Routing

- Each node maintains its local "link state" (LS)
  - i.e., a list of its directly attached links and their costs



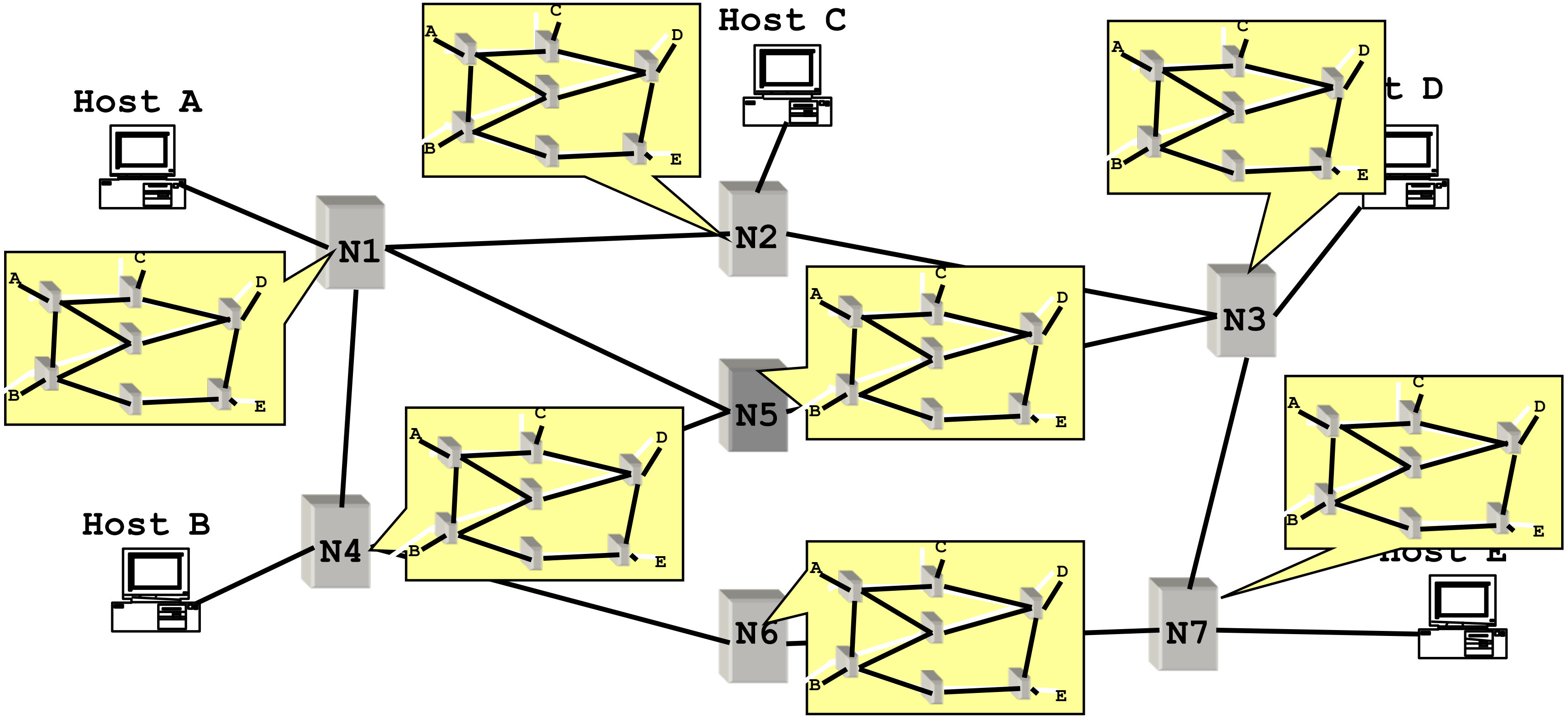
Link State Routing

- Each node maintains its local "link state" (LS)
- Each node floods its local link state
  - on receiving a new LS message, a router forwards the message to all its neighbors other than the one it received the message from



Link State Routing

- Each node maintains its local "link state" (LS)
- Each node floods its local link state
- Hence, each node learns the entire network topology
  - Can use Dijkstra's to compute the shortest paths between nodes



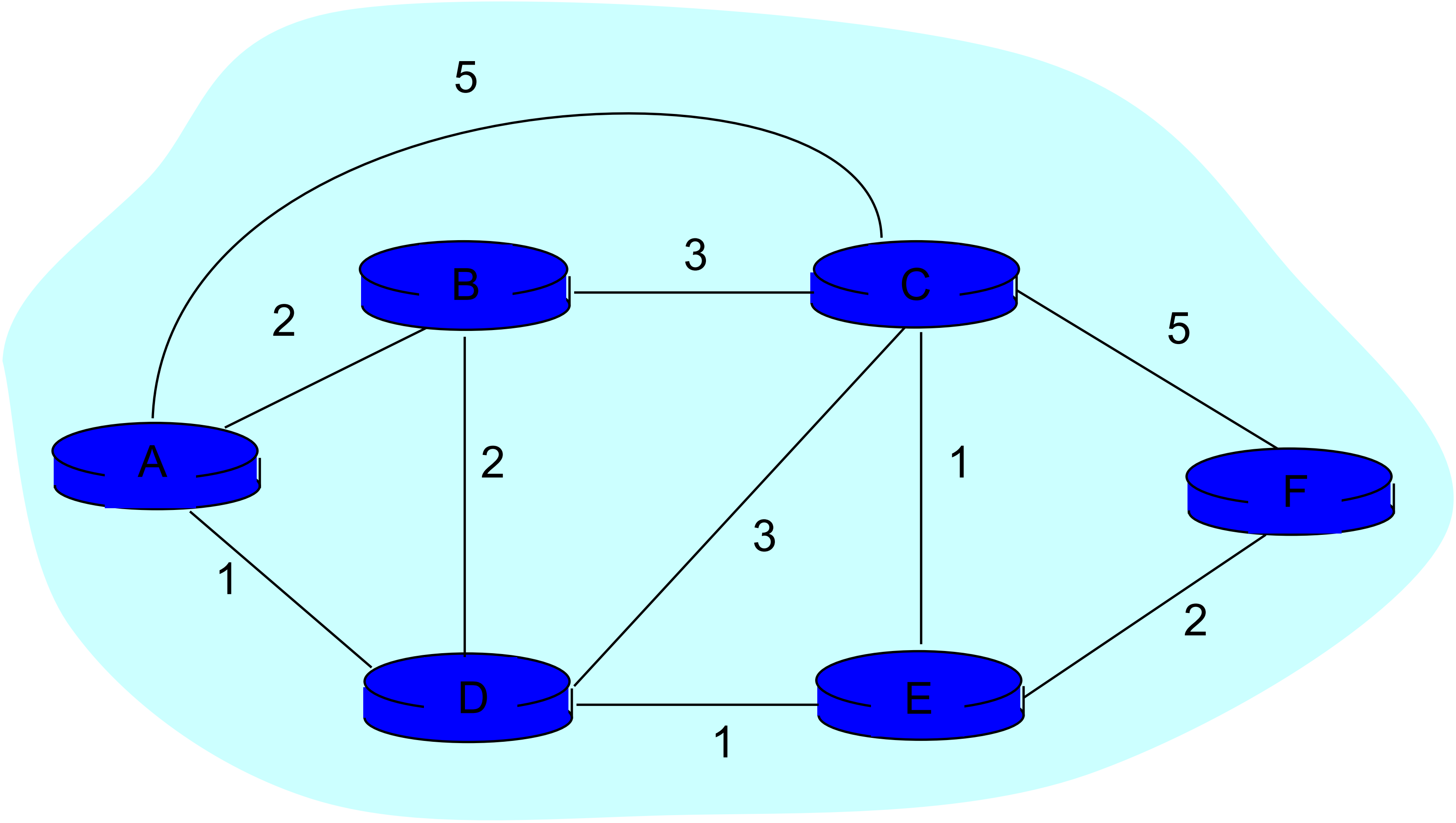
Dijkstra’s Shortest Path Algorithm

- INPUT:
  - Network topology (graph), with link costs

- OUTPUT:
  - Least cost paths from one node to all other nodes

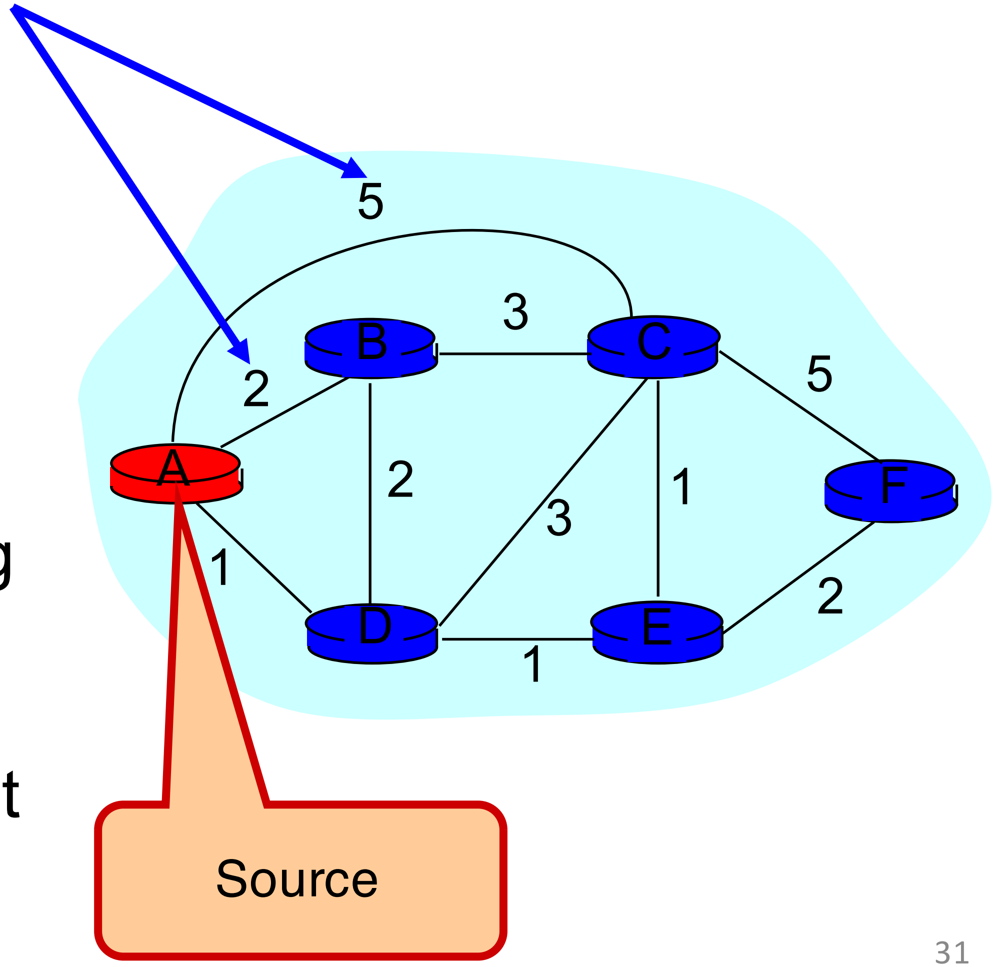
- Iterative: after *k* iterations, a node knows the least cost path to its *k* closest neighbors

Example



Notation

- c(i,j): link cost from node *i* to *j*; cost is infinite if not direct neighbors; **≥ 0**
- D(v): total cost of the current least cost path from source to destination *v*
- p(v): *v*’s predecessor along path from source to *v*
- S: set of nodes whose least cost path definitively known



Dijkstra’ s Algorithm

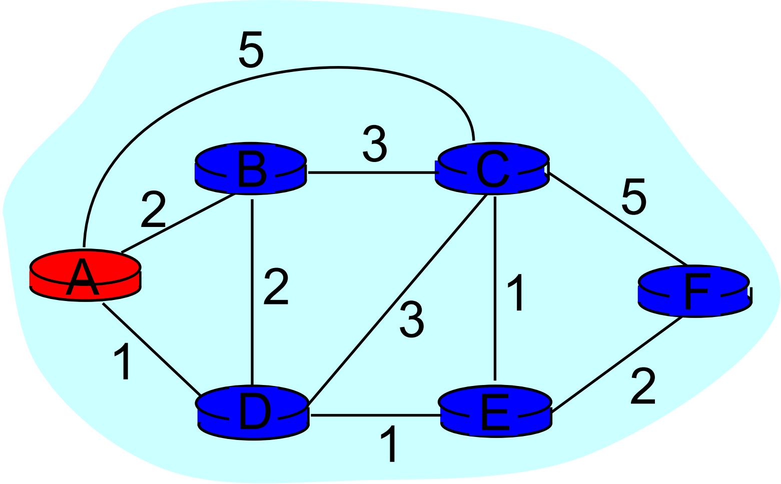
```
Initialization:
  S = {A};
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v);
      else D(v) = ∞;

Loop
    find w not in S such that D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent to w and not in S:
      if  D(w) + c(w,v) < D(v) then
        // w gives us a shorter path to v than we’ve found so far
        D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

- c(i,j): link cost from node *i* to *j*
- D(v): current cost source → *v*
- p(v): *v*’s predecessor along path from source to *v*
- S: set of nodes whose least cost path definitively known

Example: Dijkstra’s Algorithm

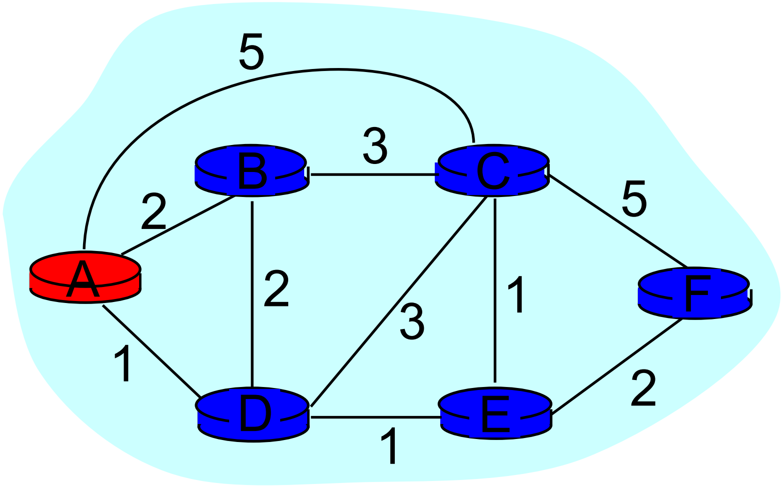
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						



```
Initialization:
  S = {A};
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v);
      else D(v) = ∞;
…
```

Example: Dijkstra’s Algorithm

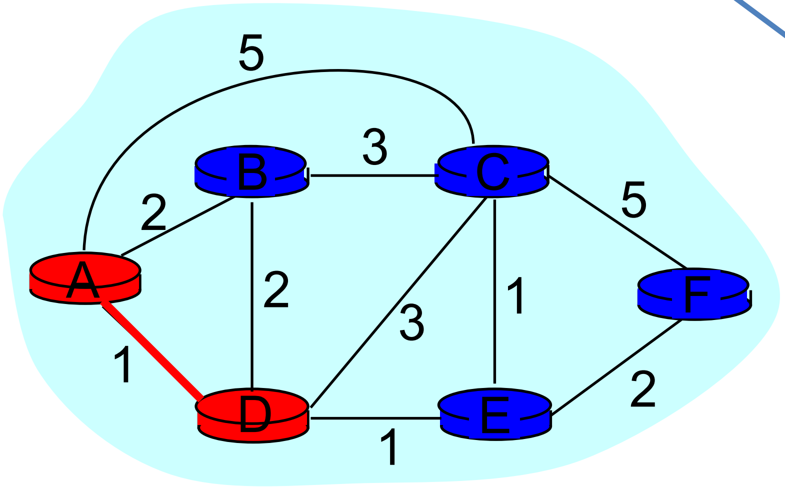
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						



```
…
Loop
  find w not in S s.t. D(w) is a minimum;
  add w to S;
  update D(v) for all v adjacent
    to w and not in S:
  If D(w) + c(w,v) < D(v) then
     D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra’s Algorithm

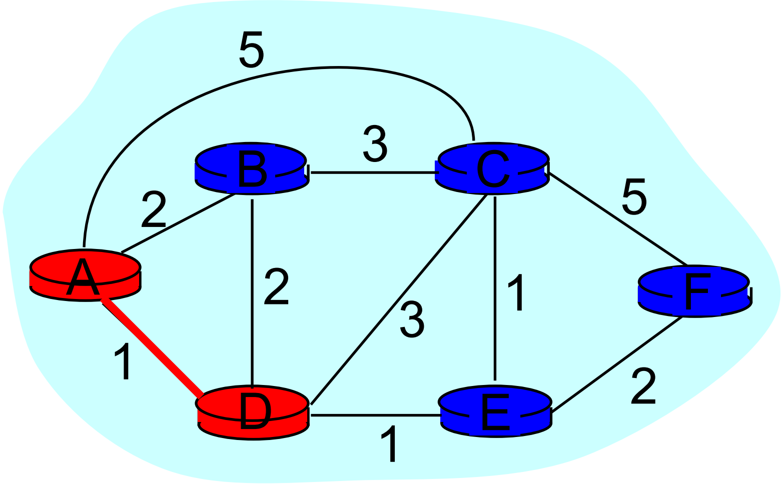
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD					
2						
3						
4						
5						



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent
      to w and not in S:
    If D(w) + c(w,v) < D(v) then
        D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra’s Algorithm

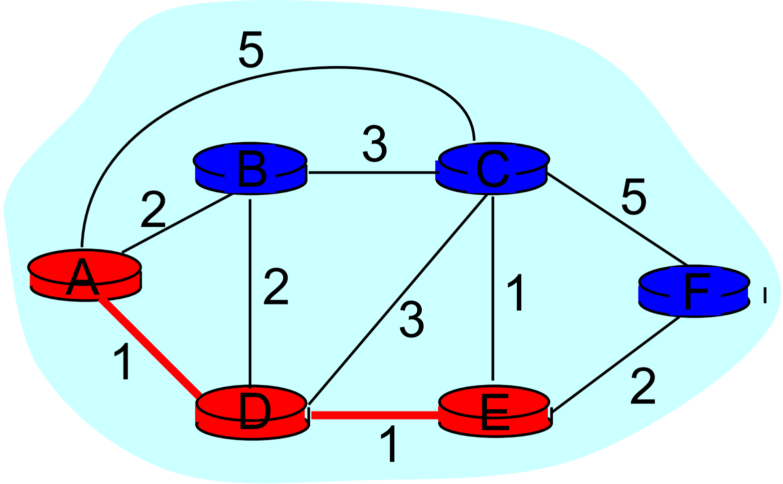
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2						
3						
4						
5						



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
  until all nodes in S;
```

Example: Dijkstra’s Algorithm

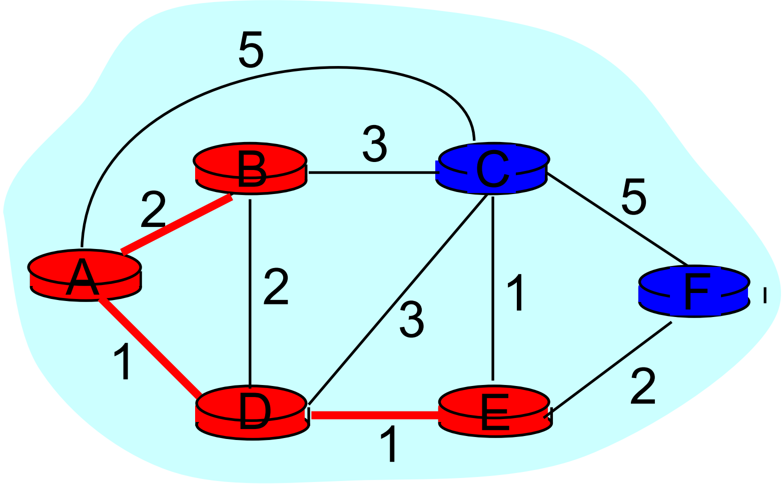
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3						
4						
5						



```
...
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
 until all nodes in S;
```

Example: Dijkstra’s Algorithm

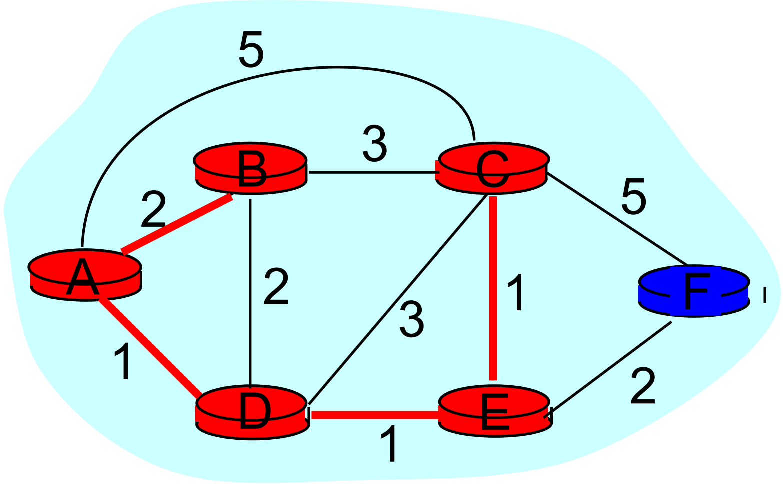
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4						
5						



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent
        to w and not in S:
    If D(w) + c(w,v) < D(v) then
        D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra’s Algorithm

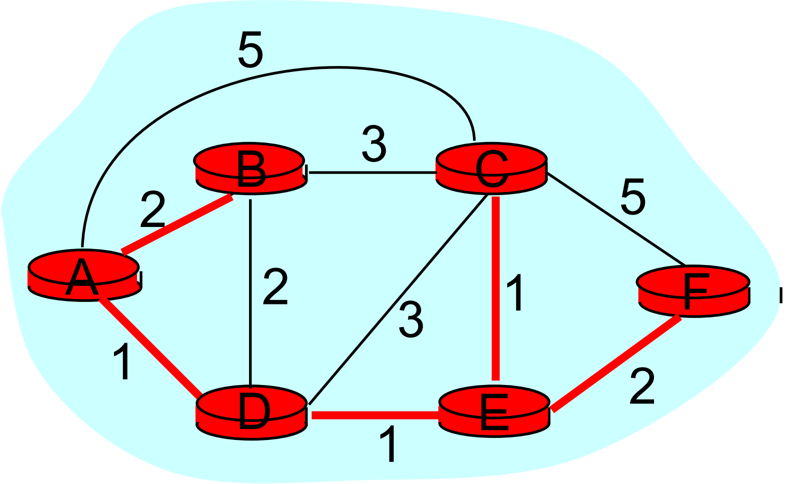
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5						



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent
      to w and not in S:
    If D(w) + c(w,v) < D(v) then
        D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra’s Algorithm

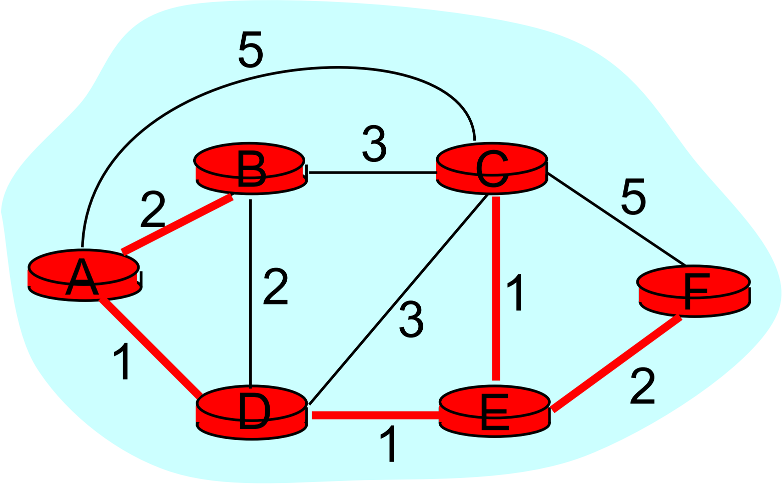
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					



```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
 until all nodes in S;
```

Example: Dijkstra’s Algorithm

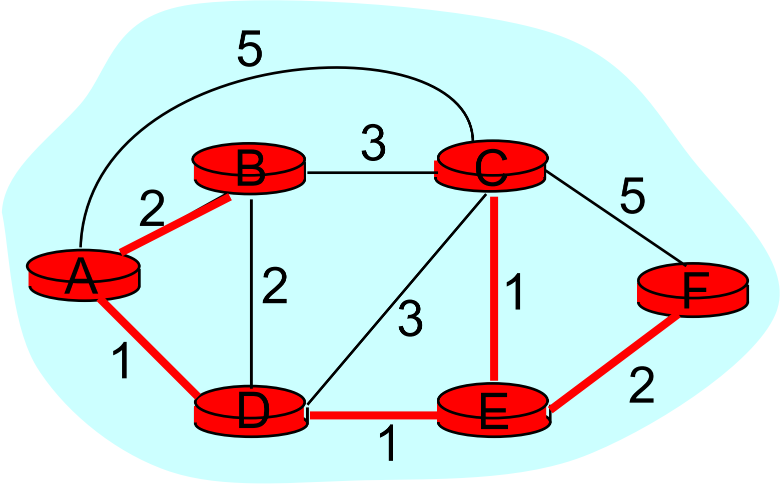
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					



To determine path A → C (say), work backward from C via p(v)

The Forwarding Table

- Running Dijkstra at node A gives the shortest path from A to all destinations
- We then construct the *forwarding table*



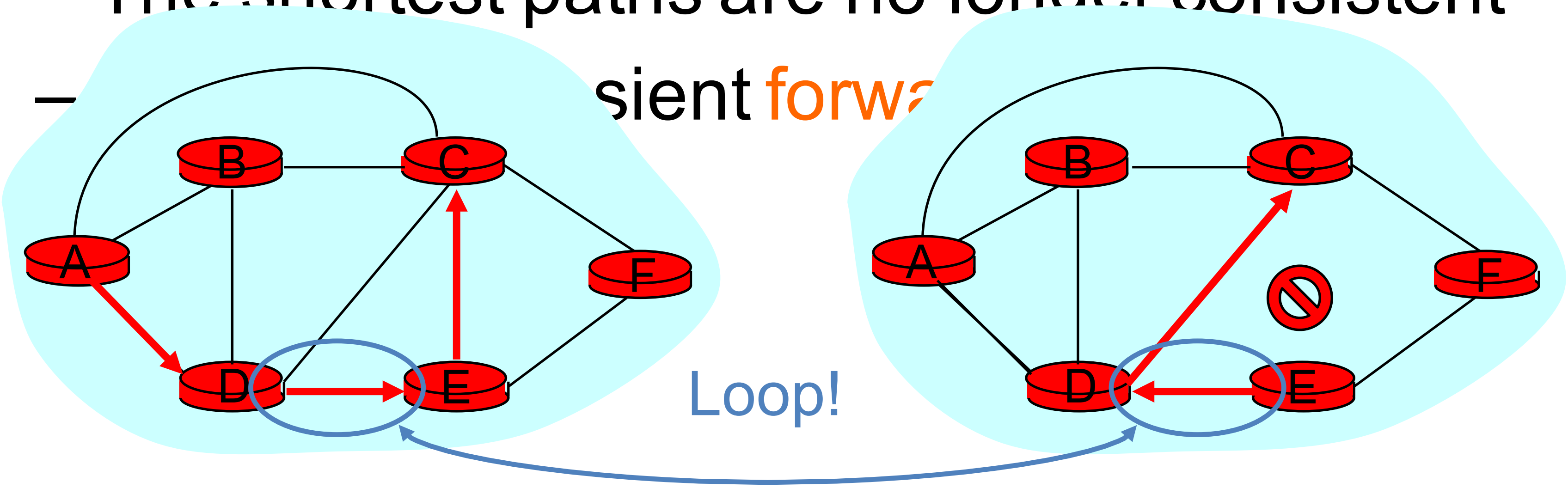
Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

Issue #1: Scalability

- How many messages needed to flood link state messages?
  - O(N x E), where N is #nodes; E is #edges in graph
- Processing complexity for Dijkstra's algorithm?
  - O(N^2), because we check all nodes w not in S at each iteration and we have O(N) iterations
  - more efficient implementations: O(N log(N))
- How many entries in the LS topology database? O(E)
- How many entries in the forwarding table? O(N)

Issue#2: Transient Disruptions

- Inconsistent link-state database
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - sient forwa



A and D think that this is the path to C

E thinks that this is the path to C

Distance Vector

Learn-By-Doing

Let’s try to collectively develop distance-vector routing from first principles

Experiment

- Your job: find the (route to) the youngest person in the room

- Ground Rules
  - **You may not** leave your seat, nor shout loudly across the class
  - **You may** talk with your immediate neighbors

    (N-S-E-W only)

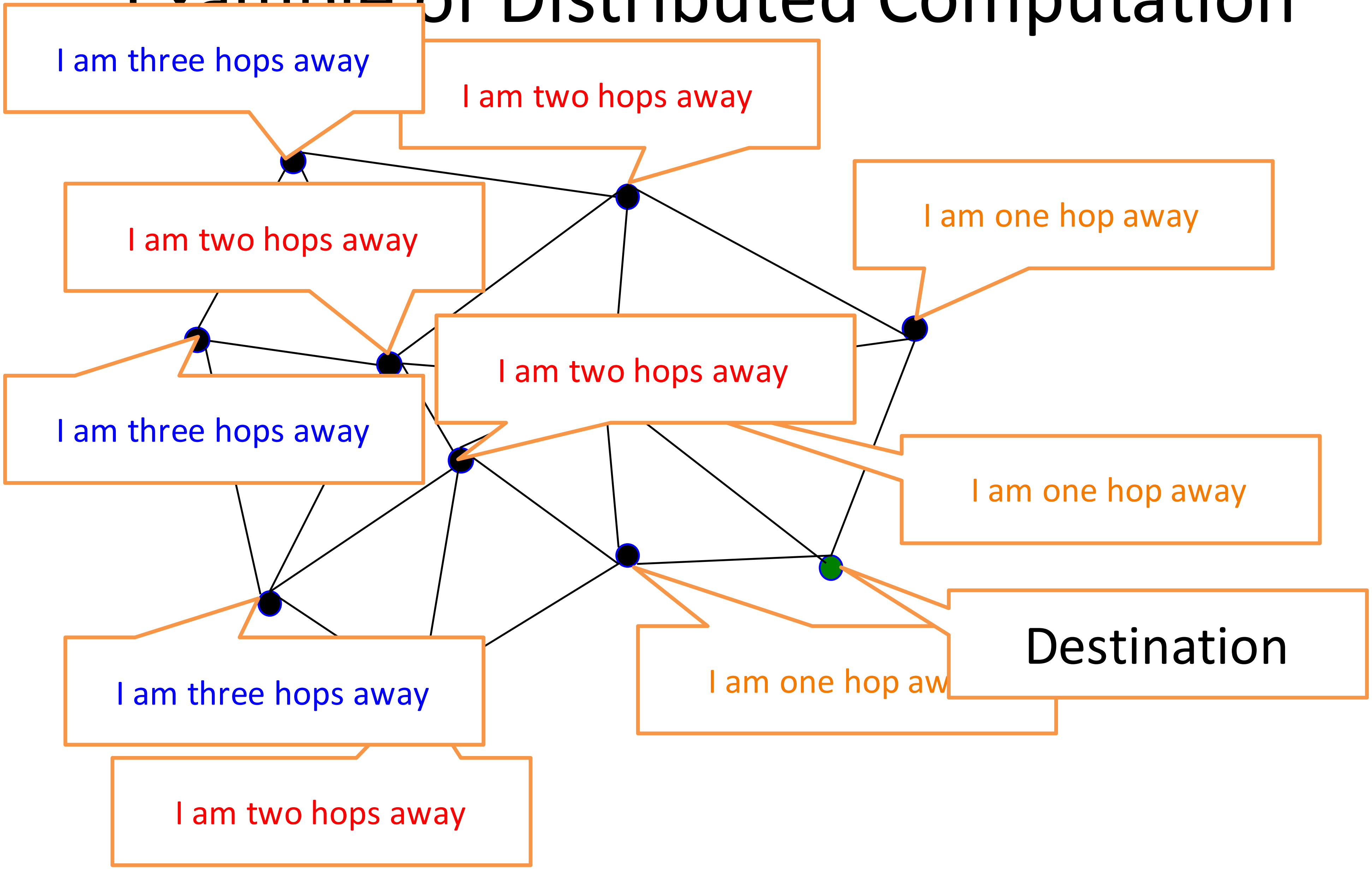
  (hint: “exchange updates” with them)

- At the end of 5 minutes, I will pick a victim and ask:
  - who is the youngest person in the room? (date&name)
  - which one of your neighbors first told you this info.?

Go!

Distance-Vector

Example of Distributed Computation



I am three hops away

I am two hops away

I am one hop away

I am two hops away

I am three hops away

I am two hops away

I am one hop away

I am three hops away

I am one hop aw

Destination

I am two hops away

Distance Vector Routing

- Each router knows the links to its neighbors
  - Does *not* flood this information to the whole network
- Each router has provisional “shortest path” to every other router
  - E.g.: Router A: “I can get to router B with cost 11”
- Routers exchange this distance vector information with their neighboring routers
  - Vector because one entry per destination
- Routers look over the set of options offered by their neighbors and select the best one
- Iterative process converges to set of shortest paths

A few other inconvenient truths

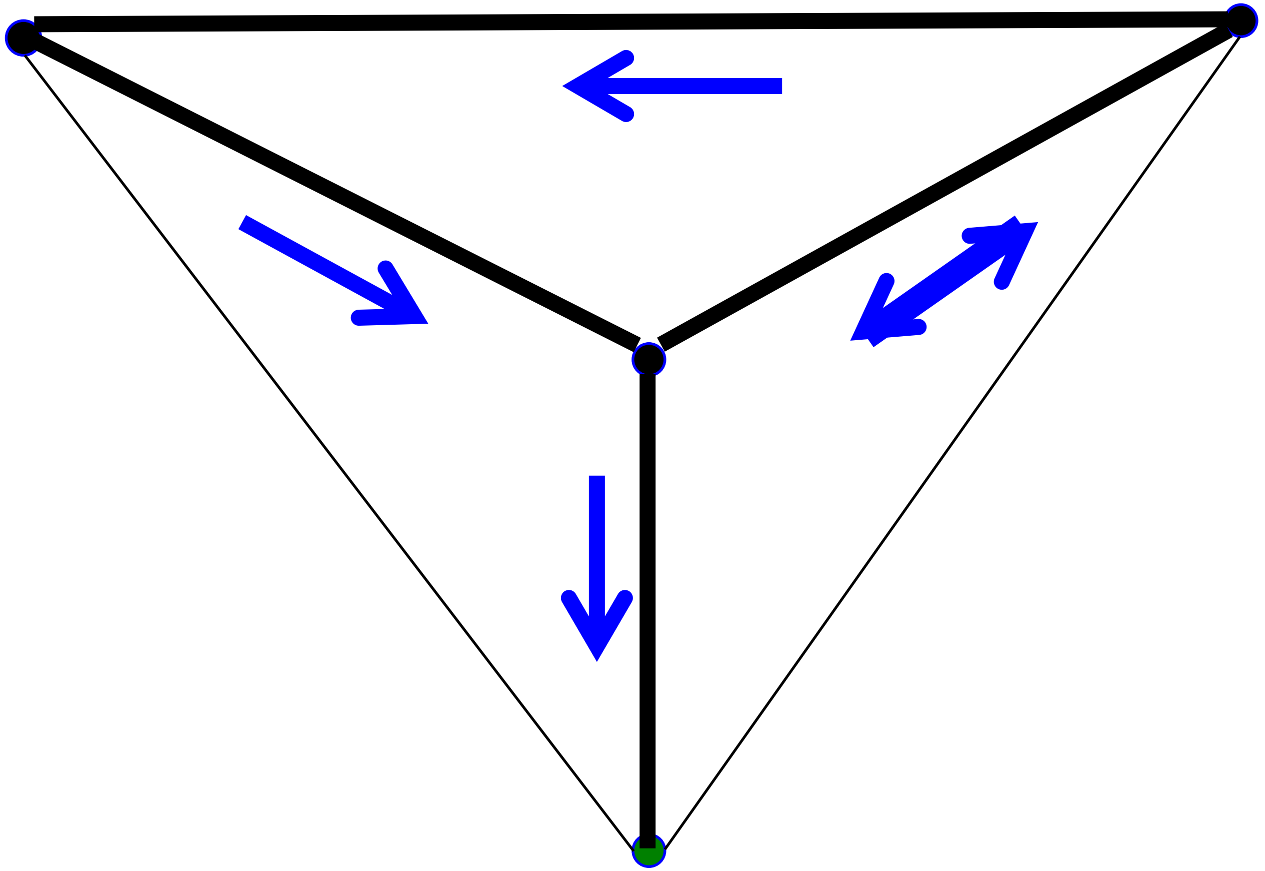
- What if we use a non-additive metric?
  - E.g., maximal capacity
- What if routers don’t use the same metric?
  - I want low delay, you want low loss rate?
- What happens if nodes lie?

Can You Use Any Metric?

- I said that we can pick any metric. Really?
- What about maximizing capacity?

What Happens Here?

Problem:"cost" does not change around loop



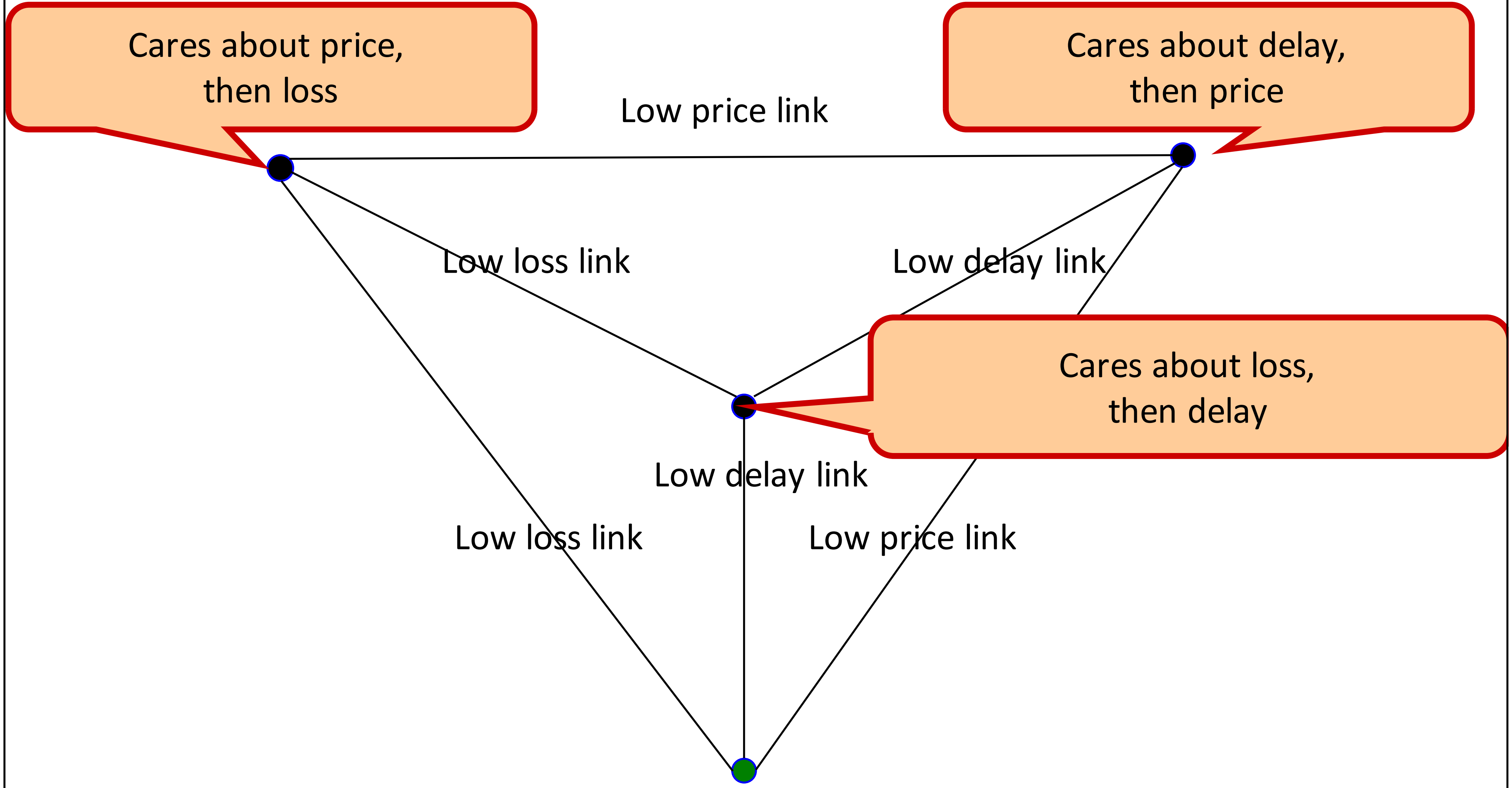
Additive measures avoid this problem!

No agreement on metrics?

- If the nodes choose their paths according to different criteria, then bad things might happen
- Example
  - Node A is minimizing latency
  - Node B is minimizing loss rate
  - Node C is minimizing price
- Any of those goals are fine, if globally adopted
  - Only a problem when nodes use different criteria

- Consider a routing algorithm where paths are described by delay, cost, loss

What Happens Here?



Cares about price, then loss

Cares about delay, then price

Cares about loss, then delay

Must agree on loop-avoiding metric

- When all nodes minimize same metric
- And that metric increases around loops
- Then process is guaranteed to converge

What happens when routers lie?

- What if a router claims a 1-hop path to everywhere?
- All traffic from nearby routers gets sent there
- How can you tell if they are lying?
- Can this happen in real life?
  - It has, several times….

Link State vs. Distance Vector

- Core idea
  - LS: tell all nodes about your immediate neighbors
  - DV: tell your immediate neighbors about (your least cost distance to) all nodes

Link State vs. Distance Vector

- LS: each node learns the complete network map; each node computes shortest paths independently and in parallel

- DV: no node has the complete picture; nodes cooperate to compute shortest paths in a distributed manner

→LS has higher messaging overhead

→LS has higher processing complexity

→LS is less vulnerable to looping

Link State vs. Distance Vector

Message complexity

- LS: O(NxE) messages;
  - N is #nodes; E is #edges
- DV: O(#Iterations x E)
  - where #Iterations is ideally O(network diameter) but varies due to routing loops or the count-to-infinity problem

Processing complexity

- LS: O(N^2)
- DV: O(#Iterations x N)

Robustness: what happens if router malfunctions?

- LS:
  - node can advertise incorrect *link* cost
  - each node computes only its *own* table
- DV:
  - node can advertise incorrect *path* cost
  - each node's table used by others; error propagates through network

Routing: Just the Beginning

- Link state and distance-vector are the deployed routing paradigms for intra-domain routing

- Inter-domain routing (BGP)
  - more Part II (Principles of Communications)
  - A version of DV

What are desirable goals for a routing solution?

- “Good” paths (least cost)
- Fast convergence after change/failures
  - no/rare loops
- Scalable
  - #messages
  - table size
  - processing complexity
- Secure
- Policy
- Rich metrics (more later)

Delivery models

- What if a node wants to send to more than one destination?
  - broadcast: send to all
  - multicast: send to all members of a group
  - anycast: send to any member of a group

- What if a node wants to send along more than one path?

Metrics

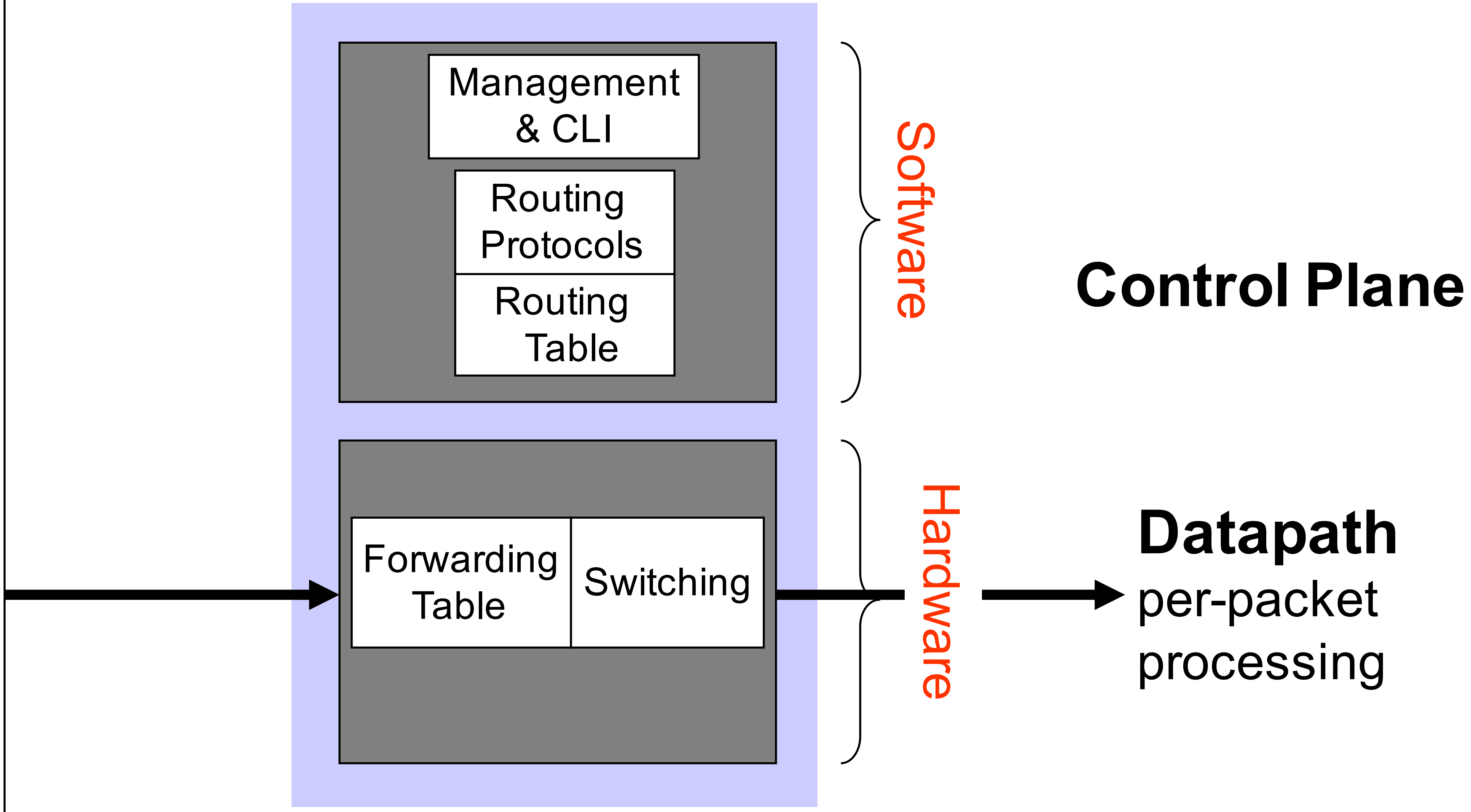
- Propagation delay
- Congestion
- Load balance
- Bandwidth (available, capacity, maximal, bbw)
- Price
- Reliability
- Loss rate
- Combinations of the above

In practice, operators set abstract “weights” (much like our costs); how exactly is a bit of a black art

From Routing back to Forwarding

- Routing: “control plane”
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Jointly creating the routing state
- Forwarding: “data plane”
  - Directing a data packet to an outgoing link
  - Individual router using routing state
- Two very different timescales….

Basic Architectural Components of an IP Router



Management & CLI

Routing Protocols

Routing Table

Forwarding Table

Switching

Software

Hardware

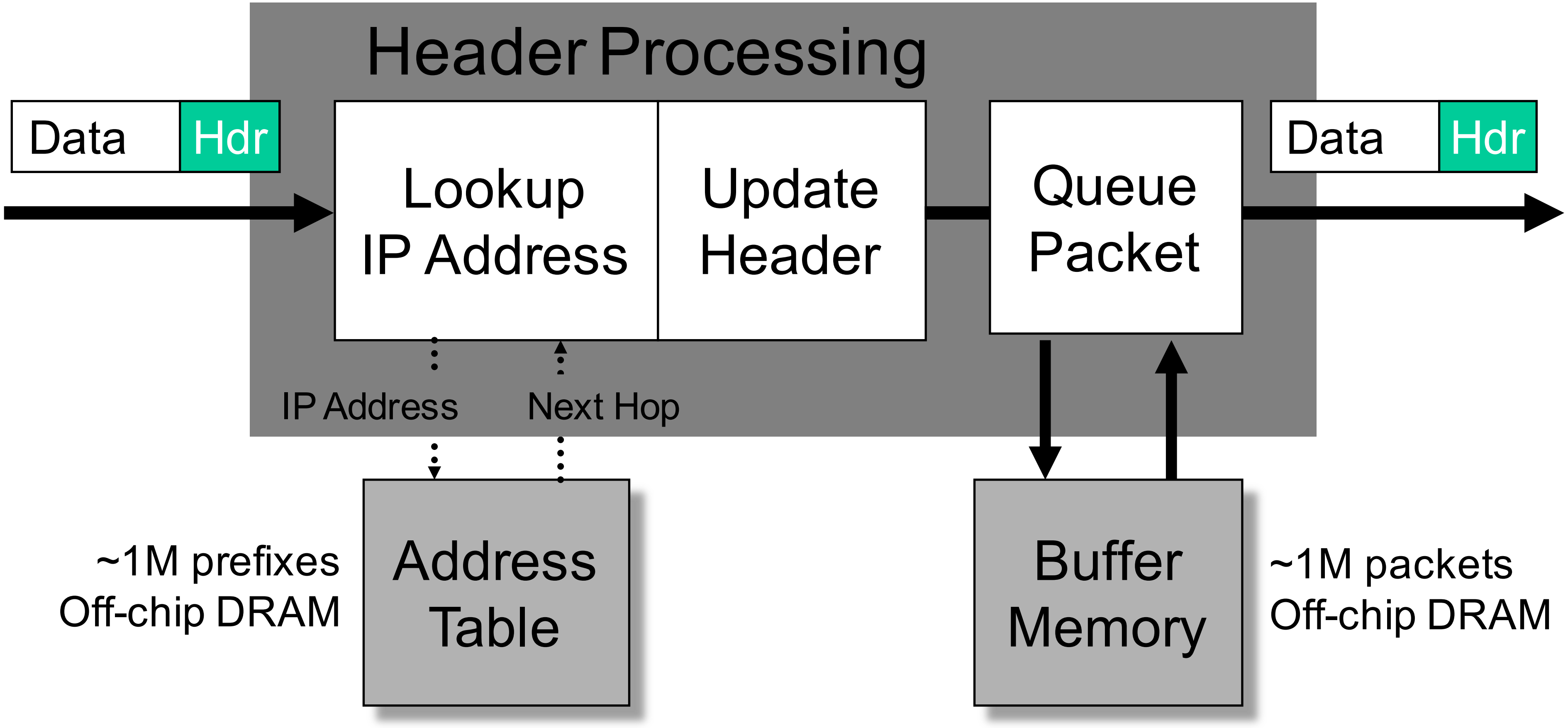
Control Plane

Datapath
per-packet processing

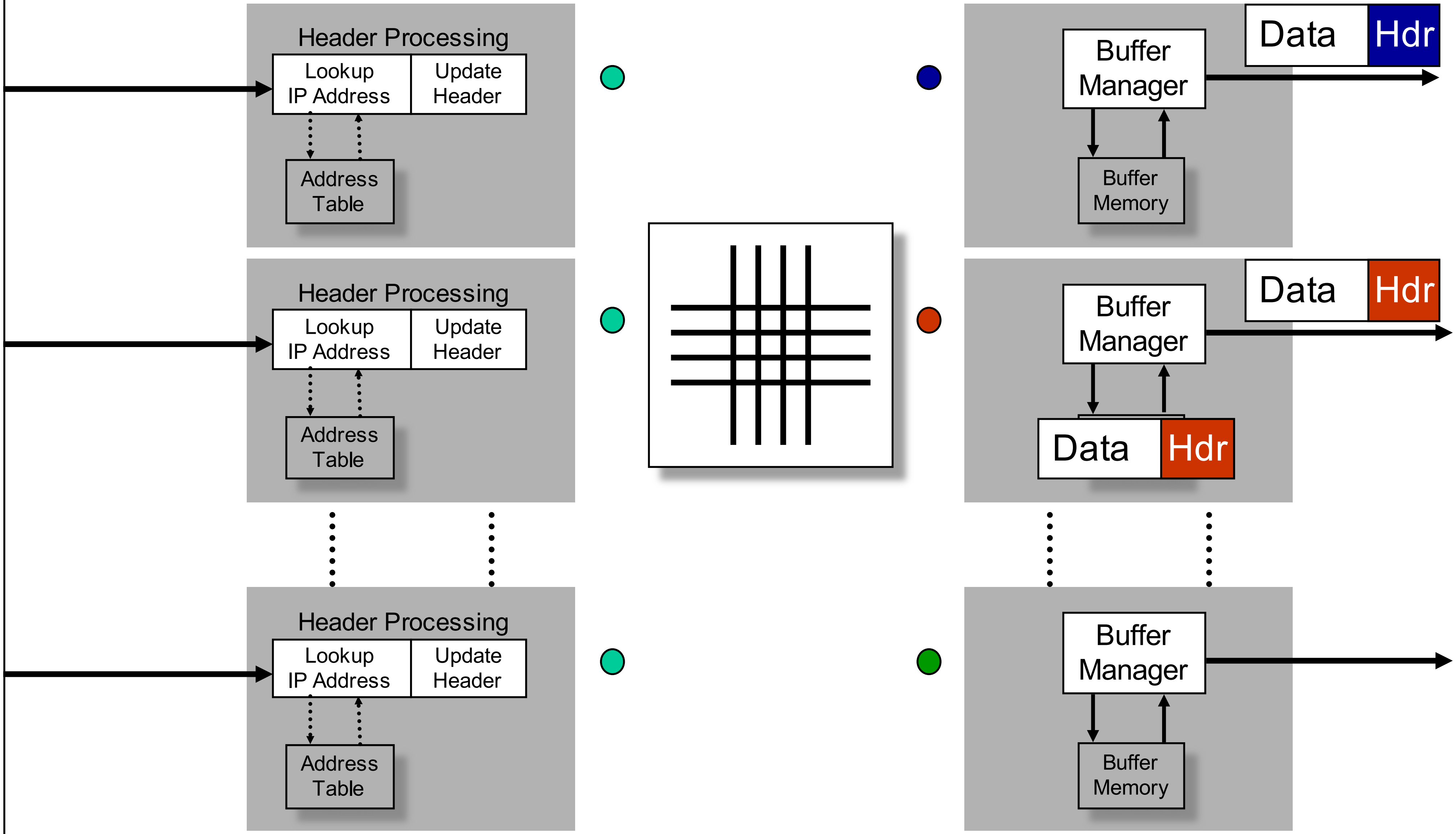
Per-packet processing in an IP Router

1. Accept packet arriving on an incoming link.
2. Lookup packet destination address in the forwarding table, to identify outgoing port(s).
3. Manipulate packet header: e.g., decrement TTL, update header checksum.
4. Send packet to the outgoing port(s).
5. Buffer packet in the queue.
6. Transmit packet onto outgoing link.

Generic Router Architecture



Generic Router Architecture



Forwarding tables

IP address } 32 bits wide → ~ 4 billion unique address

Naïve approach:

One entry per address

Entry	Destination	Port
1	0.0.0.0	1
2	0.0.0.1	2
⋮	⋮	⋮
2^{32}	255.255.255.255	12

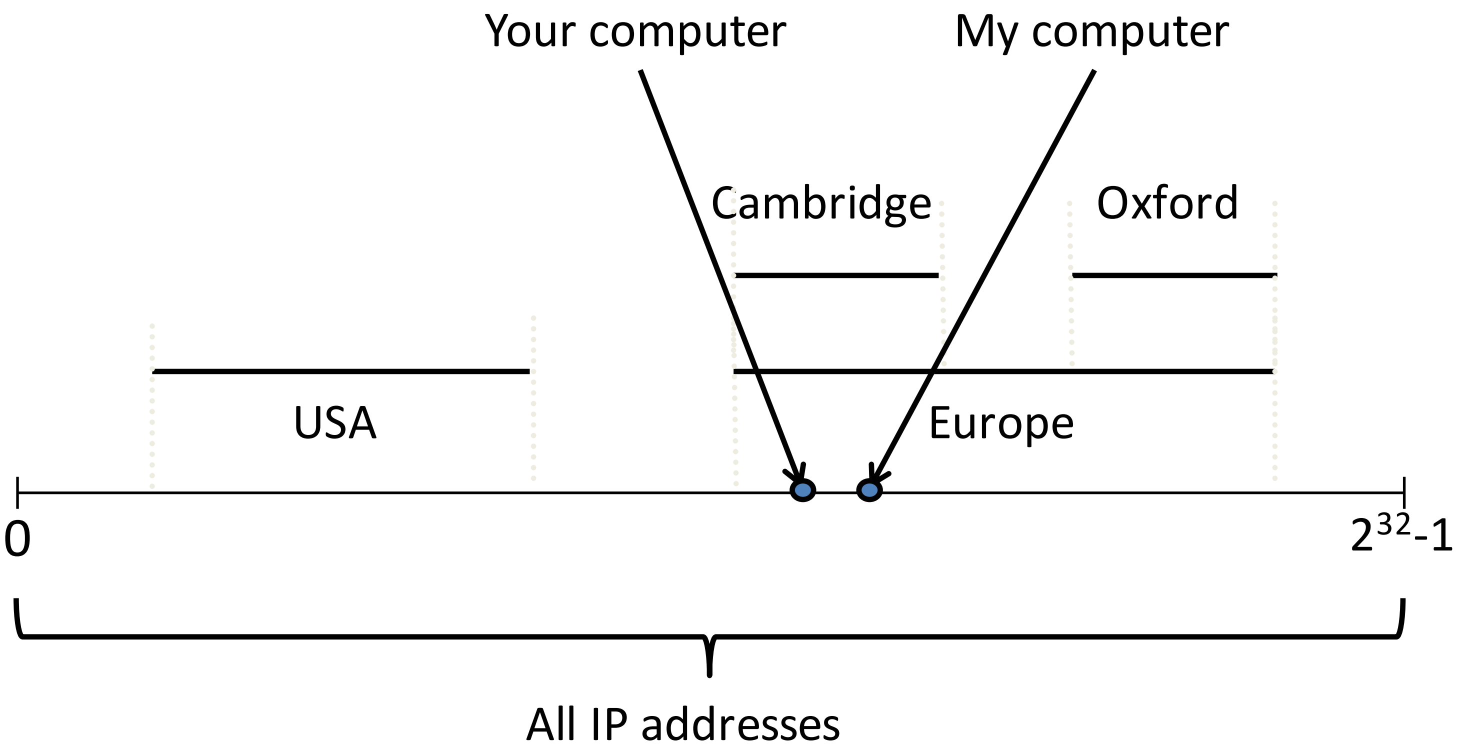
~ 4 billion entries

Improved approach:

Group entries to reduce table size

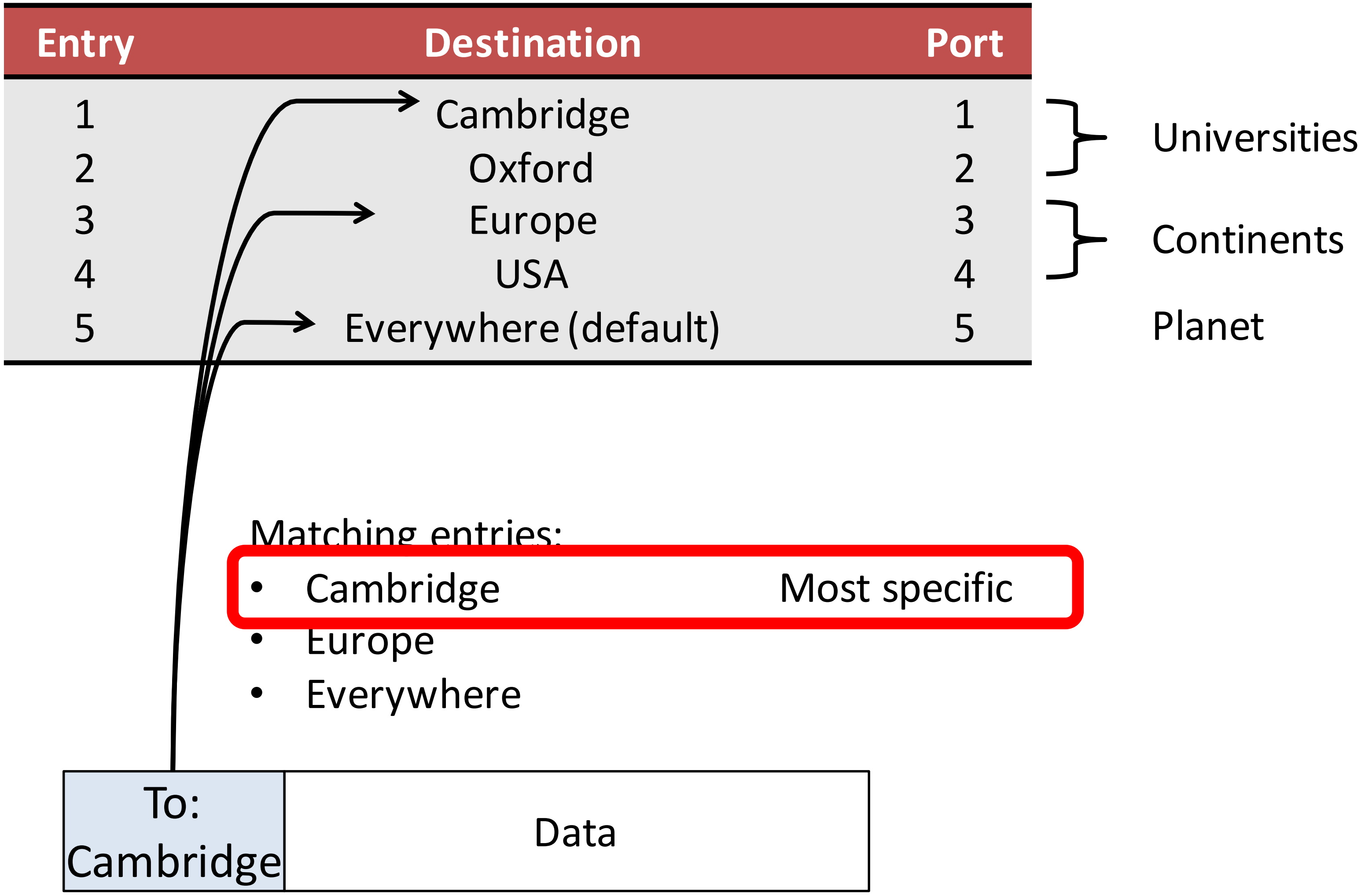
Entry	Destination	Port
1	0.0.0.0 – 127.255.255.255	1
2	128.0.0.1 – 128.255.255.255	2
⋮	⋮	⋮
50	248.0.0.0 – 255.255.255.255	12

IP addresses as a line



Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

Longest Prefix Match (LPM)



Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

Matching entries:

- Cambridge Most specific
- Europe
- Everywhere

Longest Prefix Match (LPM)

Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

Universities

Continents

Planet

Matching entries:

- Europe   Most specific
- Everywhere

To: France	Data

Implementing Longest Prefix Match

Entry	Destination	Port
1	Cambridge	1
2	Oxford	2
3	Europe	3
4	USA	4
5	Everywhere (default)	5

Searching

FOUND

Most specific

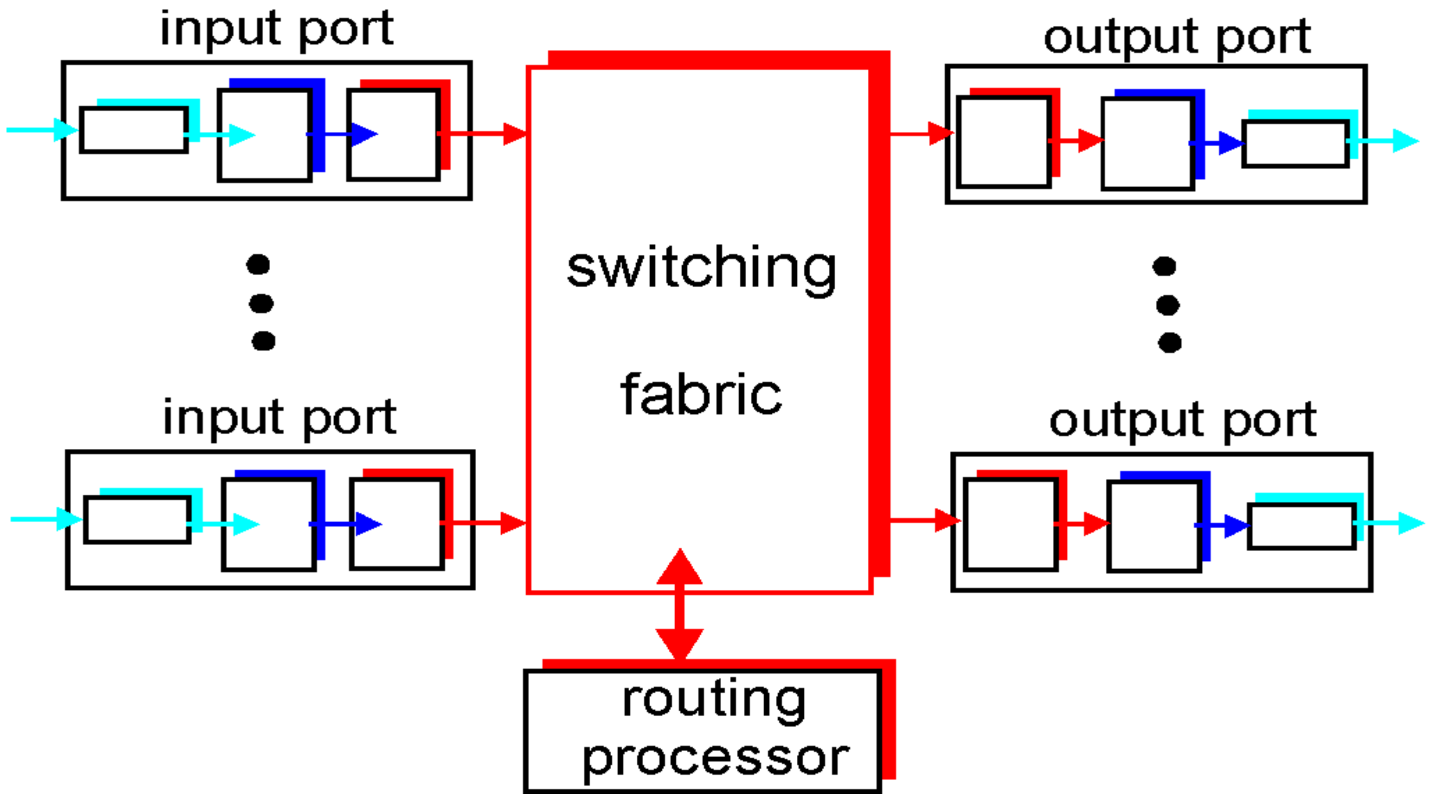
↓

Least specific

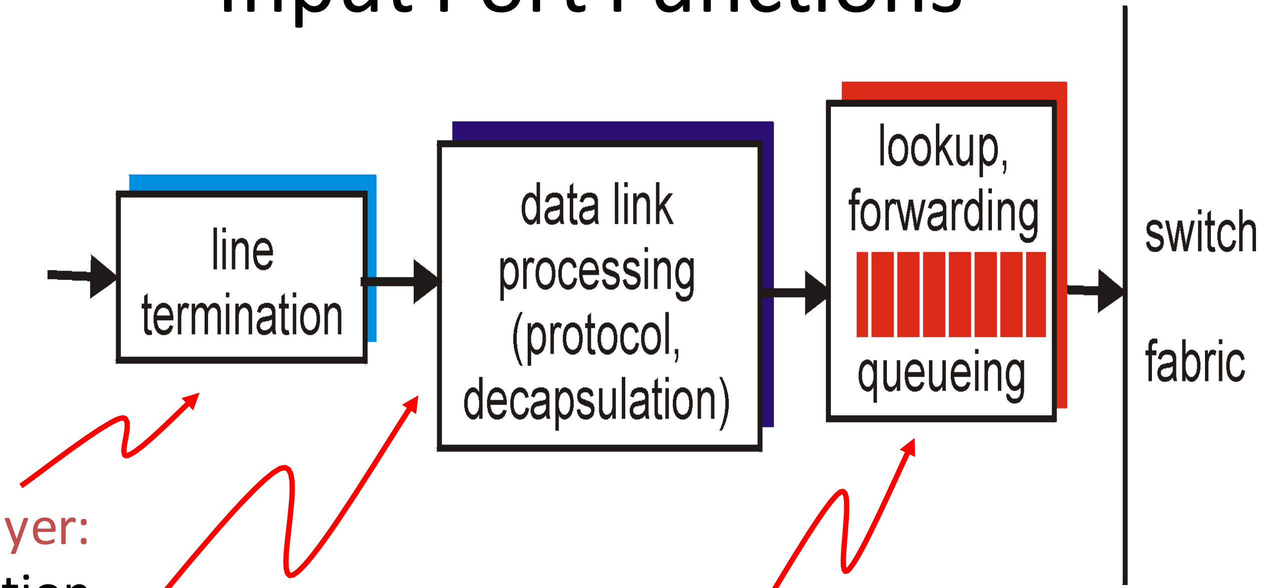
Router Architecture Overview

Two key router functions:

- run routing algorithms/protocol (RIP, OSPF, BGP)
- *forwarding* datagrams from incoming to outgoing link



Input Port Functions



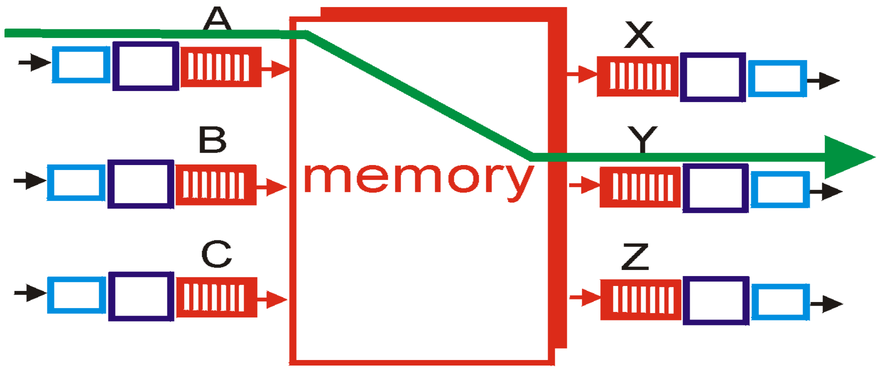
Physical layer:
bit-level reception

Data link layer:
e.g., Ethernet
see chapter 5

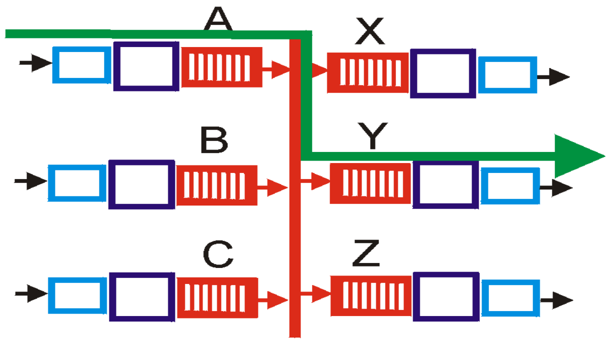
Decentralized switching*:*

- given datagram dest., lookup output port using forwarding table in input port memory
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

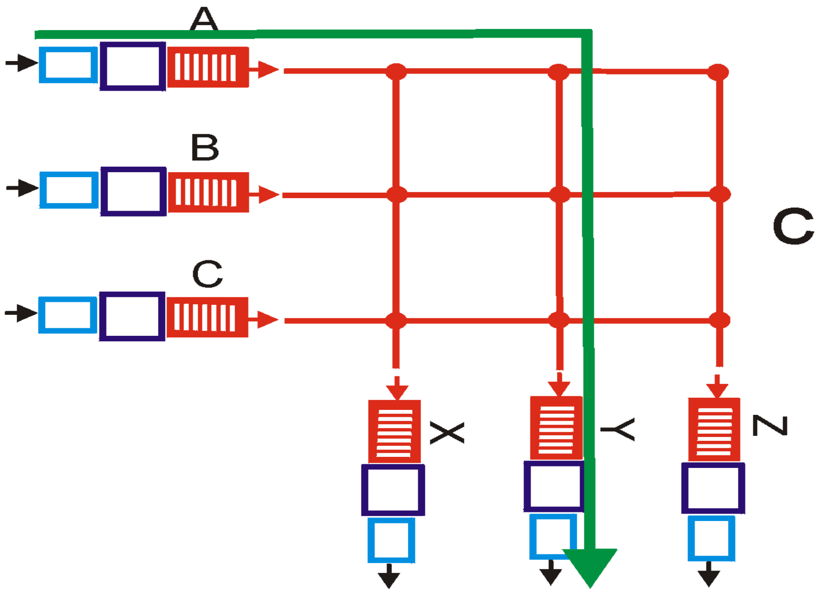
Three examples of switching fabrics
(comparison criteria: speed, contention, complexity)



memory



bus

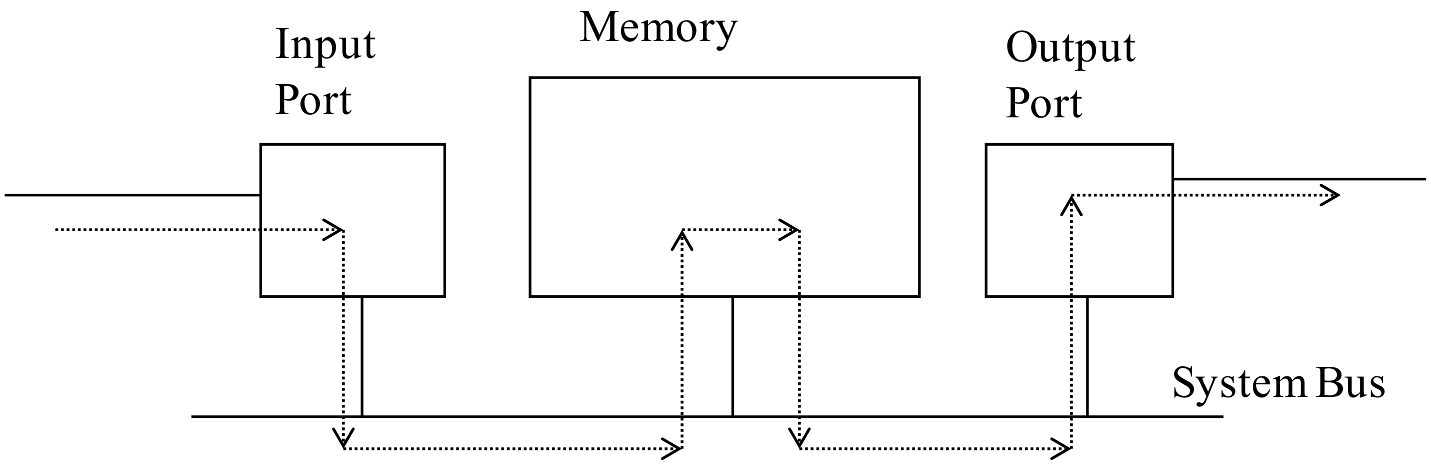


crossbar

Switching Via Memory

First generation routers:

- traditional computers with switching under direct control of CPU
- packet copied to system' s memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



Switching Via a Bus

- datagram from input port memory

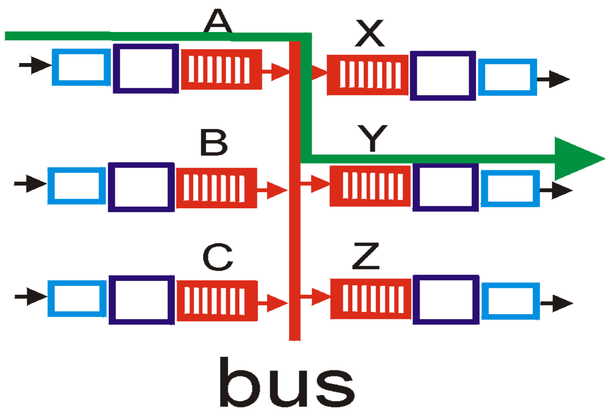
  to output port memory via a shared bus
- bus contention: switching speed limited by bus bandwidth

- Lots of ports?? speed up the bus

  no contention bus speed =

  2 x port speed x port count

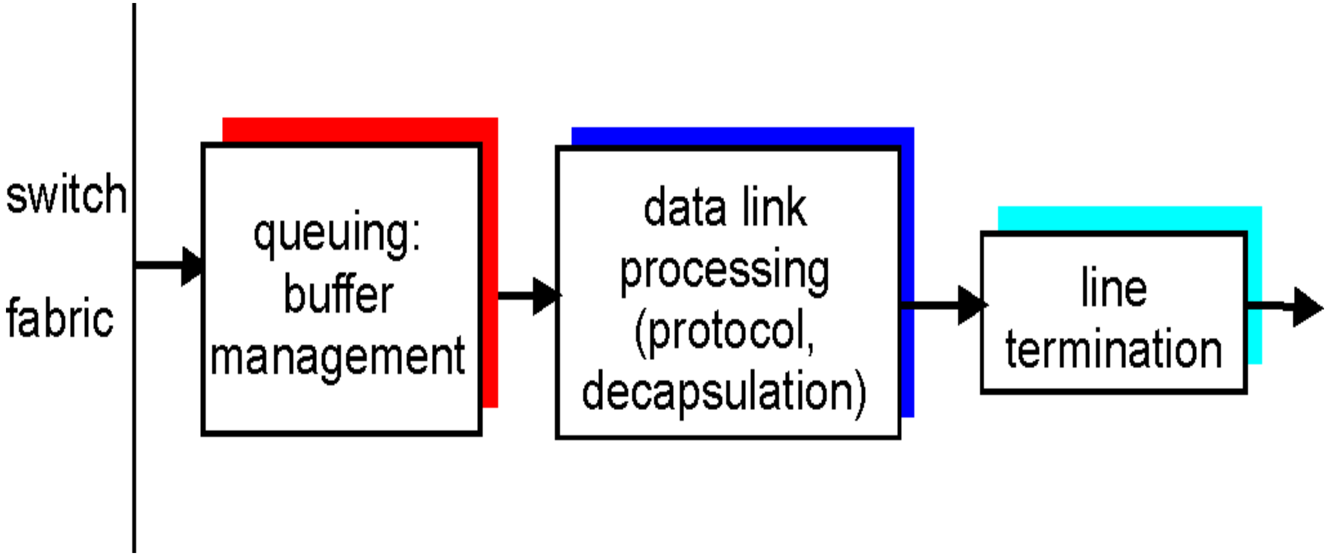
- 32 Gbps bus, Cisco 5600: sufficient speed for access routers



Switching Via An Interconnection Network

- overcome bus bandwidth limitations
- Banyan networks, other interconnection nets initially developed to connect processors in multiprocessor stages
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco CRS-1: switches 1.2 Tbps through the interconnection network

Output Ports

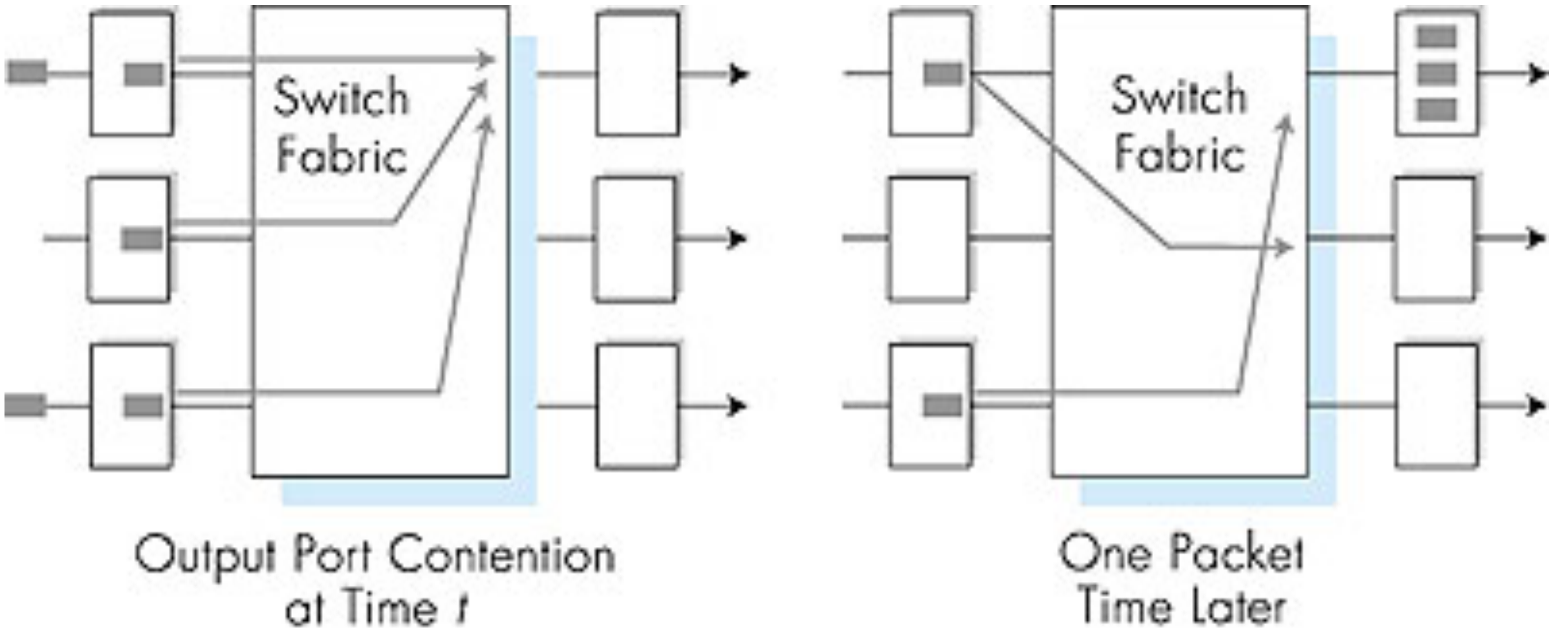

switch
fabric
queuing: buffer management
data link processing (protocol, decapsulation)
line termination

- *Buffering* required when datagrams arrive from fabric faster than the transmission rate

- *Scheduling discipline* chooses among queued datagrams for transmission

  → Who goes next?

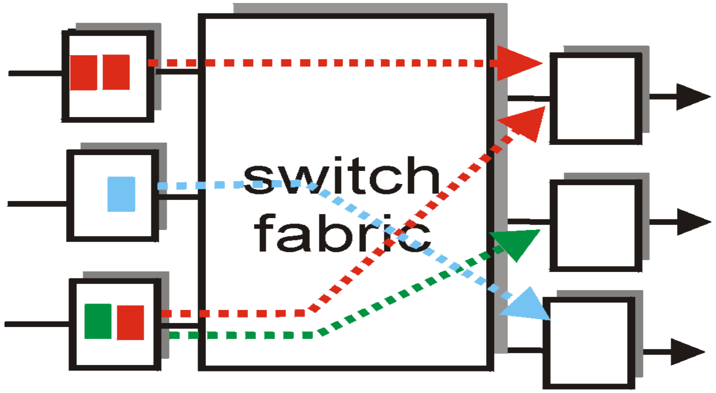
Output port queueing



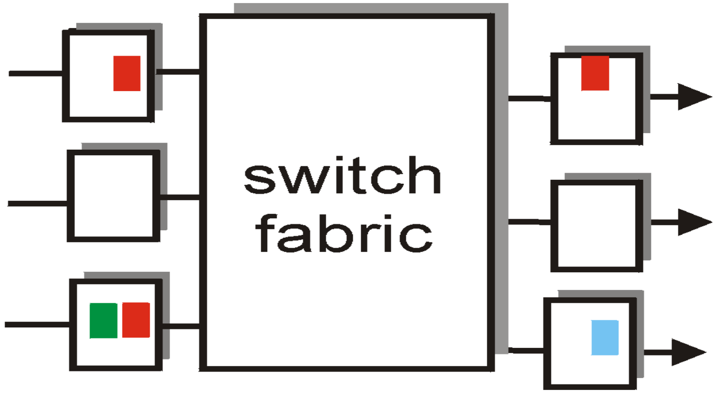
- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

Input Port Queuing

- Fabric slower than input ports combined -> queueing may occur at input queues
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward
- *queueing delay and loss due to input buffer overflow!*



output port contention
at time t - only one red
packet can be transferred



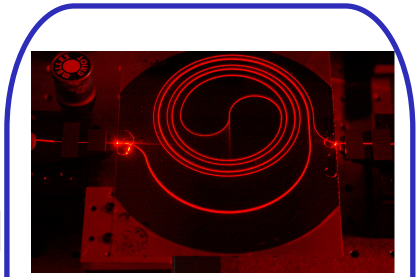
green packet
experiences HOL blocking

Buffers in Routers

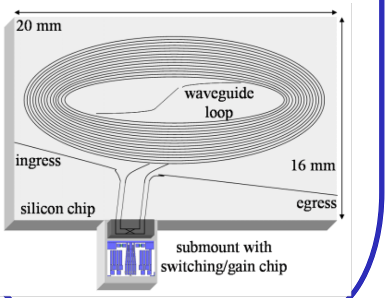
- So how large should the buffers be?

Buffer size matters

- End-to-end delay
  - Transmission, propagation, and queueing del
  - The only variable part is queueing delay
- Router architecture
  - Board space, power consumption, and cost
  - On chip buffers: higher density, higher c
  - Optical buffers: all-optical routers



1.4m long spiral waveguide with input from HeNe laser



You are now touching the edge of the *research* zone……

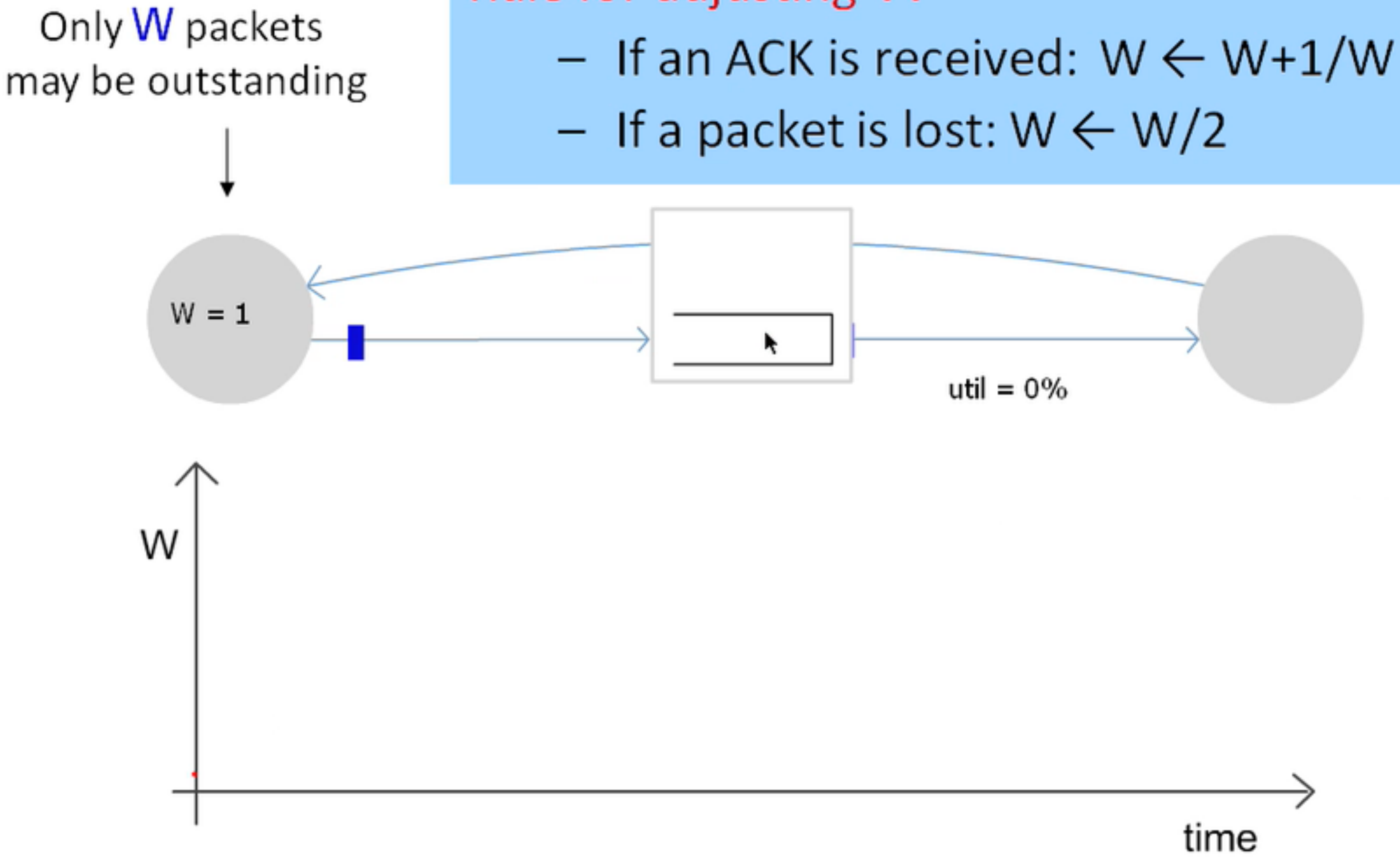
Buffer Sizing Story

	Rule-of-thumb $2T \times C$	Small Buffers $\frac{2T \times C}{\sqrt{n}}$	Tiny Buffers $O(\log W)$
# of packets	1,000,000	10,000	20 - 50
Intuition	TCP Sawtooth	Sawtooth Smoothing	Non-bursty Arrivals
Assume	Single TCP Flow, 100% Utilization	Many Flows, 100% Utilization	Paced TCP, 85-90% Utilization
Evidence	Simulation, Emulation	Simulations, Test-bed and Real Network Experiments	Simulations, Test-bed Experiments

Continuous ARQ (TCP) adapting to congestion

Rule for adjusting W

- If an ACK is received: W ← W+1/W
- If a packet is lost: W ← W/2

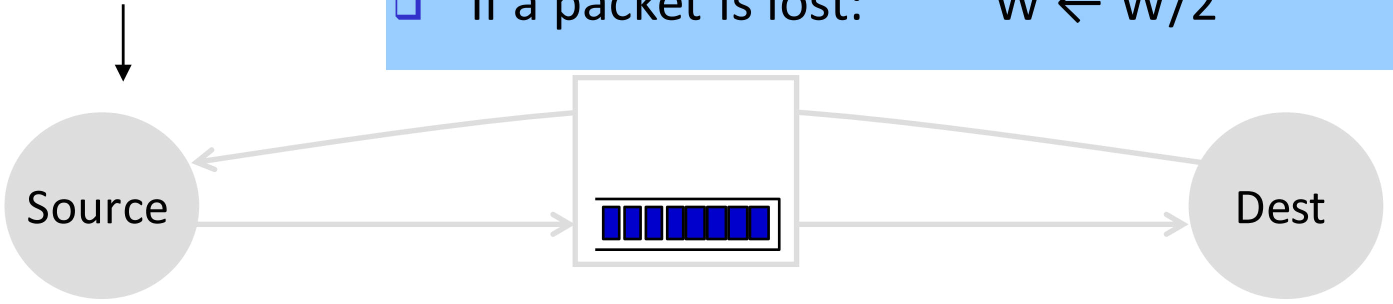


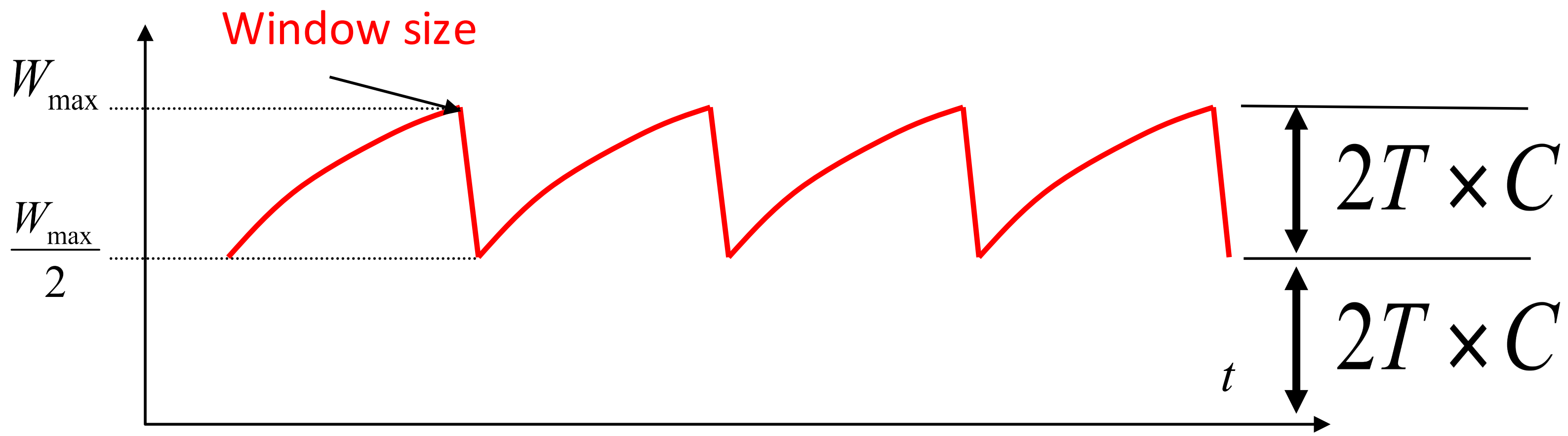
Rule-of-thumb – Intuition

Rule for adjusting W

- ❑ If an ACK is received: W ← W+1/W
- ❑ If a packet is lost: W ← W/2

Only W packets may be outstanding





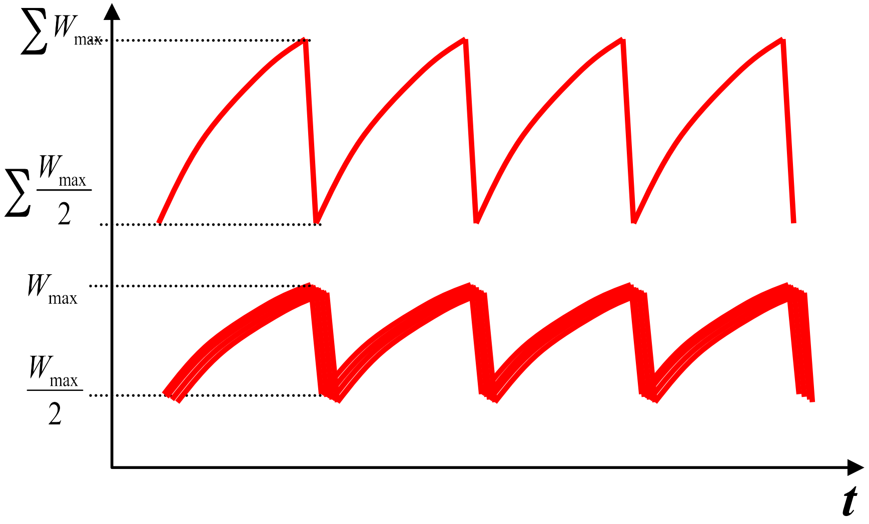
Small Buffers – Intuition

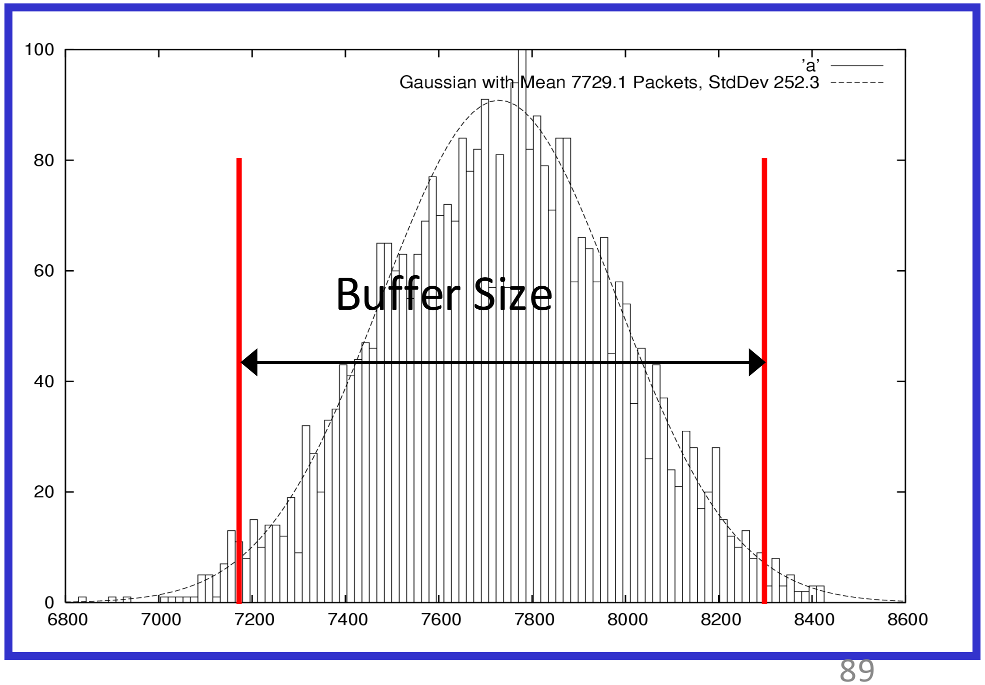
Synchronized Flows

- **Aggregate window has same dynamics**
- **Therefore buffer occupancy has same dynamics**
- **Rule-of-thumb still holds.**

Many TCP Flows

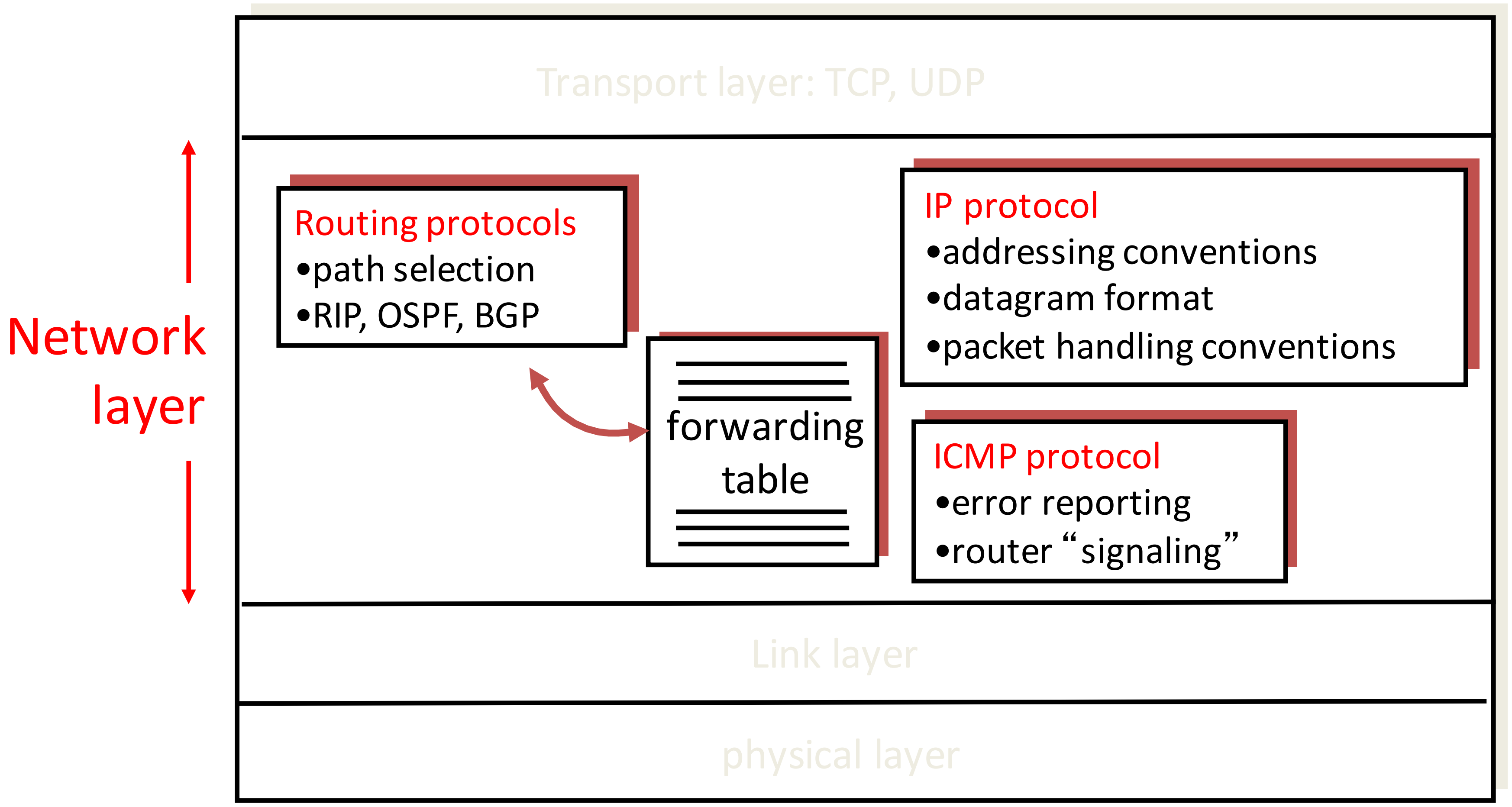
- **Independent, desynchronized**
- **Central limit theorem says the aggregate becomes Gaussian**
- **Variance (buffer size) decreases as *N* increases**





The Internet version of a Network layer

Host, router network layer functions:



Network layer

Transport layer: TCP, UDP

Routing protocols
- path selection
- RIP, OSPF, BGP

forwarding table

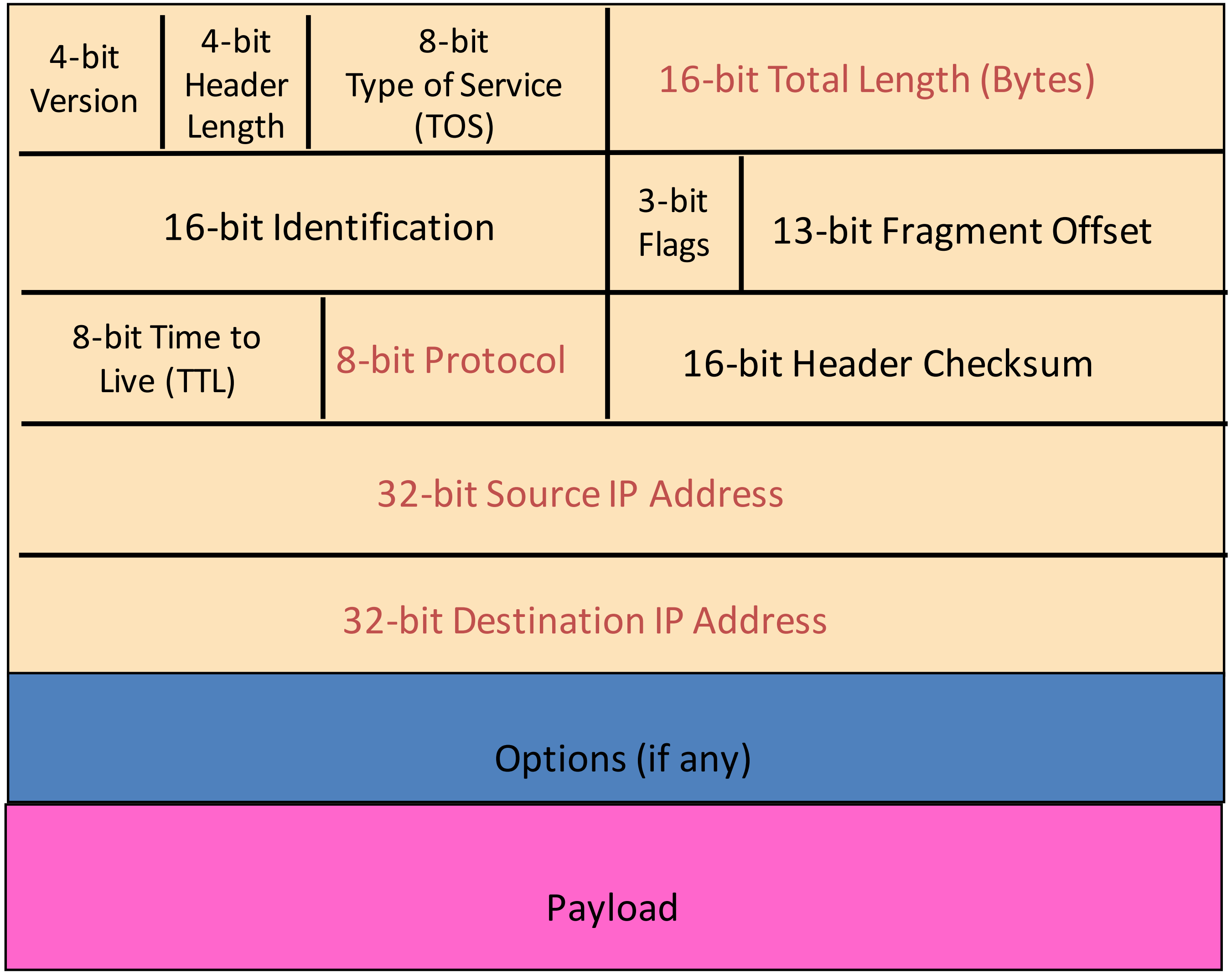
IP protocol
- addressing conventions
- datagram format
- packet handling conventions

ICMP protocol
- error reporting
- router "signaling"

Link layer

physical layer

IPv4 Packet Structure
20 Bytes of Standard Header, then Options



4-bit Version

4-bit Header Length

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

8-bit Protocol

16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

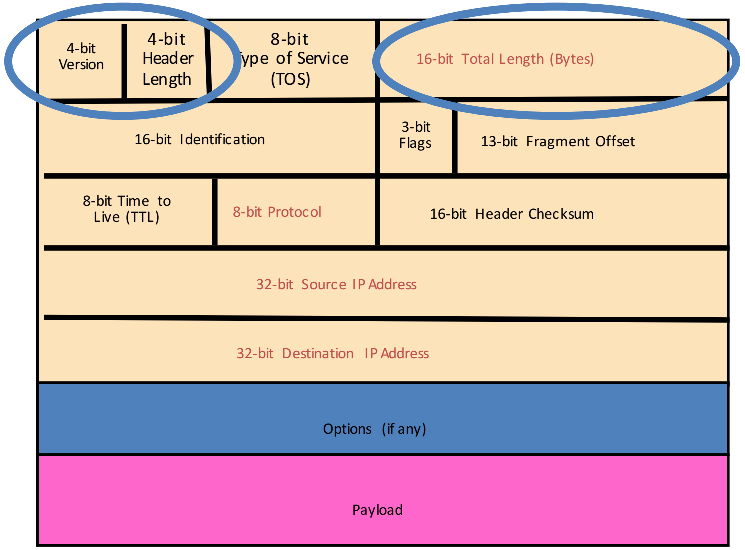
Options (if any)

Payload

(Packet) Network Tasks One-by-One

- Read packet correctly
- Get packet to the destination
- Get responses to the packet back to source
- Carry data
- Tell host what to do with packet once arrived
- Specify any special network handling of the packet
- Deal with problems that arise along the path

Reading Packet Correctly



4-bit Version

4-bit Header Length

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

8-bit Protocol

16-bit Header Checksum

32-bit Source IP Address

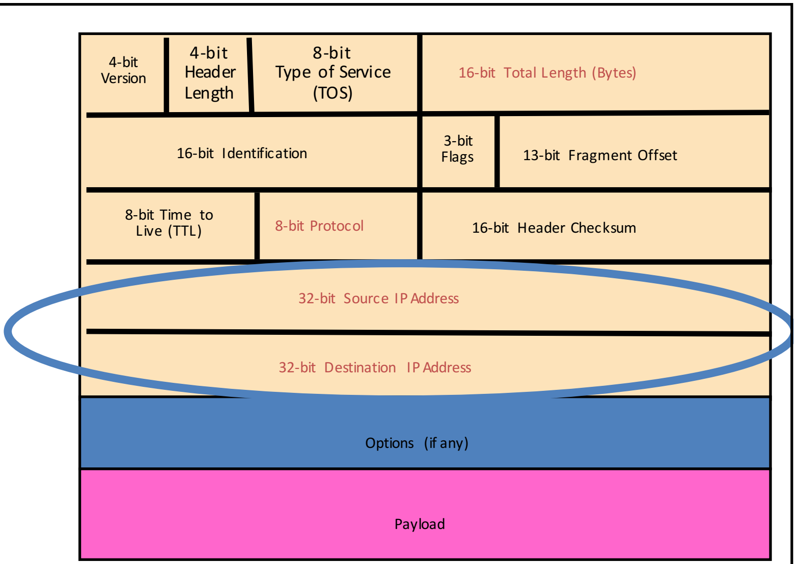
32-bit Destination IP Address

Options (if any)

Payload

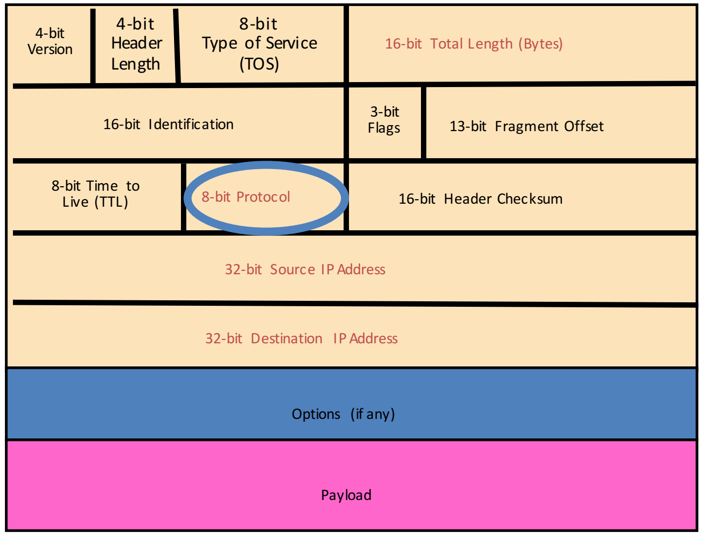
- Version number (4 bits)
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically “4” (for IPv4), and sometimes “6” (for IPv6)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically “5” (for a 20-byte IPv4 header)
  - Can be more when IP options are used
- Total length (16 bits)
  - Number of bytes in the packet
  - Maximum size is 65,535 bytes (2^{16} -1)
  - … though underlying links may impose smaller limits

Getting Packet to Destination and Back

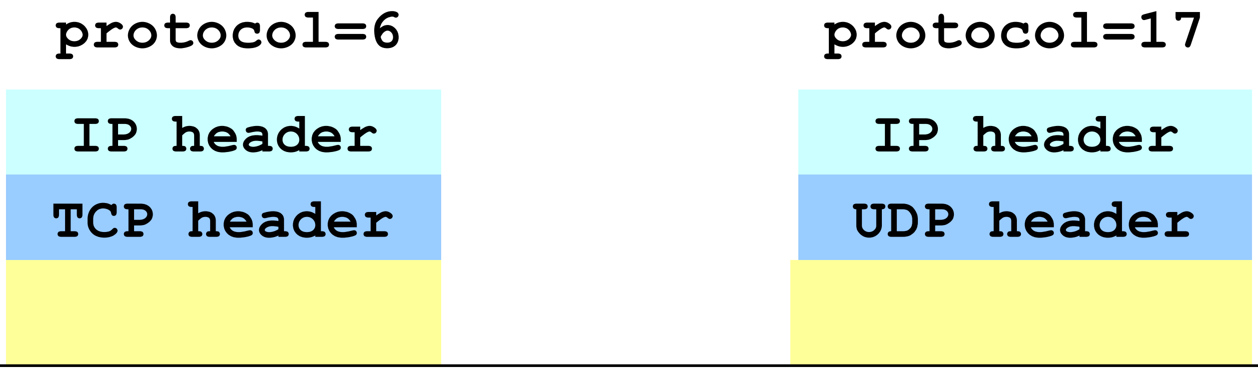

4-bit Version
4-bit Header Length
8-bit Type of Service (TOS)
16-bit Total Length (Bytes)
16-bit Identification
3-bit Flags
13-bit Fragment Offset
8-bit Time to Live (TTL)
8-bit Protocol
16-bit Header Checksum
32-bit Source IP Address
32-bit Destination IP Address
Options (if any)
Payload

- Two IP addresses
  - Source IP address (32 bits)
  - Destination IP address (32 bits)
- Destination address
  - Unique identifier/locator for the receiving host
  - Allows each node to make forwarding decisions
- Source address
  - Unique identifier/locator for the sending host
  - Recipient can decide whether to accept packet
  - Enables recipient to send a reply back to source

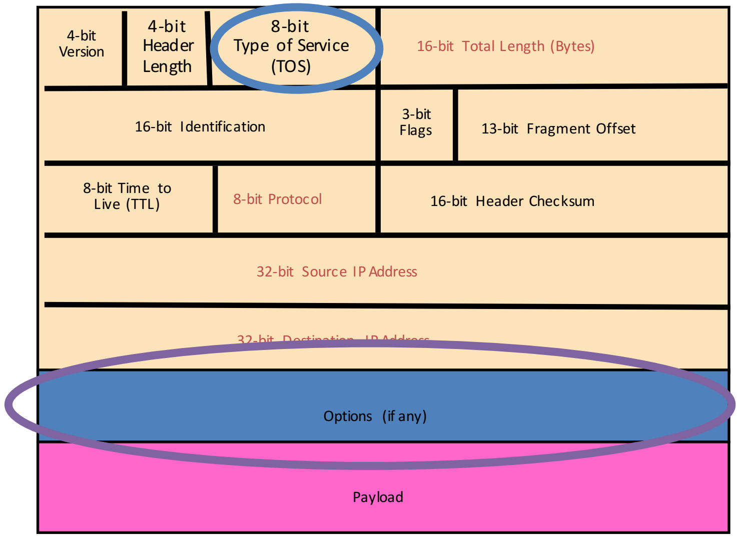
Telling Host How to Handle Packet



- Protocol (8 bits)
  - Identifies the higher-level protocol
  - Important for demultiplexing at receiving host
- Most common examples
  - E.g., “6” for the Transmission Control Protocol (TCP)
  - E.g., “17” for the User Datagram Protocol (UDP)



Special Handling



4-bit Version

4-bit Header Length

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

8-bit Protocol

16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

Options (if any)

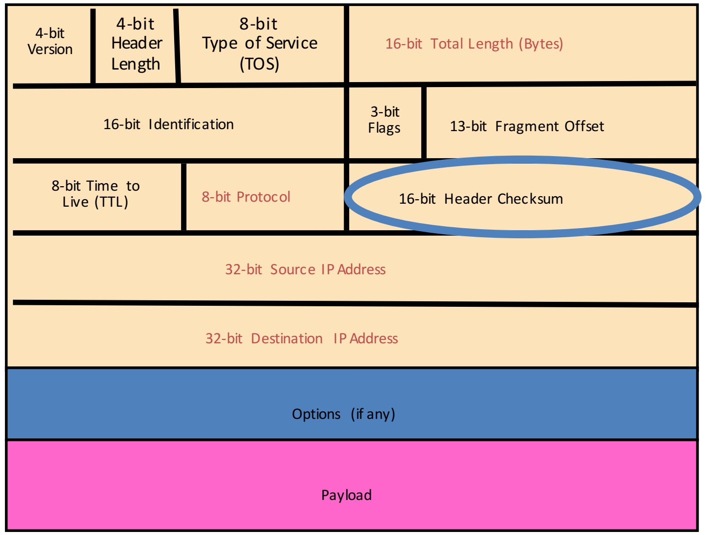
Payload

- Type-of-Service (8 bits)
  - Allow packets to be treated differently based on needs
  - E.g., low delay for audio, high bandwidth for bulk transfer
  - Has been redefined several times
- Options

Potential Problems

- Header Corrupted: **Checksum**
- Loop: **TTL**
- Packet too large: **Fragmentation**

Header Corruption



4-bit Version

4-bit Header Length

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

8-bit Protocol

16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

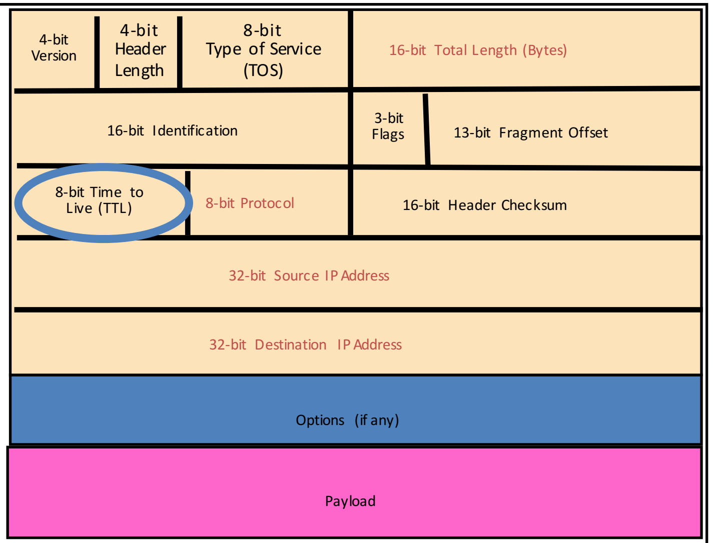
Options (if any)

Payload

- Checksum (16 bits)
  - Particular form of checksum over packet header
- If not correct, router discards packets
  - So it doesn’t act on bogus information
- Checksum recalculated at every router
  - **Why?**
  - **Why include TTL?**
  - **Why only header?**

Preventing Loops

(aka Internet Zombie plan)



4-bit Version

4-bit Header Length

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

8-bit Protocol

16-bit Header Checksum

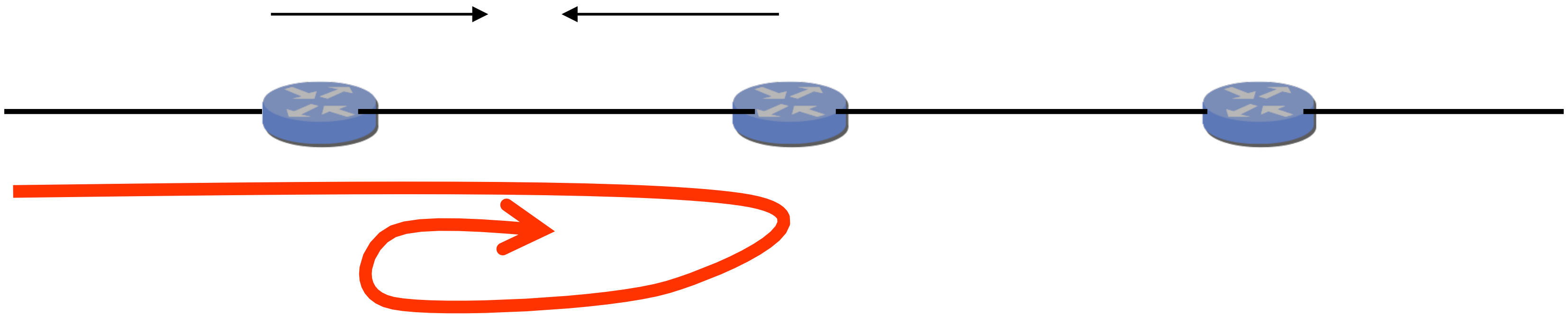
32-bit Source IP Address

32-bit Destination IP Address

Options (if any)

Payload

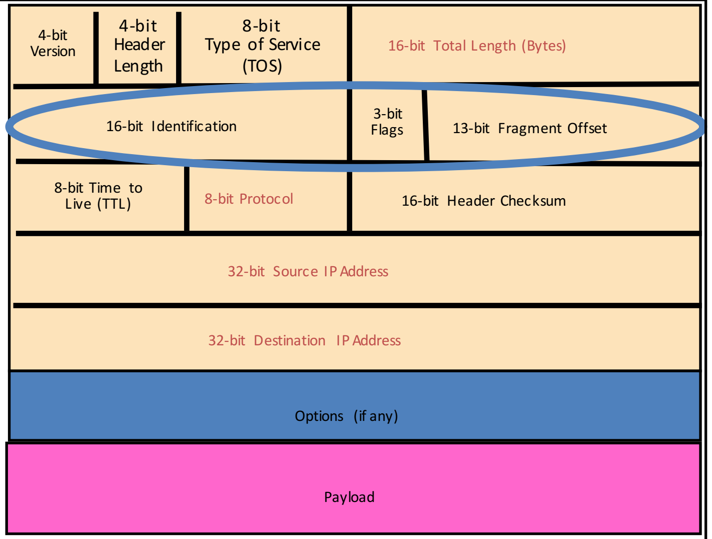
- Forwarding loops cause packets to cycle forever
  - As these accumulate, eventually consume **all** capacity



- Time-to-Live (TTL) Field (8 bits)
  - Decremented at each hop, packet discarded if reaches 0
  - …and “time exceeded” message is sent to the source
    - Using “ICMP” control message; basis for **traceroute**

Fragmentation

(some assembly required)

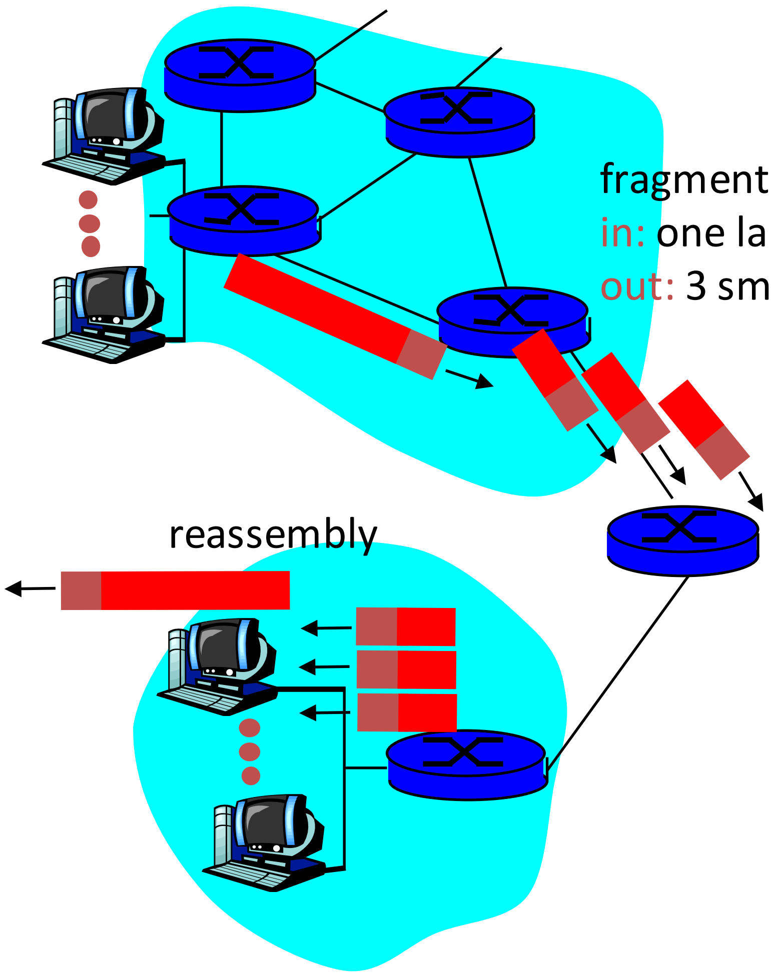


- Fragmentation: when forwarding a packet, an Internet router can split it into multiple pieces ("fragments") if too big for next hop link

- Must reassemble to recover original packet
  - Need fragmentation information (32 bits)
  - Packet identifier, flags, and fragment offset

IP Fragmentation & Reassembly

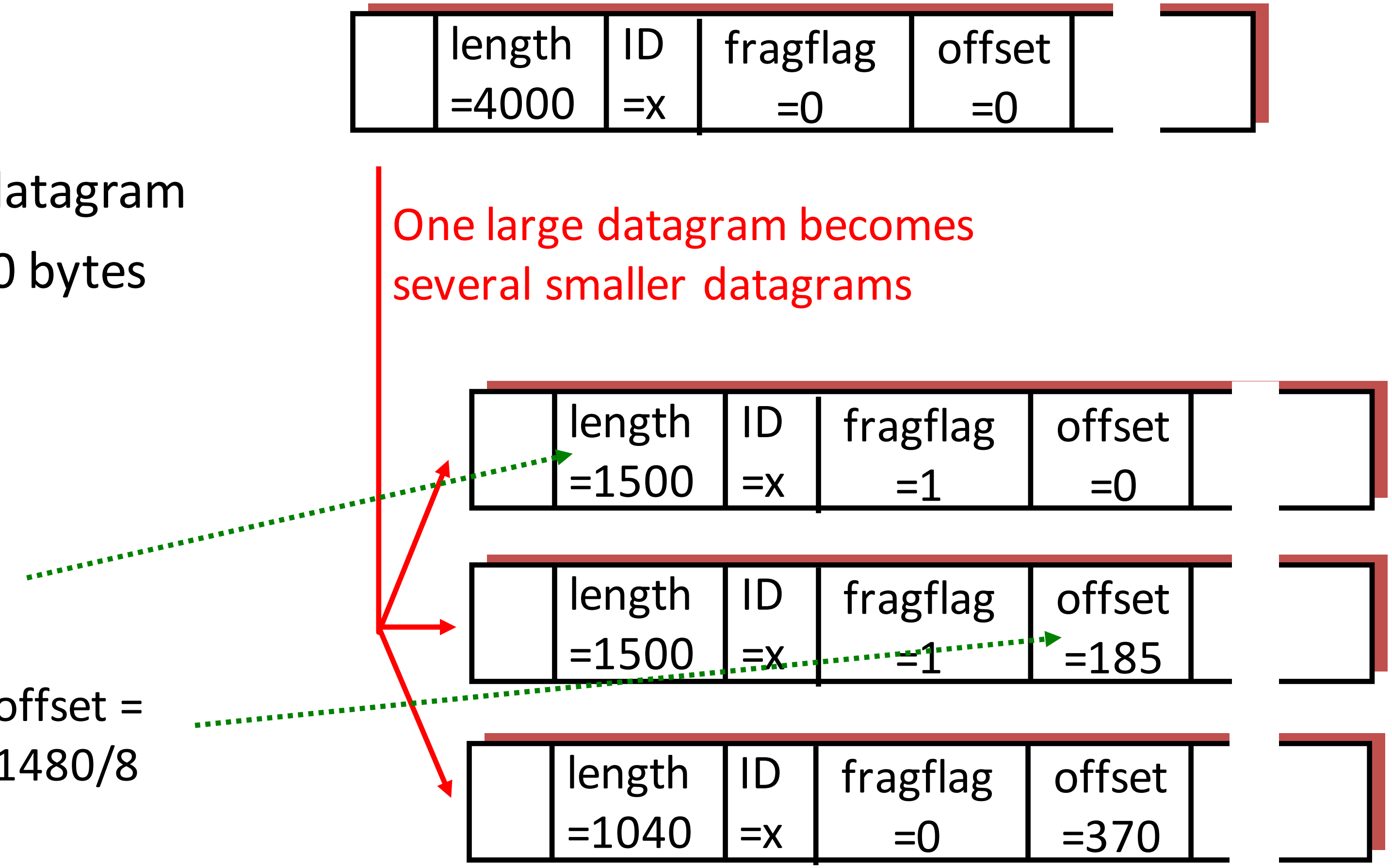
- network links have MTU (max.transfer size) - largest possible link-level frame.
  - different link types, different MTUs
- large IP datagram divided (“fragmented”) within net
  - one datagram becomes several datagrams
  - “reassembled” only at final destination
  - IP header bits used to identify, order related fragments
- IPv6 does things differently…



IP Fragmentation and Reassembly

Example

- 4000 byte datagram
- MTU = 1500 bytes



length	ID	fragflag	offset
=4000	=x	=0	=0

One large datagram becomes several smaller datagrams

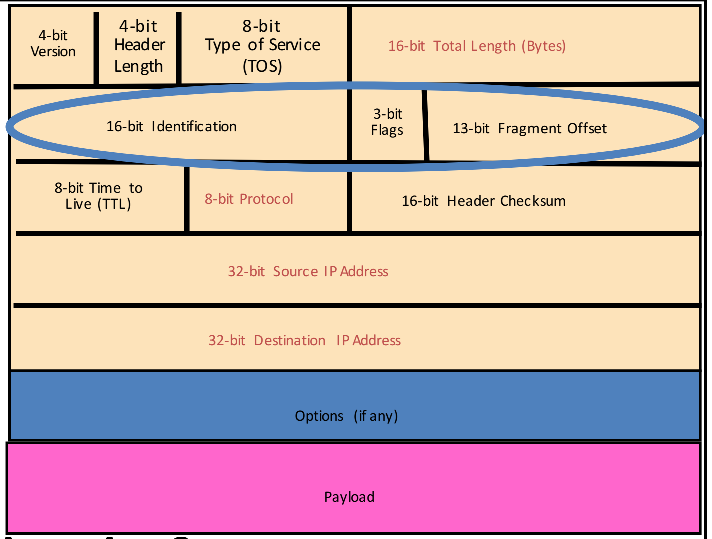
length	ID	fragflag	offset
=1500	=x	=1	=0
=1500	=x	=1	=185
=1040	=x	=0	=370

1480 bytes in data field

offset = 1480/8

Pop quiz question: What happens when a fragment is lost?

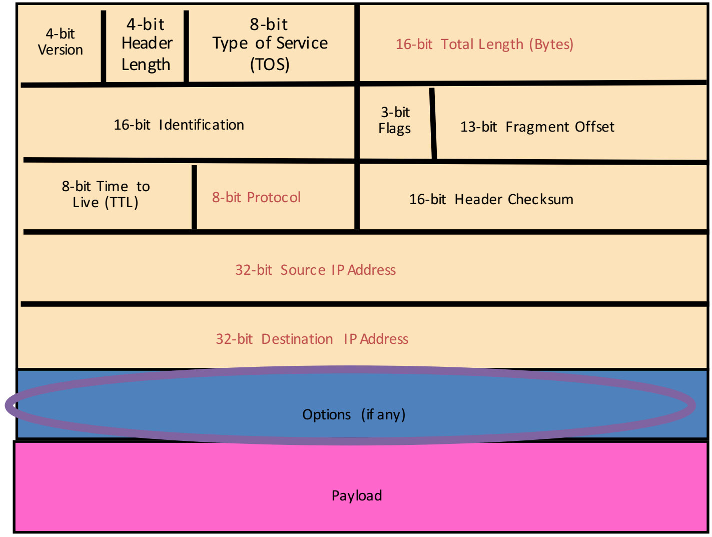
Fragmentation Details



- Identifier (16 bits): used to tell which fragments belong together
- Flags (3 bits):
  - Reserved **(RF):** unused bit
  - Don’t Fragment **(DF):** instruct routers to **not** fragment the packet even if it won’t fit
    - Instead, they **drop** the packet and send back a “Too Large” ICMP control message
    - Forms the basis for “Path MTU Discovery”
  - More (**MF**): this fragment is not the last one
- Offset (13 bits): what part of datagram this fragment covers in 8-byte units

Pop quiz question: Why do frags use offset and not a frag number?

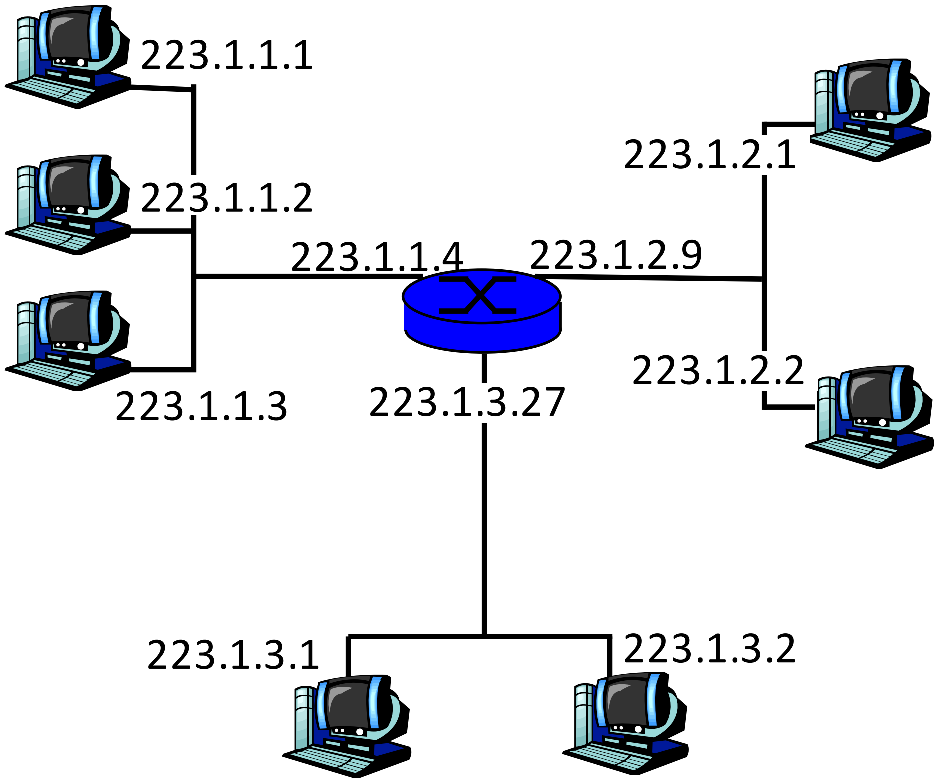
Options



- End of Options List
- No Operation (padding between options)
- Record Route
- Strict Source Route
- Loose Source Route
- Timestamp
- Traceroute
- Router Alert
- .....

IP Addressing: introduction

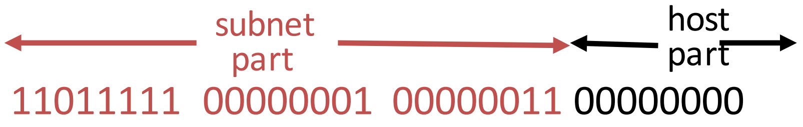
- IP address: 32-bit identifier for host, router *interface*
- *interface:* connection between host/router and physical link
  - router’s typically have multiple interfaces
  - host typically has one interface
  - IP addresses associated with each interface



$$223.1.1.1 = \underbrace{11011111}_{223}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}$$

Subnets

- IP address:
  - subnet part (high order bits)
  - host part (low order bits)
- *What's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can physically reach each other without intervening router



subnet part

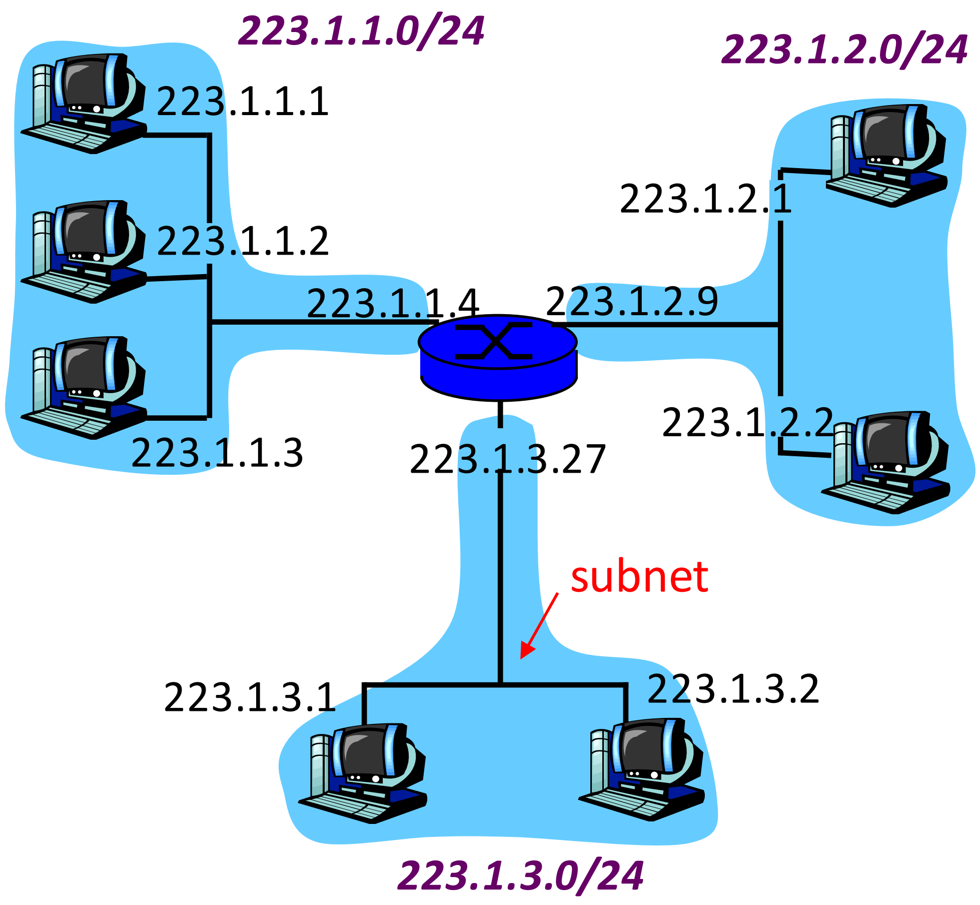
host part

11011111 00000001 00000011 00000000

223.1.3.0/24

CIDR: Classless InterDomain Routing

- subnet portion of address of arbitrary length
- address format: a.b.c.d/x, where x is # bits in subnet portion of address



Subnet mask: /24

network consisting of 3 subnets

IP addresses: how to get one?

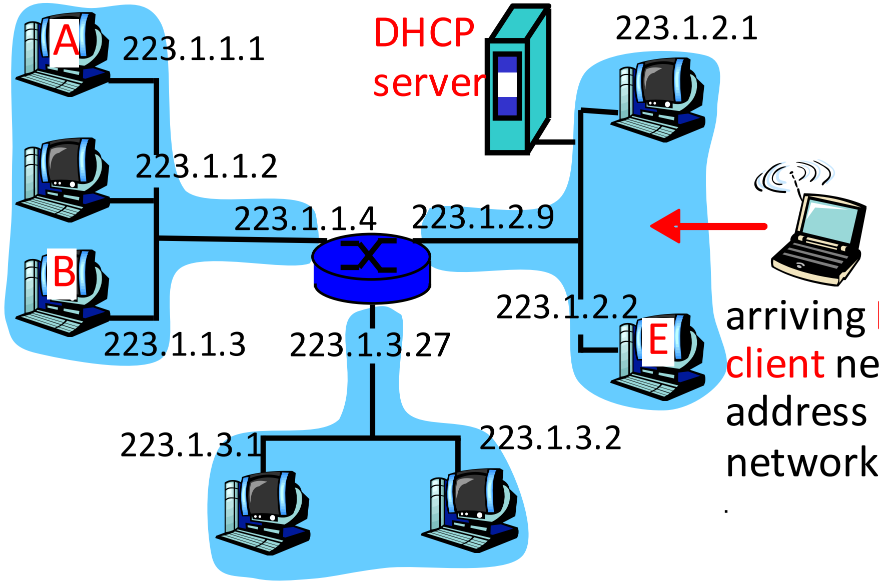
Q: How does a *host* get IP address?

- hard-coded by system admin in a file
  - Windows: control-panel->network->configuration->tcp/ip->properties
  - UNIX: /etc/rc.config (circa 1980's your mileage will vary)
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
  - "plug-and-play"

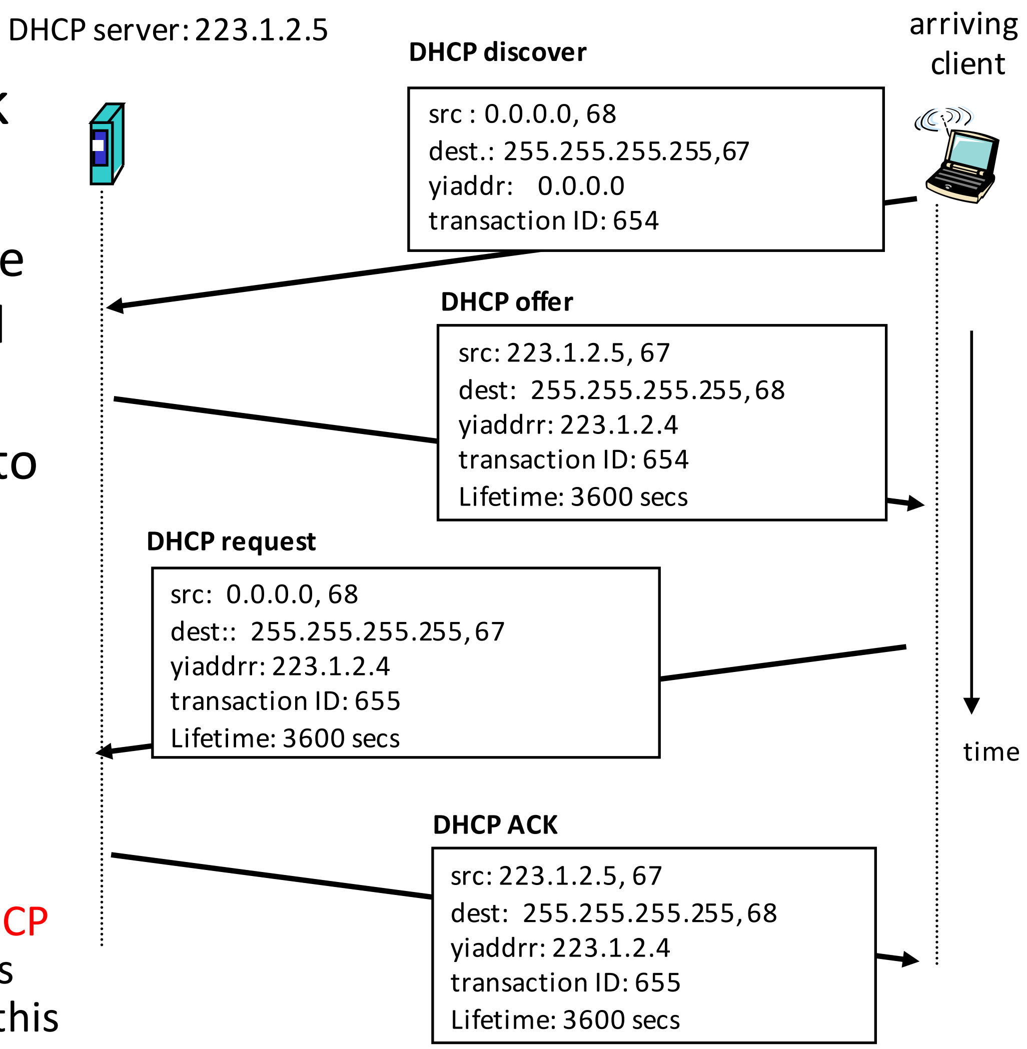
DHCP client-server scenario

Goal: allow host to *dynamically* obtain its IP address from network server when it joins network

- Can renew its lease on address in use
- Allows reuse of addresses (only hold address while connected an "on")
- Support for mobile users who want to join network (more shortly)



arriving DHCP client needs address in this network



DHCP discover

src : 0.0.0.0, 68
dest.: 255.255.255.255,67
yiaddr: 0.0.0.0
transaction ID: 654

DHCP offer

src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddrr: 223.1.2.4
transaction ID: 654
Lifetime: 3600 secs

DHCP request

src: 0.0.0.0, 68
dest:: 255.255.255.255, 67
yiaddrr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

DHCP ACK

src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddrr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

IP addresses: how to get one?

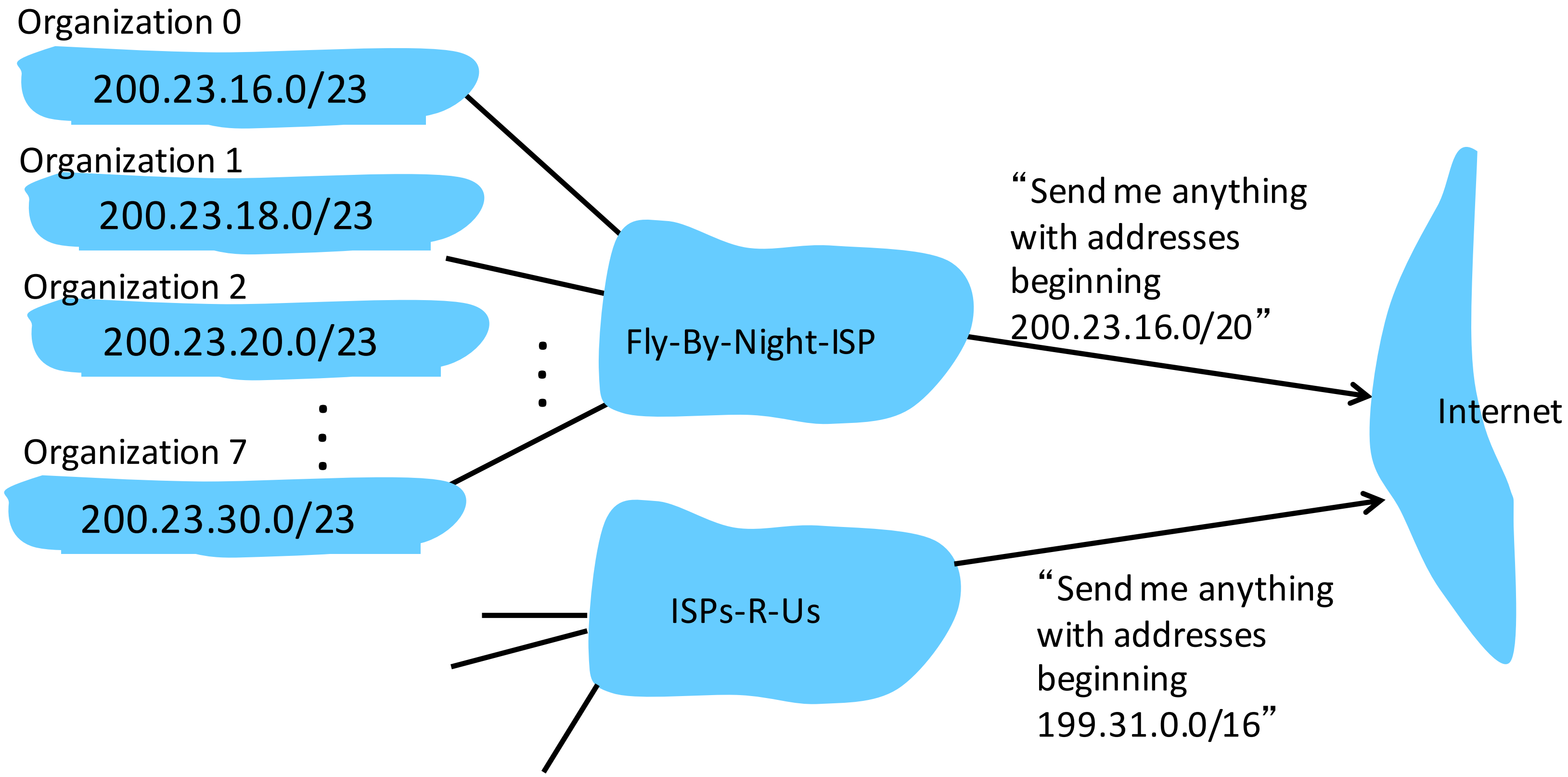
Q: How does *network* get subnet part of IP addr?

A: gets allocated portion of its provider ISP’s address space

ISP's block	11001000 00010111 00010000 00000000	200.23.16.0/20
Organization 0	11001000 00010111 00010000 00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010 00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100 00000000	200.23.20.0/23
...	....	....
Organization 7	11001000 00010111 00011110 00000000	200.23.30.0/23

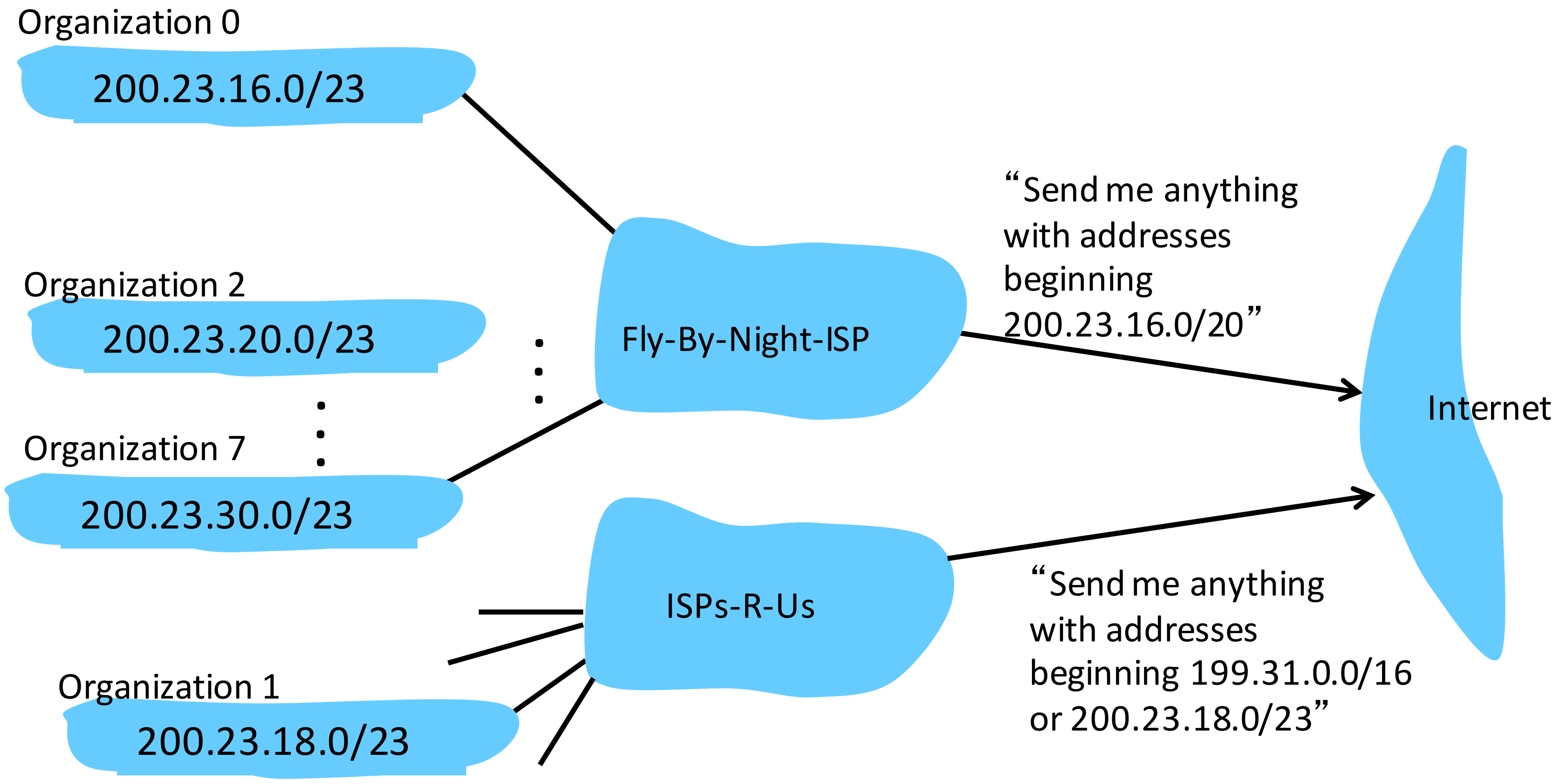
Hierarchical addressing: route aggregation

Hierarchical addressing allows efficient advertisement of routing information:



Hierarchical addressing: more specific routes

ISPs-R-Us has a more specific route to Organization 1



IP addressing: the last word...

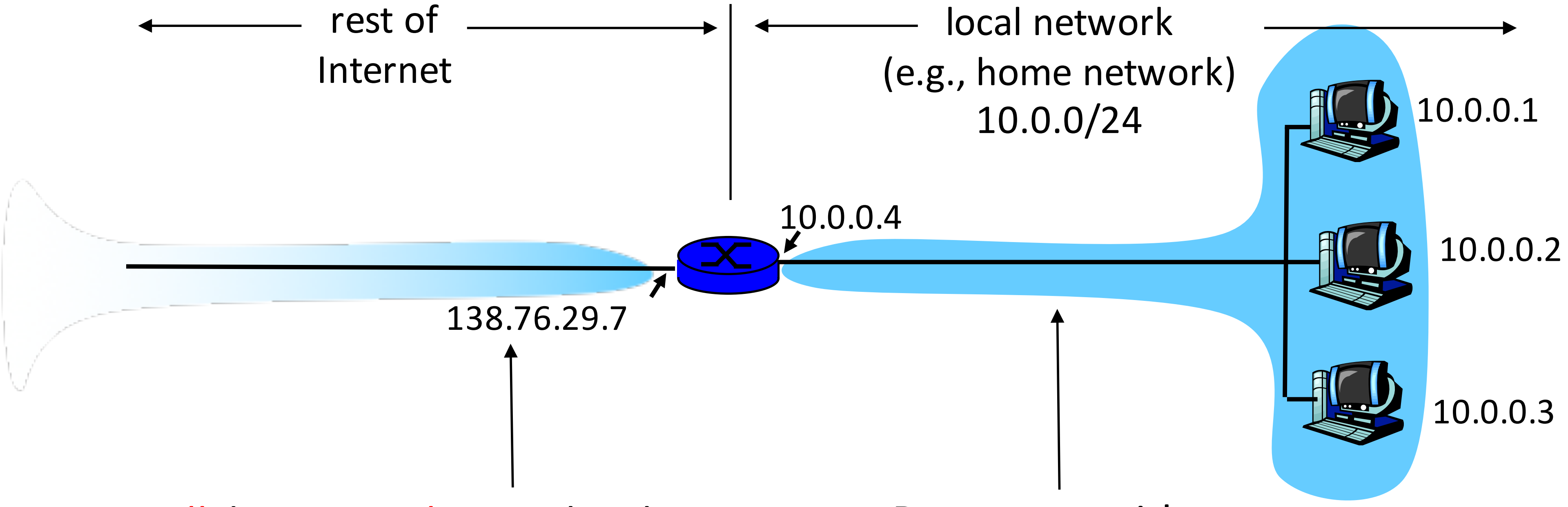
Q: How does an ISP get a block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers

- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

Cant get more IP addresses? well there is always…..

NAT: Network Address Translation



All datagrams *leaving* local network have same single source NAT IP address: 138.76.29.7, different source port numbers

Datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

NAT: Network Address Translation

- Motivation: local network uses just one IP address as far as outside world is concerned:
  - range of addresses not needed from ISP: just one IP address for all devices
  - can change addresses of devices in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - devices inside local net not explicitly addressable, visible by outside world (a security plus).

NAT: Network Address Translation

Implementation: NAT router must:

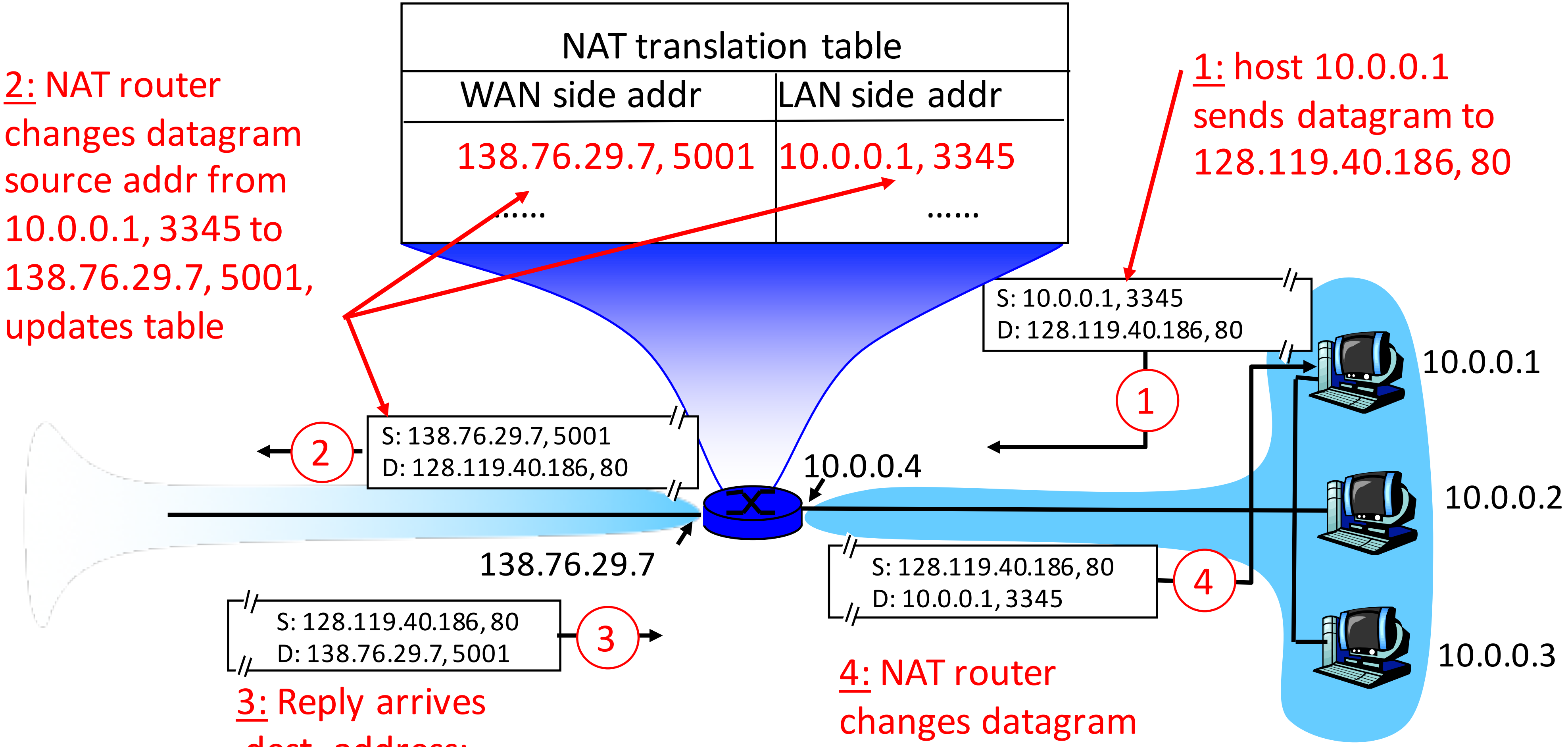
- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)

  . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr.

- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair

- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

NAT: Network Address Translation



NAT translation table

WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
……	……

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

S: 10.0.0.1, 3345
D: 128.119.40.186, 80

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

S: 138.76.29.7, 5001
D: 128.119.40.186, 80

S: 128.119.40.186, 80
D: 138.76.29.7, 5001

3: Reply arrives dest. address: 138.76.29.7, 5001

S: 128.119.40.186, 80
D: 10.0.0.1, 3345

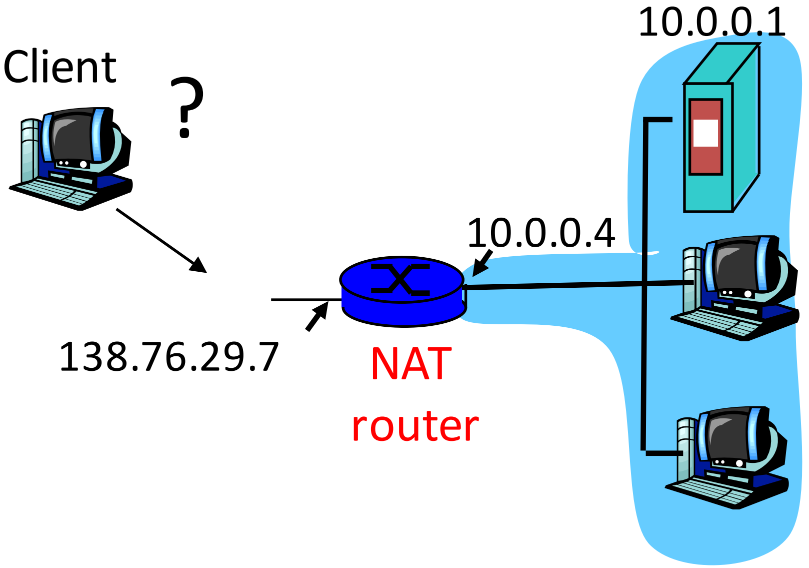
4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

NAT: Network Address Translation

- 16-bit port-number field:
  - – 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - – routers should only process up to layer 3
  - – violates end-to-end argument (?)
    - NAT possibility must be taken into account by app designers, eg, P2P applications
  - – address shortage should instead be solved by IPv6

NAT traversal problem

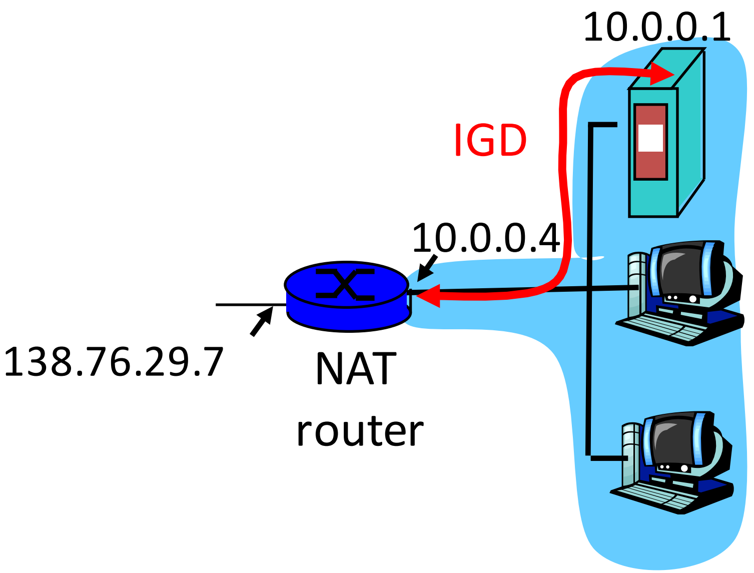
- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATted address: 138.76.29.7
- solution 1: statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



NAT traversal problem

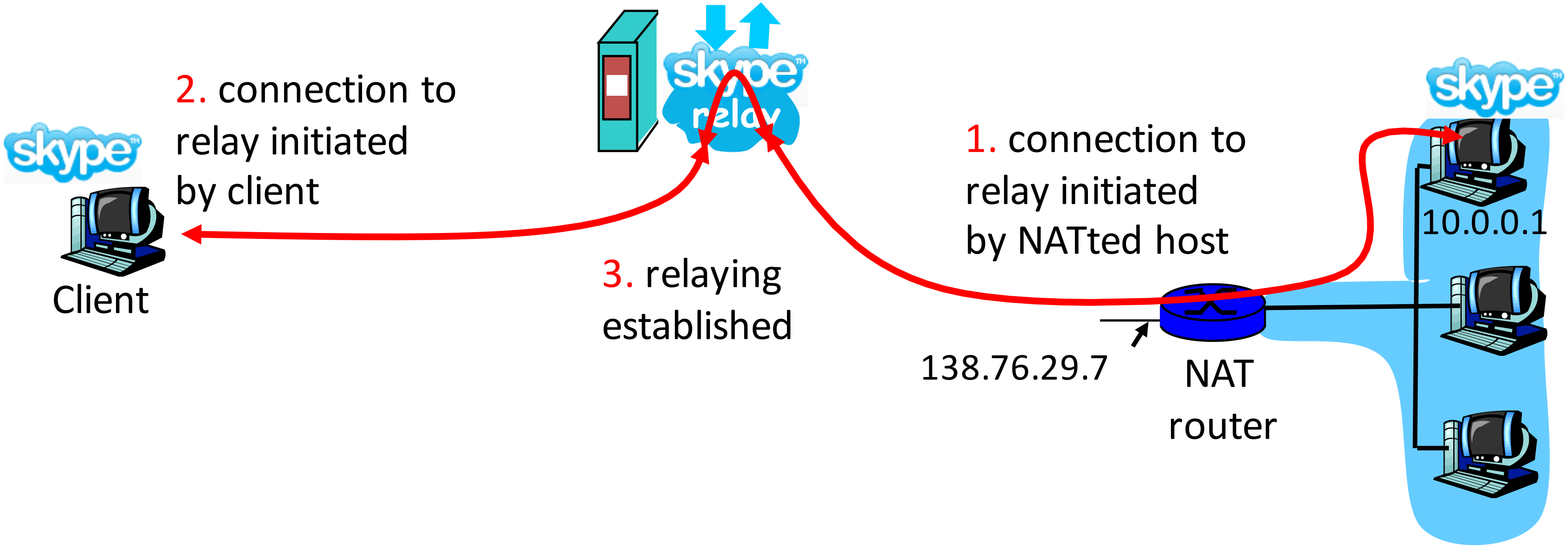
- solution 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATted host to:
  - ❖learn public IP address (138.76.29.7)
  - ❖add/remove port mappings (with lease times)

i.e., automate static NAT port map configuration



NAT traversal problem

- solution 3: relaying (used in Skype)
  - NATed client establishes connection to relay
  - External client connects to relay
  - relay bridges packets between to connections


2. connection to relay initiated by client
skype
Client
skype relay
3. relaying established
1. connection to relay initiated by NATted host
skype
10.0.0.1
138.76.29.7
NAT router

Remember this? Traceroute at work…

traceroute: rio.cl.cam.ac.uk to munnari.oz.au

(tracepath on pwf is similar)

Three delay measurements from rio.cl.cam.ac.uk to gatwick.net.cl.cam.ac.uk

```
traceroute munnari.oz.au
traceroute to munnari.oz.au (202.29.151.3), 30 hops max, 60 byte packets
 1  gatwick.net.cl.cam.ac.uk (128.232.32.2)  0.416 ms  0.384 ms  0.427 ms
 2  cl-sby.route-nwest.net.cam.ac.uk (193.60.89.9)  0.393 ms  0.440 ms  0.494 ms
 3  route-nwest.route-mill.net.cam.ac.uk (192.84.5.137)  0.407 ms  0.448 ms  0.501 ms
 4  route-mill.route-enet.net.cam.ac.uk (192.84.5.94)  1.006 ms  1.091 ms  1.163 ms
 5  xe-11-3-0.camb-rbr1.eastern.ja.net (146.97.130.1)  0.300 ms  0.313 ms  0.350 ms
 6  ae24.lowdss-sbr1.ja.net (146.97.37.185)  2.679 ms  2.664 ms  2.712 ms
 7  ae28.londhx-sbr1.ja.net (146.97.33.17)  5.955 ms  5.953 ms  5.901 ms
 8  janet.mx1.lon.uk.geant.net (62.40.124.197)  6.059 ms  6.066 ms  6.052 ms
 9  ae0.mx1.par.fr.geant.net (62.40.98.77)  11.742 ms  11.779 ms  11.724 ms
10  ae1.mx1.mad.es.geant.net (62.40.98.64)  27.751 ms  27.734 ms  27.704 ms
   -------------------------------------------------------------------------
11  mb-so-02-v4.bb.tein3.net (202.179.249.117)  138.296 ms  138.314 ms  138.282 ms
12  sg-so-04-v4.bb.tein3.net (202.179.249.53)  196.303 ms  196.293 ms  196.264 ms
13  th-pr-v4.bb.tein3.net (202.179.249.66)  225.153 ms  225.178 ms  225.196 ms
14  pyt-thairen-to-02-bdr-pyt.uni.net.th (202.29.12.10)  225.163 ms  223.343 ms  223.363 ms
15  202.28.227.126 (202.28.227.126)  241.038 ms  240.941 ms  240.834 ms
16  202.28.221.46 (202.28.221.46)  287.252 ms  287.306 ms  287.282 ms
17  * * *
18  * * *
19  * * *
20  coe-gw.psu.ac.th (202.29.149.70)  241.681 ms  241.715 ms  241.680 ms
21  munnari.OZ.AU (202.29.151.3)  241.610 ms  241.636 ms  241.537 ms
```

trans-continent link

* means no response (probe lost, router not replying)

Traceroute and ICMP

- Source sends series of UDP segments to dest
  - First has TTL =1
  - Second has TTL=2, etc.
  - Unlikely port number
- When nth datagram arrives to nth router:
  - Router discards datagram
  - And sends to source an ICMP message (type 11, code 0)
  - Message includes name of router& IP address

- When ICMP message arrives, source calculates RTT
- Traceroute does this 3 times

Stopping criterion

- UDP segment eventually arrives at destination host
- Destination returns ICMP "host unreachable" packet (type 3, code 3)
- When source gets this ICMP, stops.

ICMP: Internet Control Message Protocol

- used by hosts & routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer "above" IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

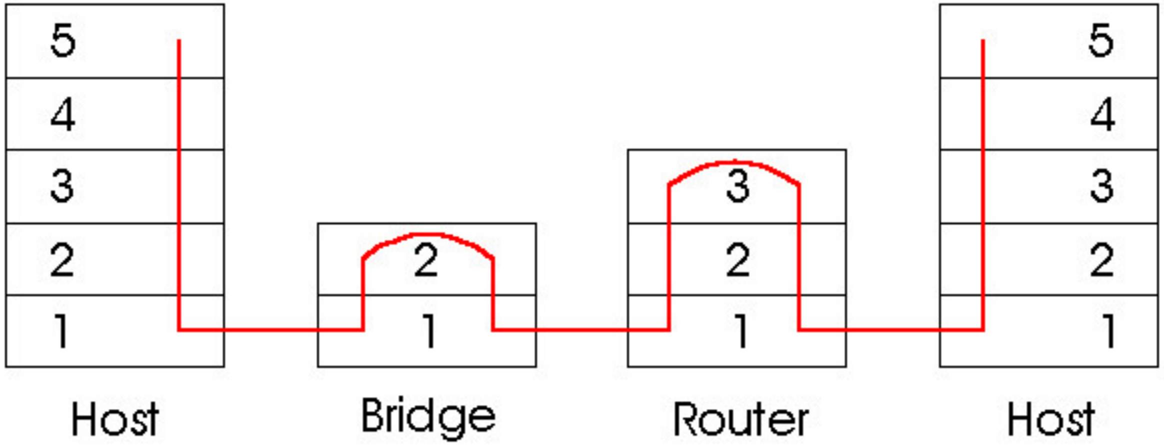
Gluing it together:

How does my Network (address) interact

with my Data-Link (address) ?

Switches vs. Routers Summary

- both store-and-forward devices
  - routers: network layer devices (examine network layer headers)
  - switches are link layer devices
- routers maintain routing tables, implement routing algorithms
- switches maintain switch tables, implement filtering, learning algorithms



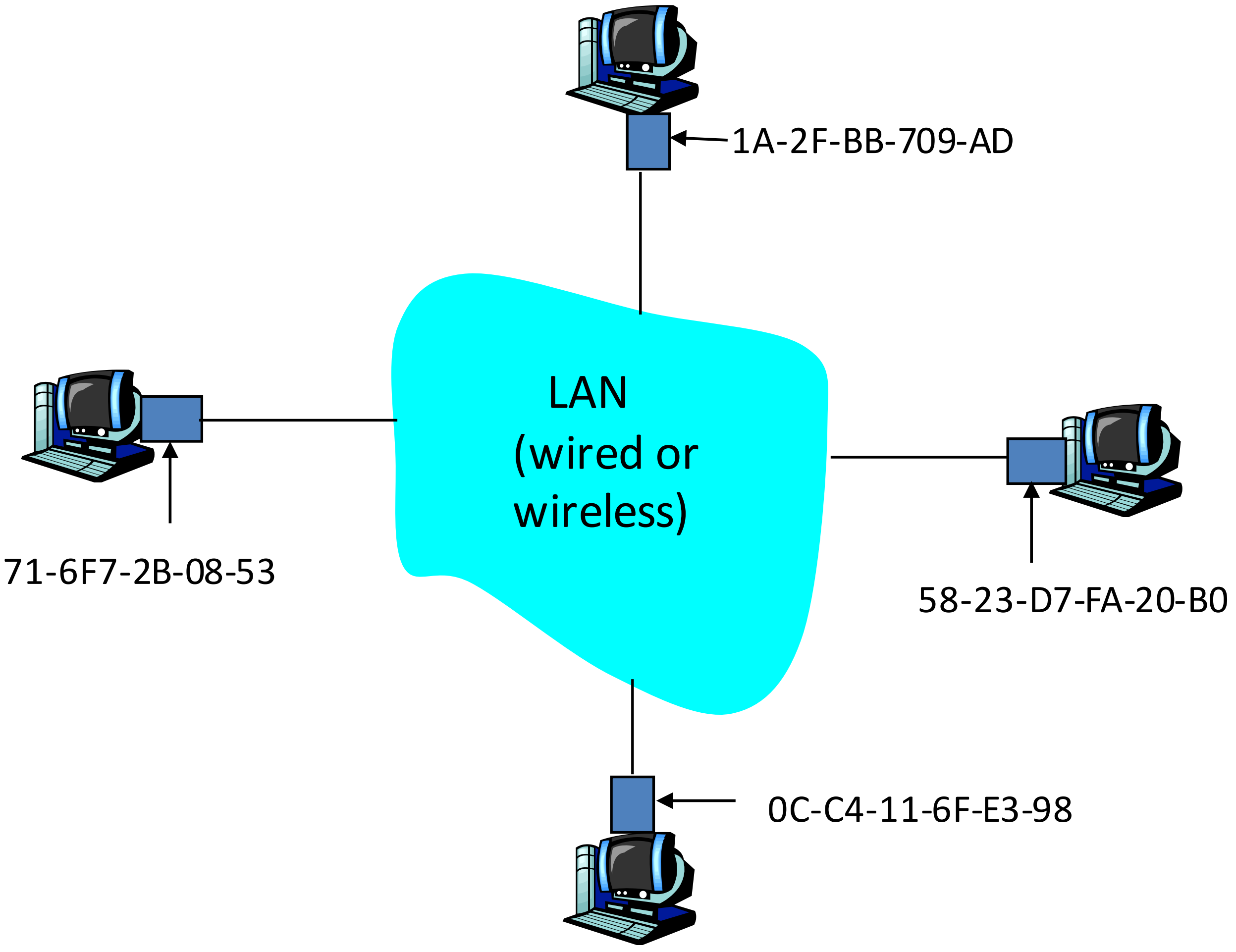
MAC Addresses (and IPv4 ARP)

or How do I glue my network to my data-link?

- 32-bit IP address:
  - *network-layer* address
  - used to get datagram to destination IP subnet
- MAC (or LAN or physical or Ethernet) address:
  - function: *get frame from one interface to another physically-connected interface (same network)*
  - 48 bit MAC address (for most LANs)
    - burned in NIC ROM, also (commonly) software settable

LAN Addresses and ARP

Each adapter on LAN has unique LAN address



Ethernet
Broadcast address =
FF-FF-FF-FF-FF-FF

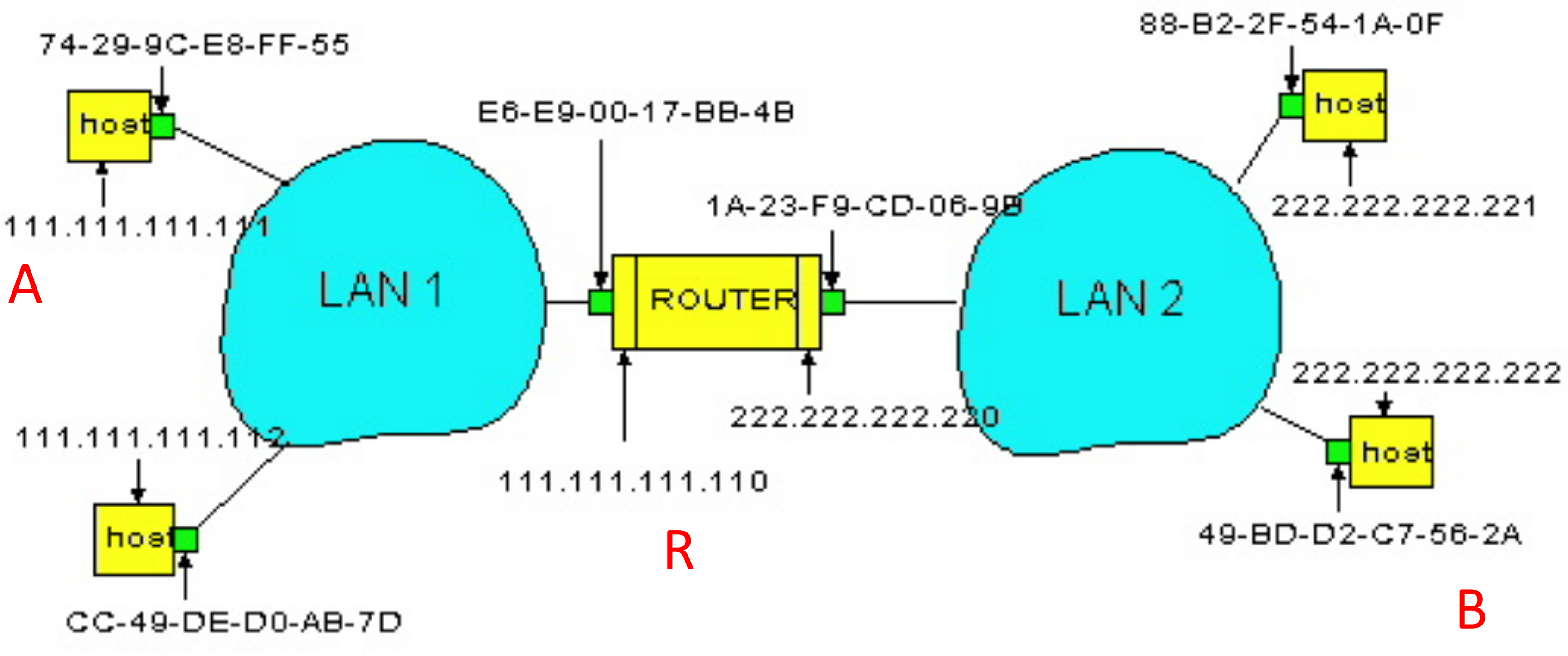
🟦 = adapter

Address Resolution Protocol

- Every node maintains an ARP table
  - <IP address, MAC address> pair
- Consult the table when sending a packet
  - Map destination IP address to destination MAC address
  - Encapsulate and transmit the data packet
- But: what if IP address not in the table?
  - Sender broadcasts: "**Who has IP address 1.2.3.156**?"
  - Receiver responds: "**MAC address 58-23-D7-FA-20-B0**"
  - Sender caches result in its ARP table

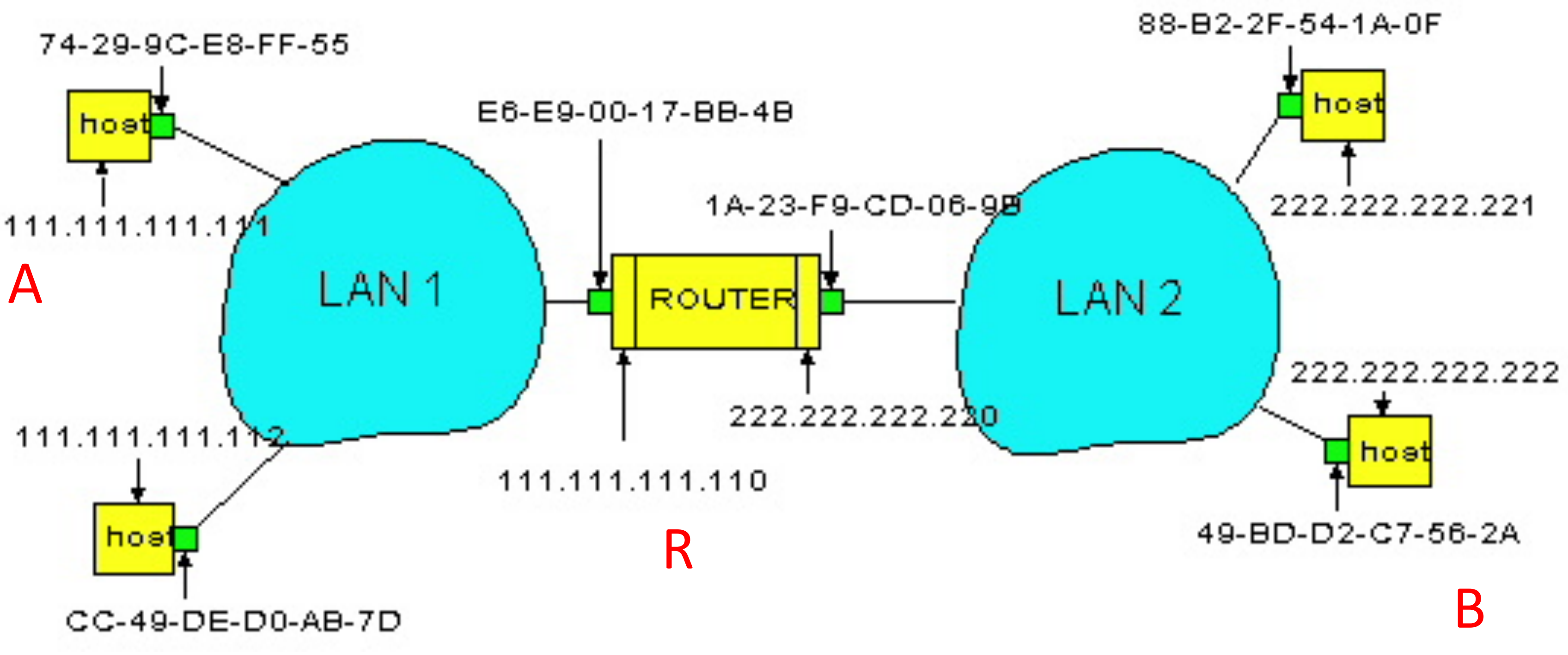
Example: A Sending a Packet to B

How does host A send an IP packet to host B?



Example: A Sending a Packet to B

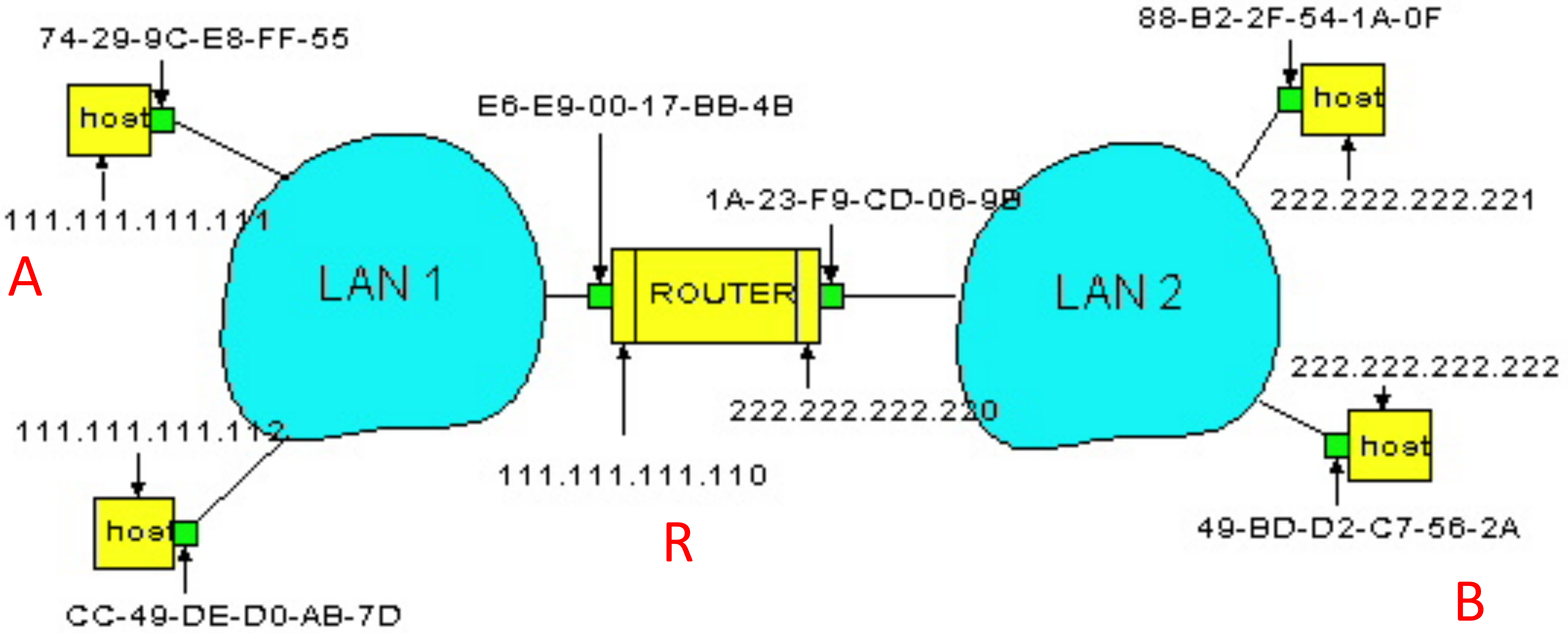
How does host A send an IP packet to host B?



1. **A sends packet to R.**
2. **R sends packet to B.**

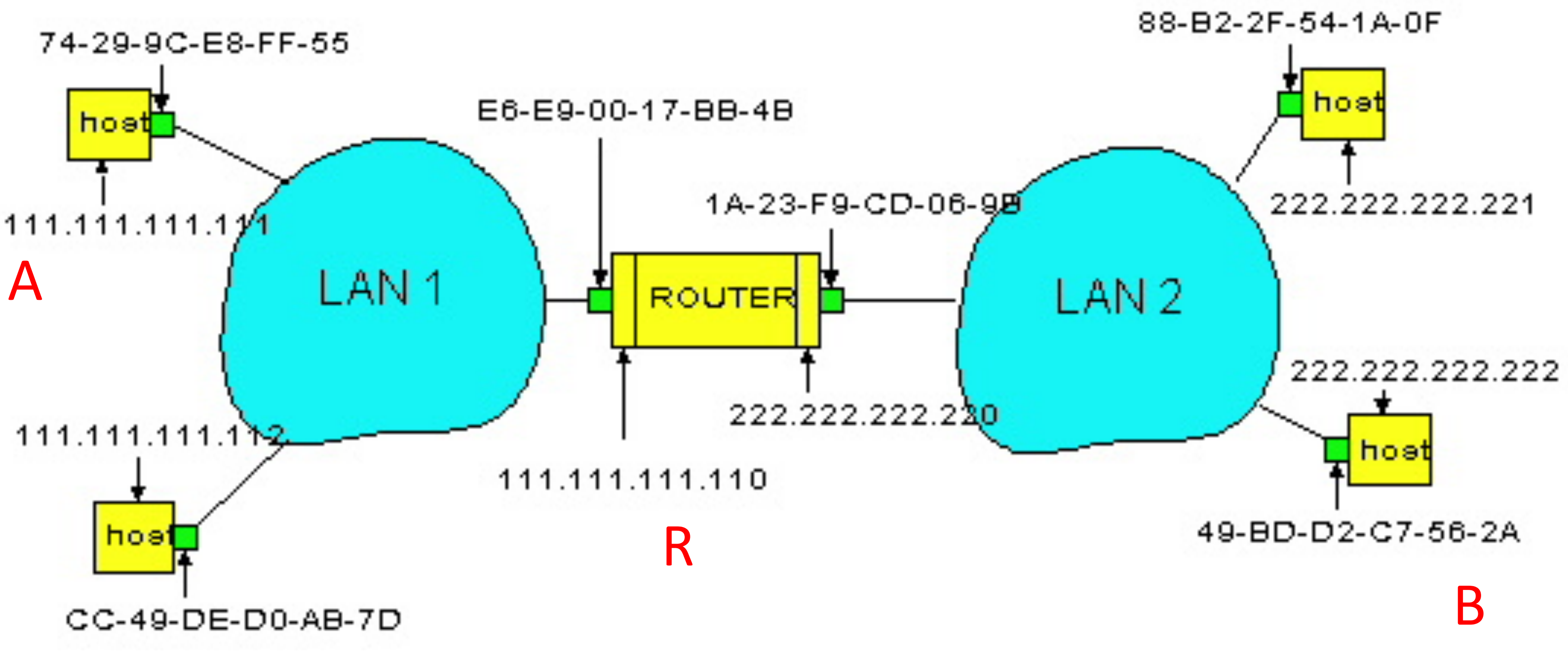
Host A Decides to Send Through R

- Host A constructs an IP packet to send to B
  - Source 111.111.111.111, destination 222.222.222.222
- Host A has a gateway router R
  - Used to reach destinations outside of 111.111.111.0/24
  - Address 111.111.111.110 for R learned via DHCP/config



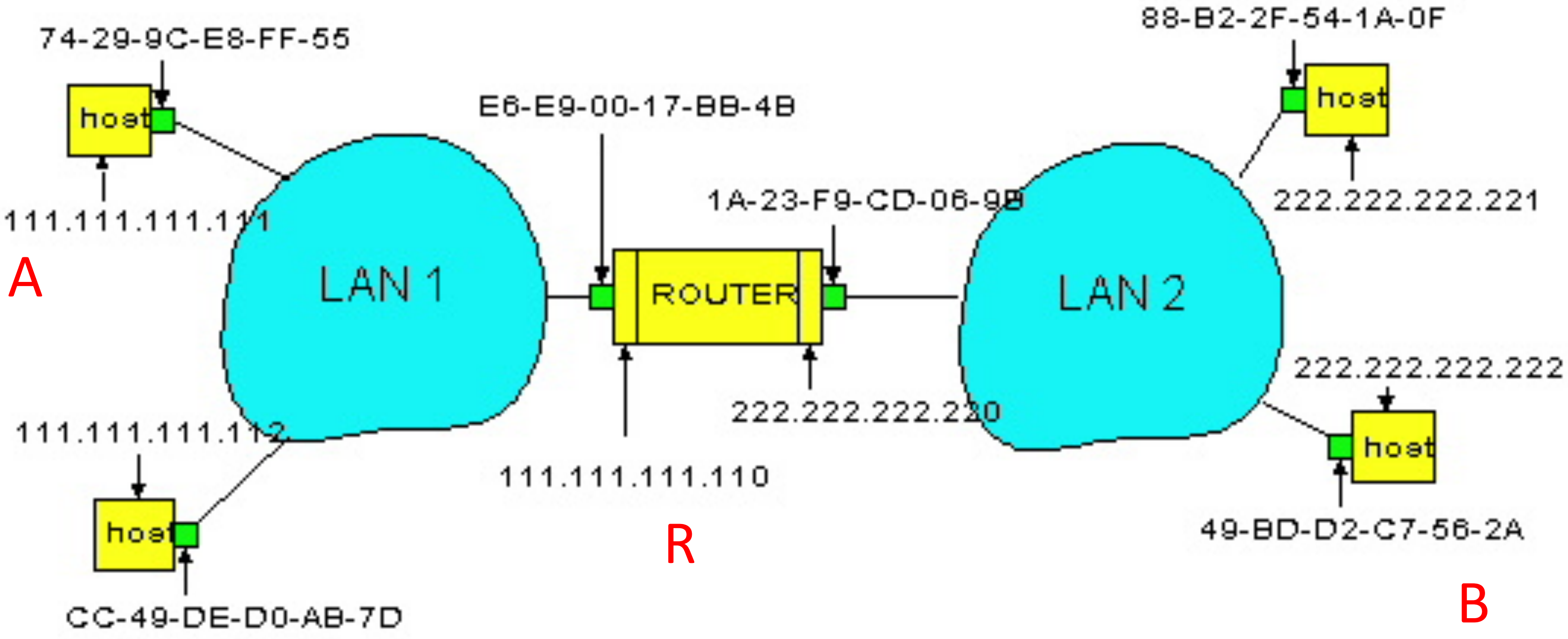
Host A Sends Packet Through R

- Host A learns the MAC address of R's interface
  - ARP request: broadcast request for 111.111.111.110
  - ARP response: R responds with E6-E9-00-17-BB-4B
- Host A encapsulates the packet and sends to R



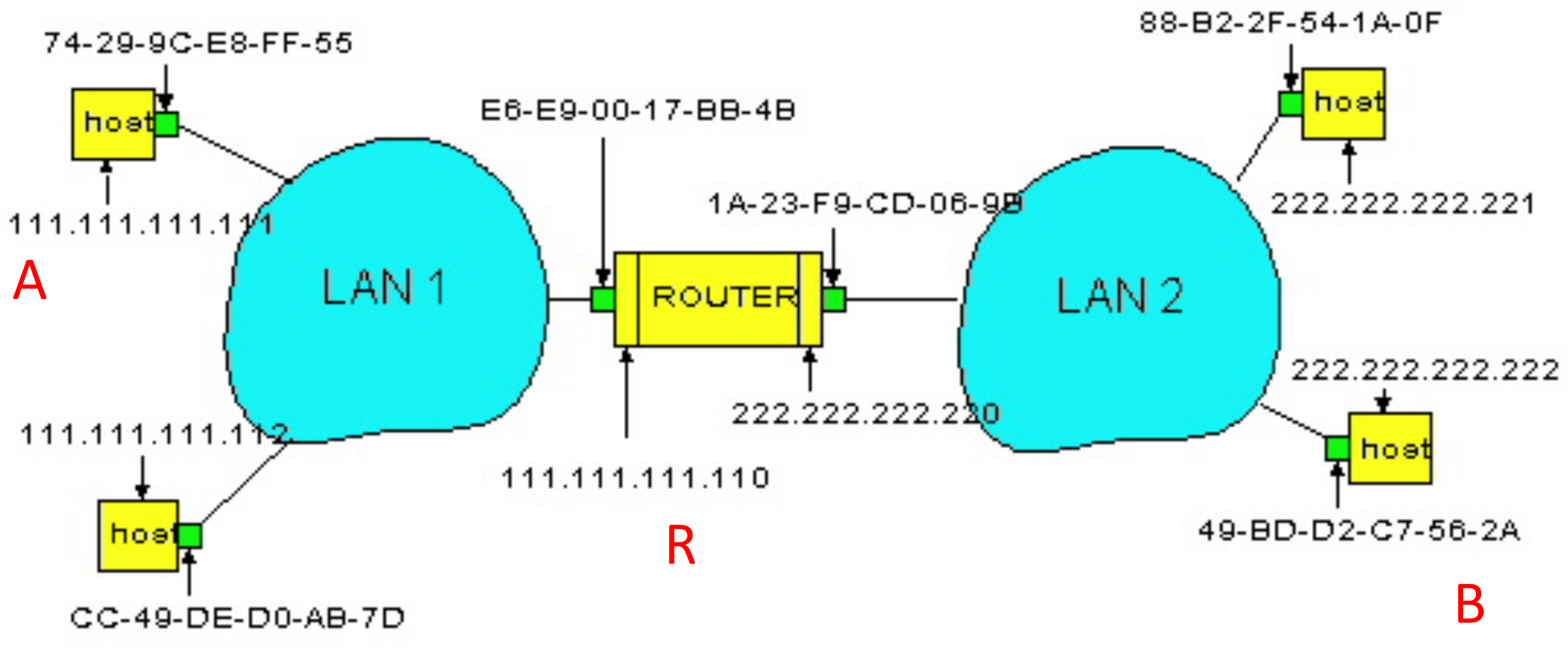
R Decides how to Forward Packet

- Router R's adaptor receives the packet
  - R extracts the IP packet from the Ethernet frame
  - R sees the IP packet is destined to 222.222.222.222
- Router R consults its forwarding table
  - Packet matches 222.222.222.0/24 via other adaptor



R Sends Packet to B

- Router R’s learns the MAC address of host B
  - ARP request: broadcast request for 222.222.222.222
  - ARP response: B responds with 49-BD-D2-C7-52A
- Router R encapsulates the packet and sends to B



Security Analysis of ARP

- Impersonation
  - Any node that hears request can answer …
  - … and can say whatever they want

- Actual legit receiver never sees a problem
  - Because even though later packets carry its IP address, its NIC doesn’t capture them since not its MAC address

Key Ideas in Both ARP and DHCP

- Broadcasting: Can use broadcast to make contact
  - Scalable because of limited size
- Caching: remember the past for a while
  - Store the information you learn to reduce overhead
  - Remember your own address & other host's addresses
- Soft state: eventually forget the past
  - Associate a time-to-live field with the information
  - … and either refresh or discard the information
  - Key for robustness in the face of unpredictable change

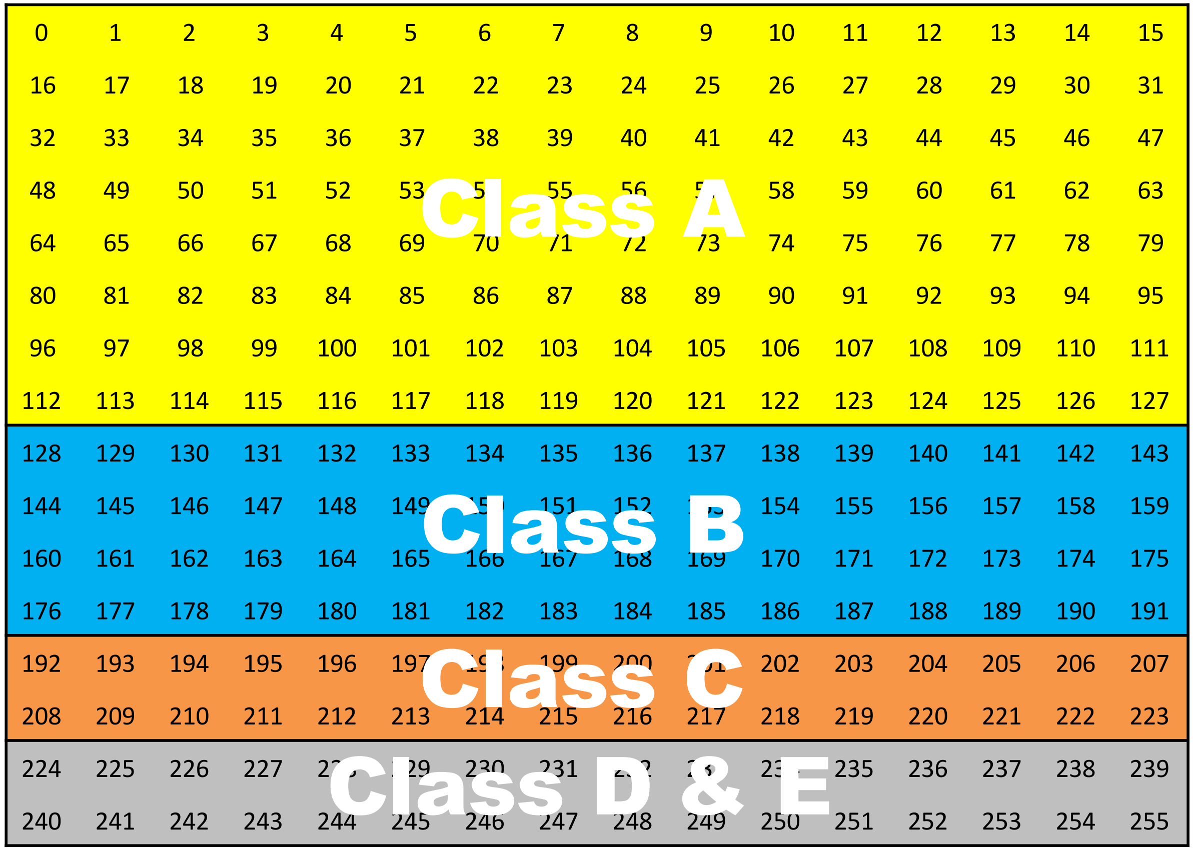
Why Not Use DNS-Like Tables?

- When host arrives:
  - Assign it an IP address that will last as long it is present
  - Add an entry into a table in DNS-server that maps MAC to IP addresses

- Answer:
  - Names: explicit creation, and are plentiful
  - Hosts: come and go without informing network
    - Must do mapping on demand
  - Addresses: not plentiful, need to reuse and remap
    - Soft-state enables dynamic reuse

No More IPv4 Addresses

- IPv4 address space in terms of /8's



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- 24 /8’s on January 12, 2010

0	1	2	3	4	**5**	6	7	8	9	10	11	12	13	**14**	15
16	17	18	19	20	21	22	**23**	24	25	26	27	28	29	30	**31**
32	33	34	35	**36**	**37**	38	**39**	40	41	**42**	43	44	45	46	47
48	**49**	**50**	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	**177**	178	**179**	180	**181**	182	183	184	**185**	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	**223**
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- 20 /8’s on April 10, 2010

0	1	2	3	4	🟩 5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	🟩 23	24	25	26	27	28	29	30	🟩 31
32	33	34	35	🟩 36	🟩 37	38	🟩 39	40	41	🟩 42	43	44	45	46	47
48	🟩 49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	🟩 100	🟩 101	🟩 102	🟩 103	🟩 104	🟩 105	🟩 106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
🟩 176	🟩 177	178	🟩 179	180	🟩 181	182	183	184	🟩 185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- 13 /8’s on May 8, 2010

0	1	2	3	4	**5**	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	**23**	24	25	26	27	28	29	30	**31**
32	33	34	35	36	**37**	38	**39**	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	**100**	101	**102**	**103**	**104**	105	**106**	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	**179**	180	181	182	183	184	**185**	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- 7 /8’s on November 30th, 2010

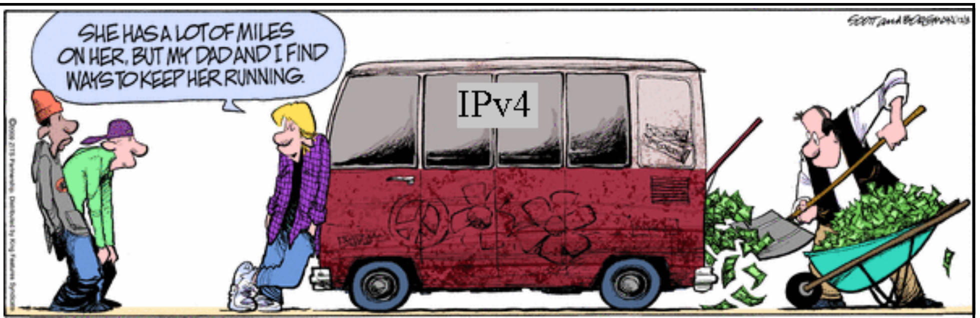
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	**39**	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	**102**	**103**	**104**	105	**106**	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	**179**	180	181	182	183	184	**185**	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

No More IPv4 Addresses

- **0 /8’s** on January 31st, 2011!

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

IPv6


SHE HAS A LOT OF MILES ON HER. BUT MY DAD AND I FIND WAYS TO KEEP HER RUNNING.
IPv4

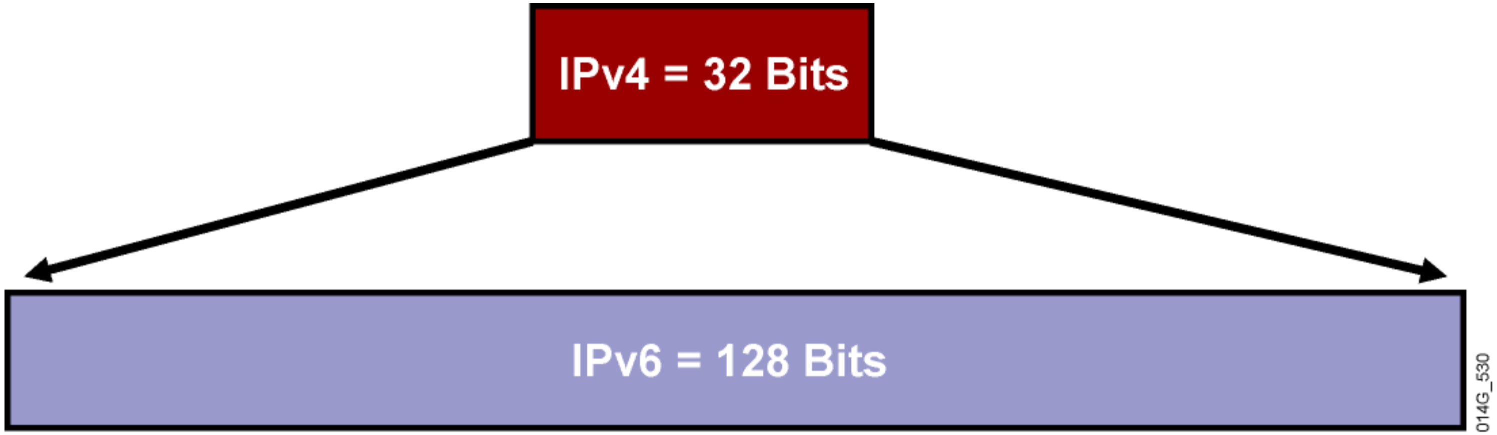
- Motivated (prematurely) by address exhaustion
  - Address field ***four*** times as long

- Steve Deering focused on simplifying IP
  - Got rid of all fields that were not absolutely necessary
  - “Spring Cleaning” for IP

- Result is an elegant, if unambitious, protocol

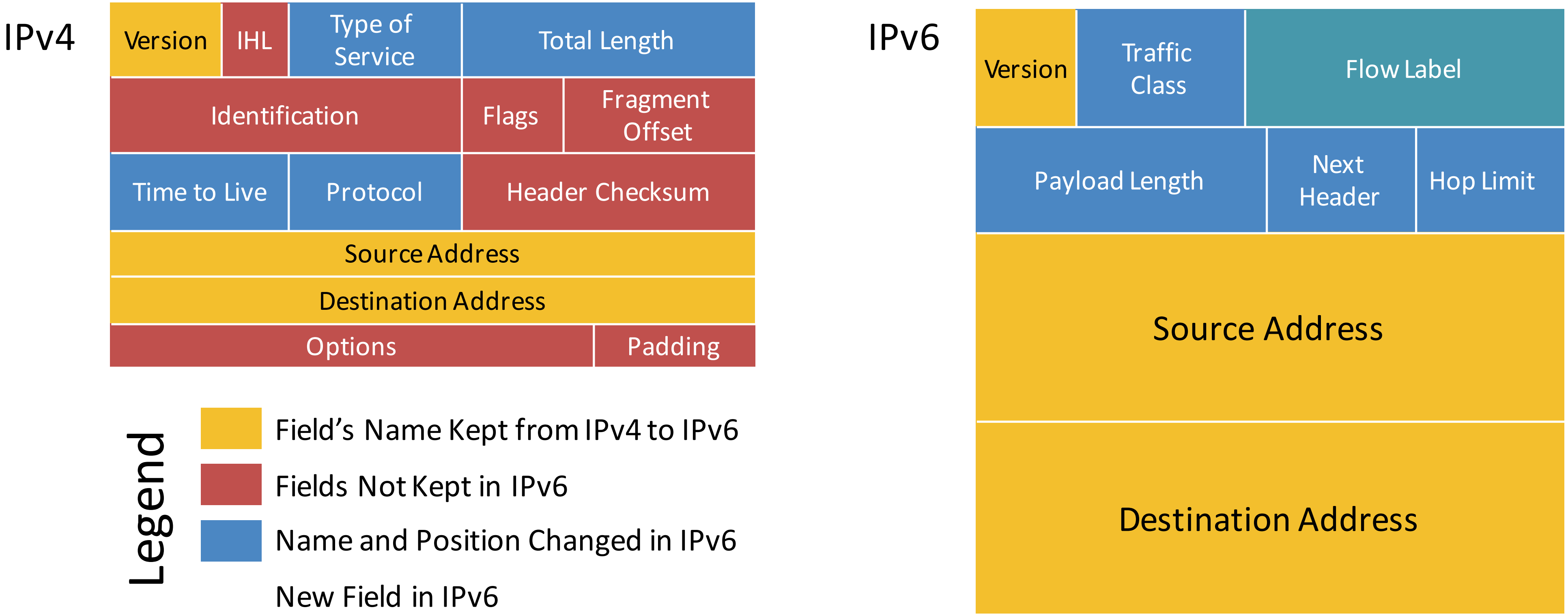
Larger Address Space

- IPv4 = 4,294,967,295 addresses
- IPv6 = 340,282,366,920,938,463,374,607,432,768,211,456 addresses
- 4x in number of bits translates to **huge** increase in address space!



Other Significant Protocol Changes

- Increased minimum MTU from 576 to 1280
- No enroute fragmentation… fragmentation only at source
- Header changes
- Replace broadcast with multicast



IPv4

Version | IHL | Type of Service | Total Length
Identification | Flags | Fragment Offset
Time to Live | Protocol | Header Checksum
Source Address
Destination Address
Options | Padding

IPv6

Version | Traffic Class | Flow Label
Payload Length | Next Header | Hop Limit
Source Address
Destination Address

Legend

- 🟨 Field's Name Kept from IPv4 to IPv6
- 🟥 Fields Not Kept in IPv6
- 🟦 Name and Position Changed in IPv6
- New Field in IPv6

IPv4	IPv6
Addresses are 32 bits (4 bytes) in length.	Addresses are 128 bits (16 bytes) in length
Address (A) resource records in DNS to map host names to IPv4 addresses.	Address (AAAA) resource records in DNS to map host names to IPv6 addresses.
Pointer (PTR) resource records in the IN-ADDR.ARPA DNS domain to map IPv4 addresses to host names.	Pointer (PTR) resource records in the IP6.ARPA DNS domain to map IPv6 addresses to host names.
IPSec is optional and should be supported externally	IPSec support is not optional
Header does not identify packet flow for QoS handling by routers	Header contains Flow Label field, which Identifies packet flow for QoS handling by router.
Both routers and the sending host fragment packets.	Routers do not support packet fragmentation. Sending host fragments packets
Header includes a checksum.	Header does not include a checksum.
Header includes options.	Optional data is supported as extension headers.
ARP uses broadcast ARP request to resolve IP to MAC/Hardware address.	Multicast Neighbor Solicitation messages resolve IP addresses to MAC addresses.
Internet Group Management Protocol (IGMP) manages membership in local subnet groups.	Multicast Listener Discovery (MLD) messages manage membership in local subnet groups.
Broadcast addresses are used to send traffic to all nodes on a subnet.	IPv6 uses a link-local scope all-nodes multicast address.
Configured either manually or through DHCP.	Does not require manual configuration or DHCP.
Must support a 576-byte packet size (possibly fragmented).	Must support a 1280-byte packet size (without fragmentation).

Roundup: Why IPv6?

- Larger address space

- Auto-configuration

- Cleanup
- Eliminate fragmentation
- Eliminate checksum
- Pseudo-header (w/o Hop Limit) covered by transport layer
- Flow label
- Increase minimum MTU from 576 to 1280
- Replace broadcasts with multicast

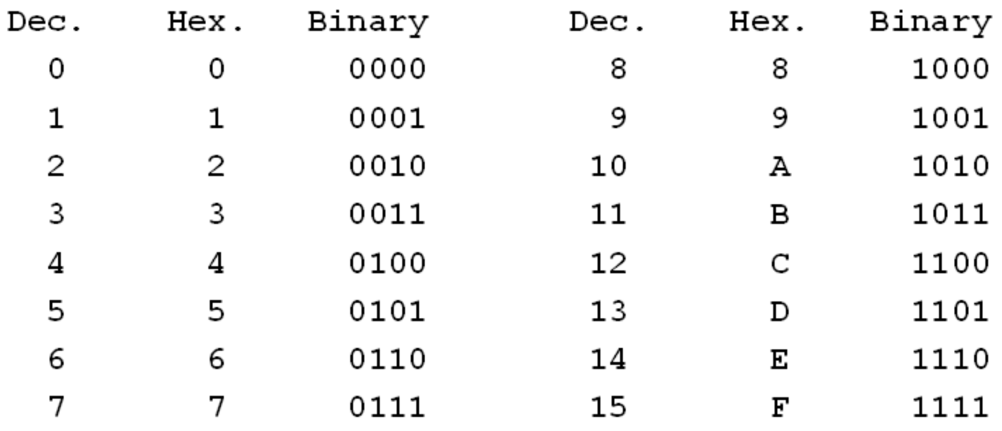
No Checksum!

- Provided by transport layer, if needed
- Ala TCP, includes pseudo-header
- Pseudo-header doesn’t include Hop Limit
  - No per-hop re-computation!
  - Allows end-to-end implementation (transport layer)
- UDP checksum required (wasn’t in IPv4) rfc6936: **No more zero**
- Pseudo-header added to ICMPv6 checksum

IPv6 Address Notation

- RFC 5952
- 128-bit IPv6 addresses are represented in:
  - Eight 16-bit segments
  - Hexadecimal (non-case sensitive) between 0000 and FFFF
  - Separated by colons
- Example:
  - `3ffe:1944:0100:000a:0000:00bc:2500:0d0b`
- Two rules for dealing with 0’s

One Hex digit = 4 bits



Dec.	Hex.	Binary	Dec.	Hex.	Binary
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

0’s Rule 1 – Leading 0’s

- The leading zeroes in any 16-bit segment do not have to be written.

- Example
  - 3ffe : 1944 : 0100 : 000a : 0000 : 00bc : 2500 : 0d0b
  - 3ffe : 1944 : 100 : a : 0 : bc : 2500 : d0b

3ffe:1944:100:a:0:bc:2500:d0b

0’s Rule 1 – Leading 0’s

- Can only apply to **leading zeros**… otherwise ambiguous results
- Example
  - 3ffe : 1944 : 100 : a : 0 : bc : 2500 : d0b
- Could be either
  - 3ffe : 1944 : **0**100 : **000**a : **000**0 : **00**bc : 2500 : **0**d0b
  - 3ffe : 1944 : 100**0** : a**000** : 0**000** : bc**00** : 2500 : d0b**0**
  - *Which is correct?*

0’s Rule 1 – Leading 0’s

- Can only apply to **leading zeros**… otherwise ambiguous results

- Example
  - `3ffe : 1944 :  100 :    a :    0 :   bc : 2500 :  d0b`

- Could be either
  - **`3ffe : 1944 : 0100 : 000a : 0000 : 00bc : 2500 : 0d0b`**
  - `3ffe : 1944 : 1000 : a000 : 0000 : bc00 : 2500 : d0b0`
  - *Which is correct?*

0's Rule 2 – Double Colon

- Any **single**, **contiguous** string of **16-bit segments** consisting **of all zeroes** can be represented with a **double colon**.

```
ff02 : 0000 : 0000 : 0000 : 0000 : 0000 : 0000 : 0005
ff02 :    0 :    0 :    0 :    0 :    0 :    0 :    5
ff02 :                                         :    5
```

ff02::5

0’s Rule 2 – Double Colon

- Only a **single** contiguous string of all-zero segments can be represented with a double colon.

- Example:

```
2001 : 0d02 : 0000 : 0000 : 0014 : 0000 : 0000 : 0095
```

- Both of these are correct

```
2001 :  d02 ::                0014 :    0 :    0 :   95
```

OR

```
2001 :  d02 :    0 :    0 :   14 ::                  95
```

0’s Rule 2 – Double Colon

- However, using double colon more than once creates ambiguity
- Example

```
2001:d02::14::95

2001:0d02:0000:0000:0000:0014:0000:0095
2001:0d02:0000:0000:0014:0000:0000:0095
2001:0d02:0000:0014:0000:0000:0000:0095
```

Network Prefixes

- In IPv4, network portion of address can by identified by either
  - **Netmask**: `255.255.255.0`
  - **Bitcount**: `/24`

- Only use bitcount with IPv6

`3ffe:1944:100:a::/64`

Special IPv6 Addresses

- Default route: **`::/0`**
- Unspecified Address: **`::/128`**
  - Used in SLAAC (coming later)
- Loopback/Local Host: **`::1/128`**
  - No longer a /8 of addresses but a single address

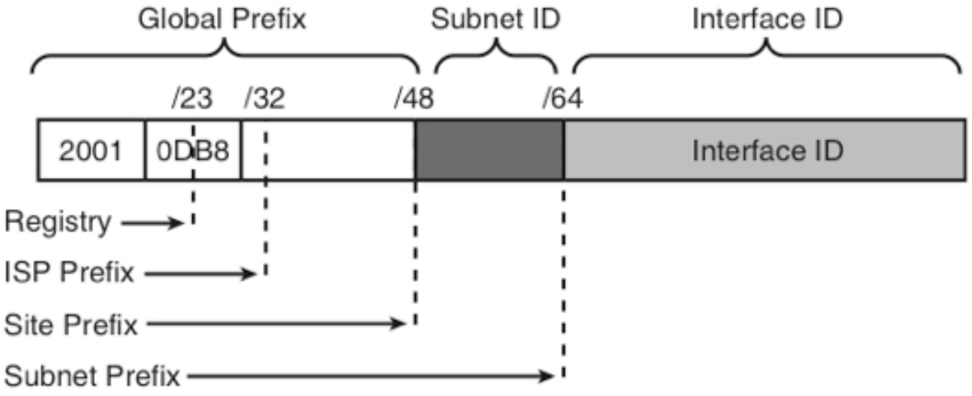
Types of IPv6 Addresses

- RFC 4291– “IPv6 Addressing Architecture”
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Types of IPv6 Addresses

- RFC 4291– "IPv6 Addressing Architecture"
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

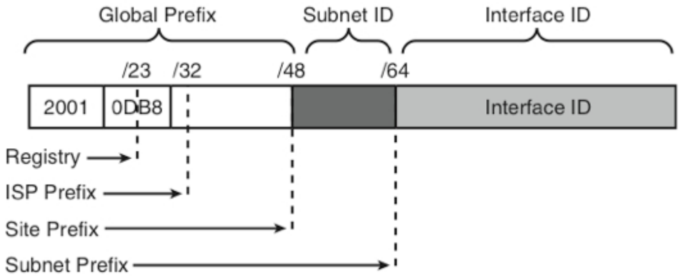
Global Unicast Addresses



- Globally routable addresses
  - RFC 3587
- 3 parts
  - 48 bit **global routing prefix**
    - Hierarchically-structured value assigned to a site
    - Further broken down into Registry, ISP Prefix, and Site Prefix fields
  - 16 bit **Subnet ID**
    - Identifier of a subnet within a site
  - 64(!) bit **Interface ID**
    - Identify an interface on a subnet
    - Motivated by expected use of MAC addresses (IEEE EUI-64 identifiers) in SLAAC…
  - Except GUAs that start with ‘000…’ binary
    - Used for, e.g., “IPv4-Mapped IPv6 Addresses” (RFC 4308)

Global Unicast Addresses

- Current **ARIN** policy is to assign no longer than /32 to an ISP
  - **A**merican **R**egistry for **I**nternet **N**umbers
  - https://www.arin.net/policy/nrpm.html
  - UCSC allocation is `2607:F5F0::/32`



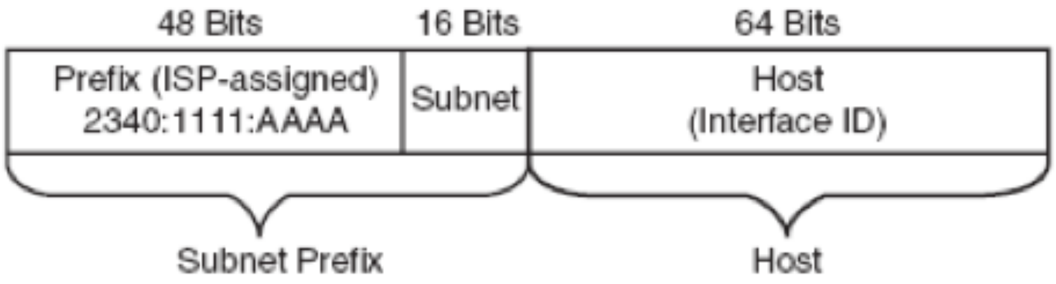
- IANA currently assigning addresses that start with ‘`001…`’ binary
  - `2000::/3`
    - `(2000:: - 3FFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF)`
  - Supports
    - Maximum 2^{29} (536,870,912… 1/8 of an **Internet address space** of) ISPs
    - 2^{45} sites (equivalent to 8,192 **IAS**s of sites!)

- ISP can delegate a minimum of 2^{16}, or 65,535 site prefixes
  - Difference between Global Prefix (48 bits) and ISP Prefix (32 bits)

Subnetting GUAs

- Each site can identify 2^{16} (65,535) subnets

```
2340:1111:AAAA:1::/64
2340:1111:AAAA:2::/64
2340:1111:AAAA:3::/64
2340:1111:AAAA:4::/64
...
```



- Subnet has address space of 2^{64}… an IAS of IASs!

- Can extend the subnet ID into the interface ID portion of the address…
  - Sacrifice ability to use EUI-64 style of SLAAC…
  - Maybe not a bad thing… more later

These are huge numbers!!

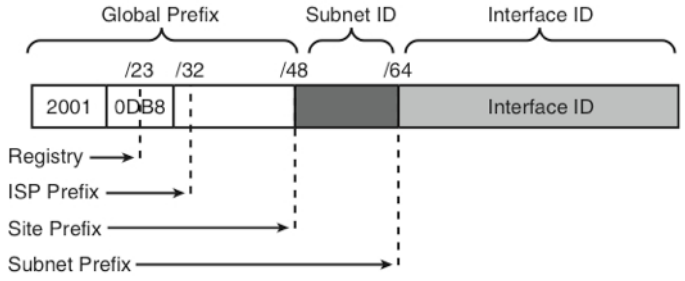
- Assume average /16's allocated to ISPs and /22's allocated to sites in IPv4

IPv6 2000::/3 block

Description	Range	Count	Scale vs IPv4
Total # ISPs	/3 – /32	2^{29} = 512M	**9,362**
Total # Sites	/3 – /48	2^{42} = 4T	**1.2M**
Sites/ISP	/48 – /64	2^{16} = 64K	**1,024**

IPv4 class A, B, and C blocks

Total # ISPs	/16 * 7/8	57K
Total # Sites	/22 * 7/8	3.6M
Sites/ISP	/16 - /22	2^{6} = 64



- ***And this keeps assumption of /64 subnets!***

IPv6 Address Space

- **Allocated**
  - 2000::/3Global Unicast
  - FC00::/7Unique Local Unicast
  - FE80::/10 Link Local Unicast
  - FF00::/8Multicast
- ***Accounts for a bit more than 2^{125} of the address space.***

- **Unallocated** (“Reserved by IETF”)
  - /3’s – 4000::, 6000::, 8000::, A000::, C000::
  - /4’s – 1000::, E000::
  - /5’s – 0800::, F000::
  - /6’s – 0400::, F800::
  - /7’s – 0200::
  - /8’s – 0000::, 0100::
  - /9’s – FE00::
  - /10’s – FEC0::
- ***Accounts for a little more than 2^{127}, or more than half, of the address space!!***

http://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xml

Problem with /64 Subnets

- Scanning a subnet becomes a DoS attack!
  - Creates IPv6 version of 2^{64} ARP entries in routers
  - Exhaust address-translation table space

- So now we have:

```
ping6 ff02::1 All nodes in broadcast domain
ping6 ff02::2 All routers in broadcast domain
```

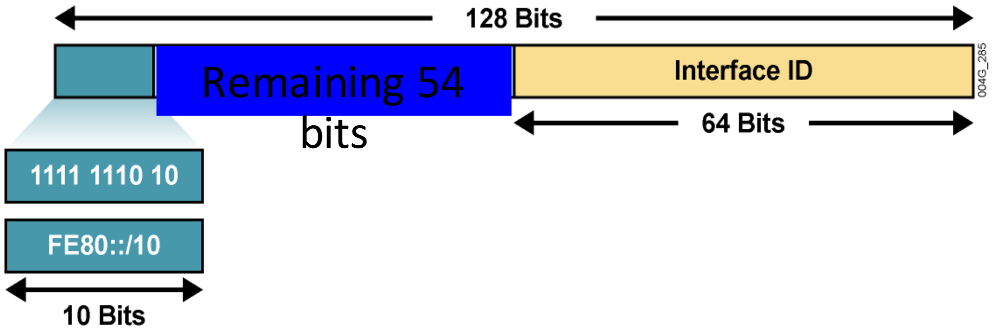
- Solutions
  - RFC 6164 recommends use of /127 to protect router-router links
  - RFC 3756 suggest "clever cache management" to address more generally

Types of IPv6 Addresses

- RFC 4291– "IPv6 Addressing Architecture"
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Link-Local Addresses

- '`11111110 10`…' binary (`FE80::/10`)
  - According to RFC 4291 bits 11-64 should be 0's… so really `FE80::/64`?
- For use on a single link.
  - Automatic address configuration
  - Neighbor discovery (IPv6 ARP)
  - When no routers are present
  - Routers must not forward



- Addresses "chicken-or-egg" problem… need an address to get an address.
- Address assignment done unilaterally by node (later)
- IPv4 has link-local address (`169.254/16`, RFC 3927)
  - Only used if no globally routable addresses available

Types of IPv6 Addresses

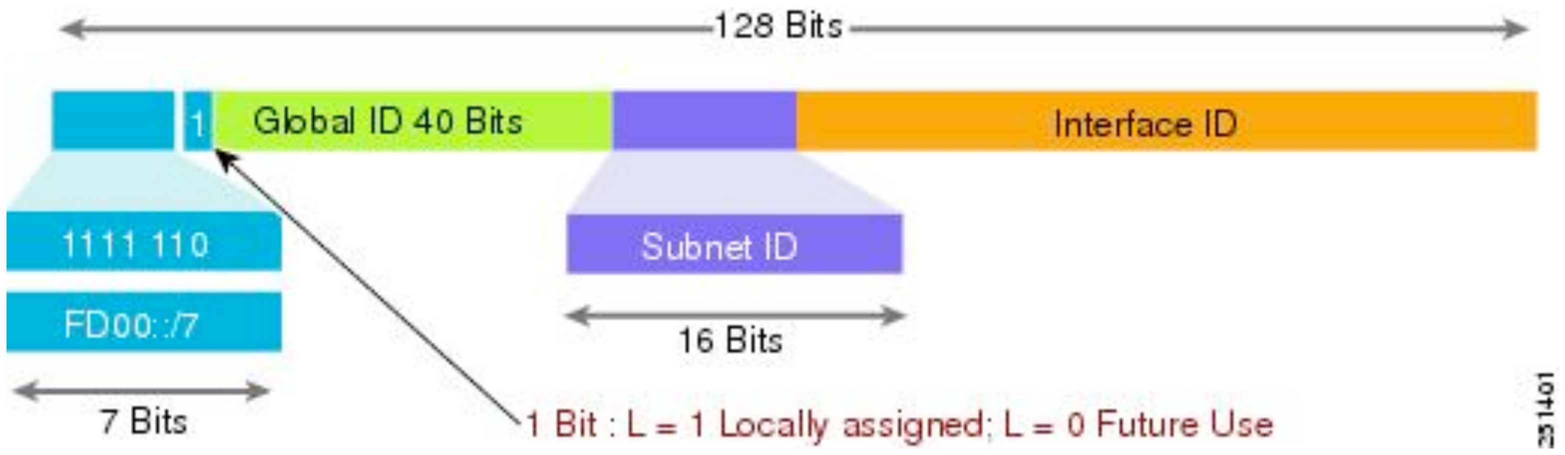
- RFC 4291– "IPv6 Addressing Architecture"
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Unique Local Addresses

- ‘1111110…’ binary (`FC00::/7`)
- Globally unique addresses intended for local communication
  - IPv6 equivalent of IPv4 RFC 1918 addresses
- Defined in RFC 4193
  - Replace “site local” addresses defined in RFC 1884, deprecated in RFC 3879
- Should not be installed in global DNS
  - Can be installed in “local DNS”

Unique Local Addresses

- 4 parts
  - "**L**" bit always 1
  - **Global ID** (40 bits) randomly generated to enforce the idea that these addresses are not to be globally routed or aggregated
  - **Subnet ID** (16 bits)… same as Globally Unique Subnet ID
  - **Interface ID** (64 bits)… same as Globally Unique Interface ID

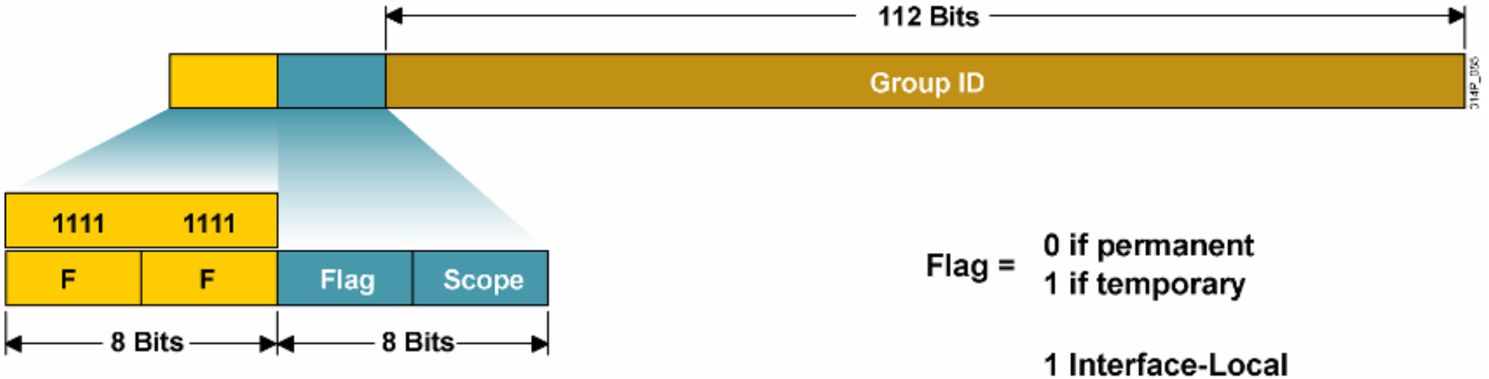


Types of IPv6 Addresses

- RFC 4291– "IPv6 Addressing Architecture"
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Multicast Addresses

- ‘11111111…’ binary (FF00::/8)
- Equivalent to IPv4 multicast (224.0.0.0/8)
- 3 parts
  - **Flag** (4 bits)
  - **Scope** (4 bits)



Flag = 0 if permanent
1 if temporary

Scope =
1 Interface-Local
2 Link-Local
3 ~~Subnet-Local~~
4 Admin-Local
5 Site-Local
8 Organization
E Global

Reserved Multicast Addresses

- All nodes
  - **FF01::1** – interface-local; used for loopback multicast transmissions
  - **FF02::1** – link-local; replaces IPv4 broadcast address (all 1's host)
- All routers
  - **FF01::2** (interface-local), **FF02::2** (link-local)
- Solicited-Node multicast
  - Used in Neighbor Discovery Protocol (later)
  - **FF02::FF00:0/104** (FF02::FFXX:XXXX)
  - Construct by replacing 'XX:XXXX' above with low-order 24 bits of a nodes unicast or anycast address
  - Example
    - For unicast address 4037::01:800:200E:8C6C
    - Solicited-Node multicast is FF02::1:FF0E:8C6C

Types of IPv6 Addresses

- RFC 4291– “IPv6 Addressing Architecture”
- **Global Unicast**
  - Globally routable IPv6 addresses
- **Link Local Unicast**
  - Addresses for use on a given subnet
- **Unique Local Unicast**
  - Globally unique address for local communication
- **Multicast**
- **Anycast**
  - A unicast address assigned to interfaces belonging to different nodes

Anycast Addresses

- Allocated from unicast address space
  - Syntactically indistinguishable from unicast addresses
- An address assigned to more than one node
- Anycast traffic routed to the “nearest” host with the anycast address
- Typically used for a service (e.g. local DNS servers)
- Nodes must be configured to know an address is anycast
  - Don’t do Duplicate Address Detection
  - Advertise a route?

A Node's Required Addresses

Red = new for IPv6

- Link-local address for each interface
- Configured unicast or anycast addresses
- Loopback address
- All-Nodes multicast interface and link addresses
- Solicited-Node multicast for each configured unicast and anycast address
- Multicast addresses for all groups the node is a member of
- Routers must add
  - Subnet-Router anycast address for each interface
    - Subnet prefix with all 0's host part
  - All-Routers multicast address

Roundup: IPv6 Addresses

- “Interface ID” (host part) is 64 bits
- New addresses required by all nodes (host or router)
  - Link-local address
  - All-nodes interface-local and link-local multicast
  - Solicited-node multicast for each unicast/anycast address
- New addresses required by routers
  - All-routers interface-local, link-local and site-local multicast
  - Subnet-Router anycast for each interface?

Host Configuration

Assigning Address to Interfaces

- Static (manual) assignment
  - Needed for network equipment
- DHCPv6
  - Needed to track who uses an IP address
- **S**tate**L**ess **A**ddress **A**uto**C**onfiguration (**SLAAC**)
  - New to IPv6
- Describe SLAAC in the following…

SLAAC

- RFC 4862 – IPv6 Stateful Address Autoconfiguration
- Used to assign unicast addresses to interfaces
  - Link-Local Unicast
  - Global Unicast
  - Unique-Local Unicast?
- Goal is to minimize manual configuration
  - No manual configuration of hosts
  - Limited router configuration
  - No additional servers
- Use when “not particularly concerned with the exact addresses hosts use”
  - Otherwise use DHCPv6 (RFC 3315)

SLAAC Building Blocks

- Interface IDs
- Neighbor Discovery Protocol
- SLAAC Process

SLAAC Building Blocks

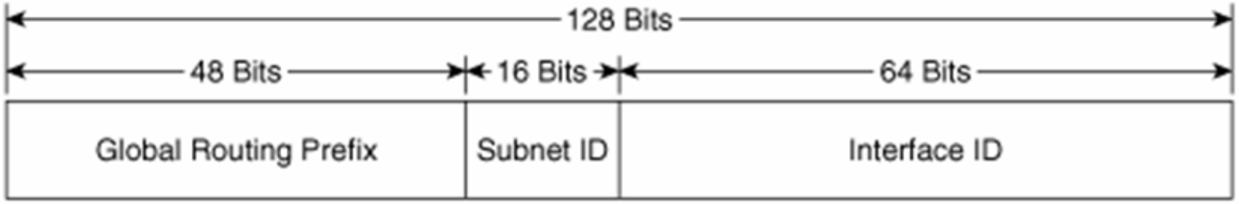
- Interface IDs

- Neighbor Discovery Protocol

- SLAAC Process

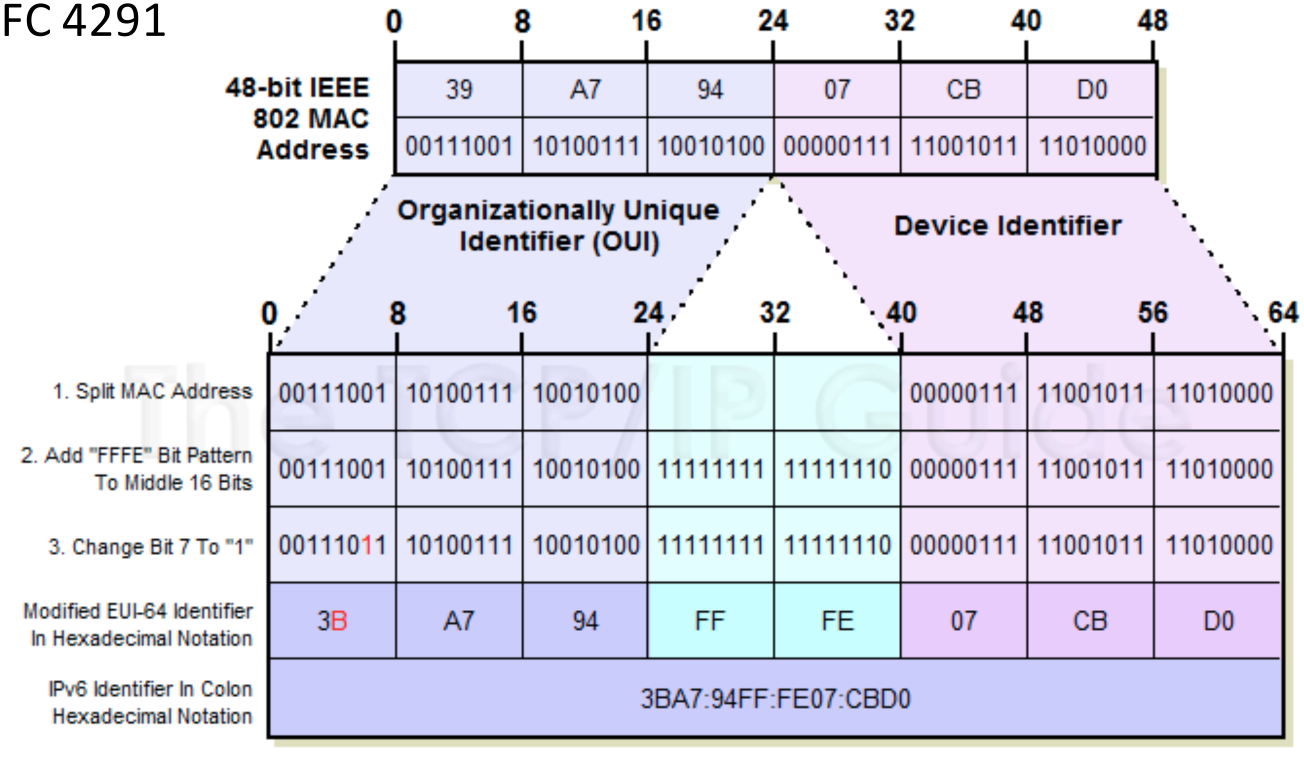
Interface IDs

- Used to identify a unique interface on a link
- Thought of as the “host portion” of an IPv6 address.
- 64 bits: To support both 48 bit and 64 bit IEEE MAC addresses
- Required to be unique on a link
- Subnets using auto addressing must be /64s.
- EUI-64 vs Privacy interface IDs



IEEE EUI-64 Option for Interface ID

- Use interface MAC address
- Insert FFFE to convert EUI-48 to EUI-64
- FlipUniversal/Local bit to “1”
    - Section 2.5.1 RFC 4291



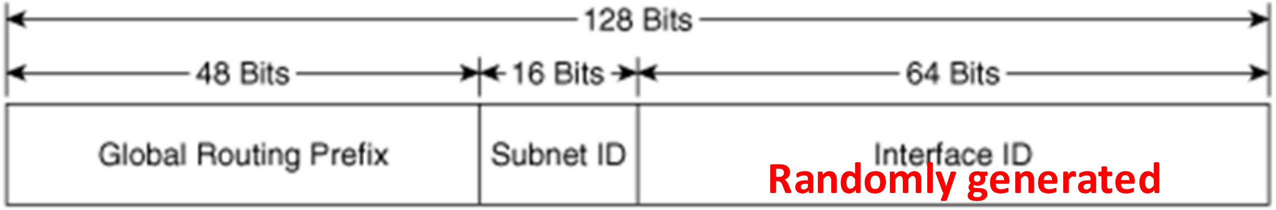
	0–8	8–16	16–24	24–32	32–40	40–48
48-bit IEEE 802 MAC Address	39	A7	94	07	CB	D0
	00111001	10100111	10010100	00000111	11001011	11010000

	0–8	8–16	16–24	24–32	32–40	40–48	48–56	56–64
1. Split MAC Address	00111001	10100111	10010100			00000111	11001011	11010000
2. Add "FFFE" Bit Pattern To Middle 16 Bits	00111001	10100111	10010100	11111111	11111110	00000111	11001011	11010000
3. Change Bit 7 To "1"	00111011	10100111	10010100	11111111	11111110	00000111	11001011	11010000
Modified EUI-64 Identifier In Hexadecimal Notation	3B	A7	94	FF	FE	07	CB	D0
IPv6 Identifier In Colon Hexadecimal Notation	3BA7:94FF:FE07:CBD0							

64-Bit IPv6 Modified EUI-64 Interface Identifier

Privacy Option for Interface ID

- Using MAC uniquely identifies a host… security/privacy concerns!
- Microsoft(!) defined an alternative solution for Interface IDs (RFC 4941)
- Hosts generates a random 64 bit Interface ID



SLAAC Building Blocks

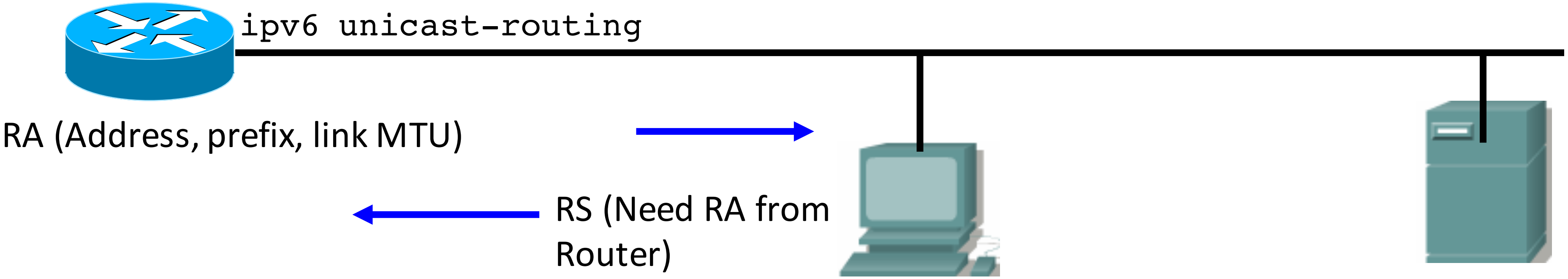
- Interface IDs
- Neighbor Discovery Protocol
- SLAAC Process

NDP

- RFC 4861 – Neighbor Discovery for IPv6
- Used to
  - Determine MAC address for nodes on same subnet (ARP)
  - Find routers on same subnet
  - Determine subnet prefix and MTU
  - Determine address of local DNS server (RFC 6106)
- Uses 5 ICMPv6 messages
  - **Router Solicitation** (**RS**) – request routers to send RA
  - **Router Advertisement** (**RA**) – router's address and subnet parameters
  - **Neighbor Solicitation** (**NS**) – request neighbor's MAC address (ARP Request)
  - **Neighbor Advertisement** (**NA**) – MAC address for an IPv6 address (ARP Reply)
  - **Redirect** – inform host of a better next hop for a destination

NDP RS & RA

- **Router Solicitation** (**RS**)
  - Originated by hosts to request that a router send an RA
  - Source = unspecified (::) or link-local address,
  - Destination = All-routers multicast (FF02::2)
- **Router Advertisement** (**RA**)
  - Originated by routers to advertise their address and link-specific parameters
  - Sent periodically and in response to Router Solicitation messages
  - Source = link-local address,
  - Destination = All-nodes multicast (FF02::1)



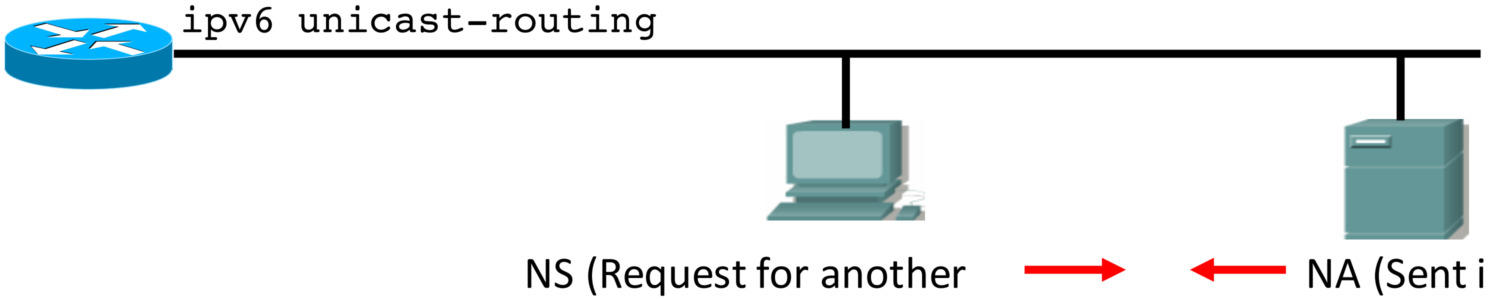
```
ipv6 unicast-routing
```

RA (Address, prefix, link MTU)

RS (Need RA from Router)

NDP NS & NA

- **Neighbor Solicitation** (**NS**)
  - Request ***target*** MAC address while providing target of source (IPv4 ARP Request)
  - Used to resolve address or verify reachability of neighbor
  - Source = unicast or “::” (Duplicate Address Detection… next slide)
  - Destination = solicited-node multicast
- **Neighbor Advertisement** (**NA**)
  - Advertise MAC address for given IPv6 address (IPv4 ARP Reply)
  - Respond to NS or communicate MAC address change
  - Source = unicast, destination = NS’s source or all-nodes multicast (if source “::”)



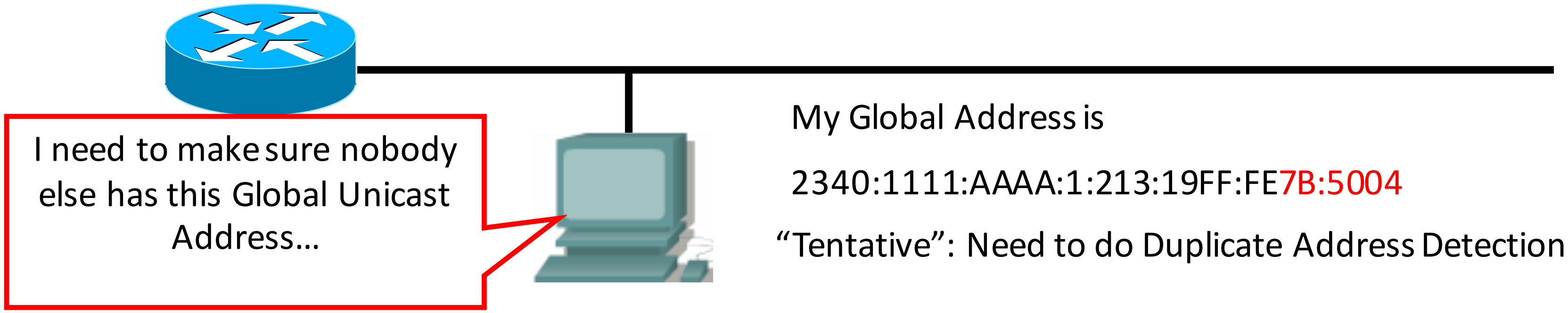
```
ipv6 unicast-routing
```

NS (Request for another node’s Link Layer Address) → ← NA (Sent in response to NS)

Duplicate Address Detection

- **Duplicate Address Detection** (**DAD**) used to verify address is unique in subnet prior to assigning it to an interface
- **MUST** take place on all unicast addresses, regardless of whether they are obtained through stateful, stateless or manual configuration
- **MUST NOT** be performed on anycast addresses
- Uses Neighbor Solicitation and Neighbor Advertisement messages
- NS sent to solicited-node multicast; if no NA received address is unique
- **Solicited-node multicast**: `FF02::1:FF:0/104` w/ last 24 bits of target

Duplicate Address Detection



I need to make sure nobody else has this Global Unicast Address…

My Global Address is

2340:1111:AAAA:1:213:19FF:FE7B:5004

“Tentative”: Need to do Duplicate Address Detection

NS (Neighbor Solicitation)

- **Target Address** = 2340:1111:AAAA:1:213:19FF:FE7B:5004

Destination: Solicited-Node Multicast Address = `FF02::1::FF7B:5004`

SLAAC Building Blocks

- Interface IDs
- Neighbor Discovery Protocol
- SLAAC Process

SLAAC Steps

- Select link-local address
- Verify “tentative” address not in use by another host with DAD
- Send RS to solicit RAs from routers
- Receive RA with
  - router address,
  - subnet MTU,
  - subnet prefix,
  - local DNS server (RFC 6106)
- Generate global unicast address
- Verify address is not in use by another host with DAD

Create Link-local address

Link-local Address =
Link-local Prefix + Interface Identifier (EUI-64 format)
FE80 [64 bits]   + [48 bit MAC u/l flipped + 16 bit FFFE]

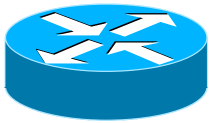
Make sure Link-local address is unique

NS (Neighbor Solicitation)

Make sure Link-local address is unique
DAD: Okay if no NA returned

Destination: Solicited-Node Multicast Address
Target address = Link-local address

Get Network Prefix to create Global unicast address



RS (Router Solicitation)

Get Prefix and other information

RA (Router Advertisement)

Source = Link-local address
Destin = FF02::1 All nodes multicast address
Query   = Prefix, Default Router, MTU, options

IPv6 Address =
Prefix + Interface ID (EUI-64 format)
[64 bits] + [48 bit MAC u/l flipped + 16 bit FFFE]

DAD

NS (Neighbor Solicitation)

Make sure IPv6 Address is unique
Target Address = IPv6 Address
DAD: Okay if no NA returned

Prefix Leases

- Prefix information contained in RA includes lifetime information
  - **Preferred lifetime**: when an address's preferred lifetime expires SHOULD only be used for existing communications
  - **Valid lifetime**: when an address's valid lifetime expires it MUST NOT be used as a source address or accepted as a destination address.

- Unsolicited RAs can reduce prefix lifetime values
  - Can be used to force re-addressing

Roundup: ICMPv6

- Implements router discovery and ARP functions
- ICMPv6 messages
  - Router Solicitation/Router Advertisement
  - Neighbor Solicitation/Neighbor Advertisement
  - (Next hop) Redirect
- Duplicate Address Detection (DAD)
  - verify unique link-local and global-unicast addresses
  - Uses:
    - NS/NA (i.e. gratuitous ARP)
    - Solicited node multicast address

Review - SLAAC

- Assigns link-local and global-unicast addresses
- Goals
  - Eliminate manual configuration
  - Require minimal router configuration
  - Require no additional servers
- Host part options
  - EUI-64
  - Random (“privacy” addresses)
- Steps
  - Generate link-local address and verify with DAD
  - Find router - RS/RA
  - Generate global unicast address and verify with DAD

Improving on IPv4 and IPv6?

- Why include unverifiable source address?
  - Would like accountability ***and*** anonymity (now neither)
  - Return address can be communicated at higher layer
- Why packet header used at edge same as core?
  - Edge: host tells network what service it wants
  - Core: packet tells switch how to handle it
    - One is local to host, one is global to network
- Some kind of payment/responsibility field?
  - Who is responsible for paying for packet delivery?
  - Source, destination, other?
- Other ideas?

Summary Network Layer

- understand principles behind network layer services:
  - network layer service models
  - forwarding versus routing (versus switching)
  - how a router works
  - routing (path selection)
  - IPv6
- Algorthims
  - Two routing approaches (LS vs DV)
  - One of these in detail (LS)
  - ARP

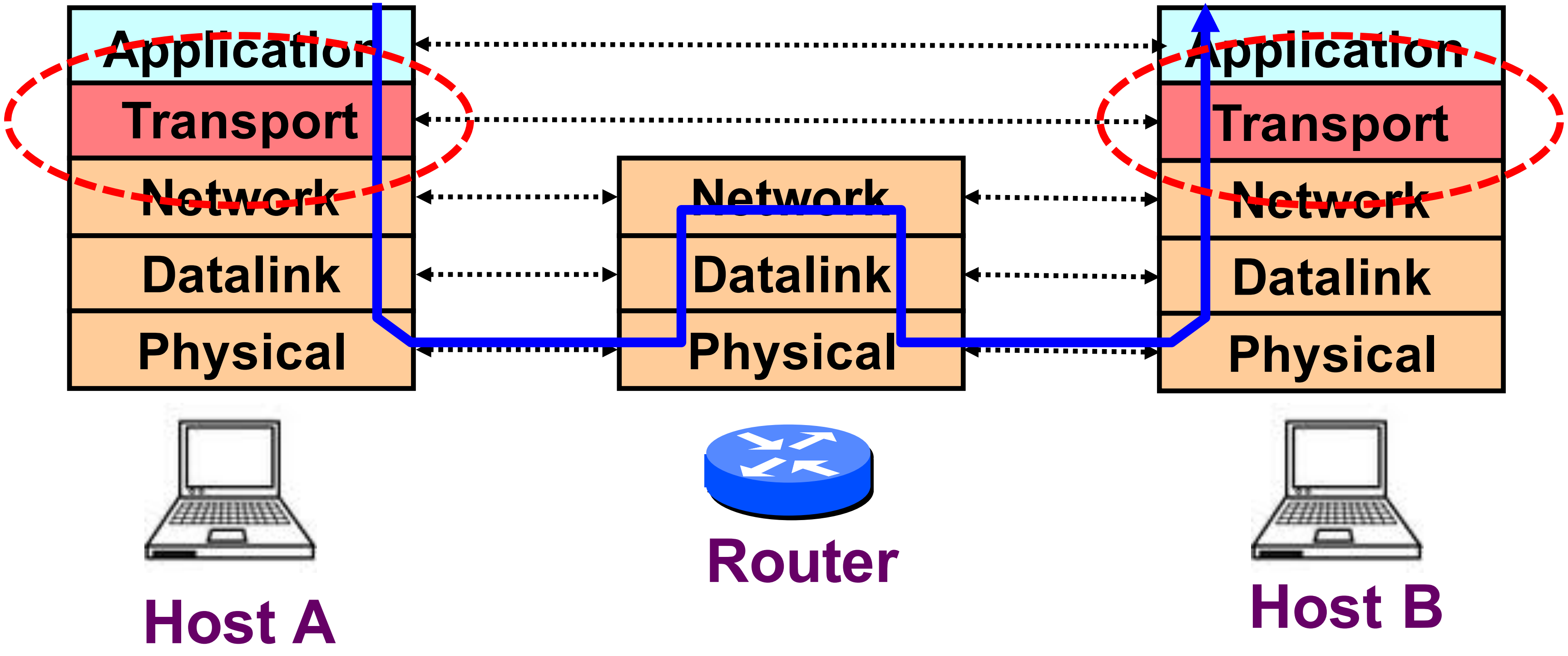
Topic 5 – Transport

Our goals:

- understand principles behind transport layer services:
  - multiplexing/demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about transport layer protocols in the Internet:
  - UDP: connectionless transport
  - TCP: connection-oriented transport
  - TCP congestion control

Transport Layer

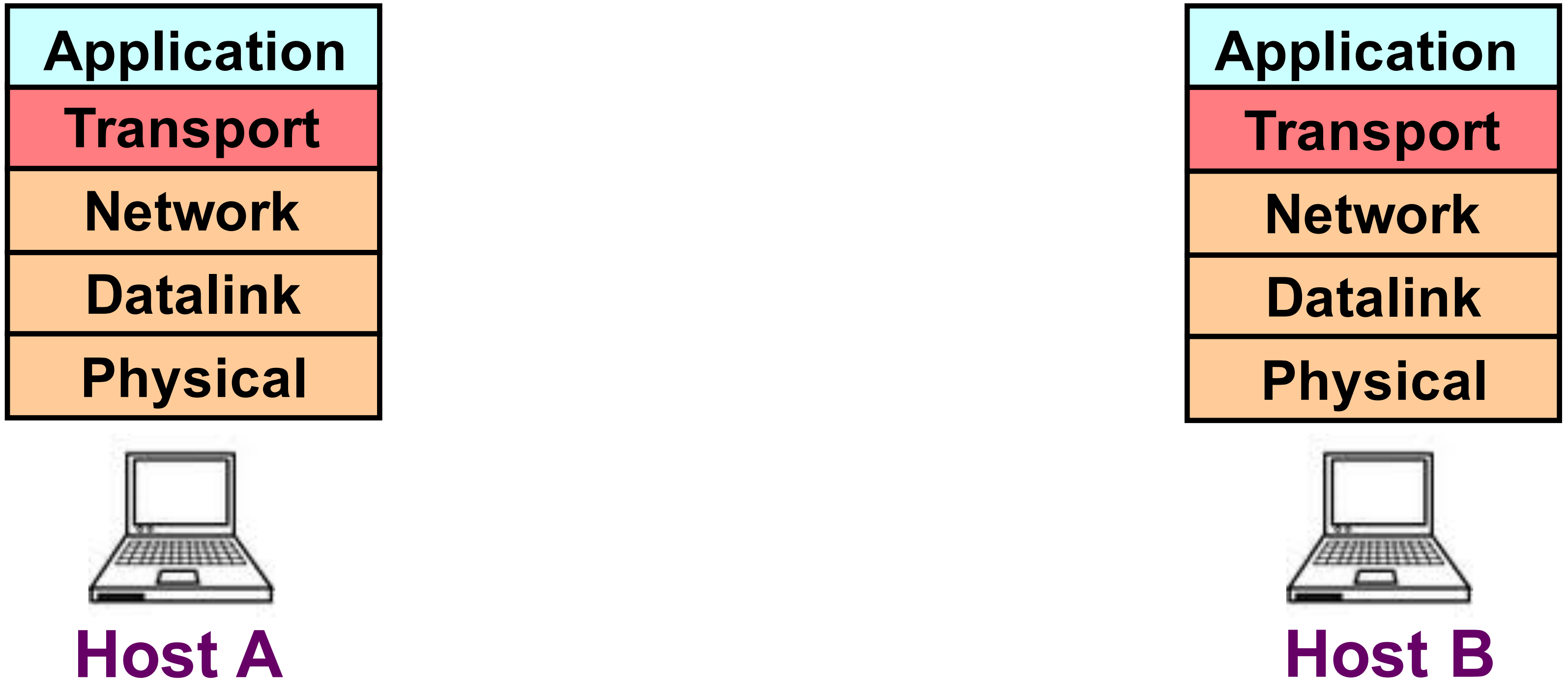
- Commonly a layer at end-hosts, between the application and network layer



Why a transport layer?

- IP packets are addressed to a host but end-to-end communication is between application processes at hosts
  - Need a way to decide which packets go to which applications (*more multiplexing*)

Why a transport layer?



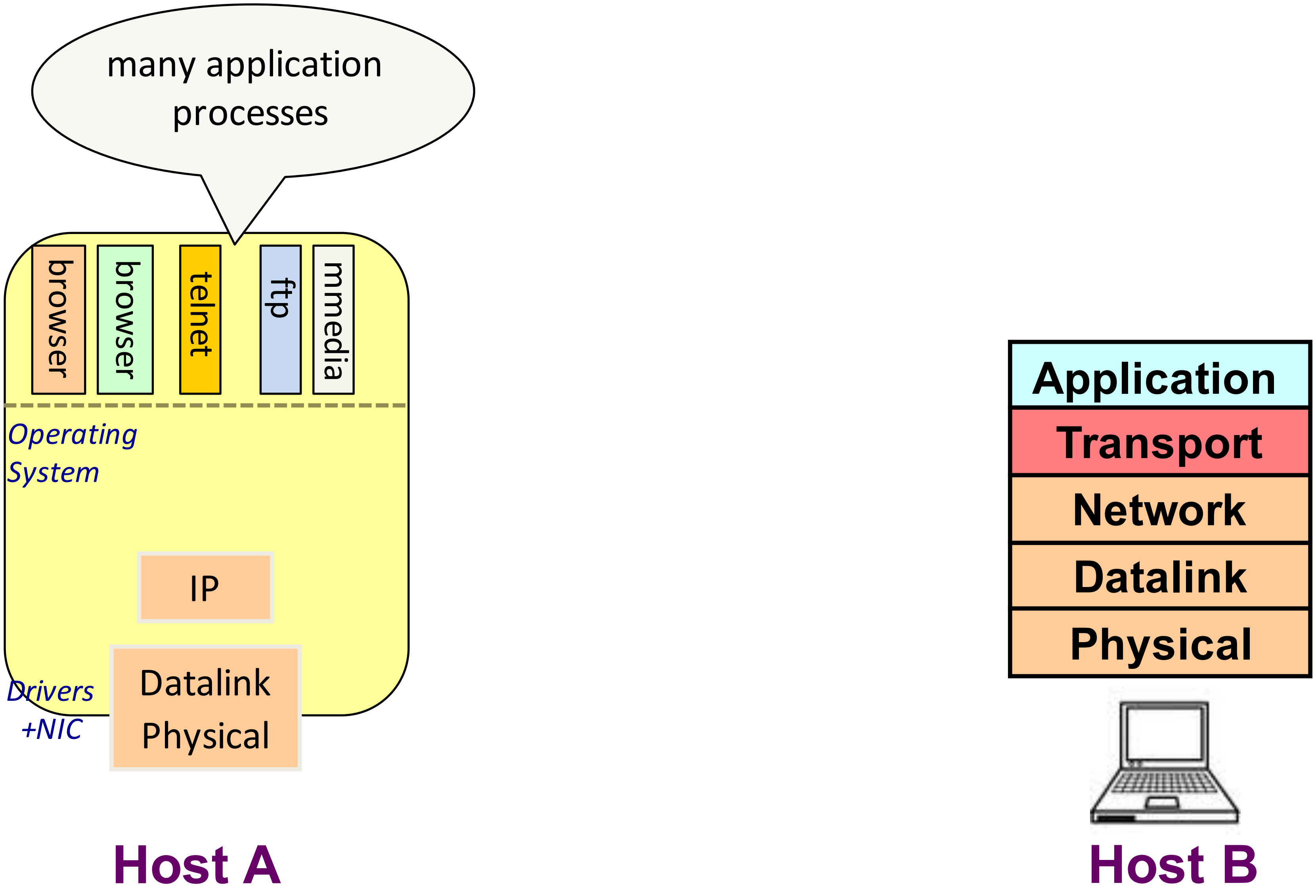
Application
Transport
Network
Datalink
Physical

Host A

Application
Transport
Network
Datalink
Physical

Host B

Why a transport layer?



many application processes

browser

browser

telnet

ftp

mmedia

Operating System

IP

Drivers +NIC

Datalink

Physical

Host A

Application

Transport

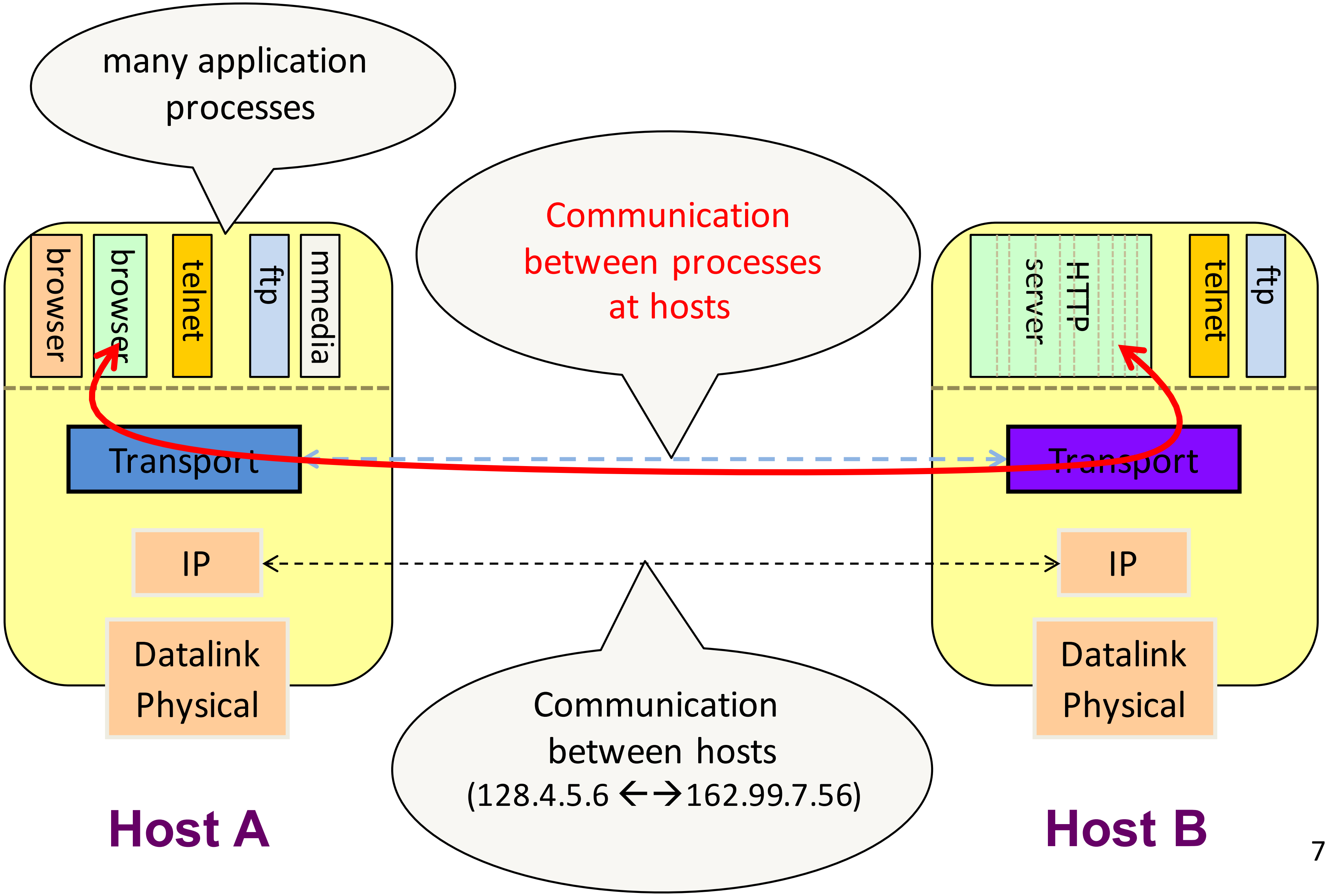
Network

Datalink

Physical

Host B

Why a transport layer?



many application processes

Communication between processes at hosts

Communication between hosts (128.4.5.6 ←→162.99.7.56)

Why a transport layer?

- IP packets are addressed to a host but end-to-end communication is between application processes at hosts
  - Need a way to decide which packets go to which applications (mux/demux)
- IP provides a weak service model (*best-effort*)
  - Packets can be corrupted, delayed, dropped, reordered, duplicated
  - No guidance on how much traffic to send and when
  - Dealing with this is tedious for application developers

Role of the Transport Layer

- Communication between application processes
  - Multiplexing between application processes
  - Implemented using *ports*

Role of the Transport Layer

- Communication between application processes
- Provide common end-to-end services for app layer [optional]
  - Reliable, in-order data delivery
  - Paced data delivery: flow and congestion-control
    - too fast may overwhelm the network
    - too slow is not efficient

Role of the Transport Layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
  - also SCTP, MTCP, SST, RDP, DCCP, …

Role of the Transport Layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
- UDP is a minimalist, no-frills transport protocol
  - only provides mux/demux capabilities

Role of the Transport Layer

- Communication between processes
- Provide common end-to-end services for app layer [optional]
- TCP and UDP are the common transport protocols
- UDP is a minimalist, no-frills transport protocol
- TCP is the *totus porcus* protocol
  - offers apps a reliable, in-order, byte-stream abstraction
  - with congestion control
  - but **no** performance (delay, bandwidth, ...) guarantees

Role of the Transport Layer

- Communication between processes
  - mux/demux from and to application processes
  - implemented using ports

Context: Applications and Sockets

- Socket: software abstraction by which an application process exchanges network messages with the (transport layer in the) operating system
  - socketID = socket(…, socket.TYPE)
  - socketID.sendto(message, …)
  - socketID.recvfrom(…)

- Two important types of sockets
  - UDP socket: TYPE is SOCK_DGRAM
  - TCP socket: TYPE is SOCK_STREAM

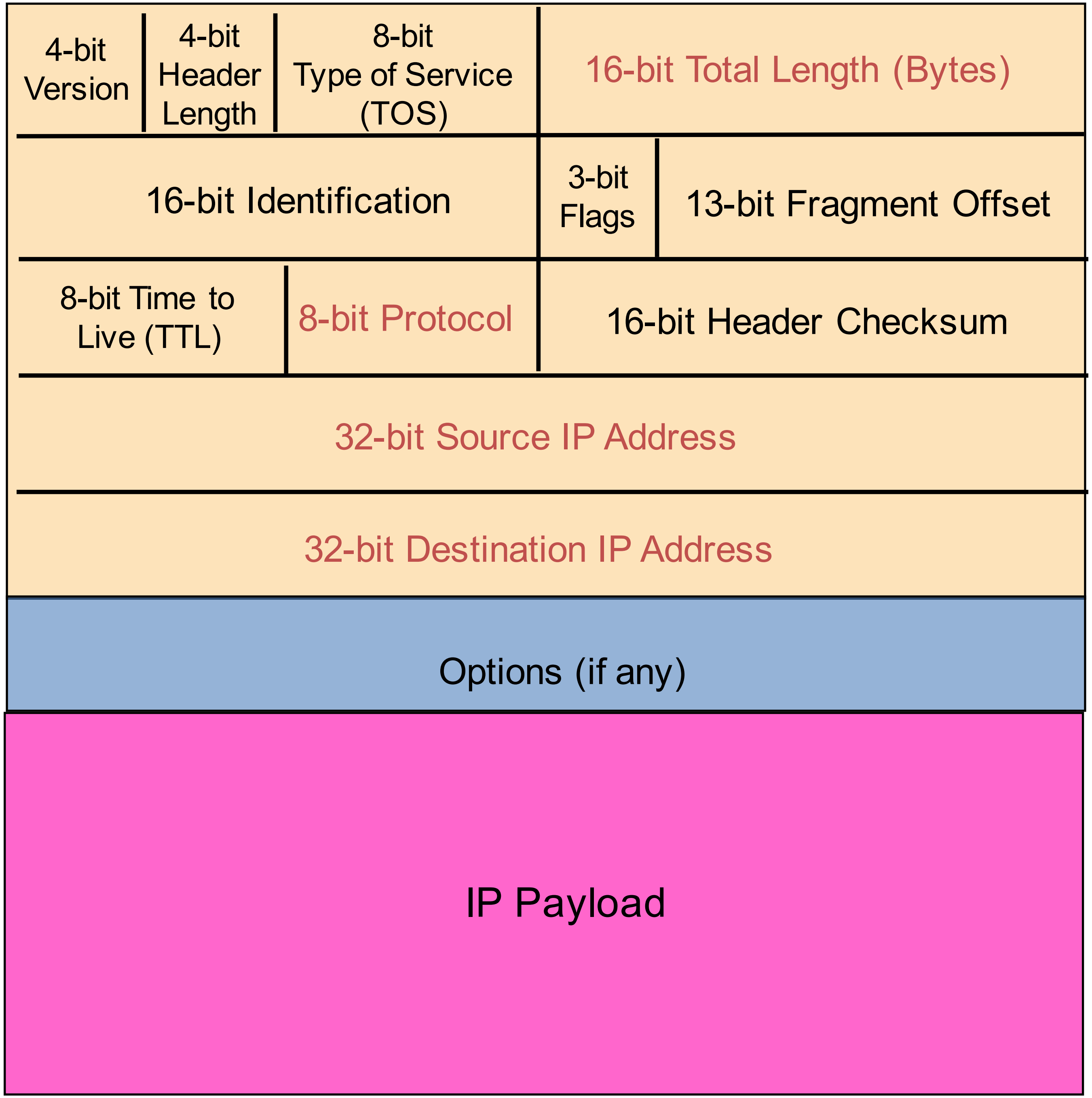
Ports

- Problem: deciding which app (socket) gets which packets

- Solution: ***port*** as a transport layer identifier
  - 16 bit identifier
  - OS stores mapping between sockets and *ports*
  - a packet carries a source and destination port number in its transport layer header

- For UDP ports (SOCK_DGRAM)
  - OS stores (local port, local IP address) ←→ socket

- For TCP ports (SOCK_STREAM)
  - OS stores (local port, local IP, remote port, remote IP) ←→ socket



4-bit Version

4-bit Header Length

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

8-bit Protocol

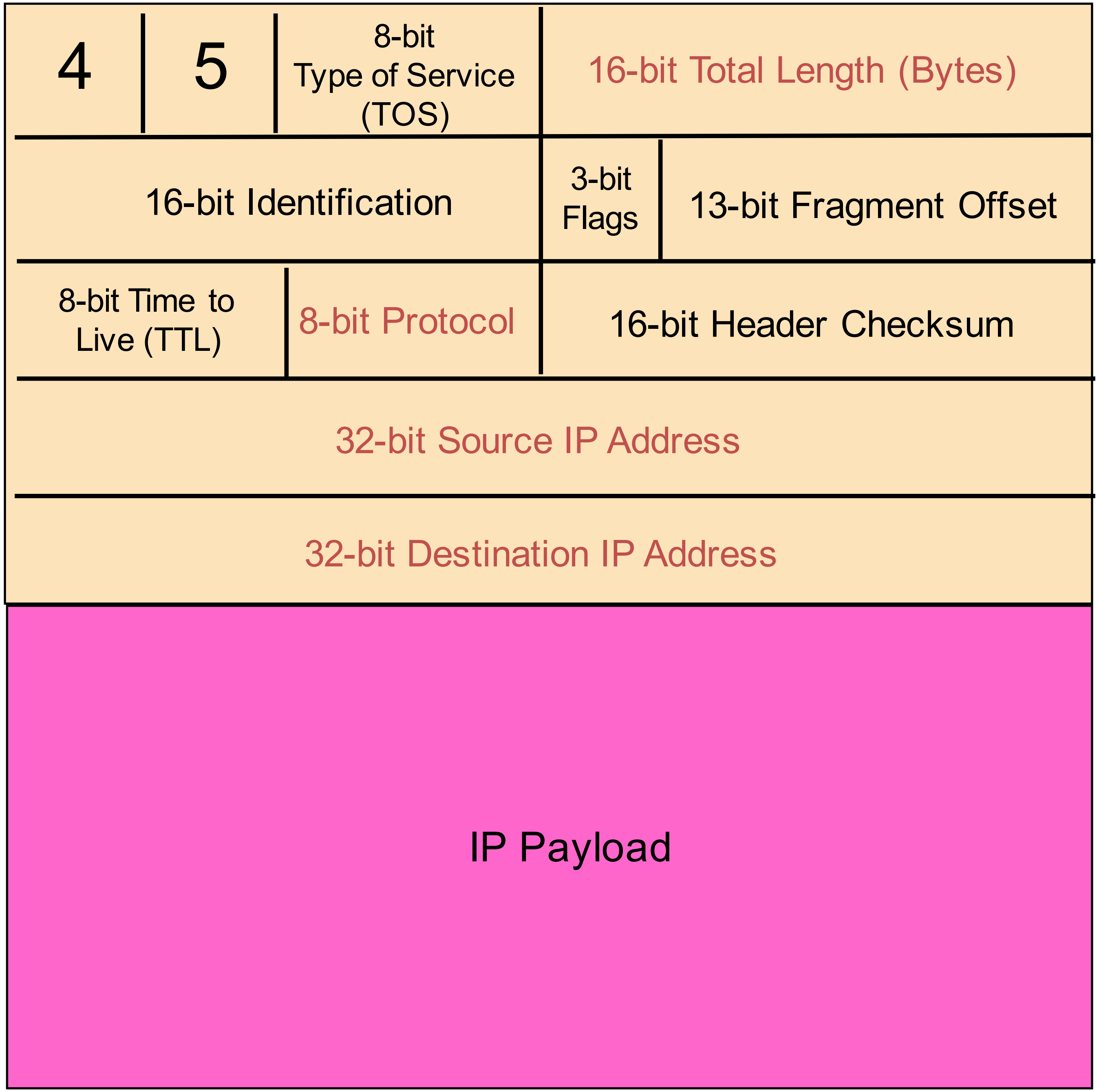
16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

Options (if any)

IP Payload



4

5

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

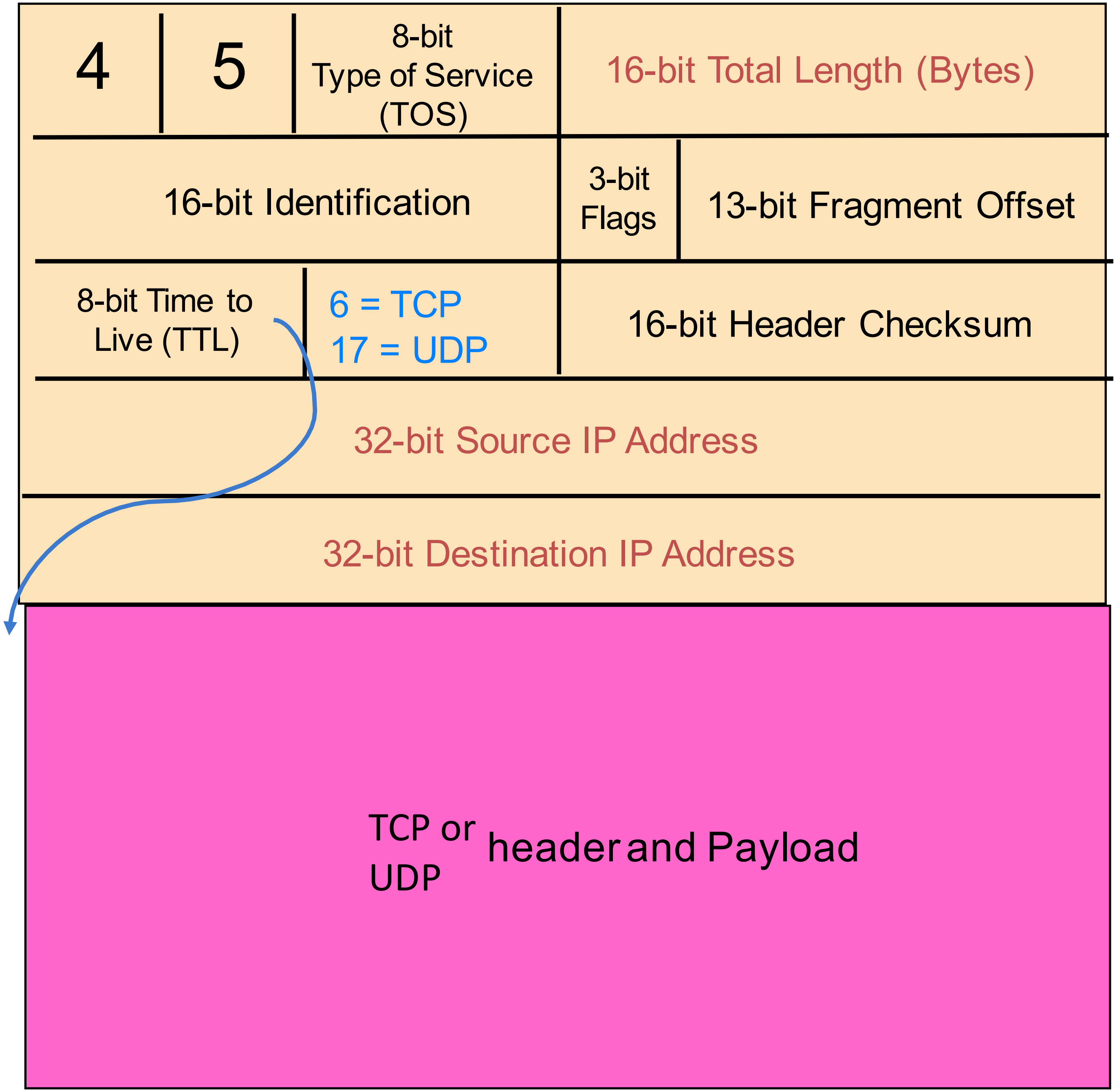
8-bit Protocol

16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

IP Payload



4

5

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

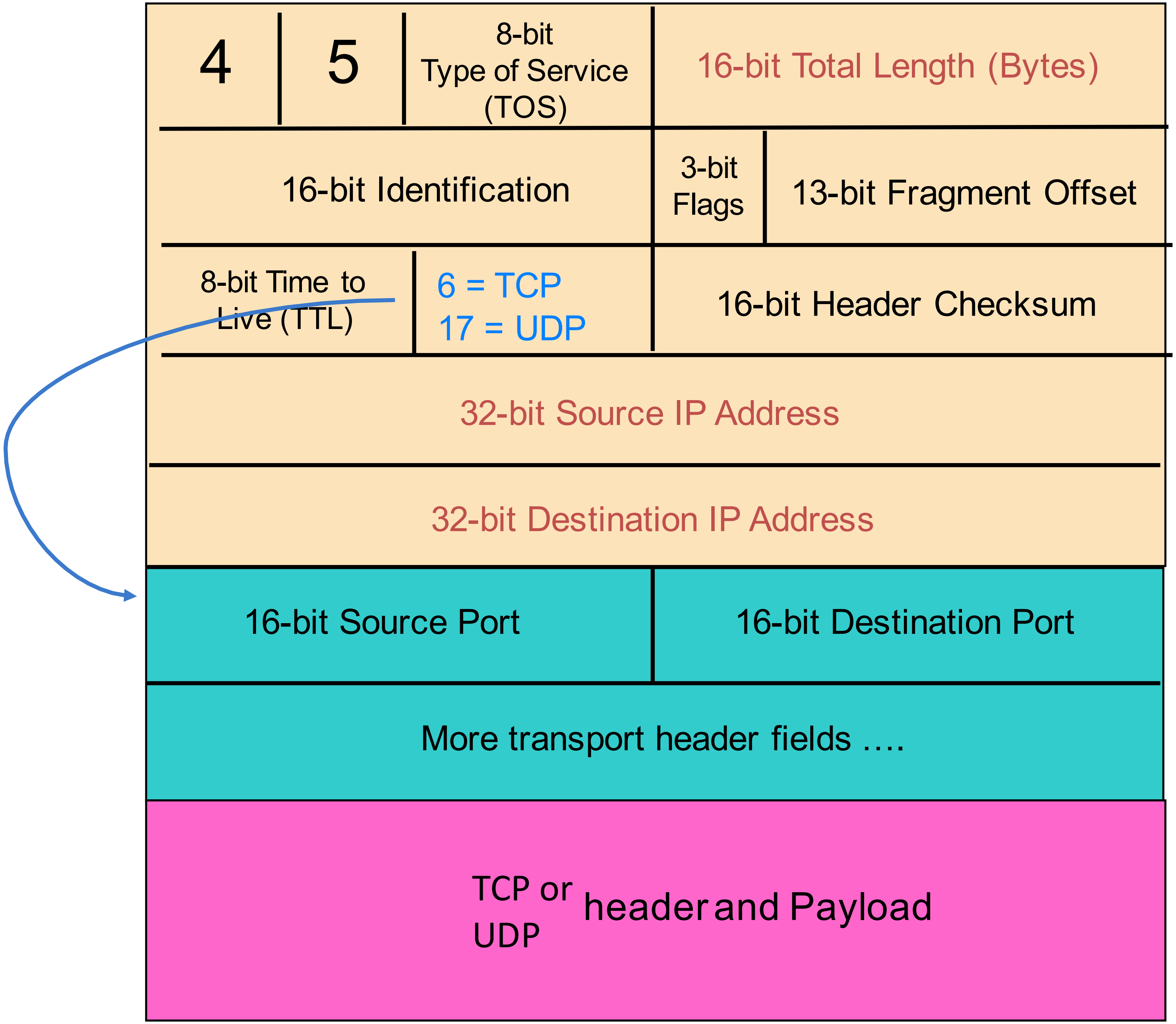
6 = TCP
17 = UDP

16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

TCP or UDP header and Payload



4

5

8-bit Type of Service (TOS)

16-bit Total Length (Bytes)

16-bit Identification

3-bit Flags

13-bit Fragment Offset

8-bit Time to Live (TTL)

6 = TCP
17 = UDP

16-bit Header Checksum

32-bit Source IP Address

32-bit Destination IP Address

16-bit Source Port

16-bit Destination Port

More transport header fields ….

TCP or UDP header and Payload

Recap: Multiplexing and Demultiplexing

- Host receives IP packets
  - Each IP header has source and destination IP address
  - Each Transport Layer header has source and destination port number
- Host uses IP addresses and port numbers to direct the message to appropriate socket

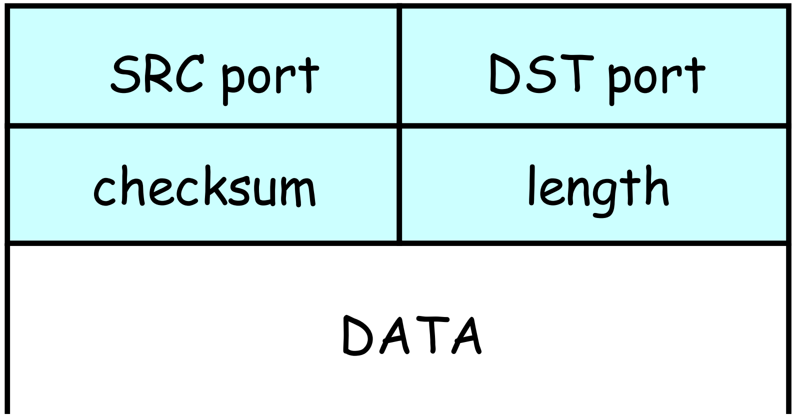
More on Ports

- Separate 16-bit port address space for UDP and TCP
- "Well known" ports (0-1023): everyone agrees which services run on these ports
  - e.g., ssh:22, http:80
  - helps client know server's port
- Ephemeral ports (most 1024-65535): dynamically selected: as the source port for a client process

UDP: User Datagram Protocol

- Lightweight communication between processes
  - Avoid overhead and delays of ordered, reliable delivery

- UDP described in RFC 768 – (1980!)
  - Destination IP address and port to support demultiplexing
  - Optional error checking on the packet contents
    - (checksum field of 0 means "don't verify checksum")



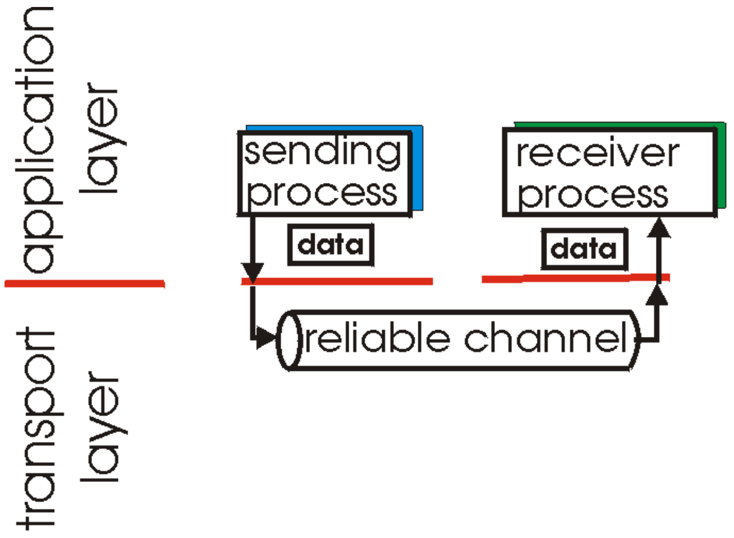
SRC port	DST port
checksum	length
DATA	

Why a transport layer?

- IP packets are addressed to a host but end-to-end communication is between application processes at hosts
  - Need a way to decide which packets go to which applications (mux/demux)
- IP provides a weak service model (*best-effort*)
  - Packets can be corrupted, delayed, dropped, reordered, duplicated

Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!



(a) provided service

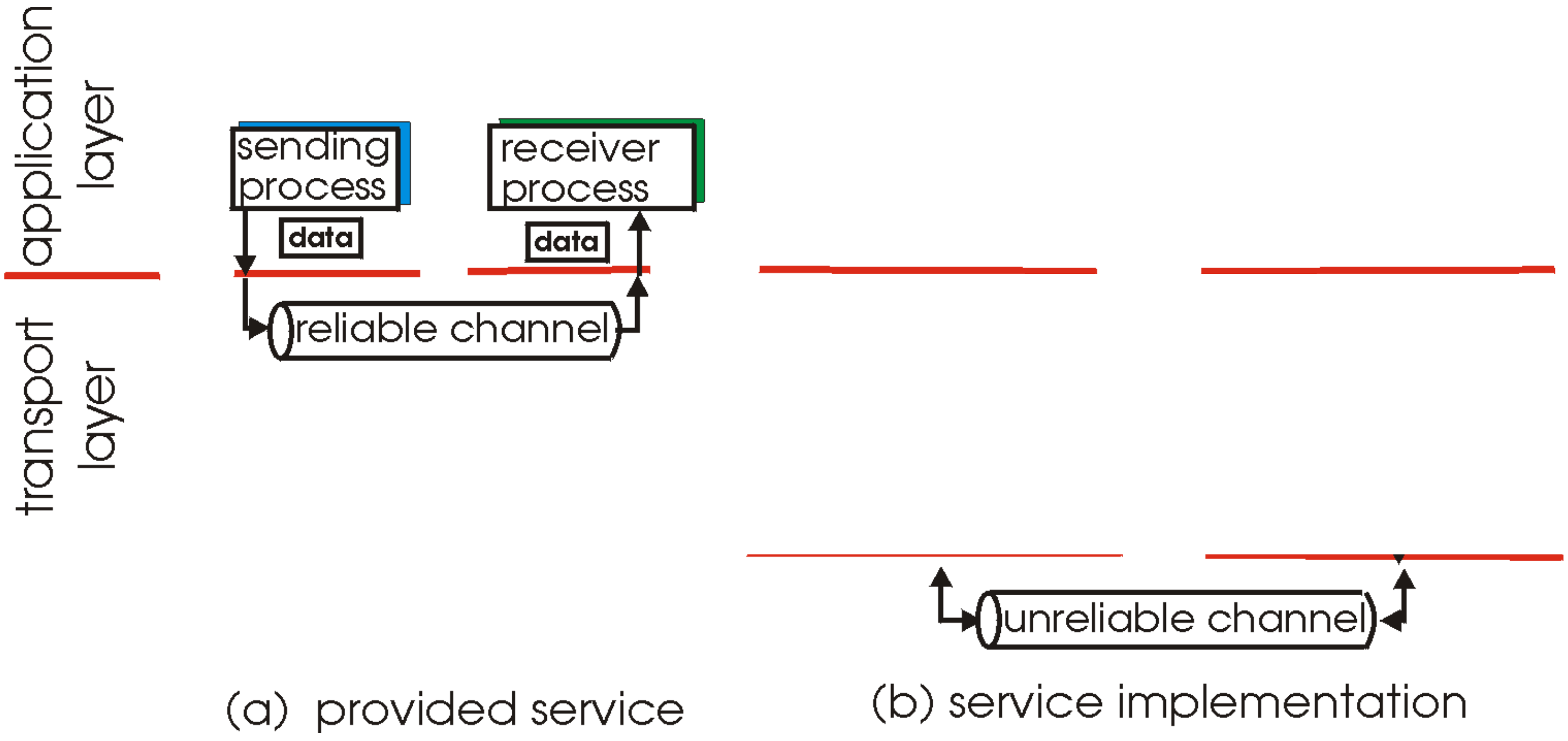
- In a perfect world, reliable transport is easy

But the Internet default is *best-effort*

- All the bad things best-effort can do
  - a packet is corrupted (bit errors)
  - a packet is lost
  - a packet is delayed (*why?*)
  - packets are reordered (*why?*)
  - a packet is duplicated (*why?*)

Principles of Reliable data transfer

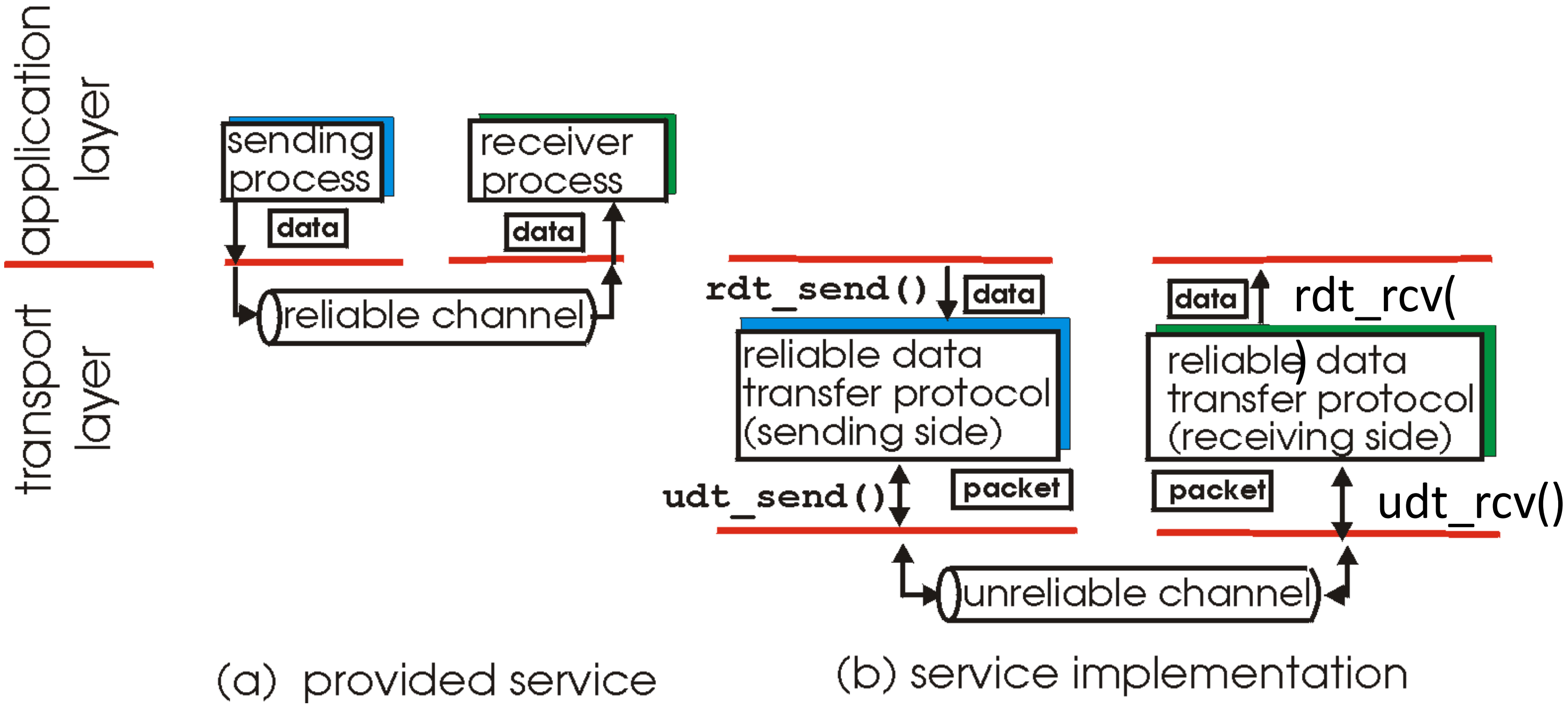
- important in app., transport, link layers
- top-10 list of important networking topics!



- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!

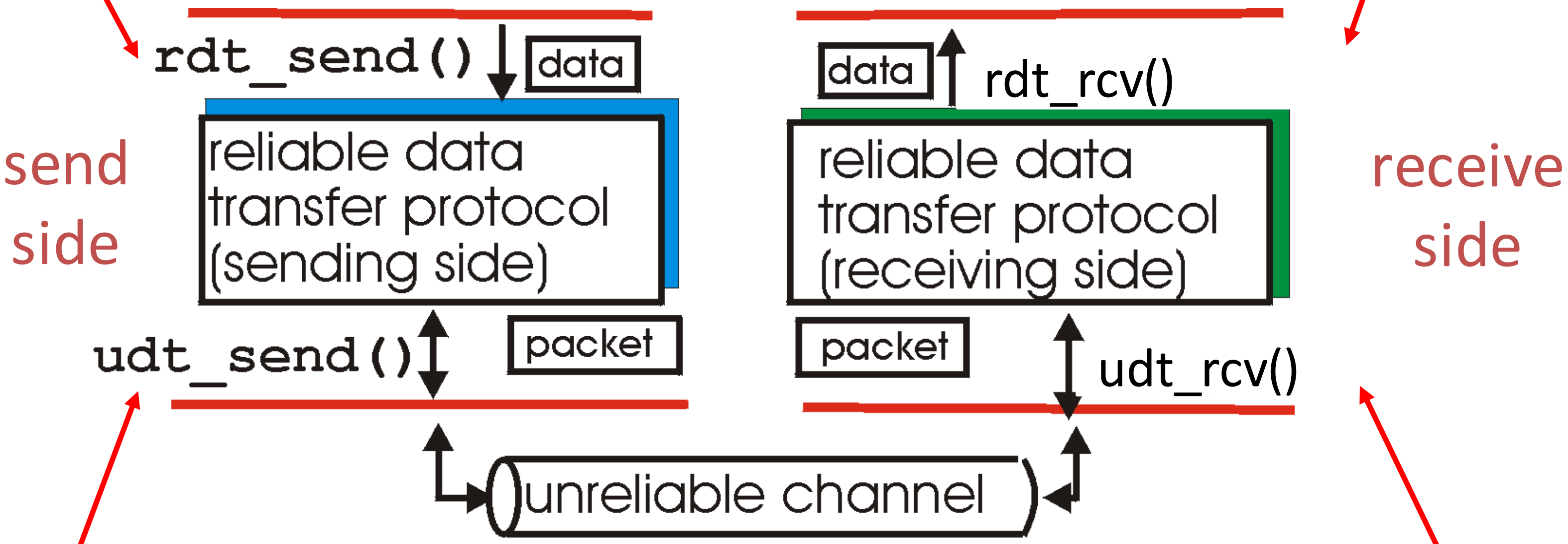


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Reliable data transfer: getting started

rdt_send(): called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

rdt_rcv(): called by **rdt** to deliver data to upper



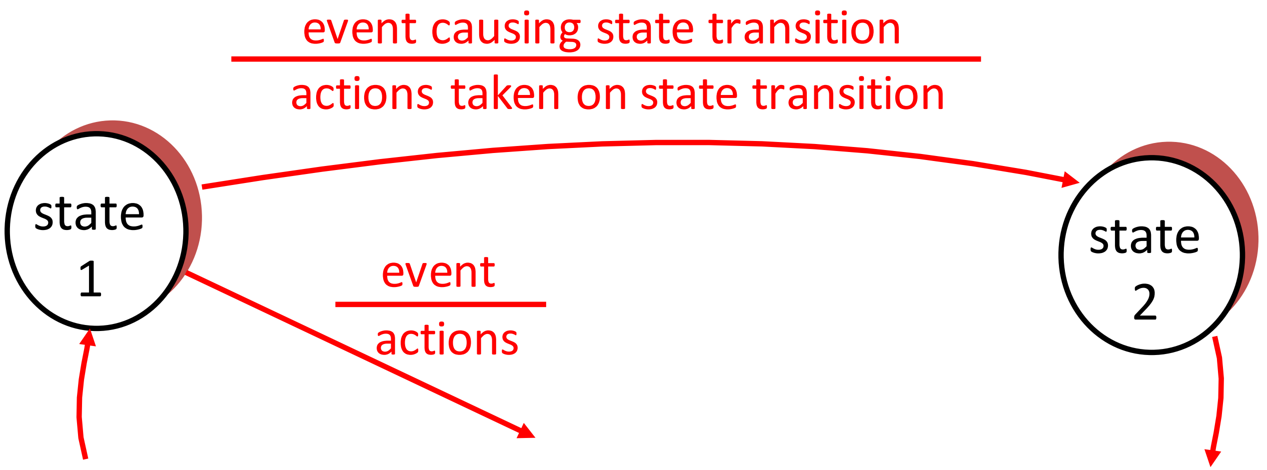
udt_send(): called by rdt, to transfer packet over unreliable channel to receiver

udt_rcv(): called when packet arrives on rcv-side of channel

Reliable data transfer: getting started

We' ll:

- incrementally develop sender, receiver sides of reliable data transfer protocol (rdt)
- consider only unidirectional data transfer
  - but control info will flow on both directions!
- use finite state machines (FSM) to specify sender, receiver



event causing state transition

actions taken on state transition

state: when in this "state" next state uniquely determined by next event

KR state machines – a note.

Beware

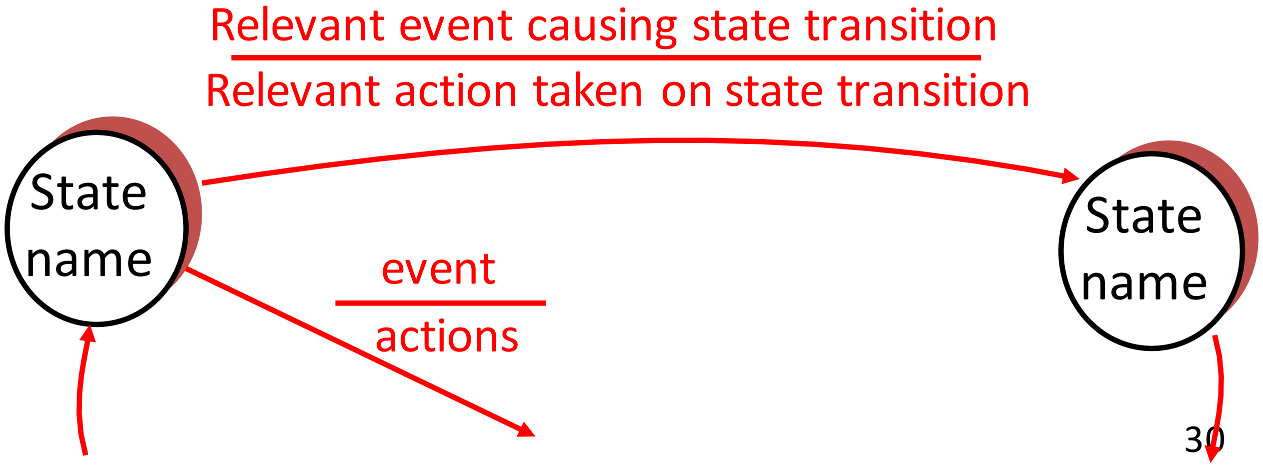
Kurose and Ross has a confusing/confused attitude to state-machines.

I’ve attempted to normalise the representation.

UPSHOT: these slides have differing information to the KR book (from which the RDT example is taken.)

in KR “actions taken” appear wide-ranging, my interpretation is more specific/relevant.

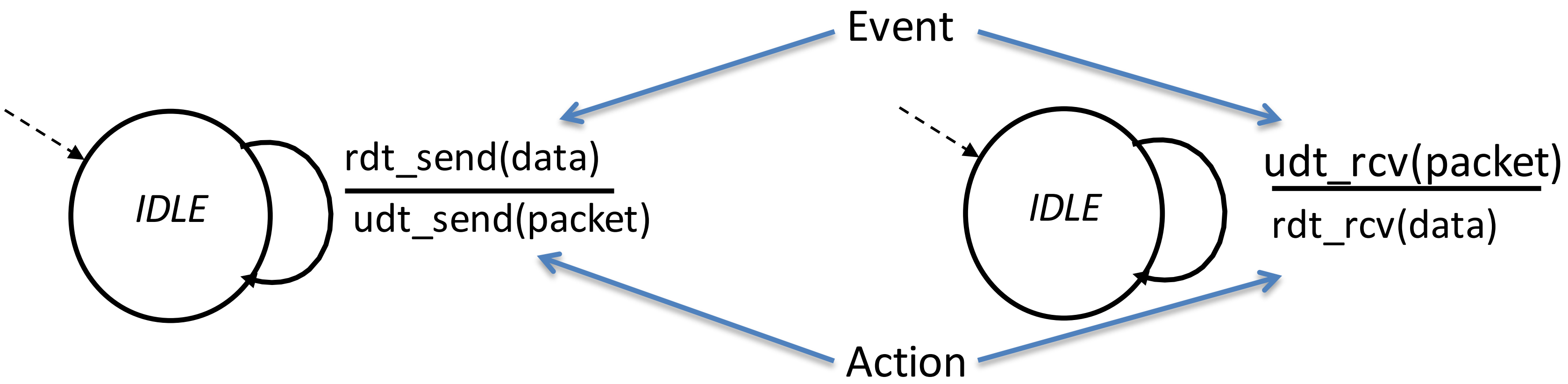
state: when in this “state” next state uniquely determined by next event



$$\frac{\text{Relevant event causing state transition}}{\text{Relevant action taken on state transition}}$$

Rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- separate FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver read data from underlying channel



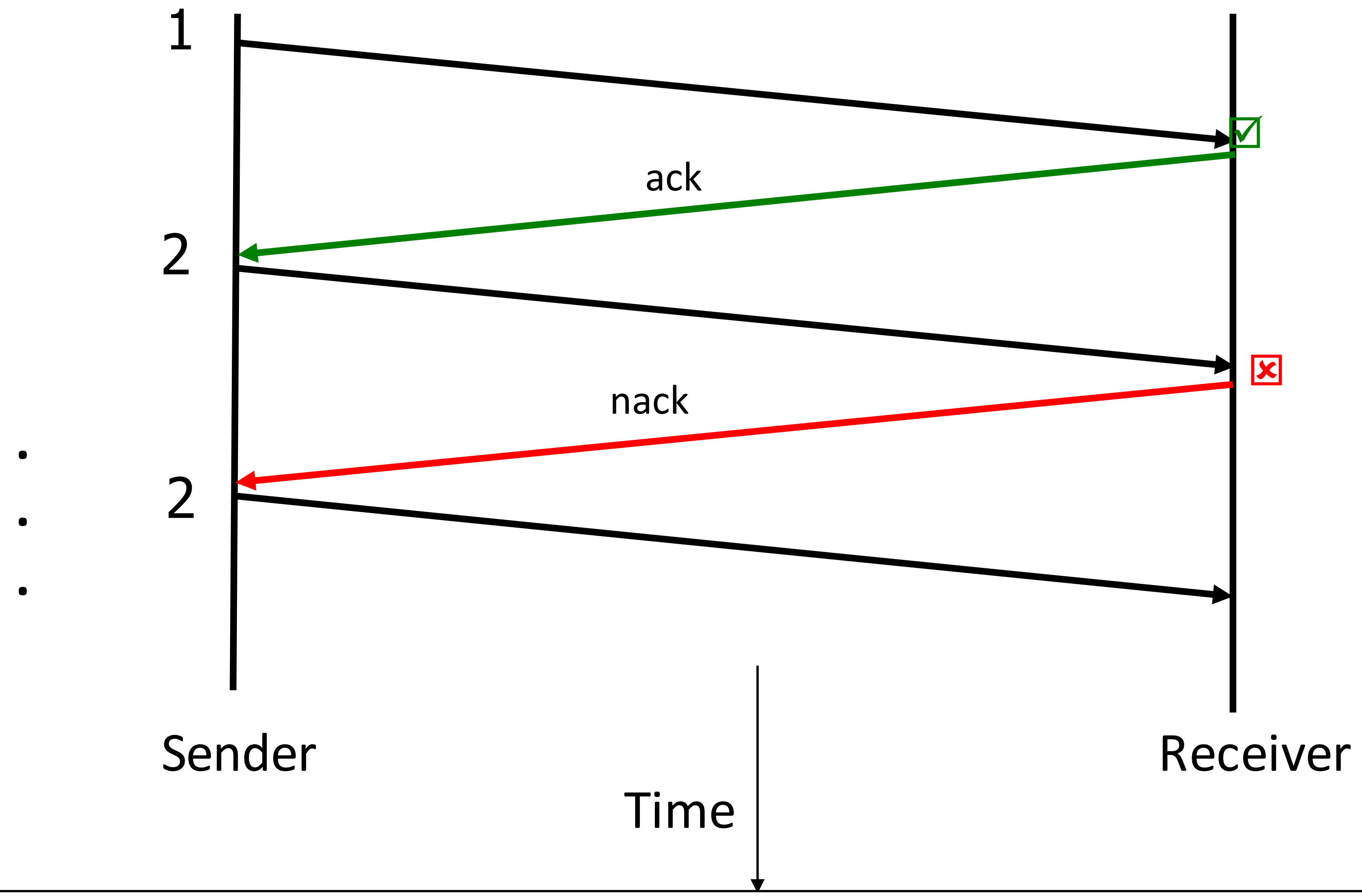
sender

receiver

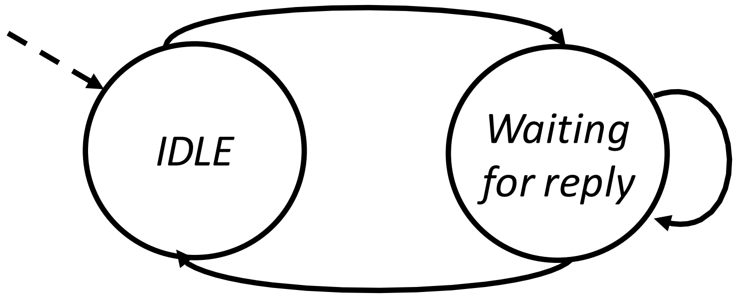
Rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors:
  - *acknowledgements (ACKs):* receiver explicitly tells sender that packet received is OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that packet had errors
  - sender retransmits packet on receipt of NAK
- new mechanisms in **`rdt2.0`** (beyond **`rdt1.0`**):
  - error detection
  - receiver feedback: control msgs (ACK,NAK) receiver->sender

Dealing with Packet Corruption



rdt2.0: FSM specification



rdt_send(data)

udt_send(packet)

IDLE

Waiting for reply

udt_rcv(reply) && isNAK(reply)

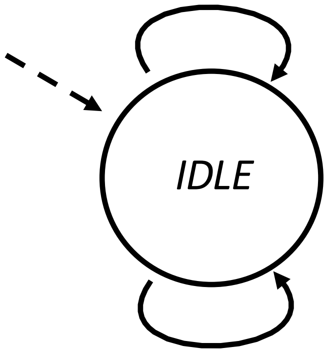
udt_send(packet)

udt_rcv(reply) && isACK(reply)

Λ

sender

receiver



udt_rcv(packet) && corrupt(packet)

udt_send(NAK)

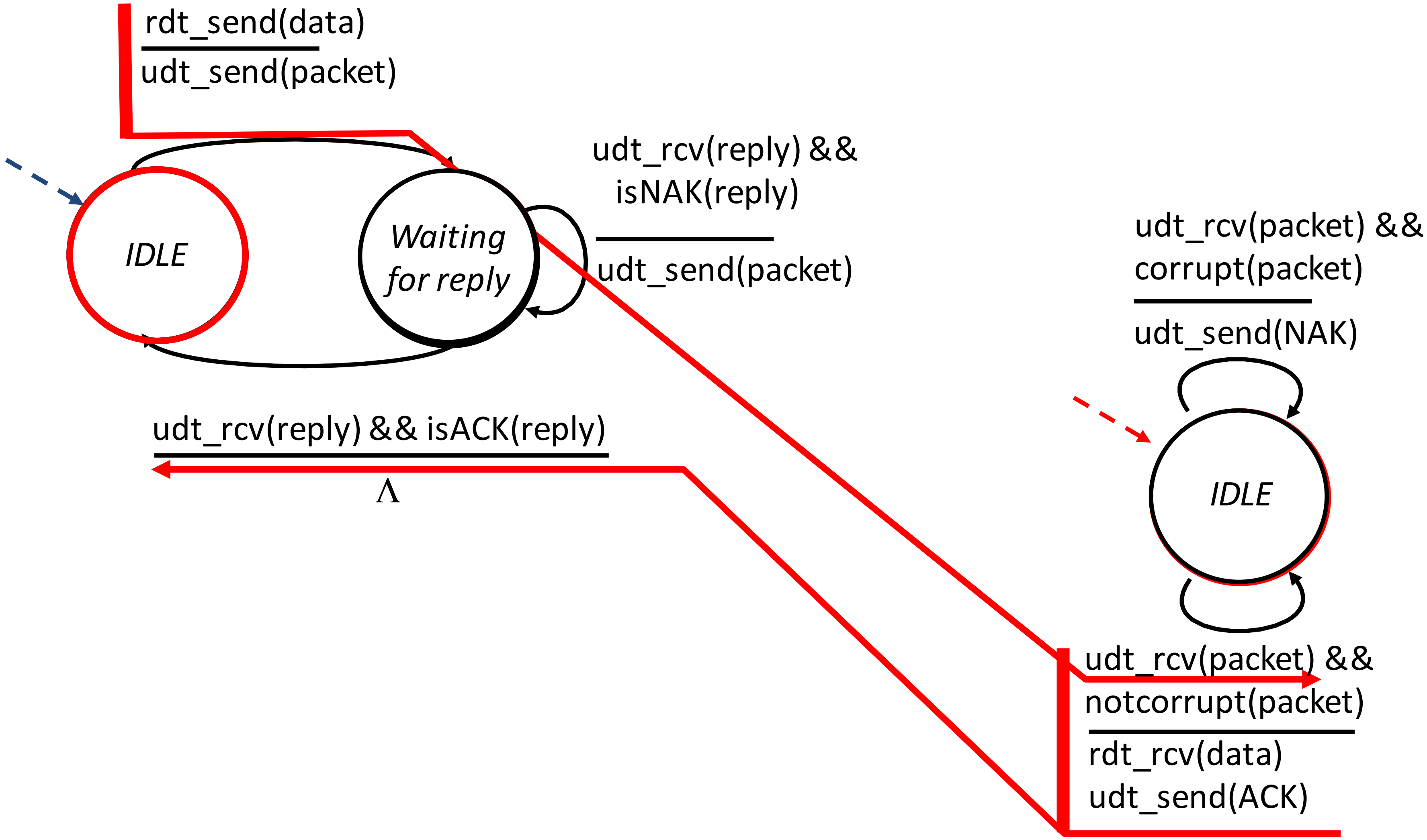
IDLE

udt_rcv(packet) && notcorrupt(packet)

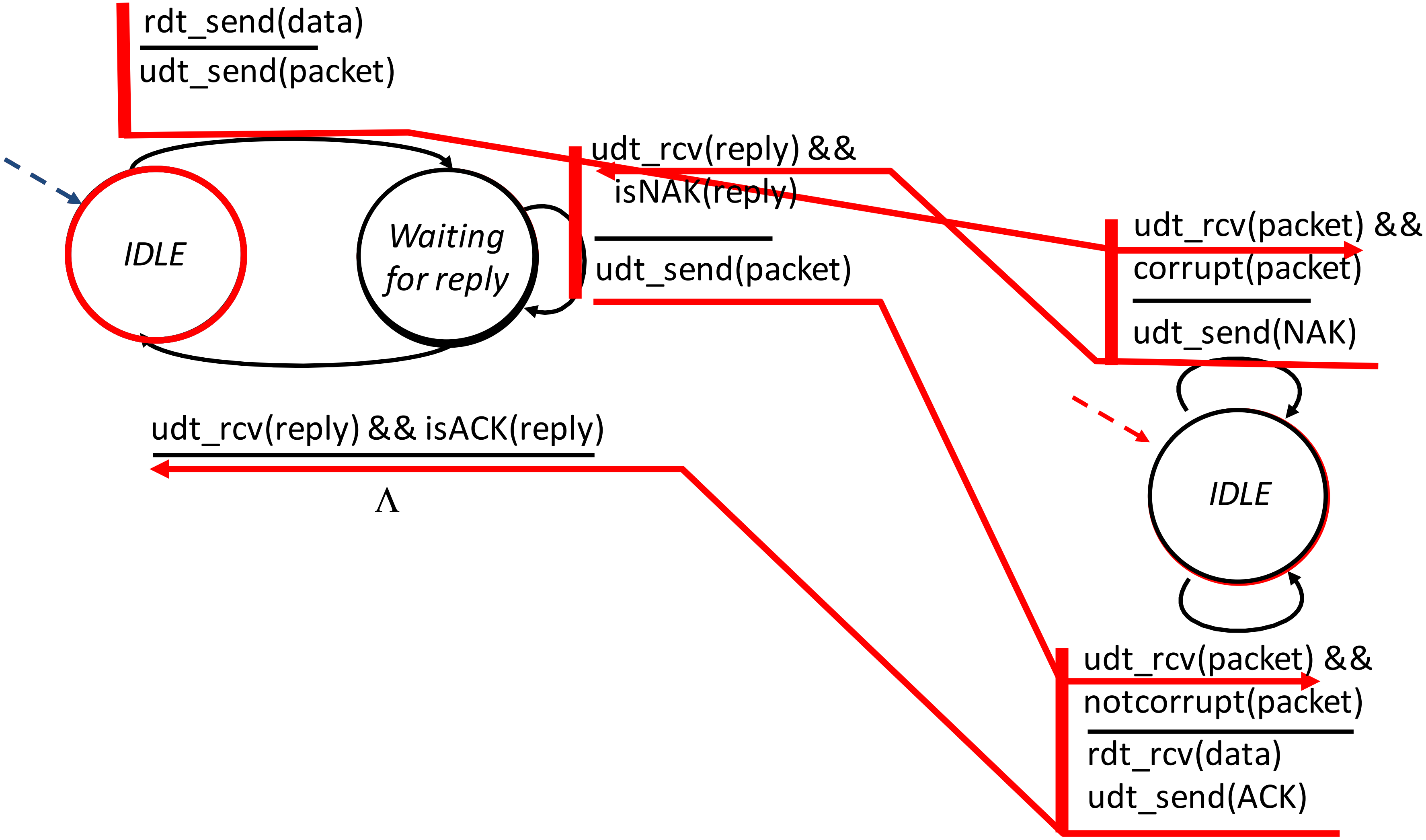
rdt_rcv(data)
udt_send(ACK)

Note: the sender holds a copy of the packet being sent until the delivery is acknowledged.

rdt2.0: operation with no errors


rdt_send(data)
udt_send(packet)
IDLE
Waiting for reply
udt_rcv(reply) && isNAK(reply)
udt_send(packet)
udt_rcv(reply) && isACK(reply)
Λ
udt_rcv(packet) && corrupt(packet)
udt_send(NAK)
IDLE
udt_rcv(packet) && notcorrupt(packet)
rdt_rcv(data)
udt_send(ACK)

rdt2.0: error scenario



rdt2.0 has a fatal flaw!

What happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

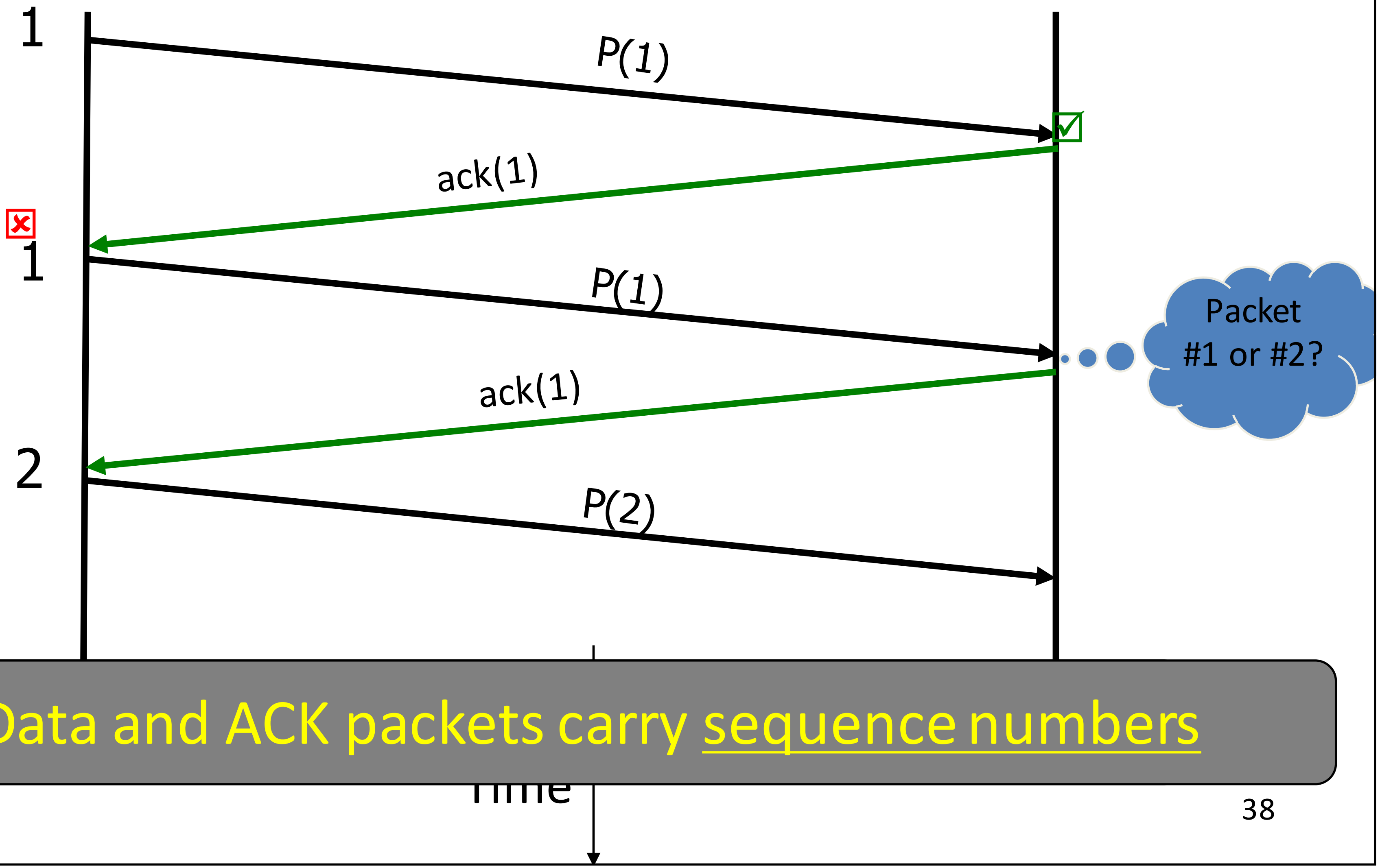
Handling duplicates:

- sender retransmits current packet if ACK/NAK garbled
- sender adds *sequence number* to each packet
- receiver discards (doesn't deliver) duplicate packet

stop and wait

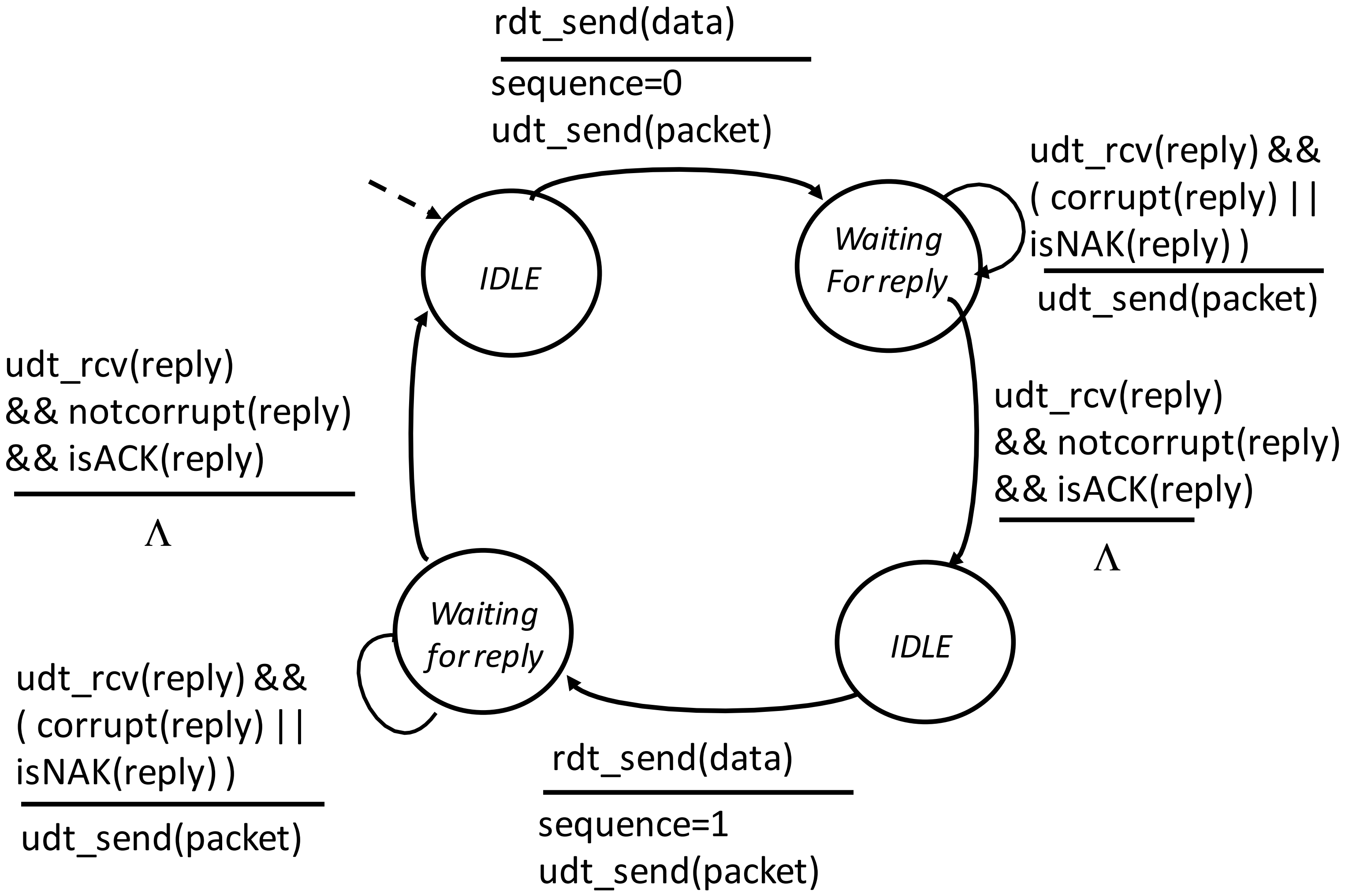
Sender sends one packet, then waits for receiver response

Dealing with Packet Corruption



Data and ACK packets carry sequence numbers

rdt2.1: sender, handles garbled ACK/NAKs



rdt_send(data)

sequence=0
udt_send(packet)

udt_rcv(reply) &&
(corrupt(reply) | |
isNAK(reply))

udt_send(packet)

IDLE

Waiting For reply

udt_rcv(reply)
&& notcorrupt(reply)
&& isACK(reply)

Λ

udt_rcv(reply)
&& notcorrupt(reply)
&& isACK(reply)

Λ

Waiting for reply

IDLE

udt_rcv(reply) &&
(corrupt(reply) | |
isNAK(reply))

udt_send(packet)

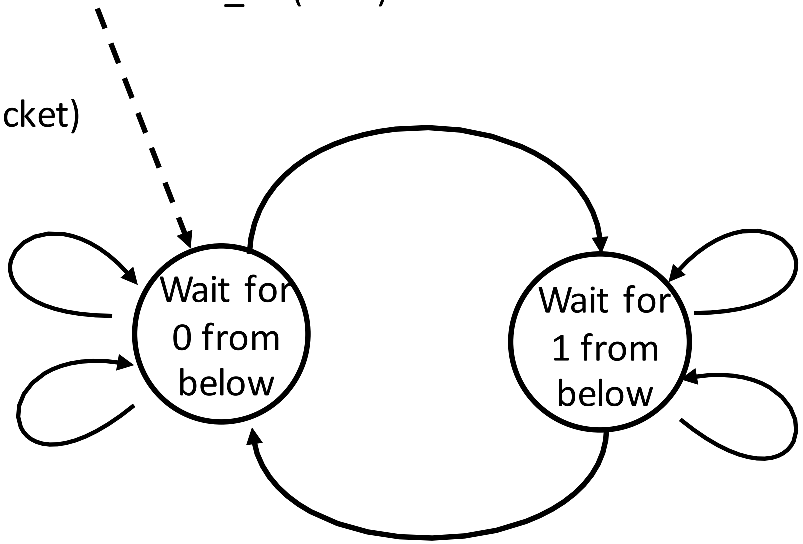
rdt_send(data)

sequence=1
udt_send(packet)

rdt2.1: receiver, handles garbled ACK/NAKs

udt_rcv(packet) && not corrupt(packet)
&& has_seq0(packet)

udt_send(ACK)
rdt_rcv(data)



receive(packet) && corrupt(packet)

udt_send(NAK)

receive(packet) &&
not corrupt(packet) &&
has_seq1(packet)

udt_send(ACK)

udt_rcv(packet) && corrupt(packet)

udt_send(NAK)

receive(packet) &&
not corrupt(packet) &&
has_seq0(packet)

udt_send(ACK)

udt_rcv(packet) && not corrupt(packet)
&& has_seq1(packet)

udt_send(ACK)
rdt_rcv(data)

rdt2.1: discussion

Sender:

- seq # added to pkt
- two seq. #’ s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must “remember” whether “current” pkt has a 0 or 1 sequence number

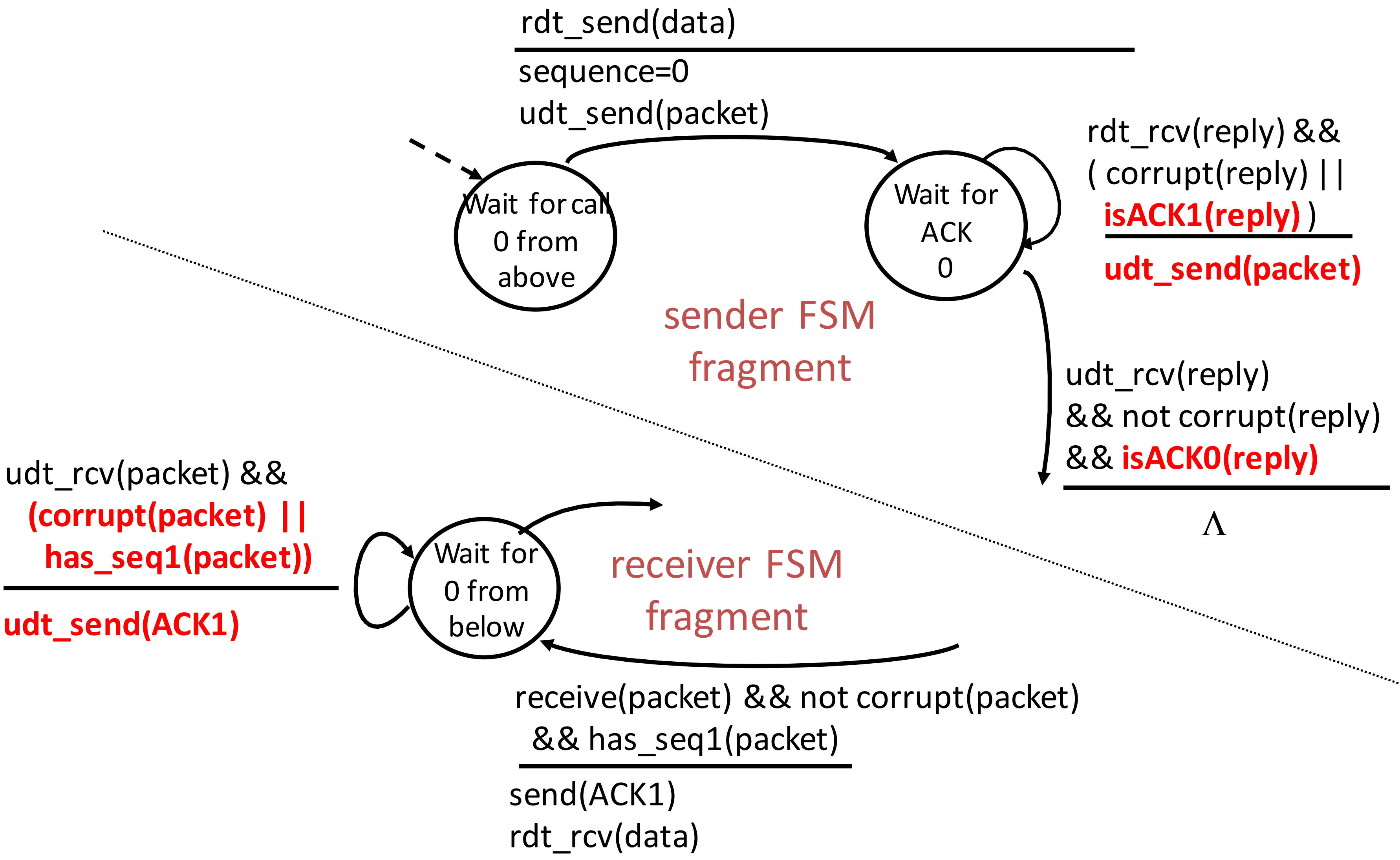
Receiver:

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

rdt2.2: a NAK-free protocol

- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

rdt2.2: sender, receiver fragments



rdt_send(data)

sequence=0
udt_send(packet)

Wait for call 0 from above

Wait for ACK 0

rdt_rcv(reply) &&
(corrupt(reply) ||
isACK1(reply))

udt_send(packet)

sender FSM fragment

udt_rcv(reply)
&& not corrupt(reply)
&& **isACK0(reply)**

Λ

udt_rcv(packet) &&
(corrupt(packet) ||
has_seq1(packet))

udt_send(ACK1)

Wait for 0 from below

receiver FSM fragment

receive(packet) && not corrupt(packet)
&& has_seq1(packet)

send(ACK1)
rdt_rcv(data)

rdt3.0: channels with errors *and* loss

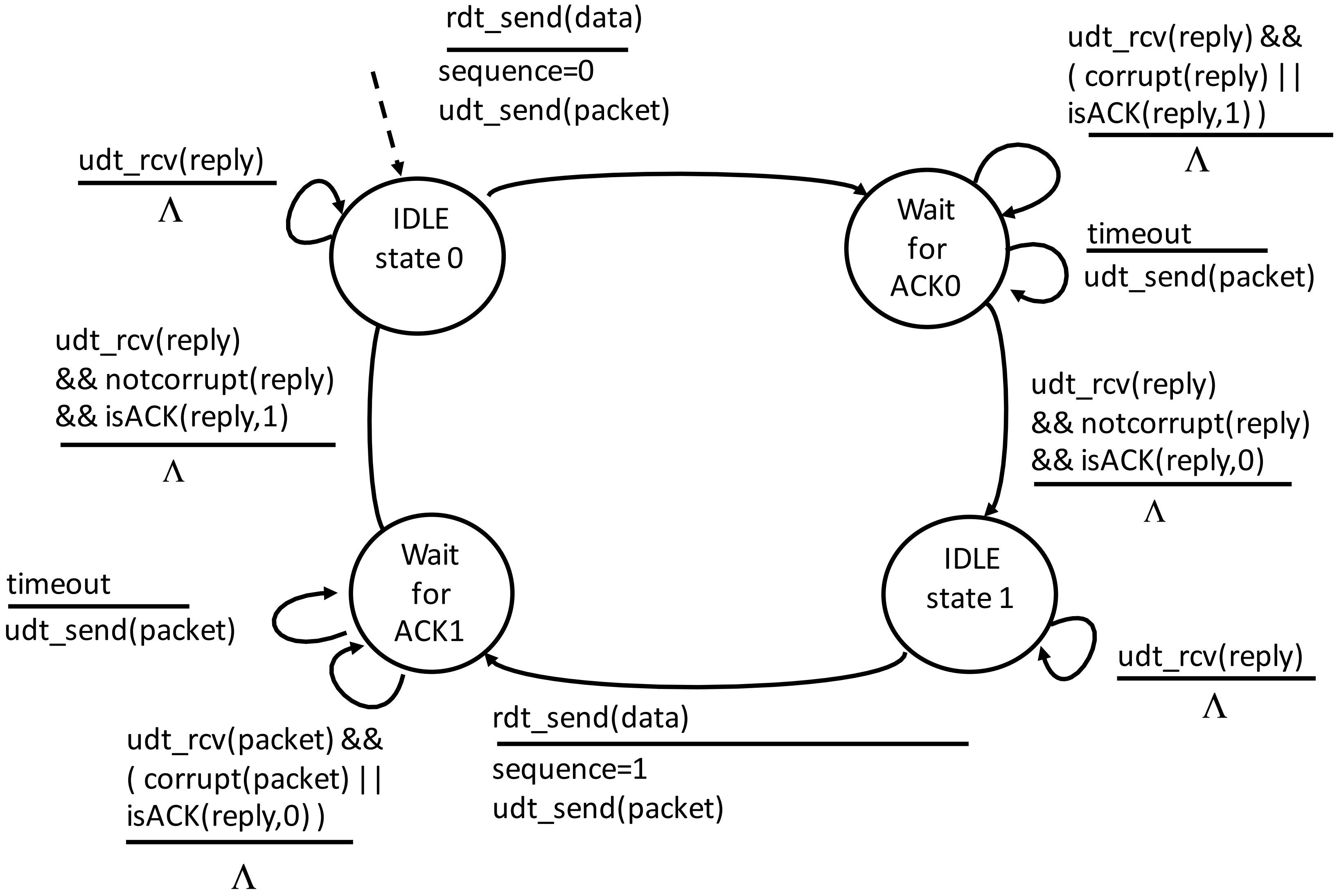
New assumption: underlying channel can also lose packets (data or ACKs)

- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

Approach: sender waits "reasonable" amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #'s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer

rdt3.0 sender



rdt_send(data)

sequence=0
udt_send(packet)

udt_rcv(reply) &&
(corrupt(reply) ||
isACK(reply,1))

Λ

udt_rcv(reply)

Λ

IDLE
state 0

Wait
for
ACK0

timeout

udt_send(packet)

udt_rcv(reply)
&& notcorrupt(reply)
&& isACK(reply,1)

Λ

udt_rcv(reply)
&& notcorrupt(reply)
&& isACK(reply,0)

Λ

timeout

udt_send(packet)

Wait
for
ACK1

IDLE
state 1

udt_rcv(reply)

Λ

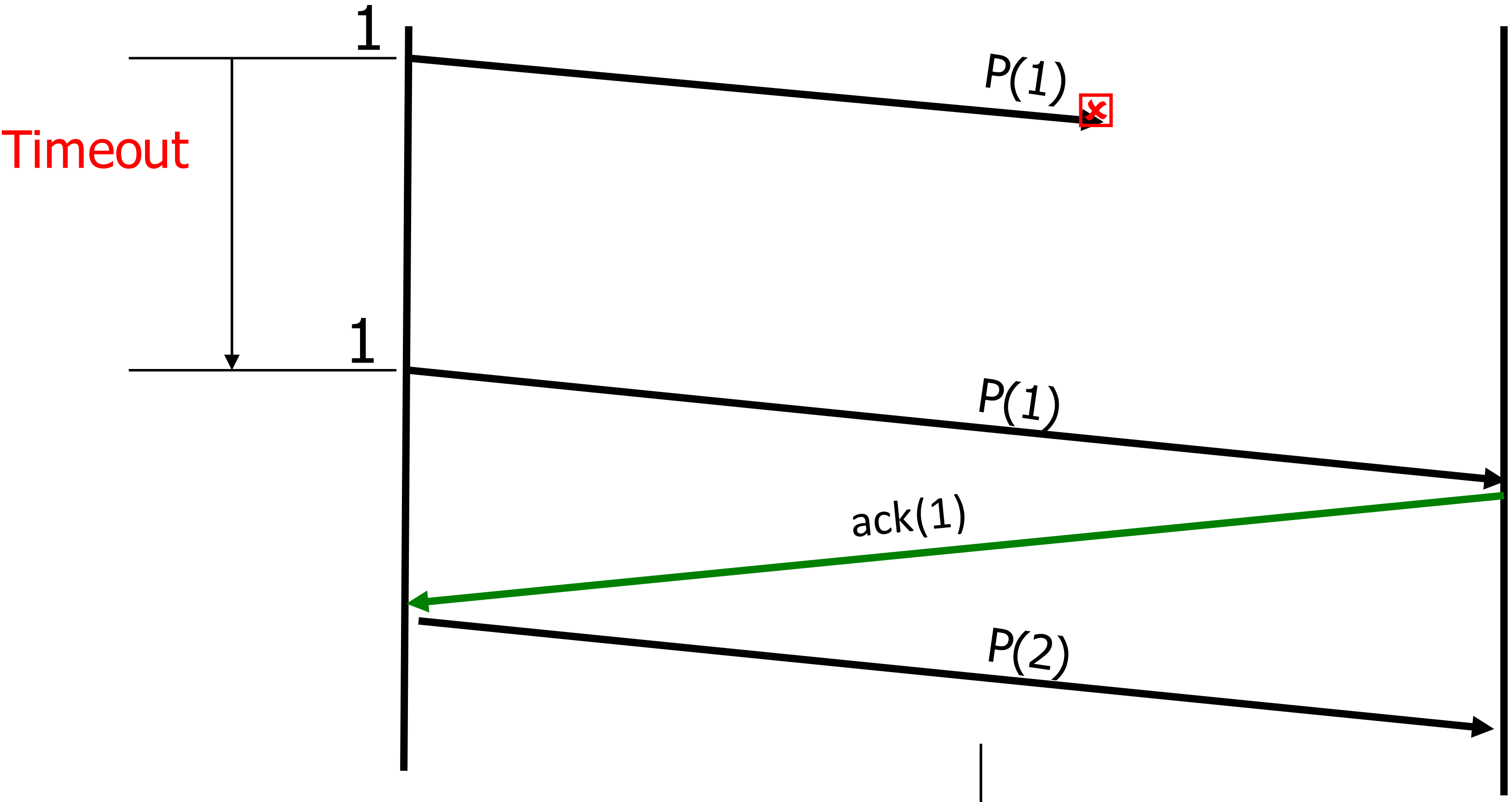
udt_rcv(packet) &&
(corrupt(packet) ||
isACK(reply,0))

Λ

rdt_send(data)

sequence=1
udt_send(packet)

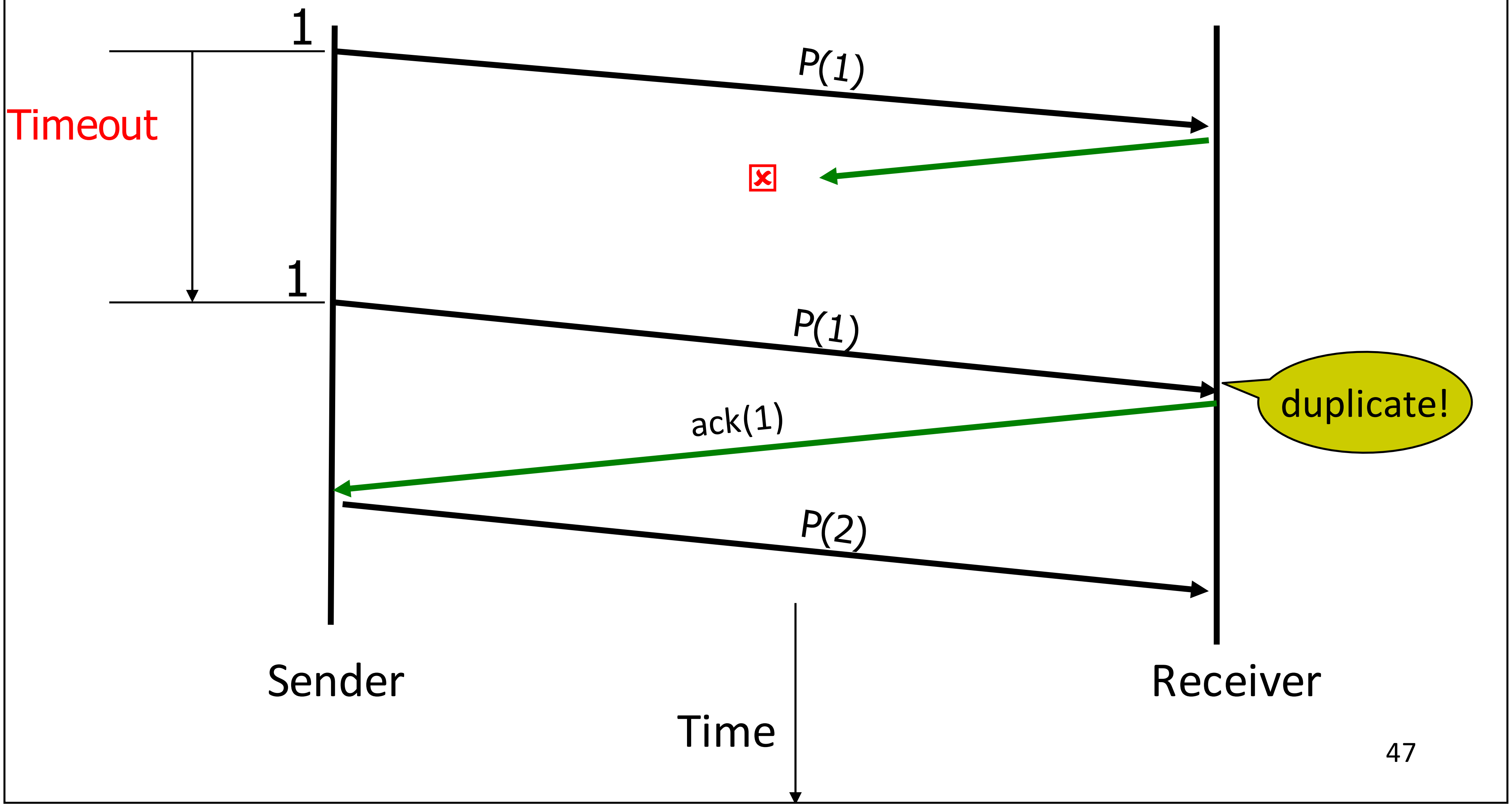
Dealing with Packet Loss



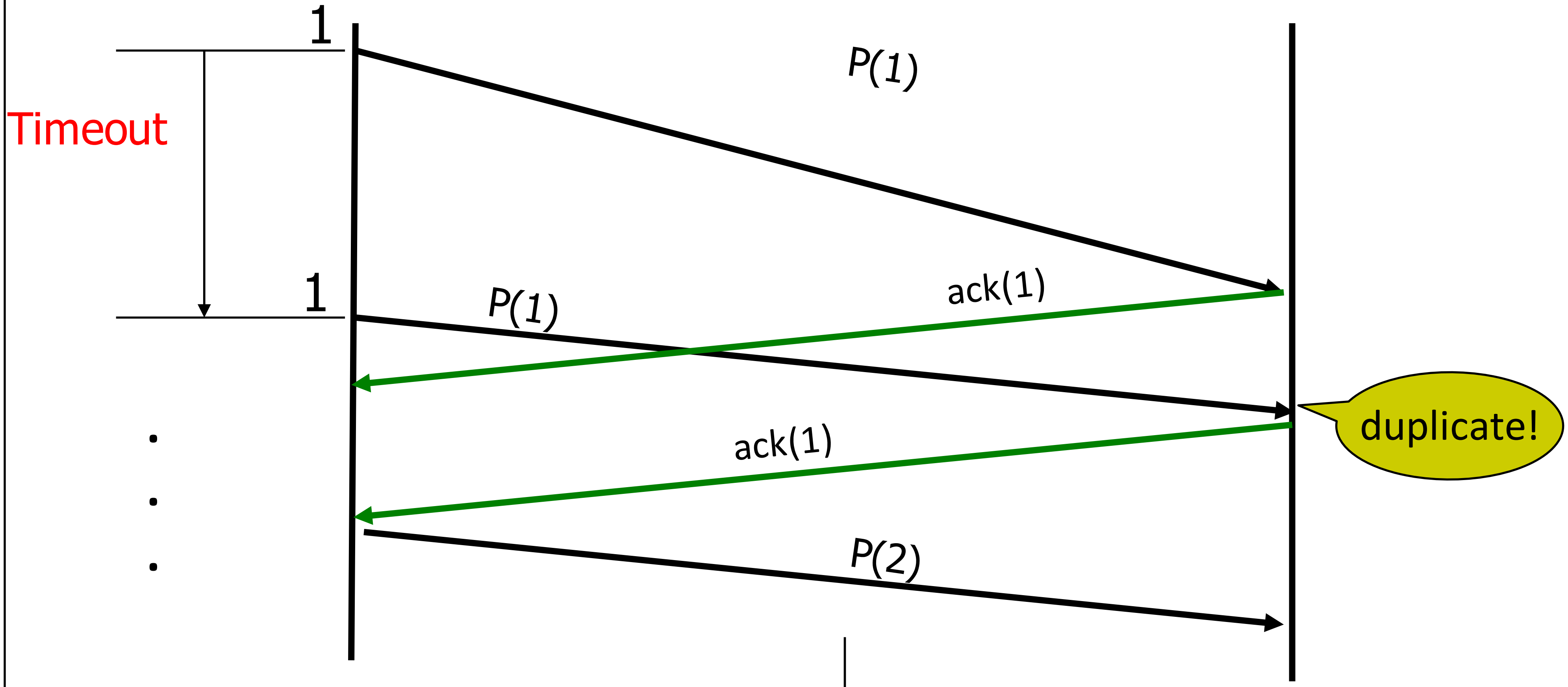
Timer-driven loss detection

Set timer when packet is sent; retransmit on timeout

Dealing with Packet Loss



Dealing with Packet Loss



Timer-driven retx. can lead to duplicates

Performance of rdt3.0

- rdt3.0 works, but performance stinks
- ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

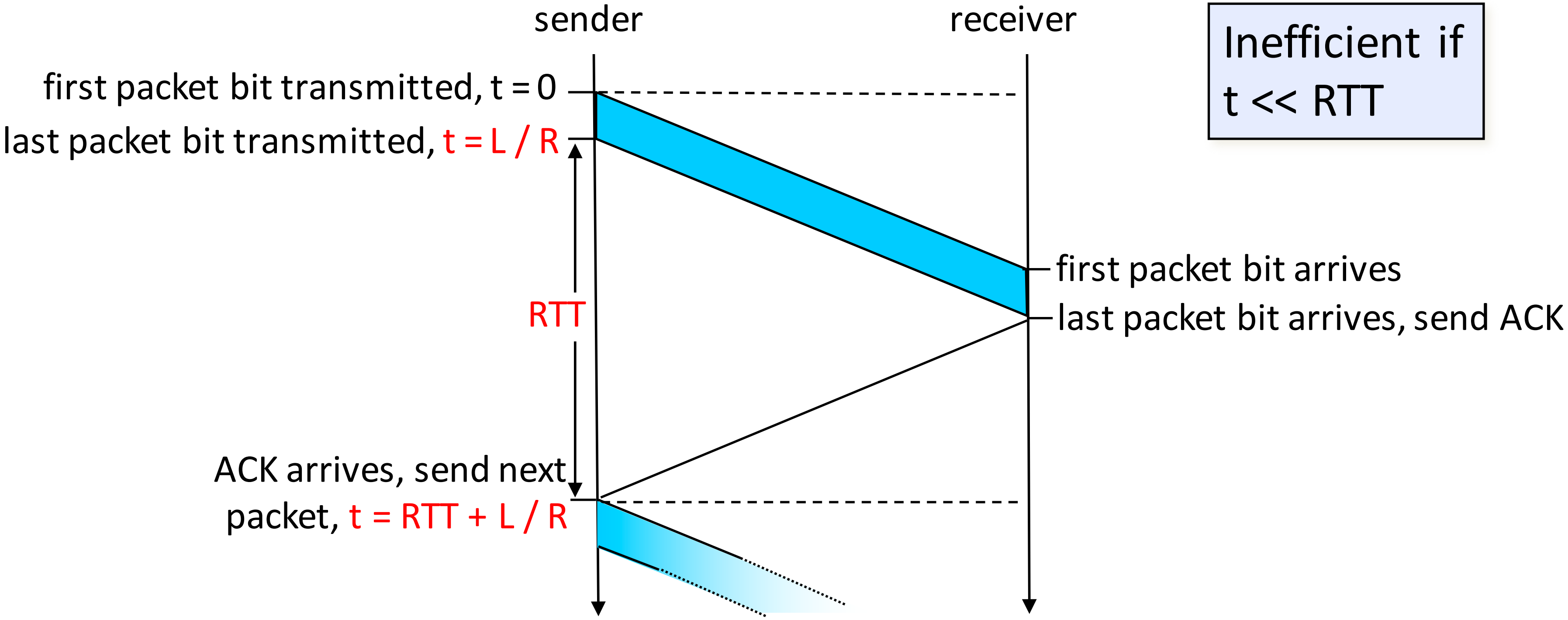
$$d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{bps}} = 8\,\text{microseconds}$$

- U sender: utilization – fraction of time sender busy sending

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- 1KB pkt every 30 msec -> 33kB/sec throughput over 1 Gbps link
- network protocol limits use of physical resources!

rdt3.0: stop-and-wait operation


sender
receiver
first packet bit transmitted, t = 0
last packet bit transmitted, t = L / R
RTT
first packet bit arrives
last packet bit arrives, send ACK
ACK arrives, send next packet, t = RTT + L / R

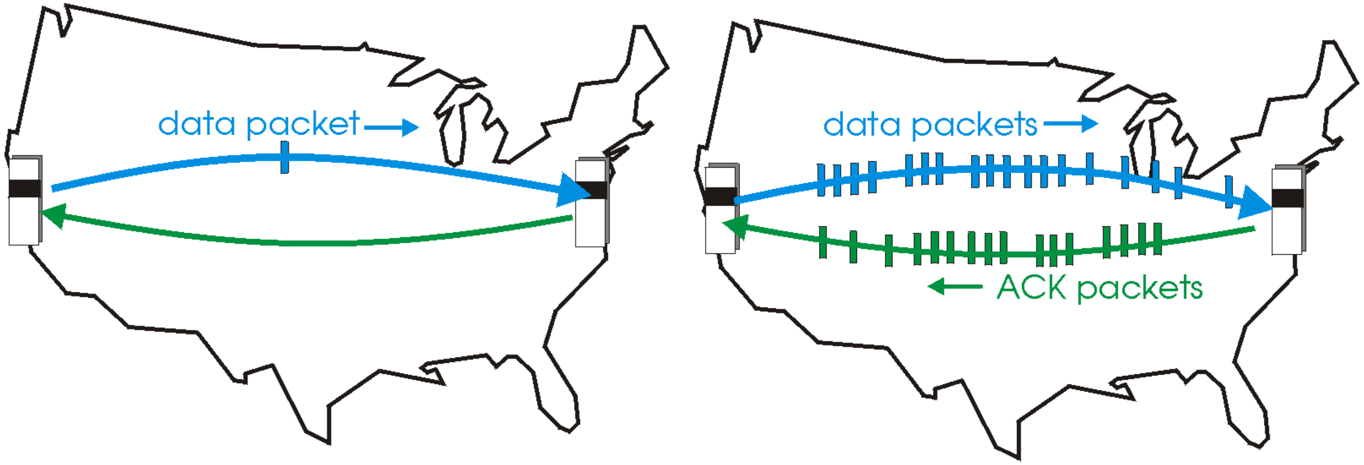
Inefficient if t << RTT

$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Pipelined (Packet-Window) protocols

Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

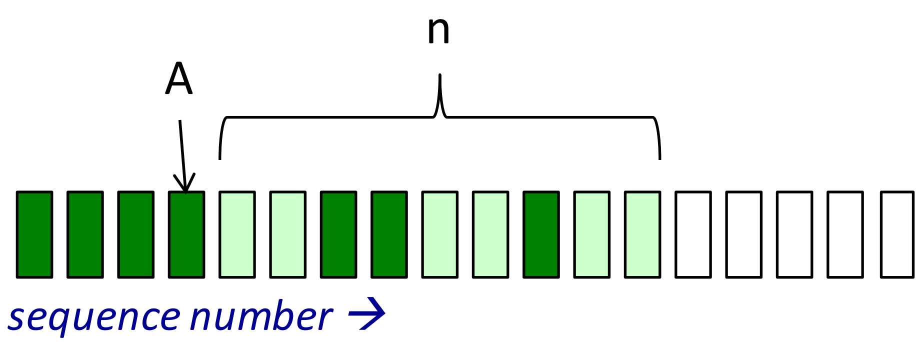
A Sliding Packet Window

- window = set of adjacent sequence numbers
  - The size of the set is the window size; assume window size is *n*

- General idea: send up to *n* packets at a time
  - Sender can send packets in its window
  - Receiver can accept packets in its window
  - Window of acceptable packets “slides” on successful reception/acknowledgement

A Sliding Packet Window

- Let A be the last ack'd packet of sender without gap; then window of sender = {A+1, A+2, …, A+n}

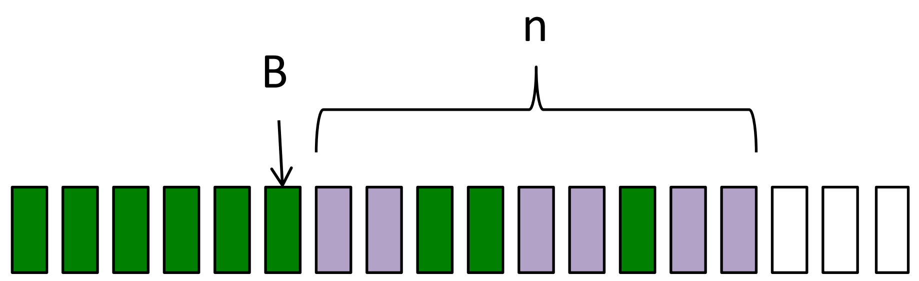


🟩 Already ACK'd

▧ Sent but not ACK'd

□ Cannot be sent

- Let B be the last received packet without gap by receiver, then window of receiver = {B+1,…, B+n}



🟩 Received and ACK'd

🟪 Acceptable but not yet received

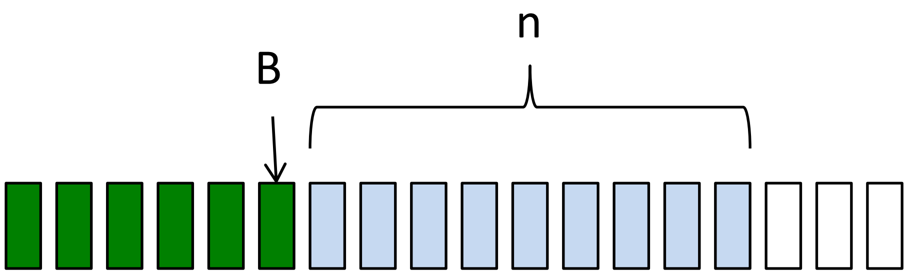
□ Cannot be received

Acknowledgements w/ Sliding Window

- Two common options
  - cumulative ACKs: ACK carries next in-order sequence number that the receiver expects

Cumulative Acknowledgements (1)

- At receiver

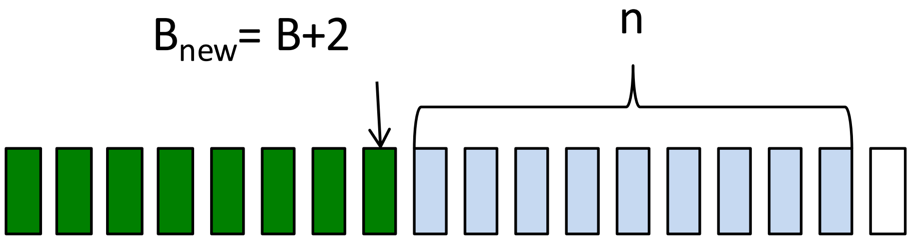


🟩 Received and ACK’d

🟦 Acceptable but not yet received

□ Cannot be received

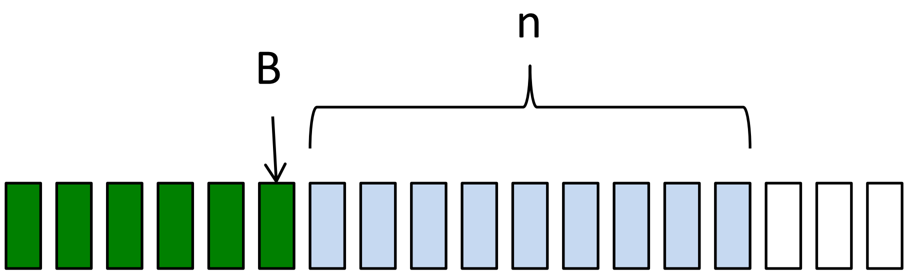
- After receiving B+1, B+2



- Receiver sends ACK(B_{new}+1)

Cumulative Acknowledgements (2)

- At receiver

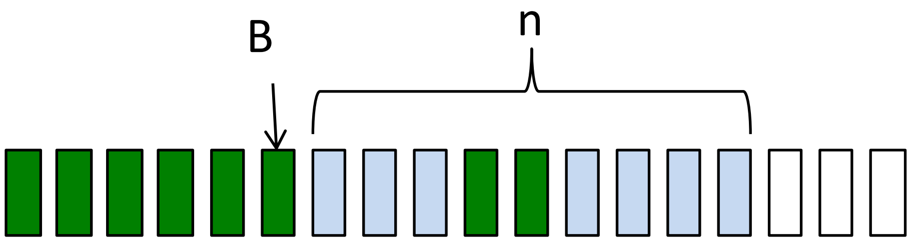


🟩 Received and ACK’d

🟦 Acceptable but not yet received

□ Cannot be received

- After receiving B+4, B+5



How do we recover?

- Receiver sends ACK(B+1)

Go-Back-N (GBN)

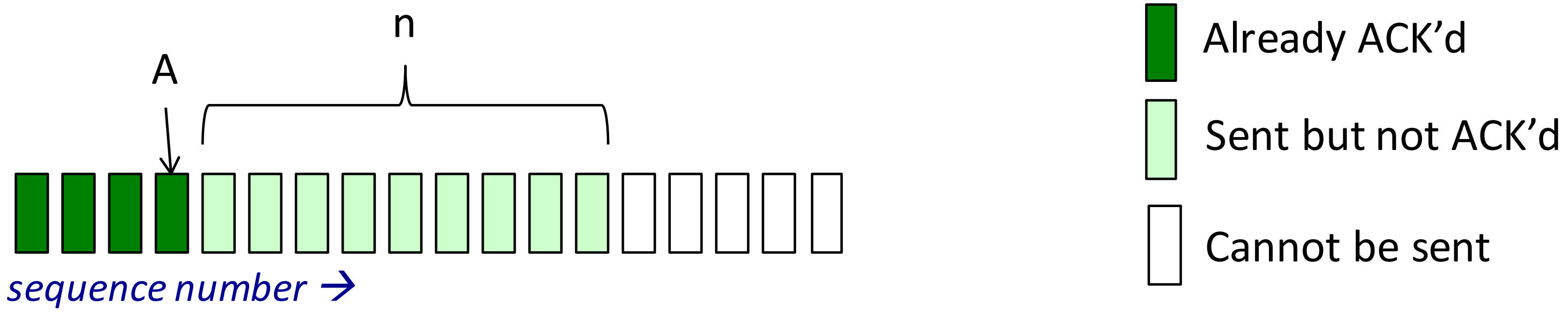
- Sender transmits up to *n* unacknowledged packets

- Receiver only accepts packets in order
  - discards out-of-order packets (i.e., packets other than *B+1*)
- Receiver uses cumulative acknowledgements
  - i.e., sequence# in ACK = next expected in-order sequence#

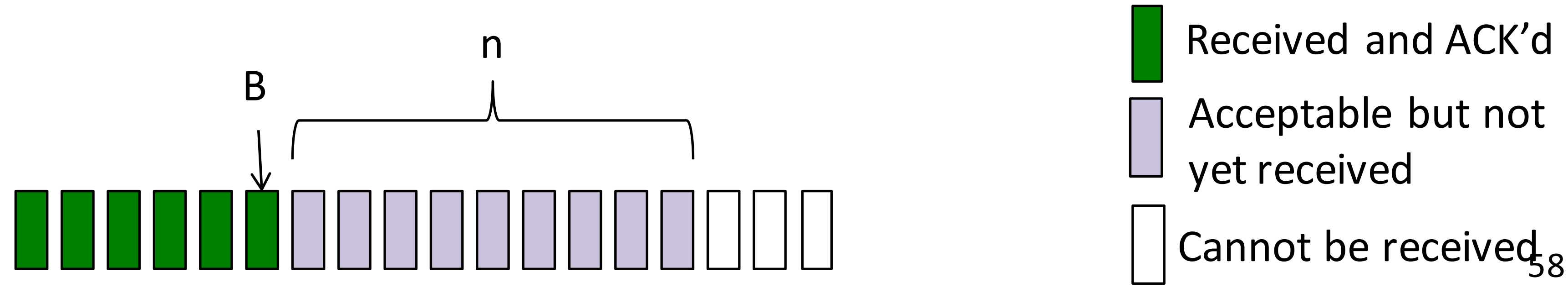
- Sender sets timer for 1st outstanding ack (A+1)
- If timeout, retransmit *A+1*, … , *A+n*

Sliding Window with GBN

- Let A be the last ack'd packet of sender without gap; then window of sender = {A+1, A+2, …, A+n}

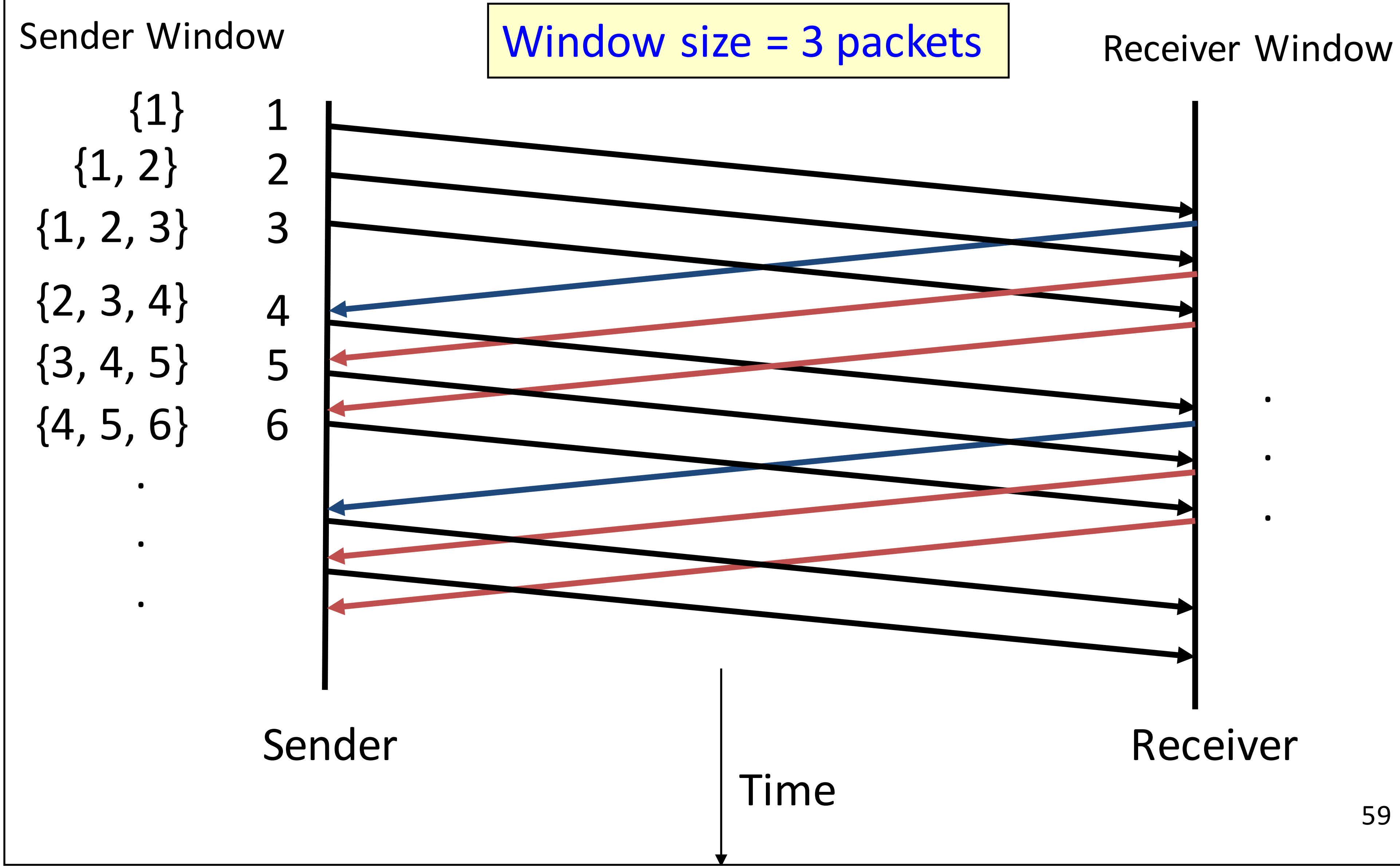


- Let B be the last received packet without gap by receiver, then window of receiver = {B+1,…, B+n}



GBN Example w/o Errors

Window size = 3 packets



Sender Window

Receiver Window

{1} 1

{1, 2} 2

{1, 2, 3} 3

{2, 3, 4} 4

{3, 4, 5} 5

{4, 5, 6} 6

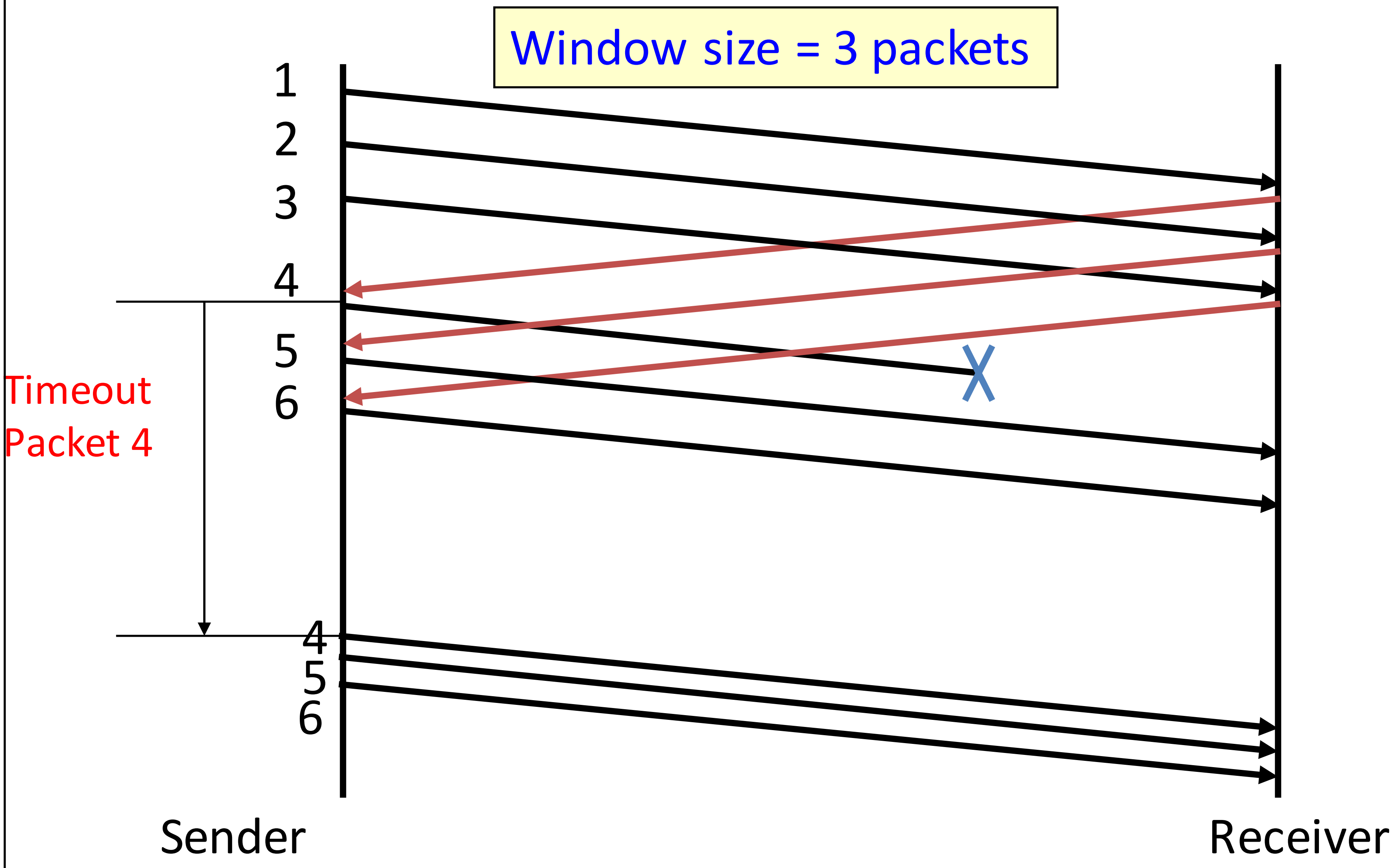
.
.
.

Sender

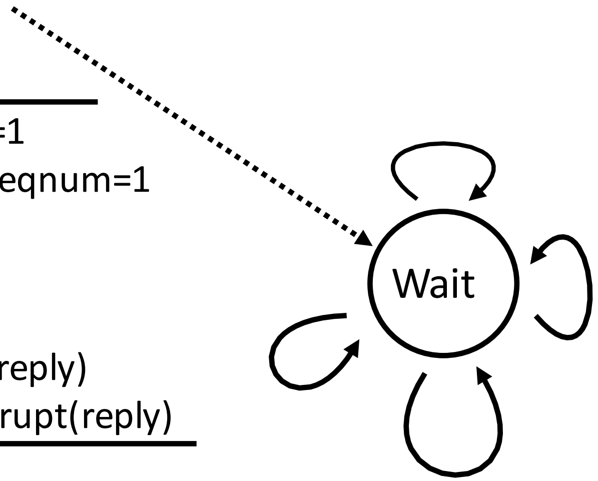
Receiver

Time

GBN Example with Errors



GBN: sender extended FSM



```
rdt_send(data)
______________________________
if (nextseqnum < base+N) {
   udt_send(packet[nextseqnum])
   nextseqnum++
   }
else
  refuse_data(data)   Block?
```

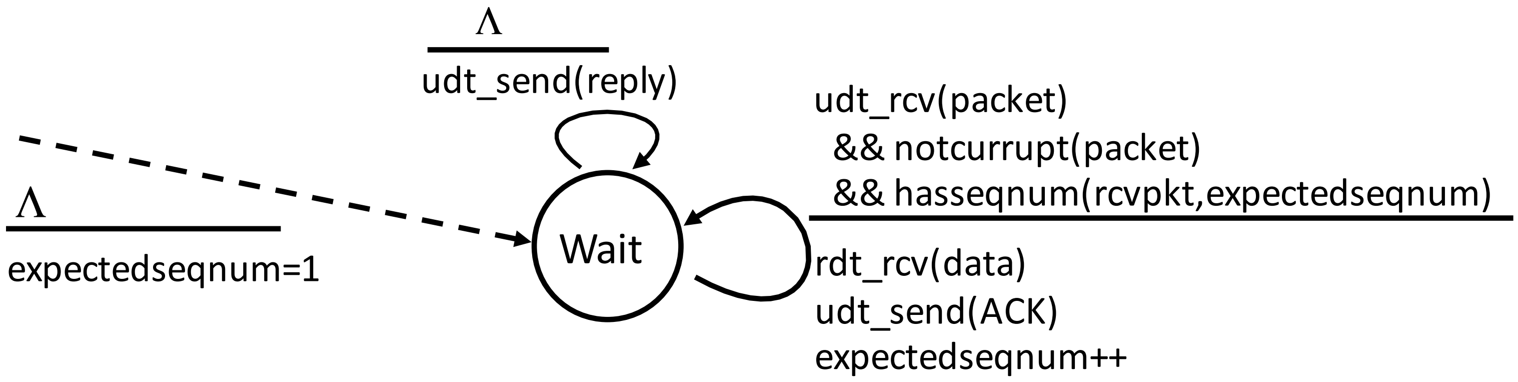
```
Λ
______________
base=1
nextseqnum=1
```

```
timeout
______________________________
udt_send(packet[base])
udt_send(packet[base+1])
…
udt_send(packet[nextseqnum-1])
```

```
udt_rcv(reply)
  && corrupt(reply)
______________________
      Λ
```

```
udt_rcv(reply) &&
  notcorrupt(reply)
______________________
base = getacknum(reply)+1
```

GBN: receiver extended FSM



ACK-only: always send an ACK for correctly-received packet with the highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

- out-of-order packet:
  - discard (don't buffer) -> no receiver buffering!
  - Re-ACK packet with highest in-order seq #

Acknowledgements w/ Sliding Window

- Two common options
  - cumulative ACKs: ACK carries next in-order sequence number the receiver expects
  - selective ACKs: ACK individually acknowledges correctly received packets

- Selective ACKs offer more precise information but require more complicated book-keeping

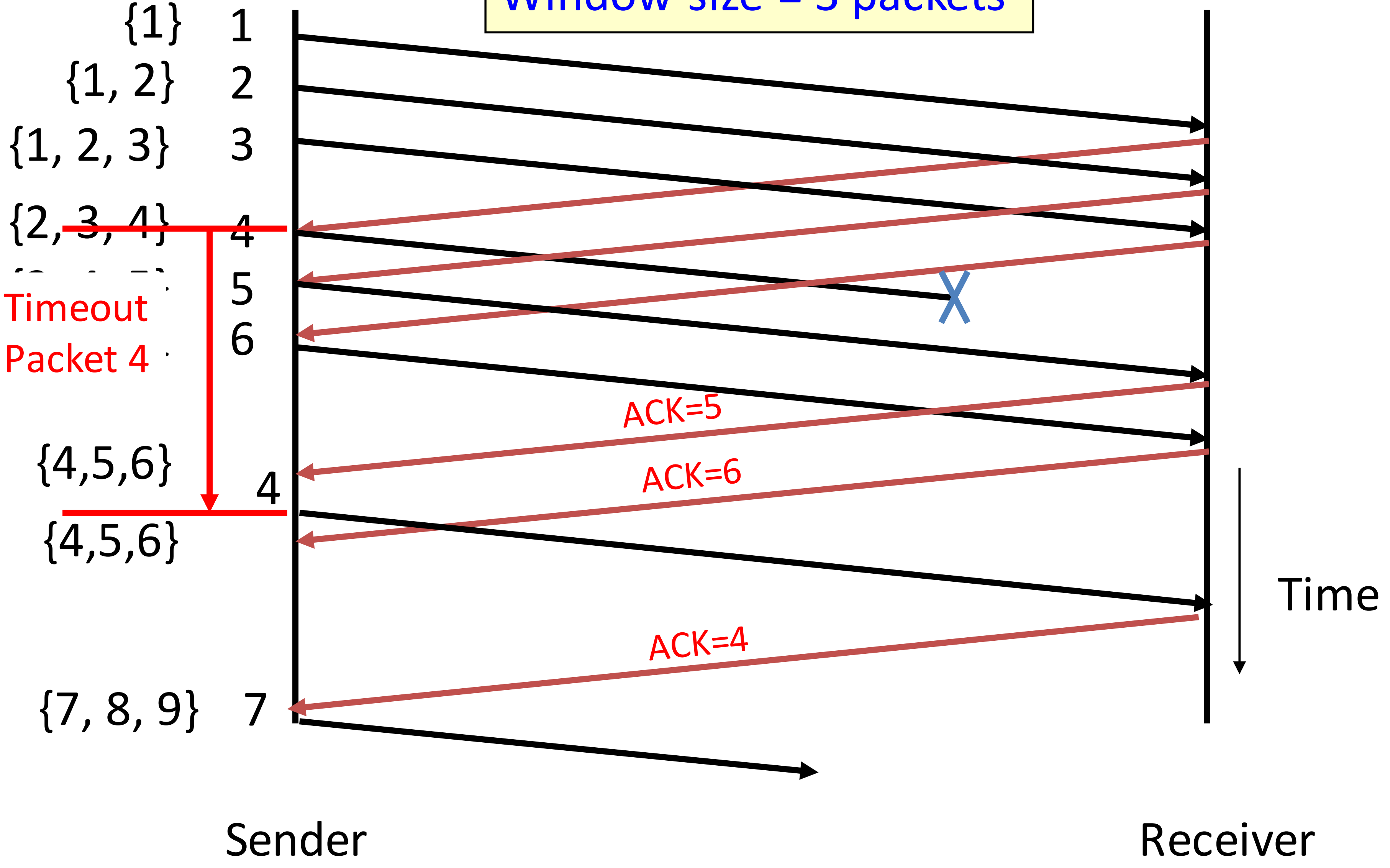
- Many variants that differ in implementation details

Selective Repeat (SR)

- Sender: transmit up to *n* unacknowledged packets
- Assume packet *k* is lost, *k+1* is not
- Receiver: indicates packet *k+1* correctly received
- Sender: retransmit only packet *k* on timeout
- Efficient in retransmissions but complex book-keeping
  - need a timer per packet

SR Example with Errors

Window size = 3 packets



Observations

- With sliding windows, it is possible to fully utilize a link, provided the window size (n) is large enough. Throughput is ~ (n/RTT)
  - Stop & Wait is like n = 1.
- Sender has to buffer all unacknowledged packets, because they may require retransmission
- Receiver may be able to accept out-of-order packets, but only up to its buffer limits
- Implementation complexity depends on protocol details (GBN vs. SR)

Recap: components of a solution

- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgments
  - cumulative
  - selective
- Sequence numbers (duplicates, windows)
- Sliding Windows (for efficiency)

- Reliability protocols use the above to decide when and what to retransmit or acknowledge

What does TCP do?

Most of our previous tricks + a few differences

- Sequence numbers are byte offsets
- Sender and receiver maintain a sliding window
- Receiver sends cumulative acknowledgements (like GBN)
- Sender maintains a single retx. timer
- Receivers do not drop out-of-sequence packets (like SR)
- Introduces fast retransmit : optimization that uses duplicate ACKs to trigger early retx
- Introduces timeout estimation algorithms

Automatic Repeat Request (ARQ)

+ Self-clocking (Automatic)

+ Adaptive

+ Flexible

- Slow to start / adapt

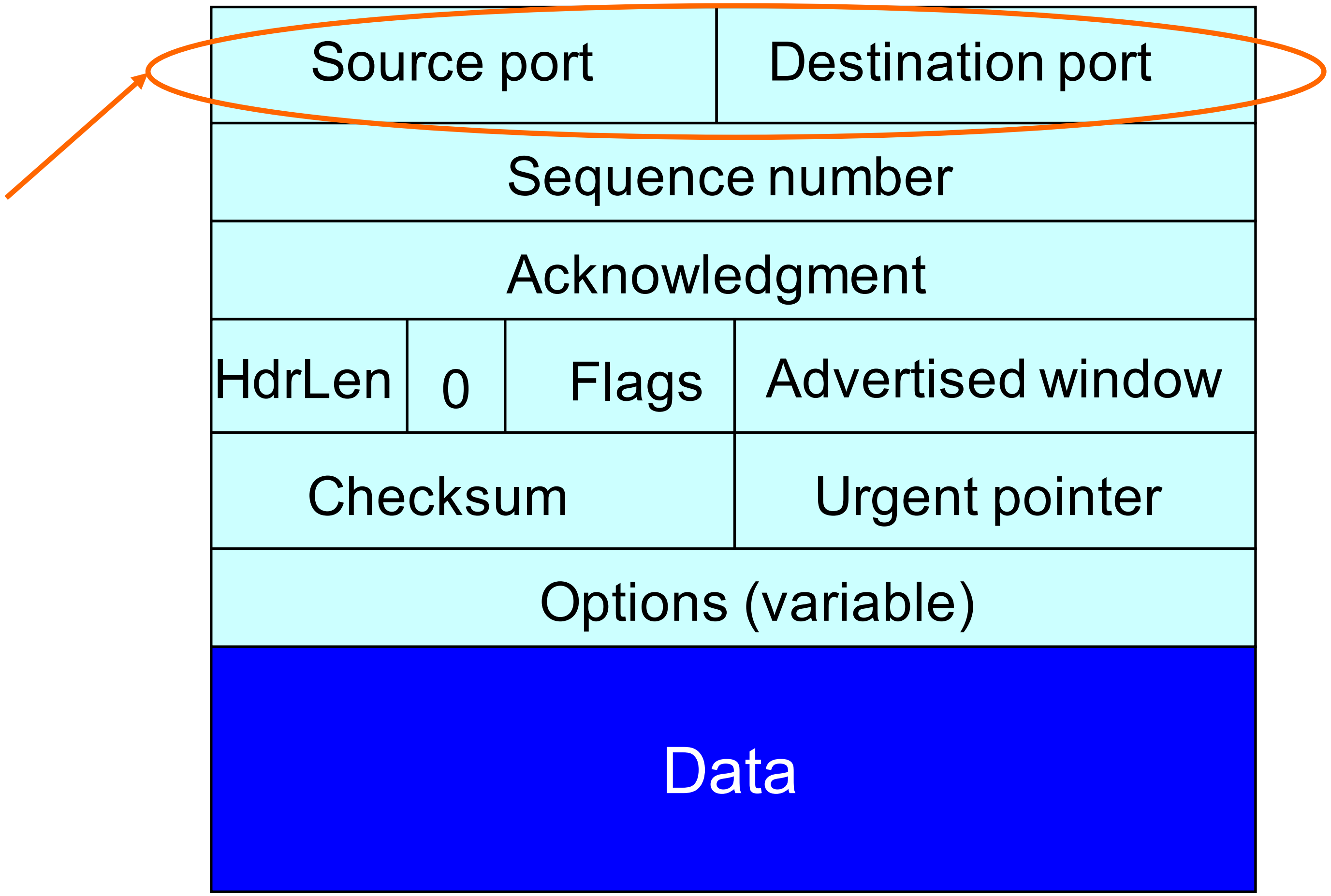
consider high Bandwidth/Delay product

Next lets move from the generic to the specific....

TCP arguably the most successful protocol in the Internet.....

its an ARQ protocol

TCP Header



Used to mux and demux

Source port | Destination port

Sequence number

Acknowledgment

HdrLen | 0 | Flags | Advertised window

Checksum | Urgent pointer

Options (variable)

Data

Last time: Components of a solution for reliable transport

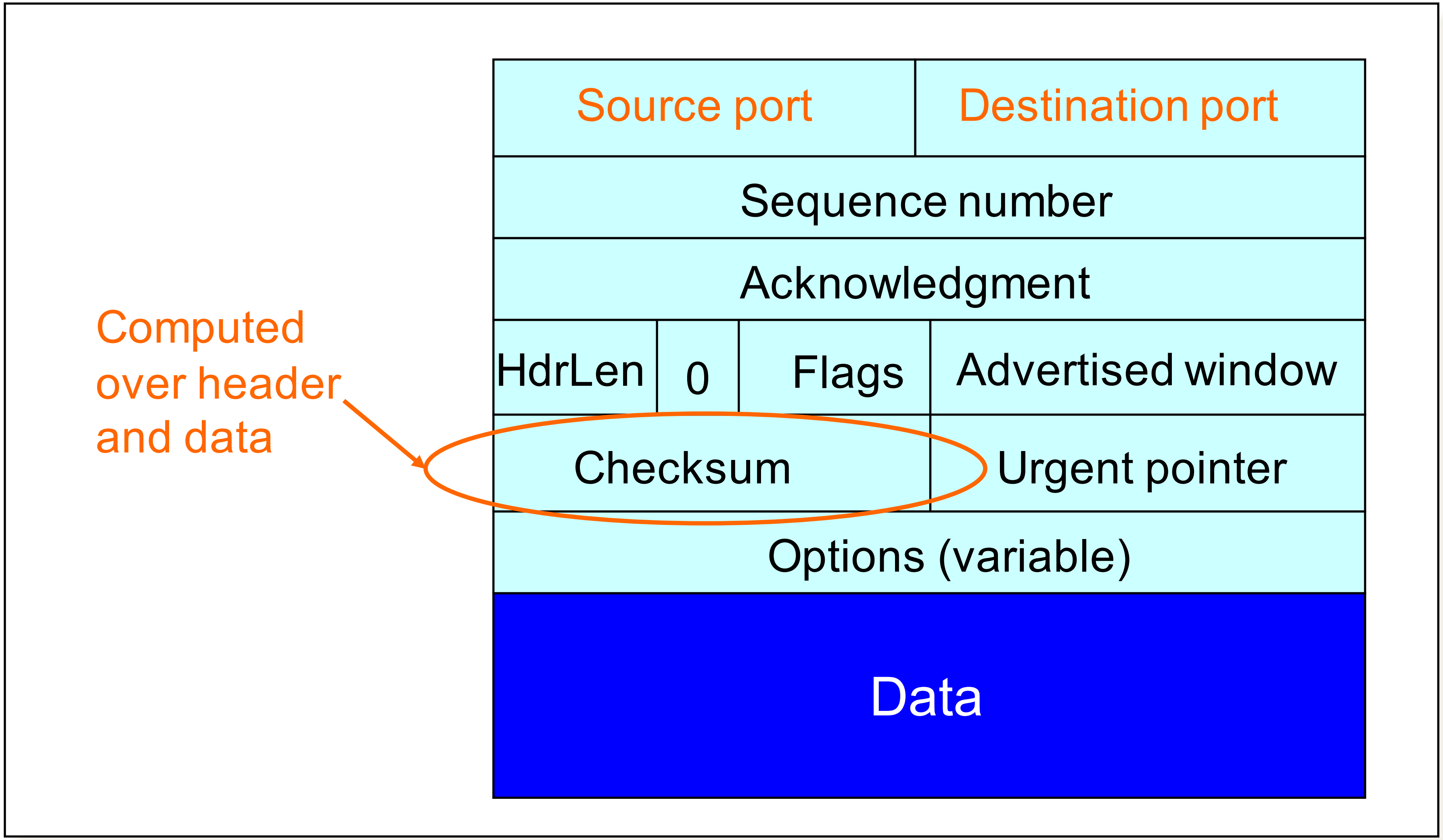
- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgments
  - cumulative
  - selective
- Sequence numbers (duplicates, windows)
- Sliding Windows (for efficiency)
  - Go-Back-N (GBN)
  - Selective Replay (SR)

What does TCP do?

Many of our previous ideas, but some key differences

- Checksum

TCP Header



Source port | Destination port

Sequence number

Acknowledgment

HdrLen | 0 | Flags | Advertised window

Checksum | Urgent pointer

Options (variable)

Data

Computed over header and data

What does TCP do?

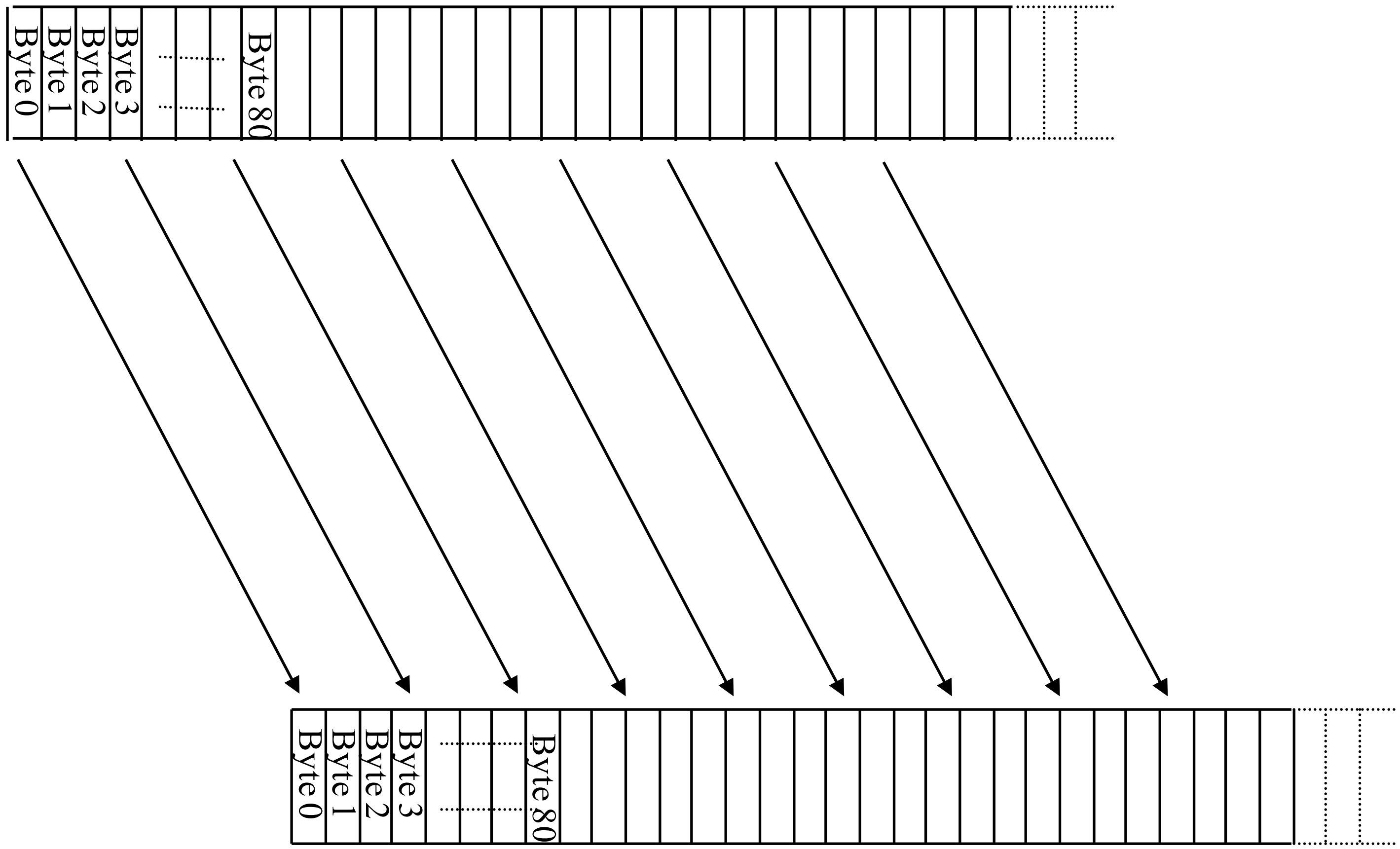
Many of our previous ideas, but some key differences

- Checksum
- **Sequence numbers are byte offsets**

TCP: Segments and Sequence Numbers

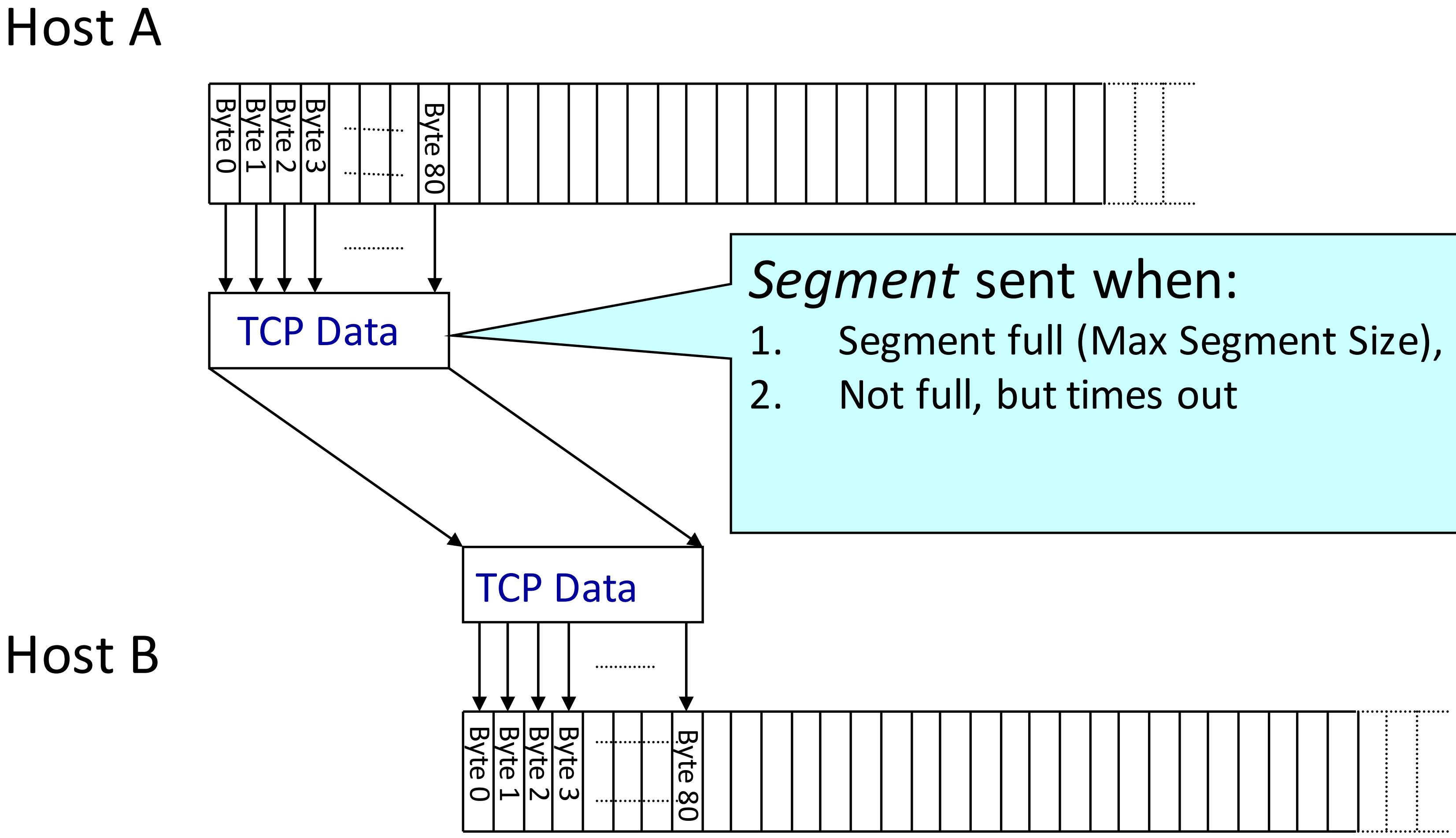
TCP "Stream of Bytes" Service…

Application @ Host A



Application @ Host B

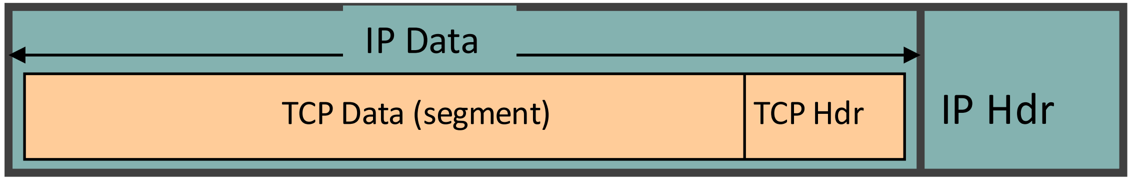
… Provided Using TCP “Segments”



Segment sent when:

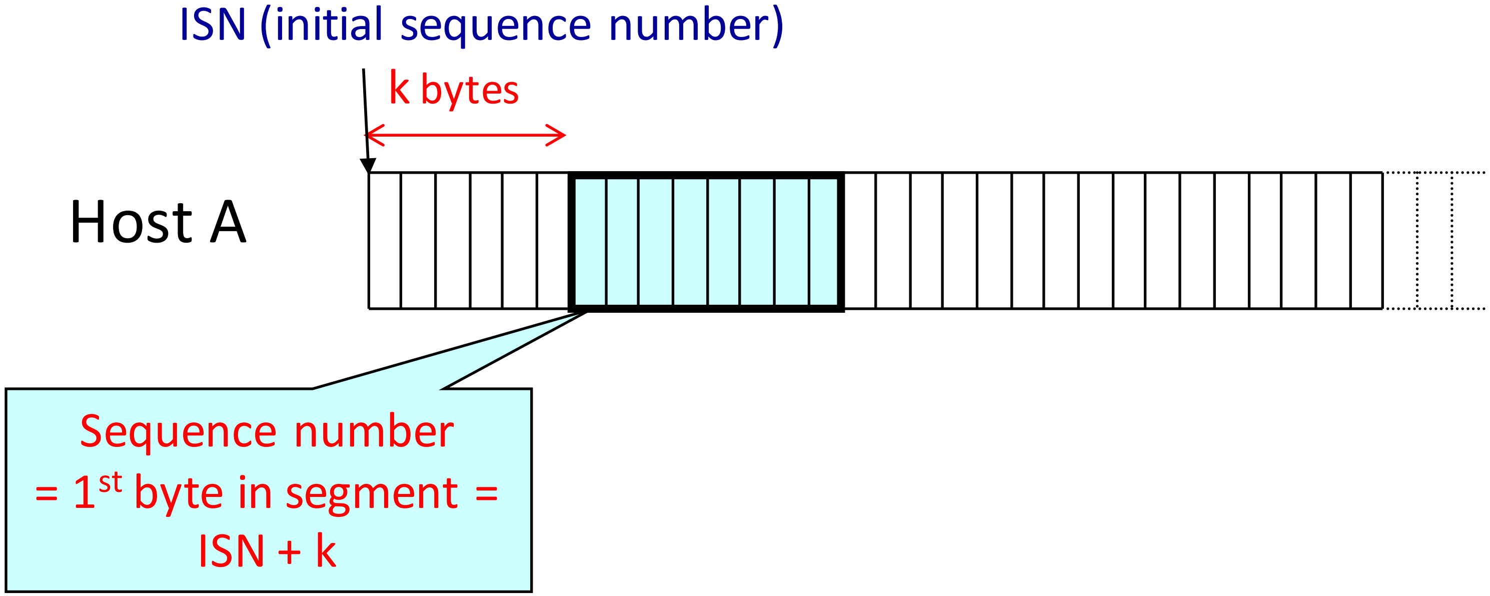
1. Segment full (Max Segment Size),
2. Not full, but times out

TCP Segment



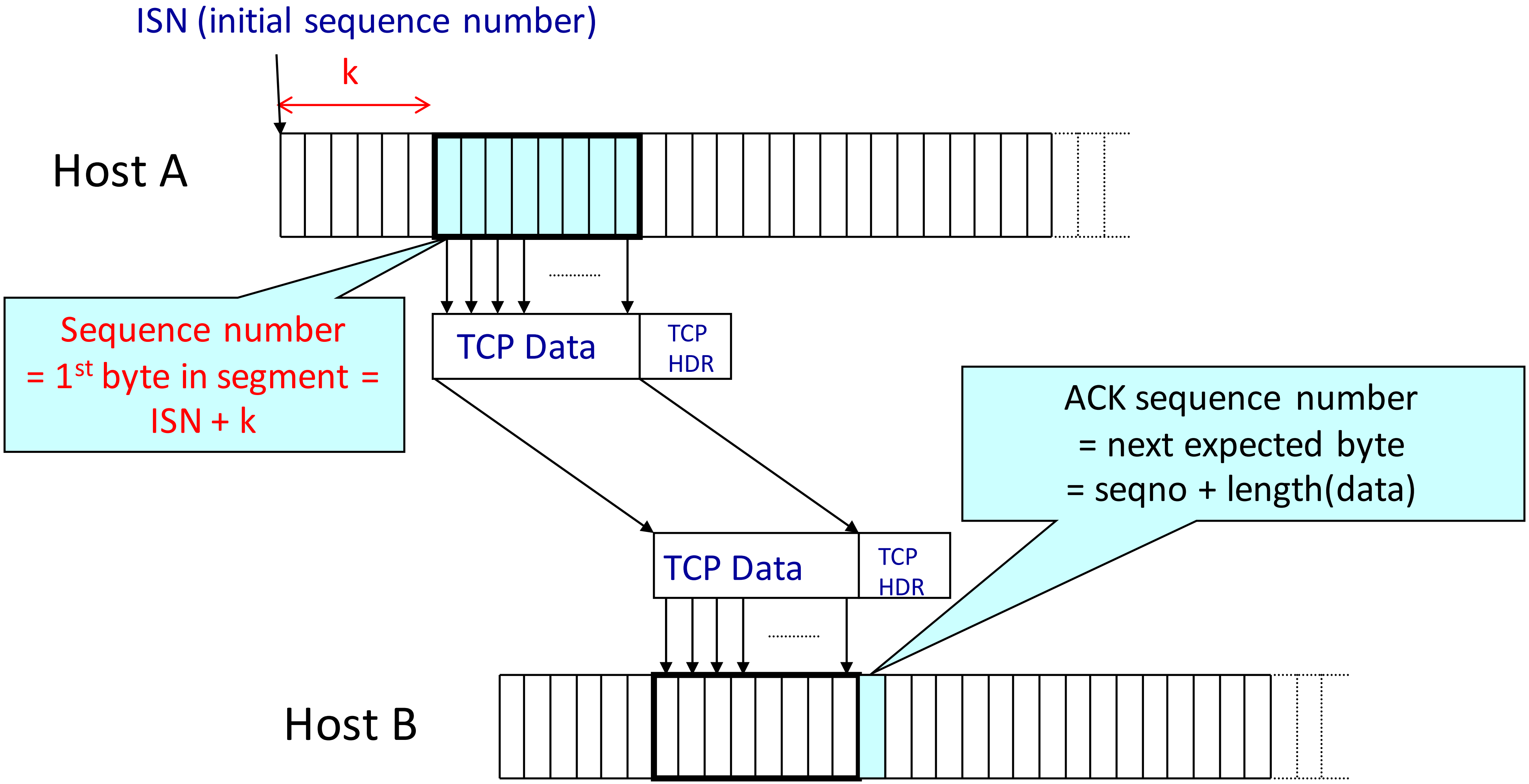
- IP packet
  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes with Ethernet
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header ≥ 20 bytes long
- TCP **segment**
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes from the stream
  - MSS = MTU – (IP header) – (TCP header)

Sequence Numbers



Sequence number = 1st byte in segment = ISN + k

Sequence Numbers

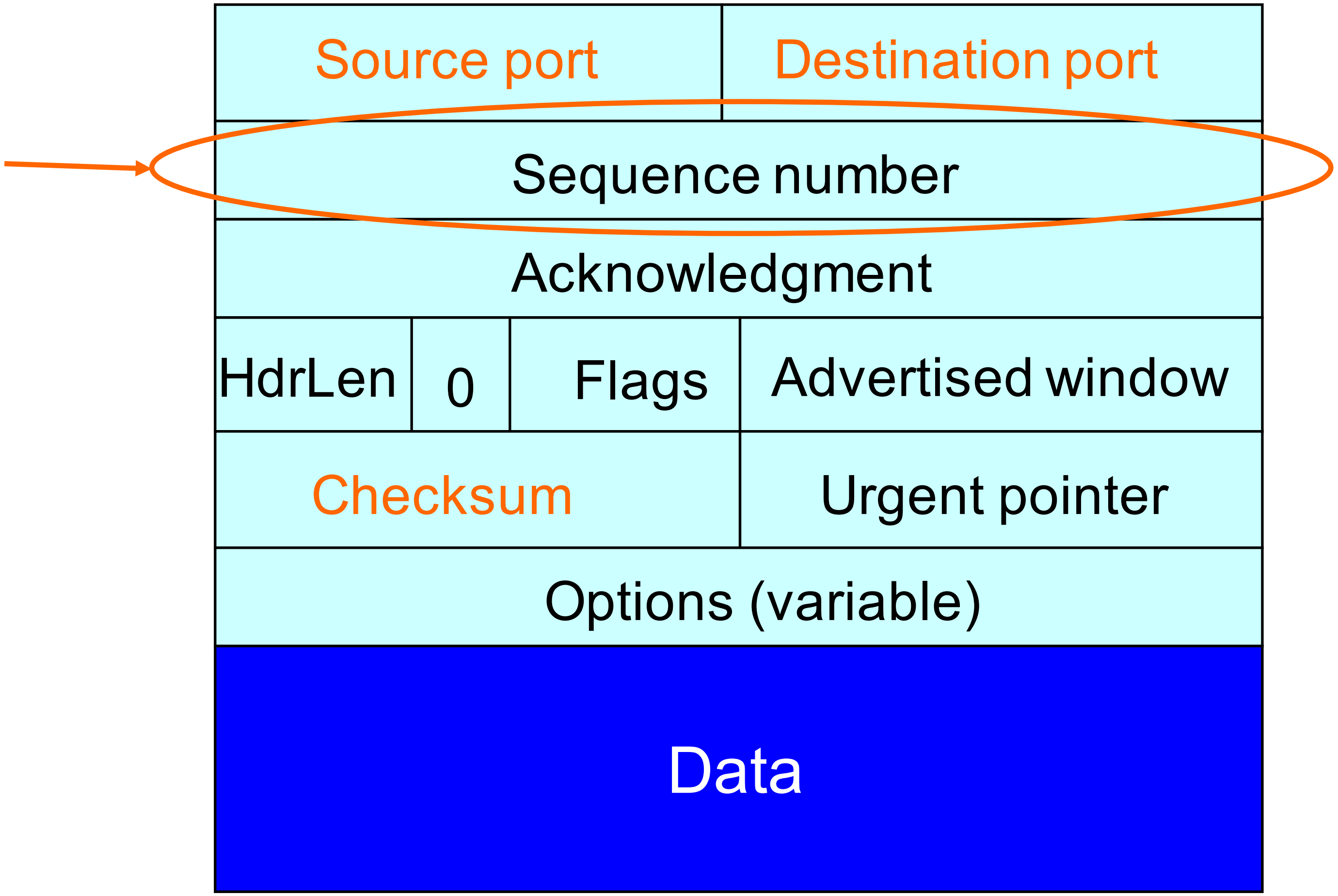


Sequence number = 1st byte in segment = ISN + k

ACK sequence number
= next expected byte
= seqno + length(data)

TCP Header

Starting byte offset of data carried in this segment



Source port | Destination port

Sequence number

Acknowledgment

HdrLen | 0 | Flags | Advertised window

Checksum | Urgent pointer

Options (variable)

Data

- What does TCP do?

What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)

ACKing and Sequence Numbers

- Sender sends packet
  - Data starts with sequence number X
  - Packet contains B bytes [X, X+1, X+2, ….X+B-1]
- Upon receipt of packet, receiver sends an ACK
  - If all data prior to X already received:
    - ACK acknowledges X+B (because that is next expected byte)
  - If highest in-order byte received is Y s.t. (Y+1) < X
    - ACK acknowledges Y+1
    - Even if this has been ACKed before

Normal Pattern

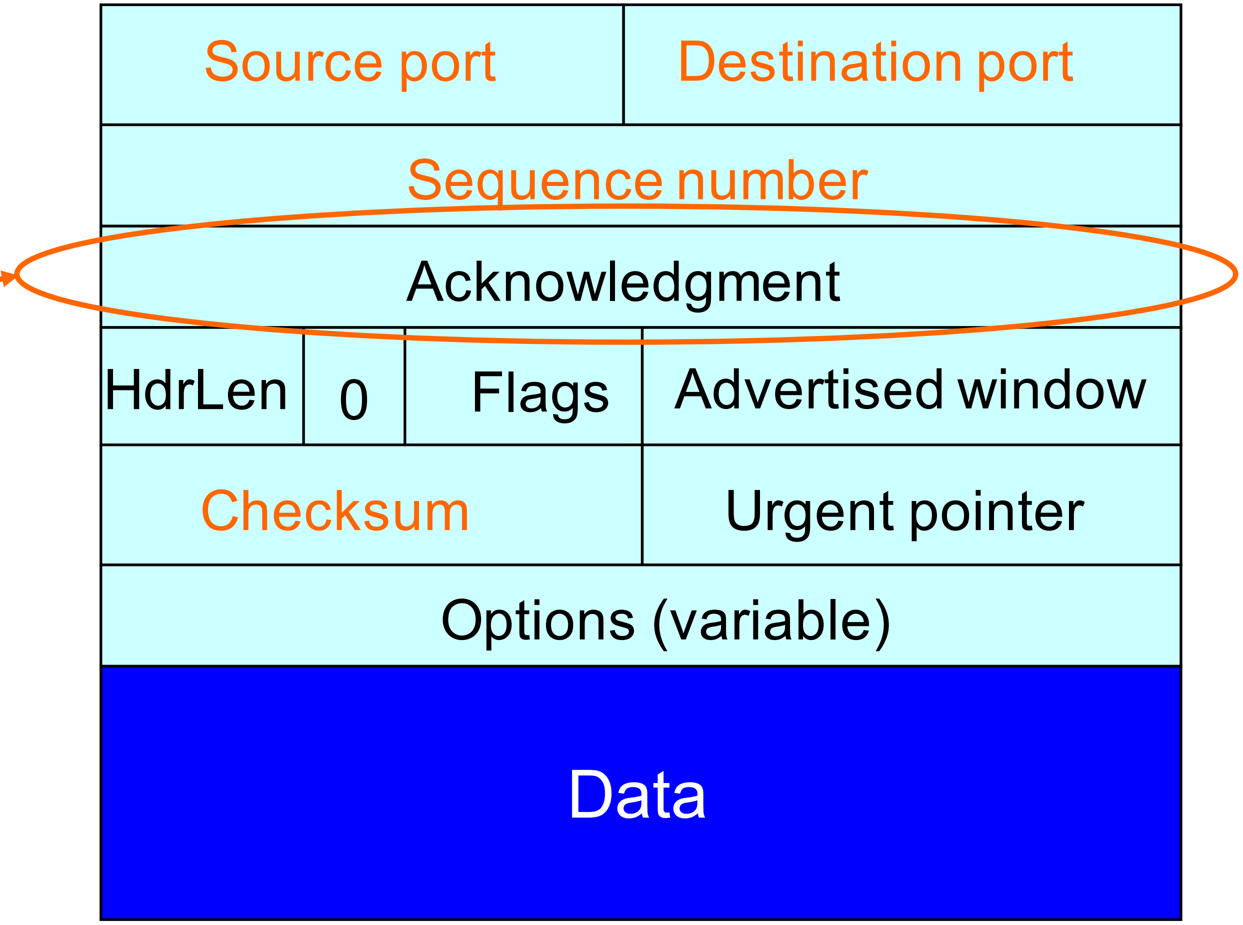
- Sender: seqno=X, length=B
- Receiver: ACK=X+B
- Sender: seqno=X+B, length=B
- Receiver: ACK=X+2B
- Sender: seqno=X+2B, length=B

- Seqno of next packet is same as last ACK field

TCP Header

Acknowledgment gives seqno just beyond highest seqno received **in order**

("What Byte is Next")



Source port

Destination port

Sequence number

Acknowledgment

HdrLen

0

Flags

Advertised window

Checksum

Urgent pointer

Options (variable)

Data

What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers can buffer out-of-sequence packets (like SR)

Loss with cumulative ACKs

- Sender sends packets with 100B and seqnos.:
  - 100, 200, 300, 400, 500, 600, 700, 800, 900, …
- Assume the fifth packet (seqno 500) is lost, but no others
- Stream of ACKs will be:
  - 200, 300, 400, 500, 500, 500, 500,…

What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers may not drop out-of-sequence packets (like SR)
- Introduces fast retransmit: optimization that uses duplicate ACKs to trigger early retransmission

Loss with cumulative ACKs

- "Duplicate ACKs" are a sign of an isolated loss
  - The lack of ACK progress means 500 hasn't been delivered
  - Stream of ACKs means some packets are being delivered

- Therefore, could trigger resend upon receiving k duplicate ACKs
    - TCP uses k=3

- But response to loss is trickier....

Loss with cumulative ACKs

- Two choices:
  - Send missing packet and increase W by the number of dup ACKs
  - Send missing packet, and wait for ACK to increase W

- Which should TCP do?

What does TCP do?

Most of our previous tricks, but a few differences

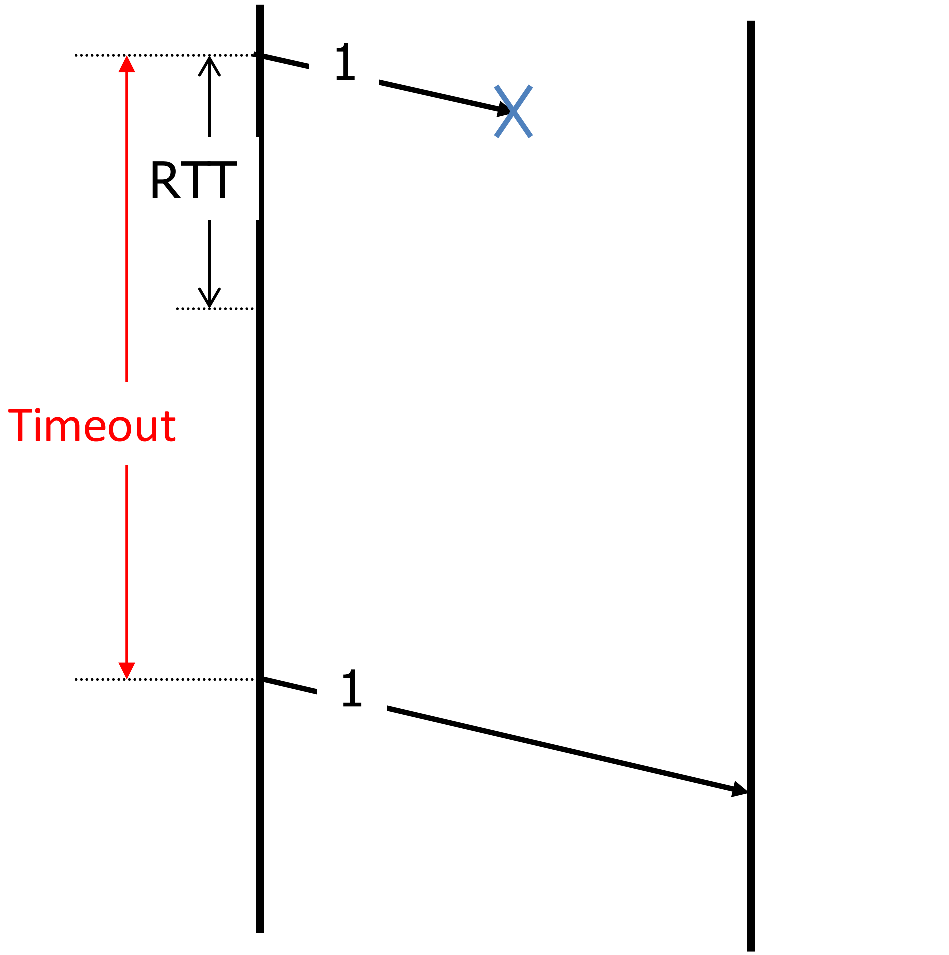
- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers do not drop out-of-sequence packets (like SR)
- Introduces fast retransmit: optimization that uses duplicate ACKs to trigger early retransmission
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout

Retransmission Timeout

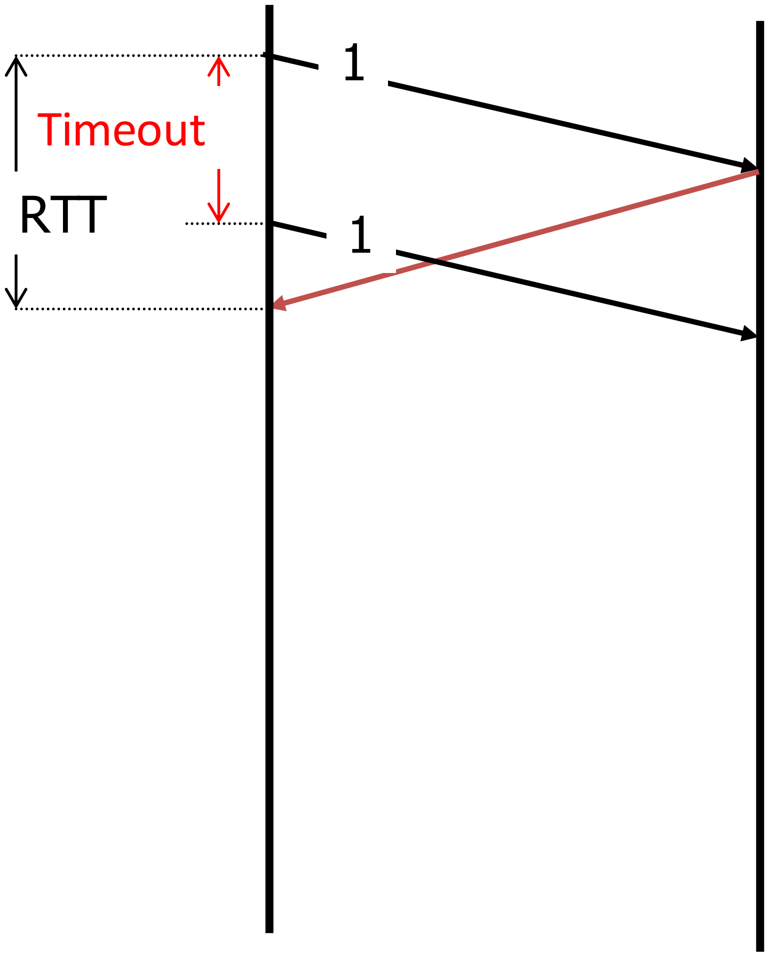
- If the sender hasn’t received an ACK by timeout, retransmit the first packet in the window

- How do we pick a timeout value?

Timing Illustration



Timeout too long → inefficient



Timeout too short → duplicate packets

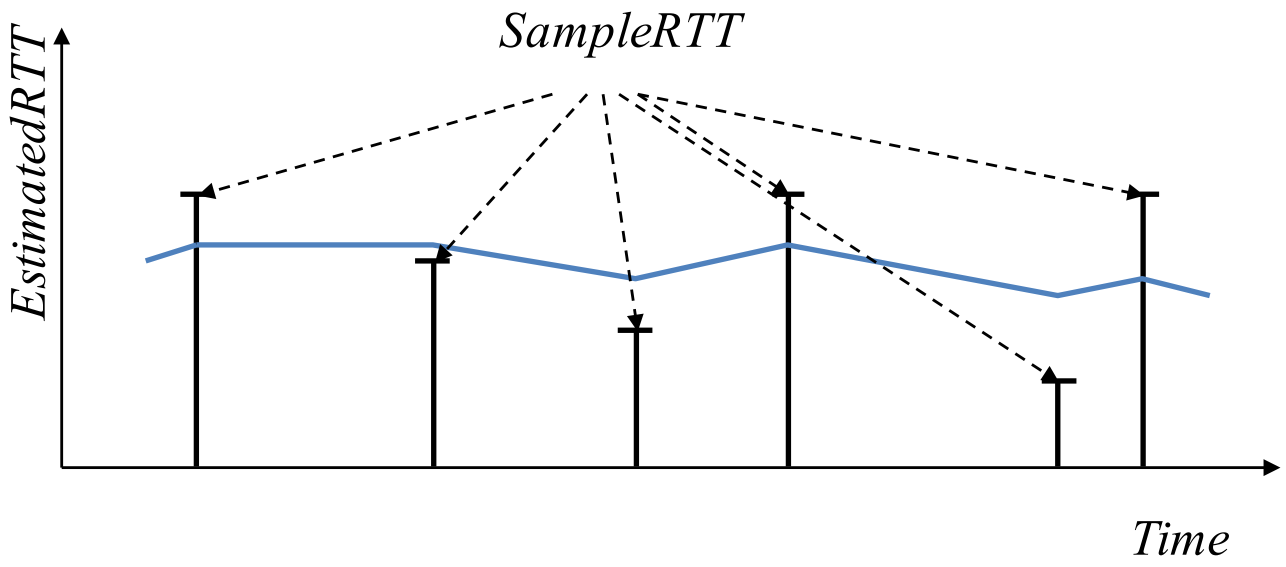
Retransmission Timeout

- If haven’t received ack by timeout, retransmit the first packet in the window
- How to set timeout?
  - Too long: connection has low throughput
  - Too short: retransmit packet that was just delayed
- Solution: make timeout proportional to RTT
- But how do we measure RTT?

RTT Estimation

- Use exponential averaging of RTT samples

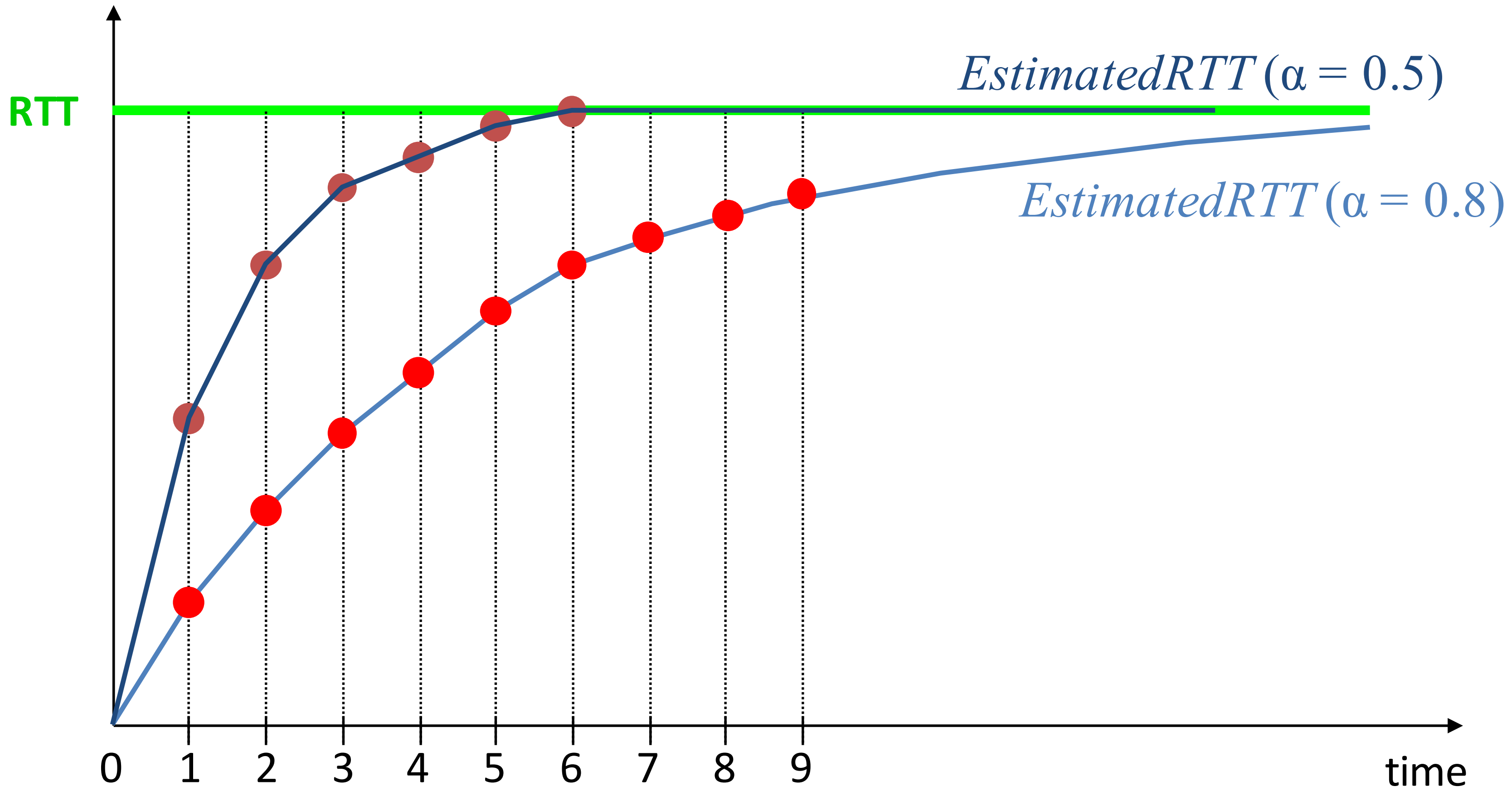
$$SampleRTT = AckRcvdTime - SendPacketTime$$
$$EstimatedRTT = \alpha \times EstimatedRTT + (1-\alpha) \times SampleRTT$$
$$0 < \alpha \leq 1$$



Exponential Averaging Example

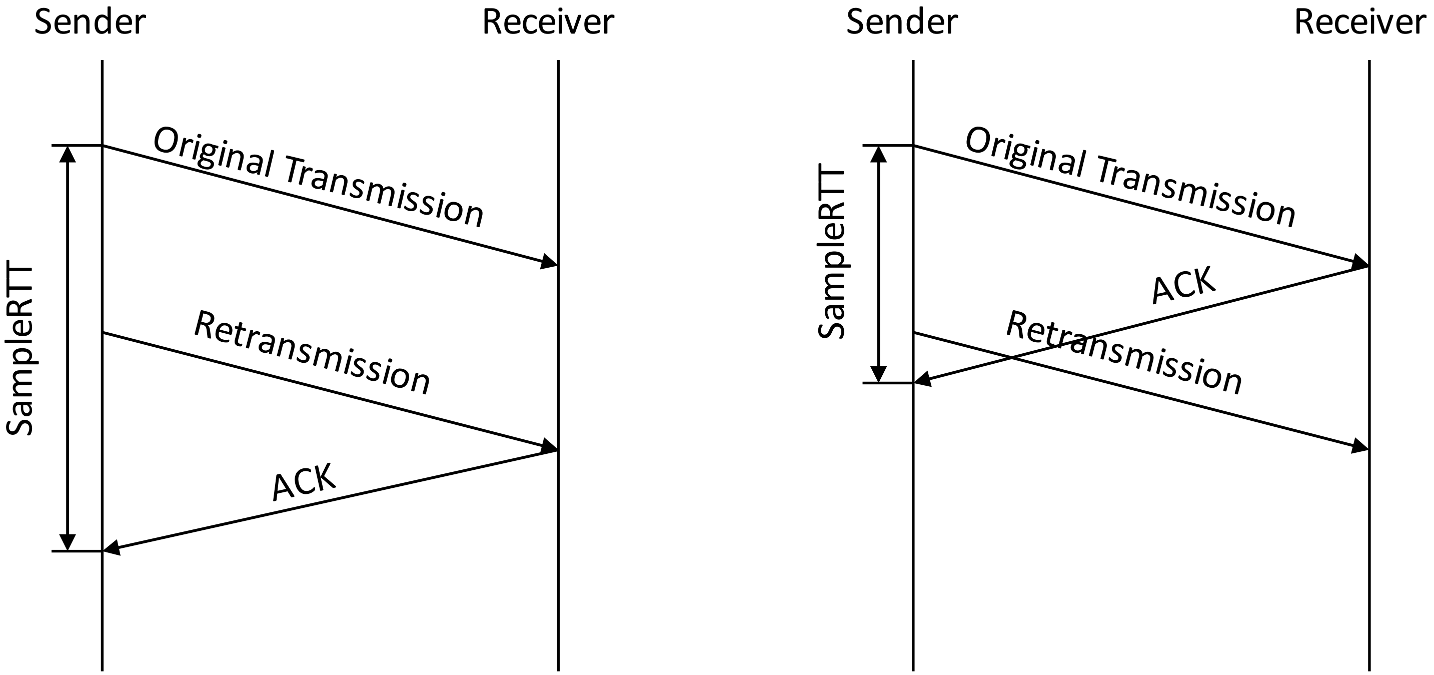
EstimatedRTT = *α*EstimatedRTT* + (1 – *α*)**SampleRTT*

Assume RTT is constant → *SampleRTT* = RTT



Problem: Ambiguous Measurements

- How do we differentiate between the real ACK, and ACK of the retransmitted packet?



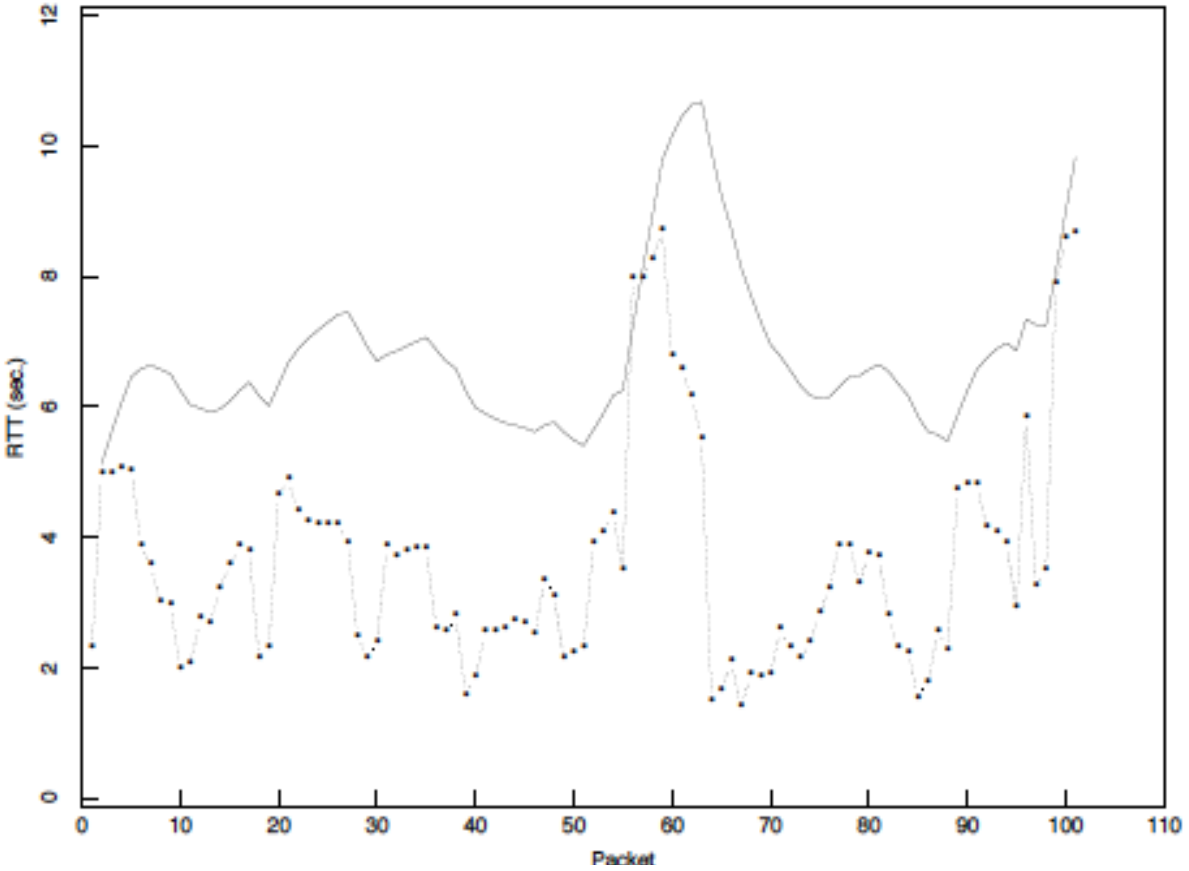
Karn/Partridge Algorithm

- Measure *SampleRTT* only for original transmissions
  - Once a segment has been retransmitted, do not use it for any further measurements
- Computes EstimatedRTT using *α = 0.875*

- Timeout value (RTO) = 2 × EstimatedRTT
- Employs exponential backoff
  - Every time RTO timer expires, set RTO ← 2·RTO
  - (Up to maximum ≥ 60 sec)
  - Every time new measurement comes in (= successful original transmission), collapse RTO back to 2 × EstimatedRTT

Karn/Partridge in action

Figure 5: Performance of an RFC793 retransmit timer



from Jacobson and Karels, SIGCOMM 1988

Jacobson/Karels Algorithm

- Problem: need to better capture variability in RTT
  - Directly measure deviation

- Deviation = | SampleRTT – EstimatedRTT |
- EstimatedDeviation: exponential average of Deviation

- RTO = EstimatedRTT + 4 x EstimatedDeviation

With Jacobson/Karels

Figure 5: Performance of an RFC793 retransmit timer

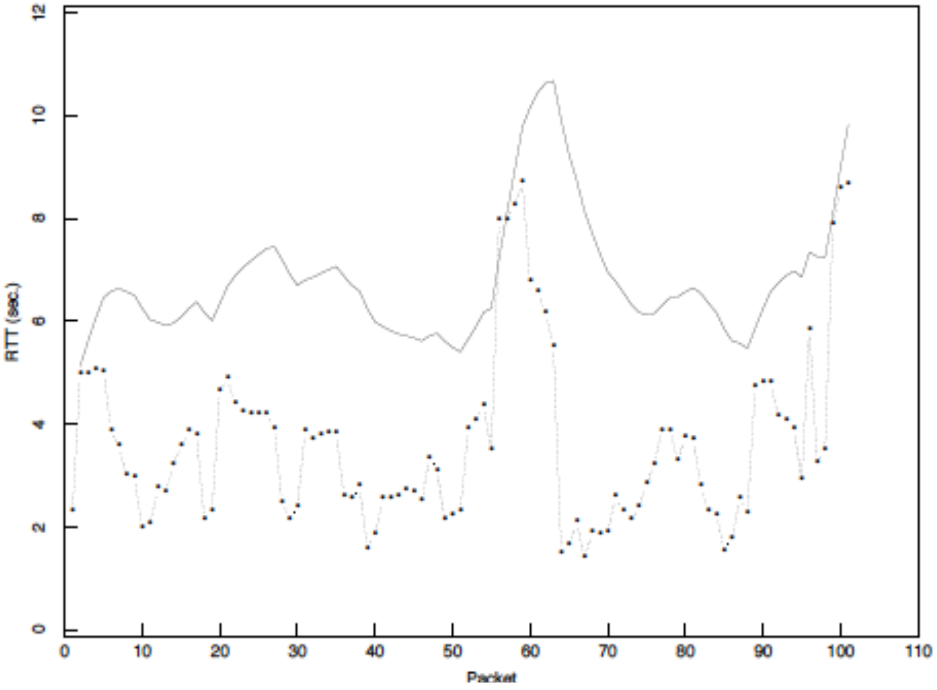
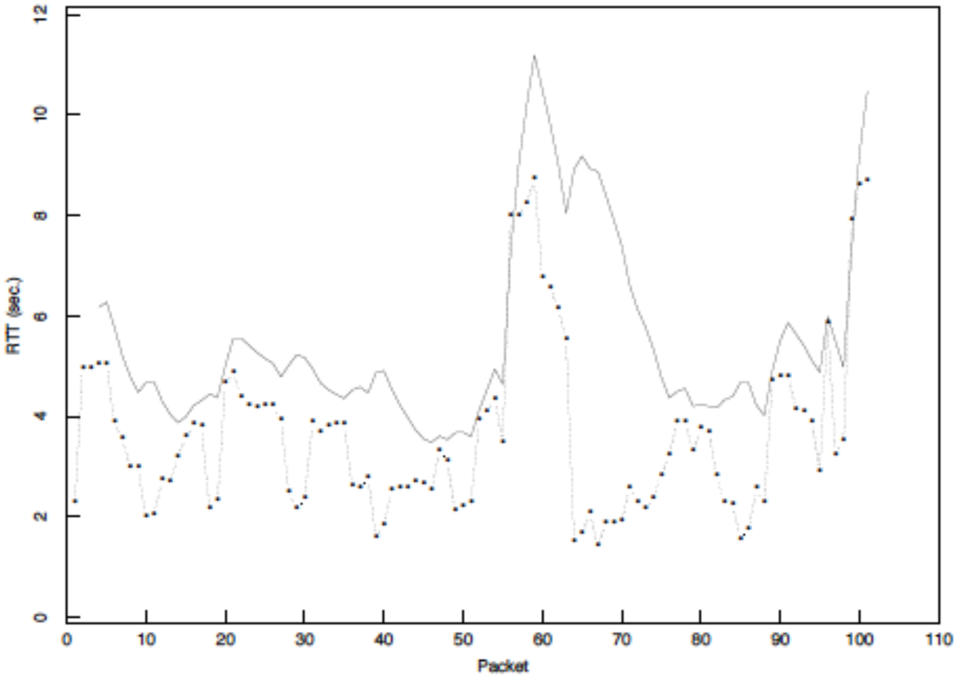


Figure 6: Performance of a Mean+Variance retransmit timer



What does TCP do?

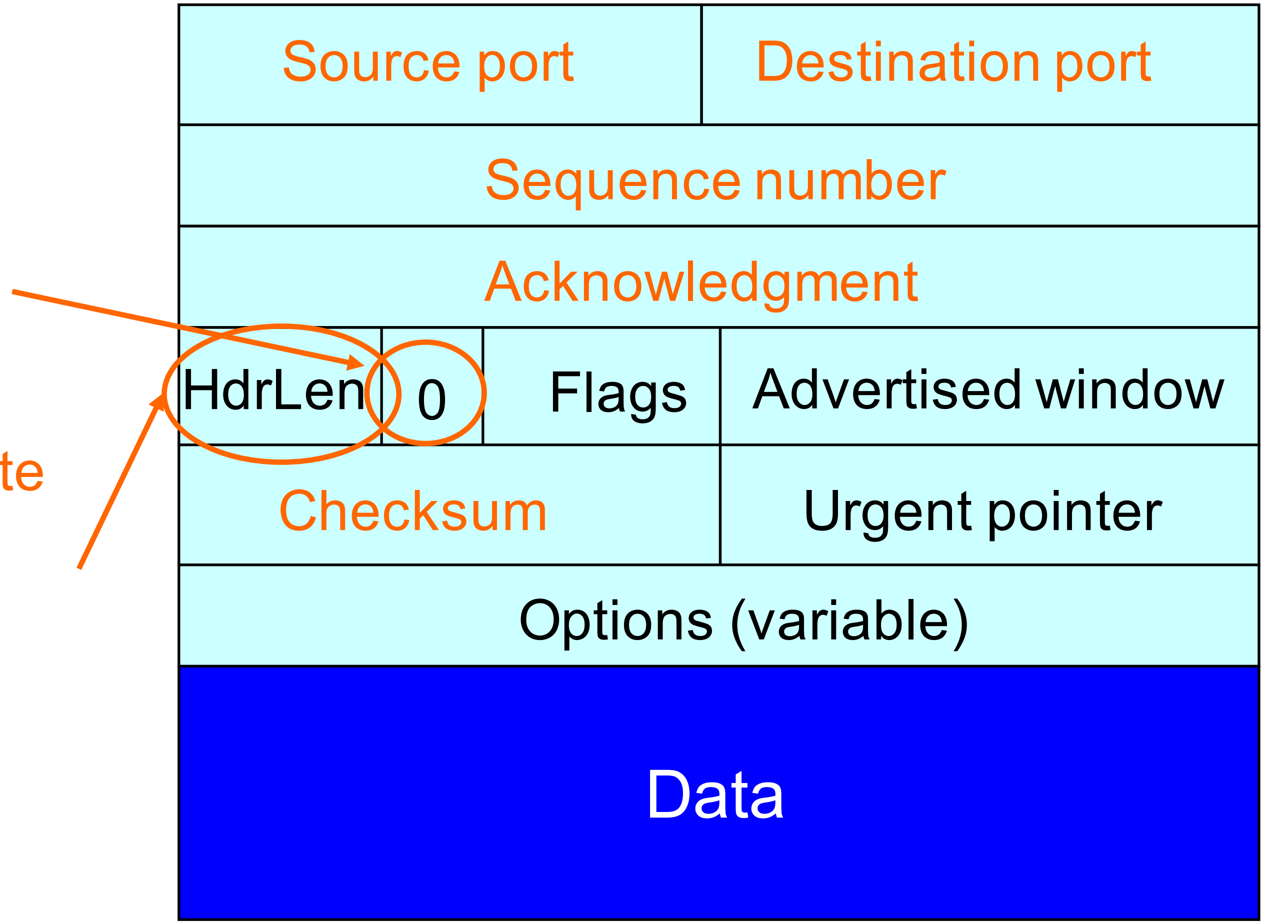
Most of our previous ideas, but some key differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers do not drop out-of-sequence packets (like SR)
- Introduces fast retransmit: optimization that uses duplicate ACKs to trigger early retransmission
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout

TCP Header: What’s left?

“Must Be Zero”
6 bits reserved

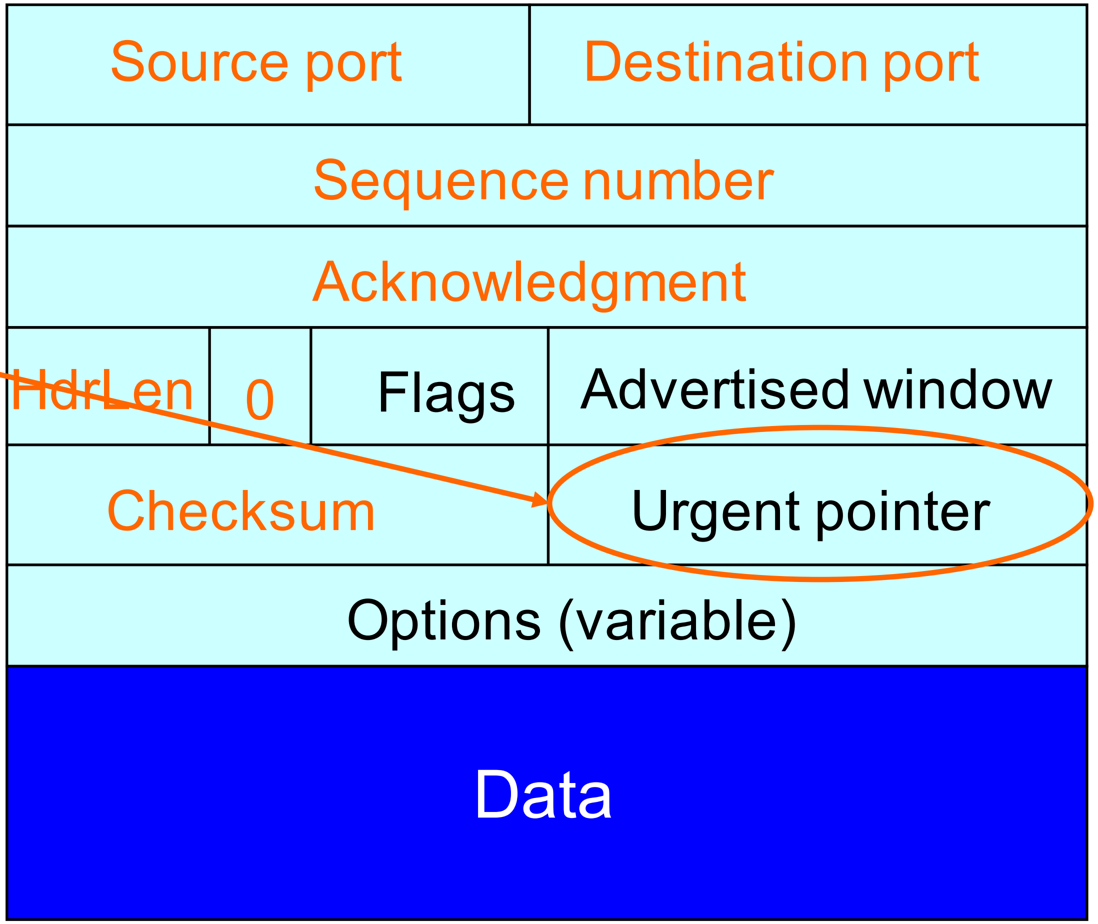
Number of 4-byte words in TCP header;
5 = no options



Source port			Destination port
Sequence number			
Acknowledgment			
HdrLen	0	Flags	Advertised window
Checksum			Urgent pointer
Options (variable)			
Data			

TCP Header: What's left?

Used with **URG** flag to indicate urgent data (not discussed further)



Source port

Destination port

Sequence number

Acknowledgment

HdrLen

0

Flags

Advertised window

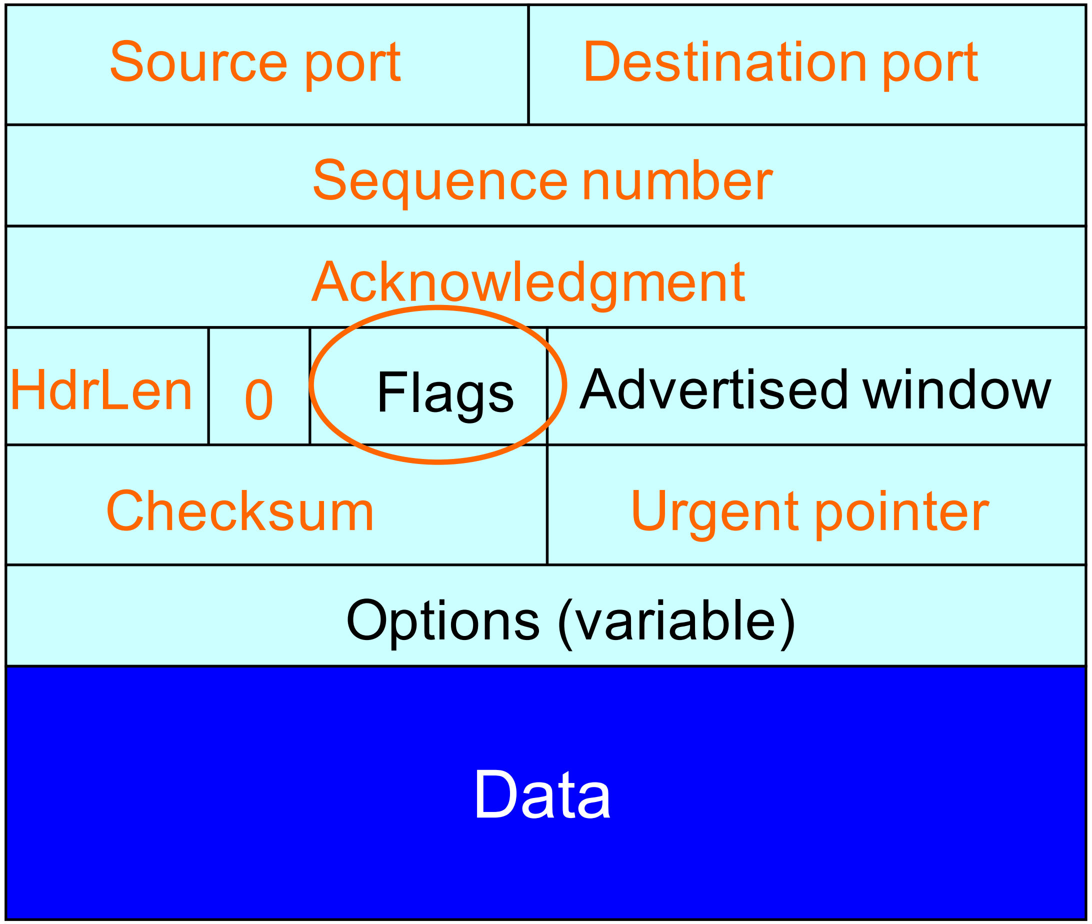
Checksum

Urgent pointer

Options (variable)

Data

TCP Header: What's left?



Source port

Destination port

Sequence number

Acknowledgment

HdrLen

0

Flags

Advertised window

Checksum

Urgent pointer

Options (variable)

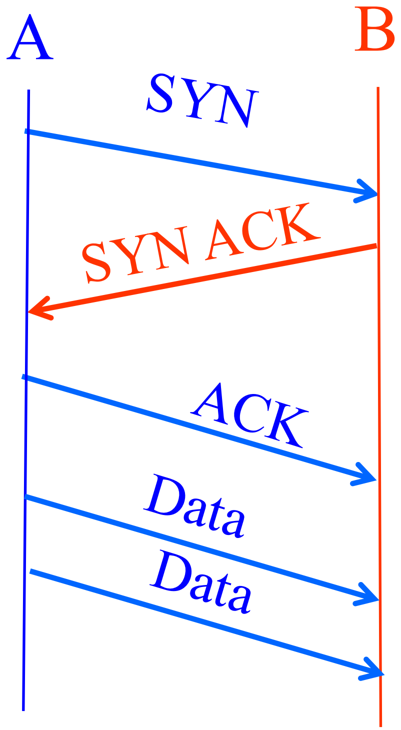
Data

TCP Connection Establishment and Initial Sequence Numbers

Initial Sequence Number (ISN)

- Sequence number for the very first byte
- Why not just use ISN = 0?
- Practical issue
  - IP addresses and port #s uniquely identify a connection
  - Eventually, though, these port #s do get used again
  - … small chance an old packet is still in flight
- TCP therefore requires changing ISN
- Hosts exchange ISNs when they establish a connection

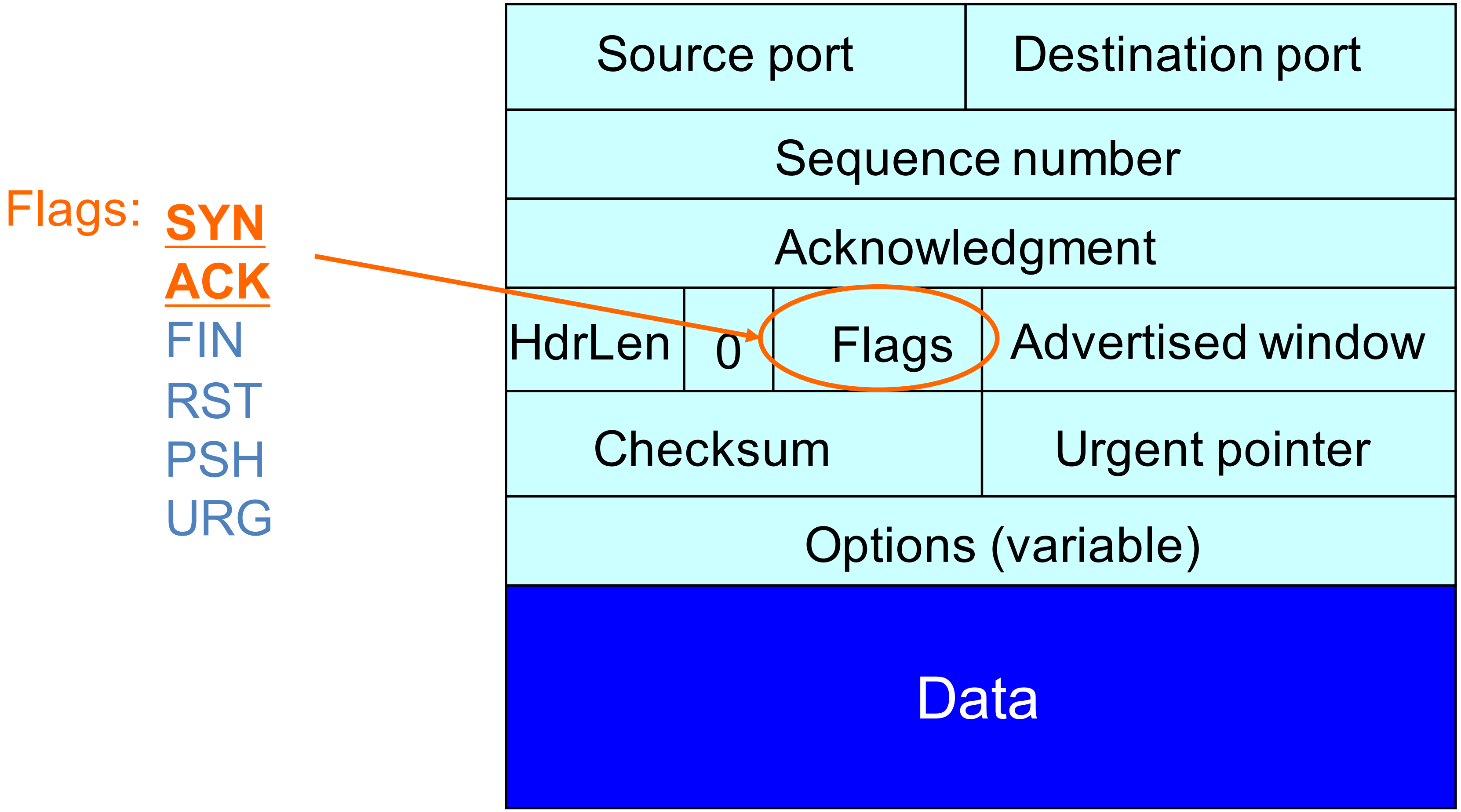
Establishing a TCP Connection



Each host tells its ISN to the other host.

- Three-way handshake to establish connection
  - Host A sends a **SYN** (open; "synchronize sequence numbers") to host B
  - Host B returns a SYN acknowledgment (**SYN ACK**)
  - Host A sends an **ACK** to acknowledge the SYN ACK

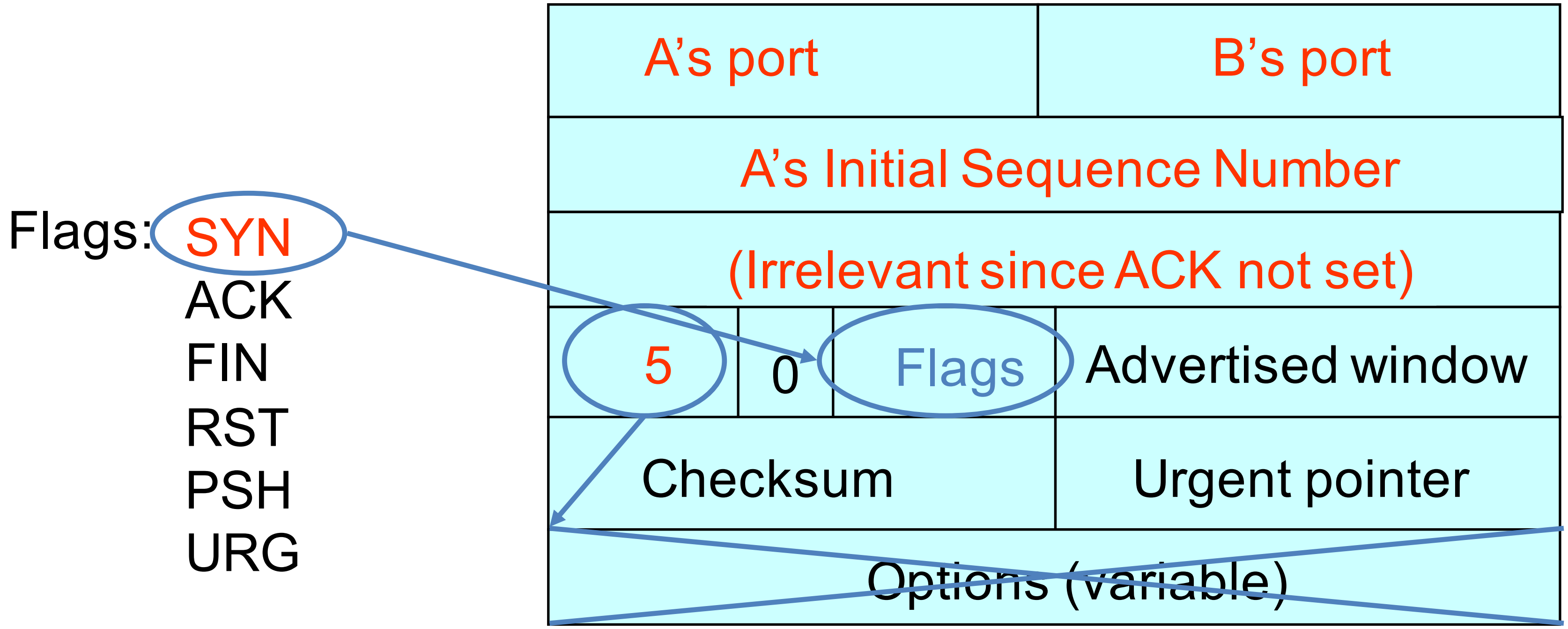
TCP Header



Flags: **SYN**
ACK
FIN
RST
PSH
URG

Source port			Destination port
Sequence number			
Acknowledgment			
HdrLen	0	Flags	Advertised window
Checksum			Urgent pointer
Options (variable)			
Data			

Step 1: A’ s Initial SYN Packet



Flags: SYN
ACK
FIN
RST
PSH
URG

A’s port | B’s port

A’s Initial Sequence Number

(Irrelevant since ACK not set)

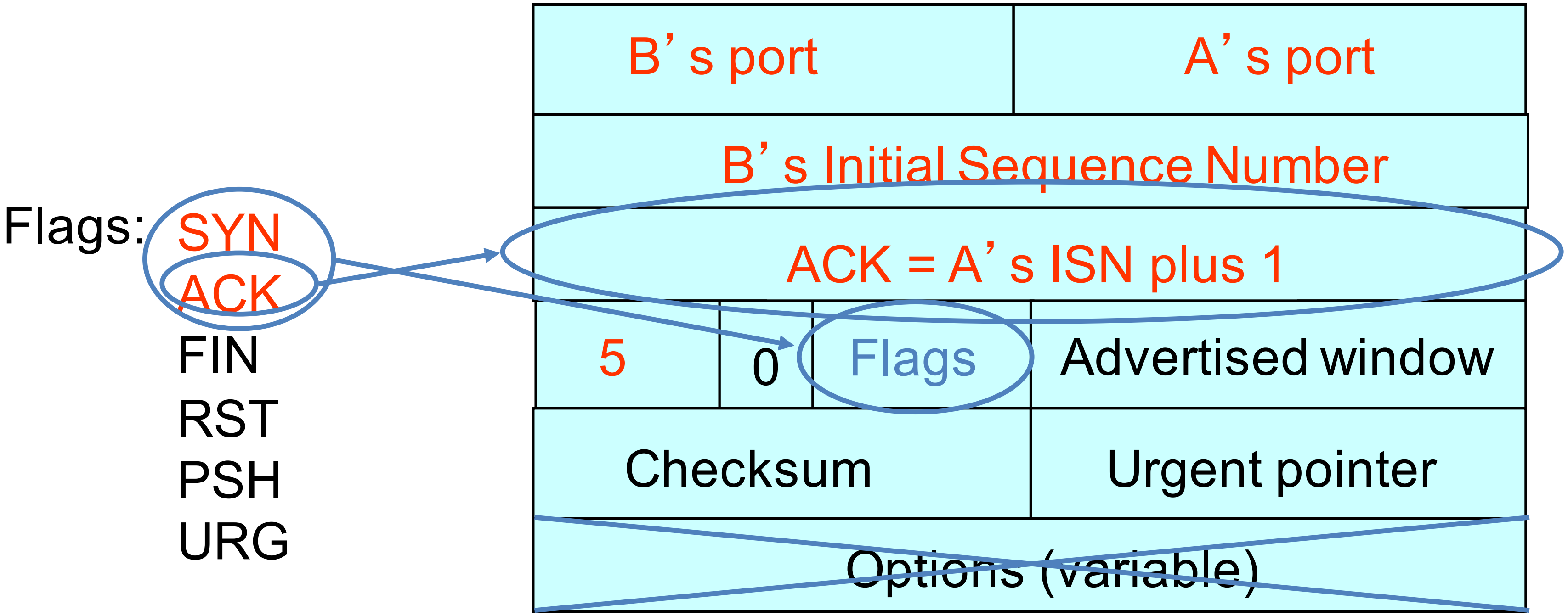
5 | 0 | Flags | Advertised window

Checksum | Urgent pointer

~~Options (variable)~~

A tells B it wants to open a connection…

Step 2: B’s SYN-ACK Packet



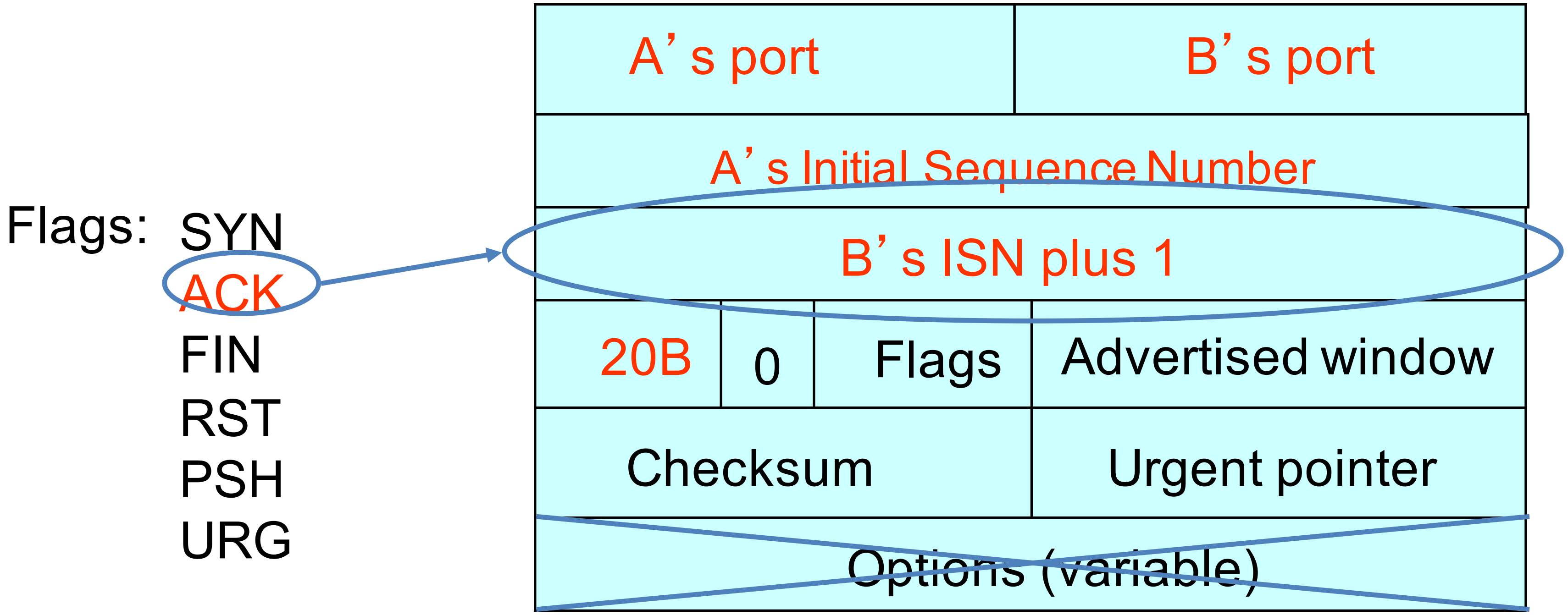
Flags: SYN
ACK
FIN
RST
PSH
URG

B’s port		A’s port	
B’s Initial Sequence Number			
ACK = A’s ISN plus 1			
5	0	Flags	Advertised window
Checksum		Urgent pointer	
~~Options (variable)~~			

B tells A it accepts, and is ready to hear the next byte…

… upon receiving this packet, A can start sending data

Step 3: A’s ACK of the SYN-ACK



Flags: SYN
ACK
FIN
RST
PSH
URG

A’s port

B’s port

A’s Initial Sequence Number

B’s ISN plus 1

20B

0

Flags

Advertised window

Checksum

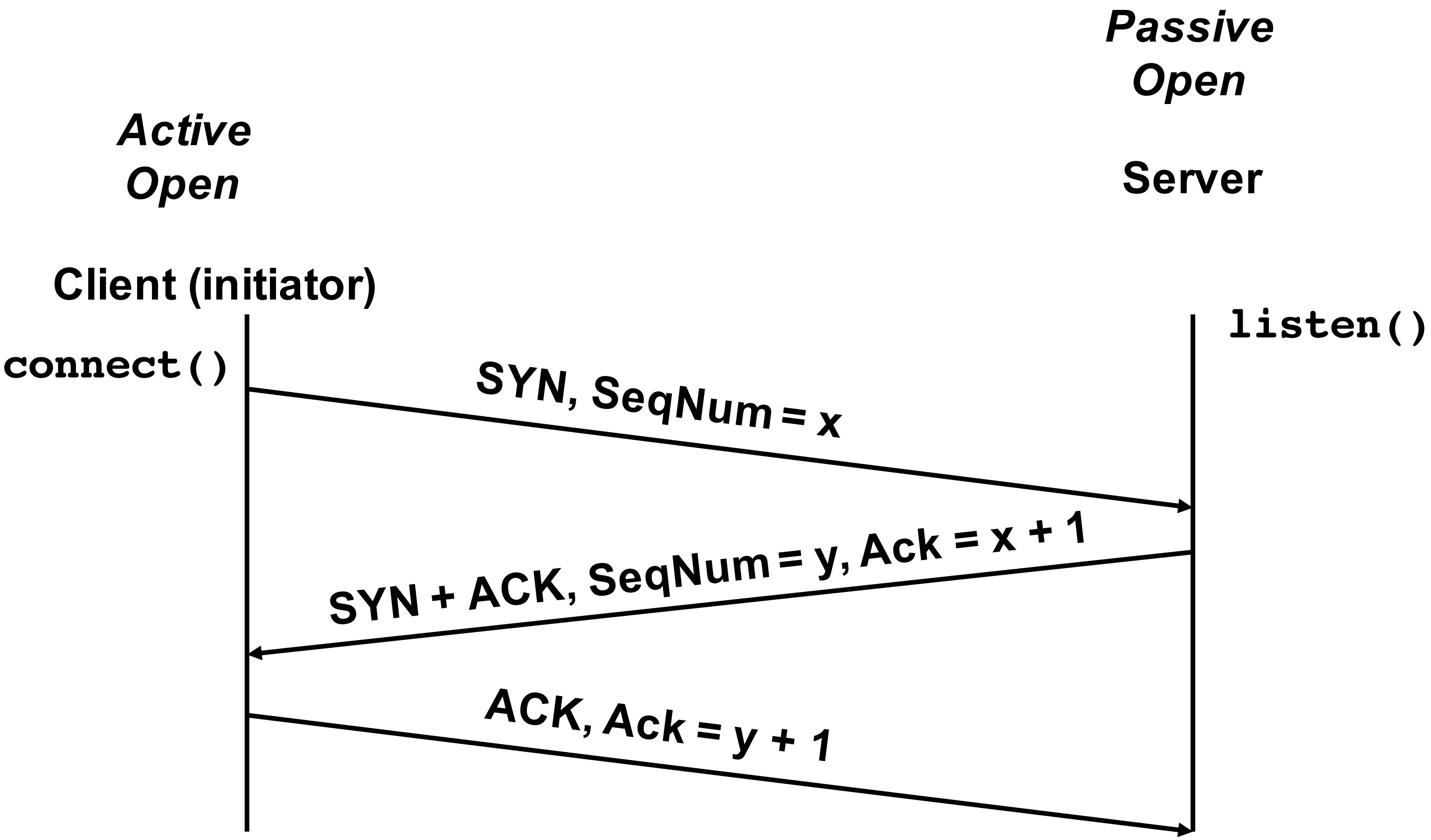
Urgent pointer

~~Options (variable)~~

A tells B it’s likewise okay to start sending

… upon receiving this packet, B can start sending data

Timing Diagram: 3-Way Handshaking



What if the SYN Packet Gets Lost?

- Suppose the SYN packet gets lost
  - Packet is lost inside the network, or:
  - Server discards the packet (e.g., it's too busy)

- Eventually, no SYN-ACK arrives
  - Sender sets a timer and waits for the SYN-ACK
  - … and retransmits the SYN if needed

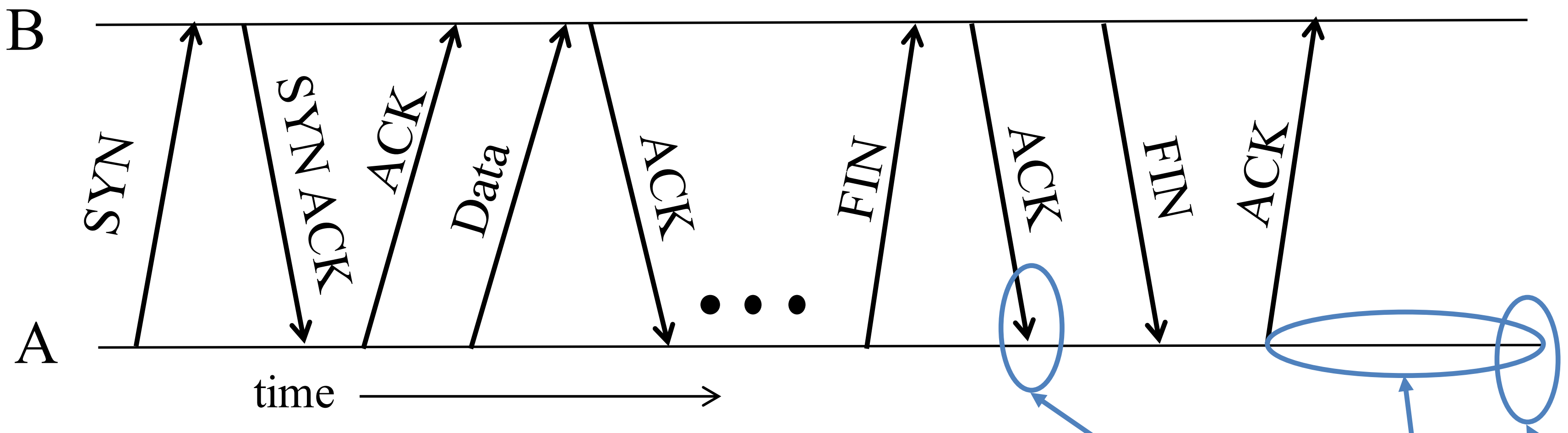
- How should the TCP sender set the timer?
  - Sender has no idea how far away the receiver is
  - Hard to guess a reasonable length of time to wait
  - **SHOULD** (RFCs 1122 & 2988) use default of 3 seconds
    - Some implementations instead use 6 seconds

SYN Loss and Web Downloads

- User clicks on a hypertext link
  - Browser creates a socket and does a “connect”
  - The “connect” triggers the OS to transmit a SYN
- If the SYN is lost…
  - 3-6 seconds of delay: can be very long
  - User may become impatient
  - … and click the hyperlink again, or click “reload”
- User triggers an “abort” of the “connect”
  - Browser creates a new socket and another “connect”
  - Essentially, forces a faster send of a new SYN packet!
  - Sometimes very effective, and the page comes quickly

Tearing Down the Connection

Normal Termination, One Side At A Time



- Finish (**FIN**) to close and receive remaining bytes
  - **FIN** occupies one byte in the sequence space
- Other host acks the byte to confirm
- Closes A’s side of the connection, but not B’s
  - Until B likewise sends a **FIN**
  - Which A then acks

Connection now **closed**

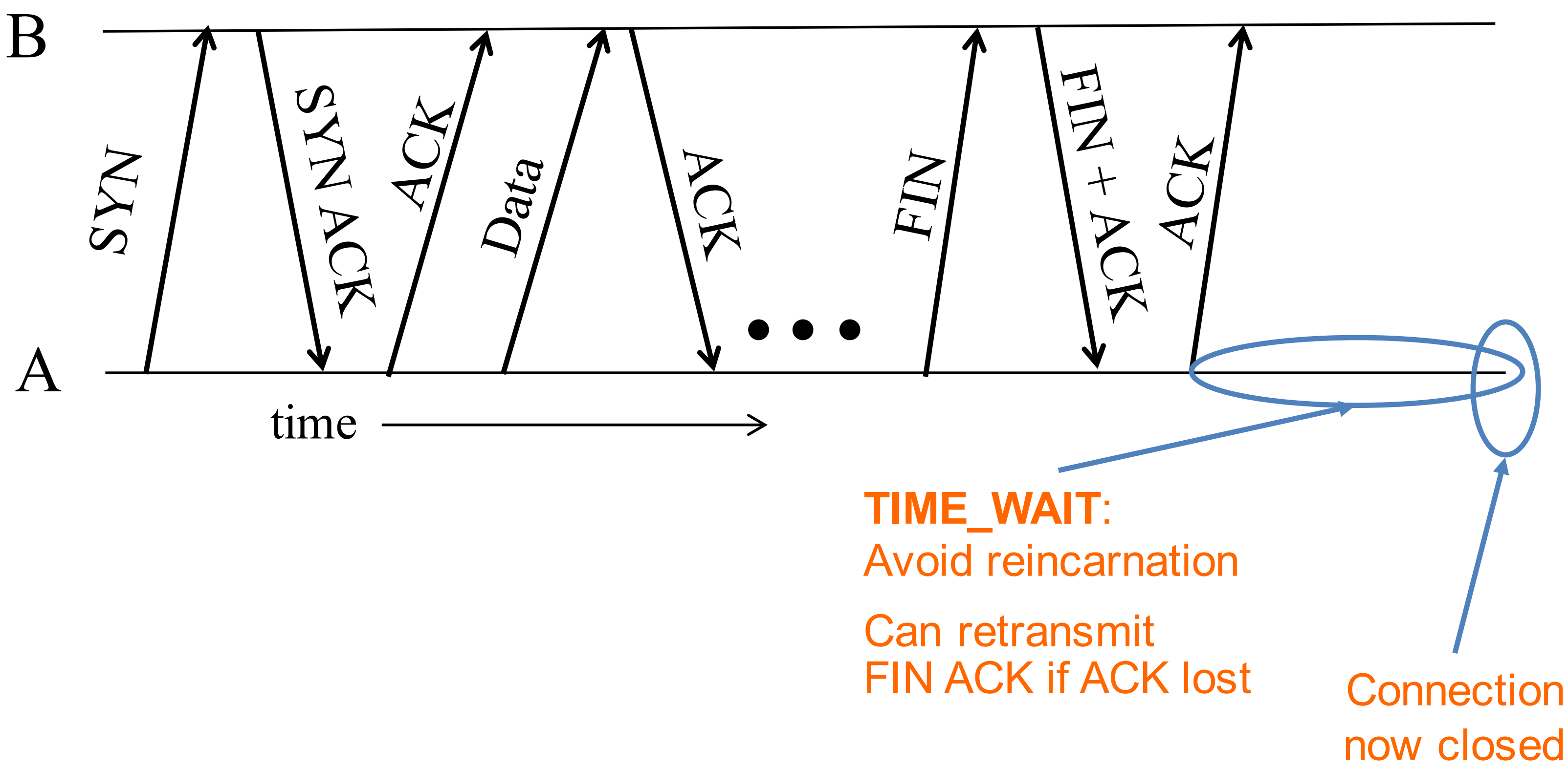
Connection now **half-closed**

TIME_WAIT:

Avoid reincarnation

B will retransmit FIN if ACK is lost

Normal Termination, Both Together



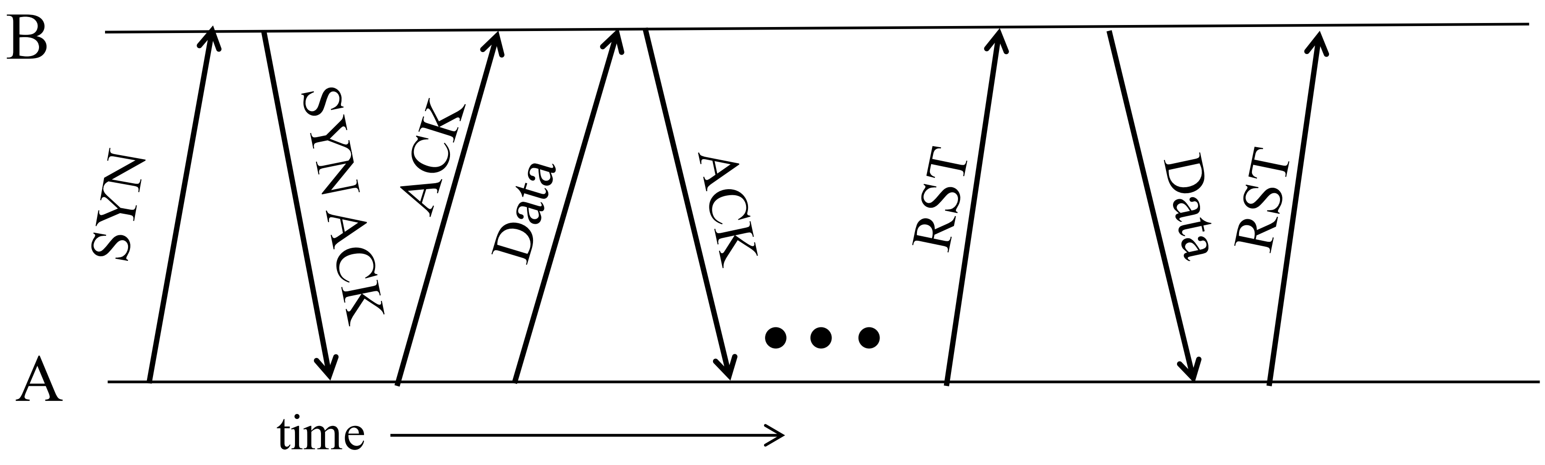
TIME_WAIT:
Avoid reincarnation

Can retransmit
FIN ACK if ACK lost

Connection
now closed

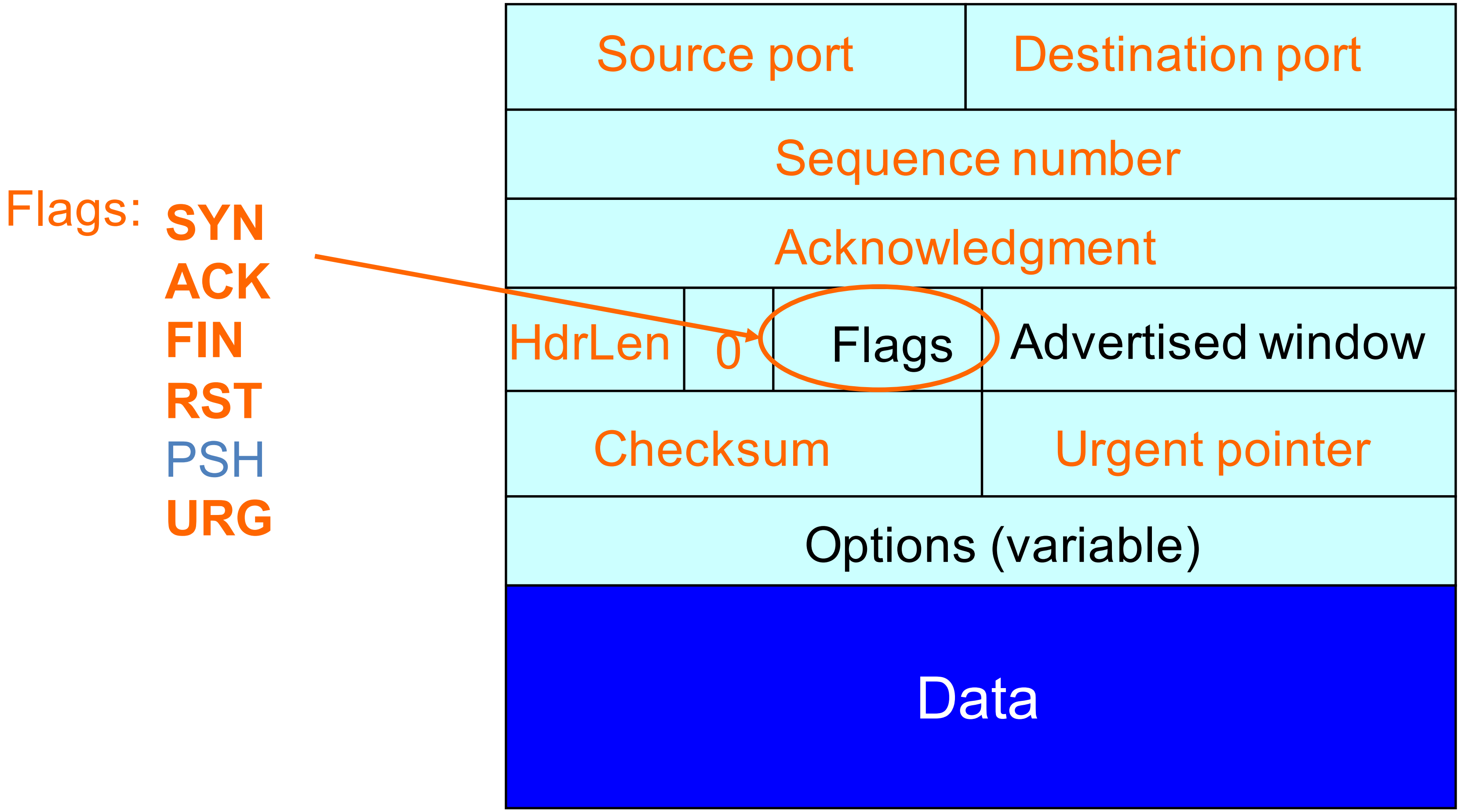
- Same as before, but B sets **FIN** with their ack of A's **FIN**

Abrupt Termination



- A sends a RESET (**RST**) to B
  - E.g., because application process on A crashed
- That's it
  - B does not ack the **RST**
  - Thus, **RST** is not delivered reliably
  - And: any data in flight is lost
  - But: if B sends anything more, will elicit another **RST**

TCP Header



Flags: SYN
ACK
FIN
RST
PSH
URG

Source port | Destination port

Sequence number

Acknowledgment

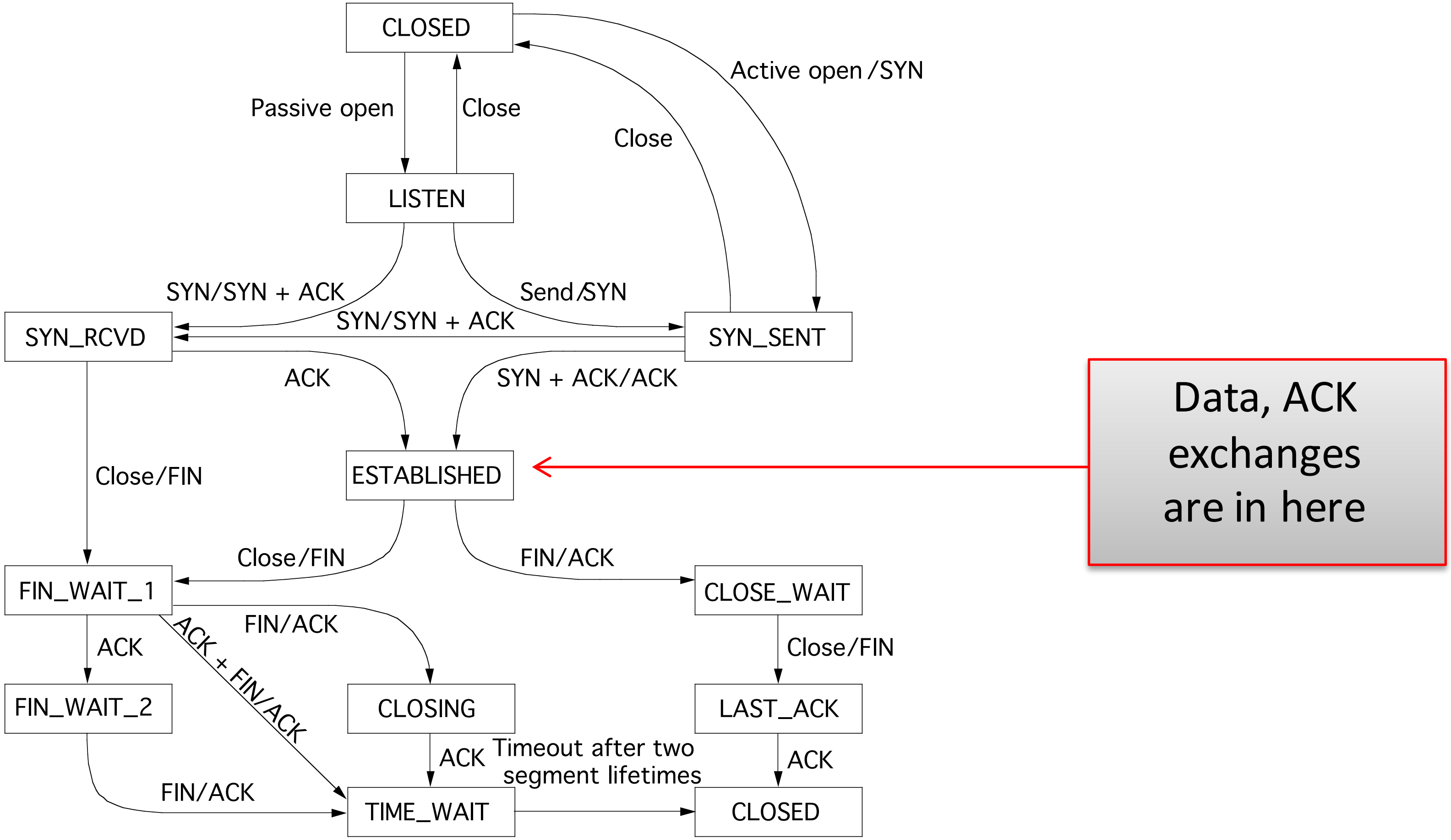
HdrLen | 0 | Flags | Advertised window

Checksum | Urgent pointer

Options (variable)

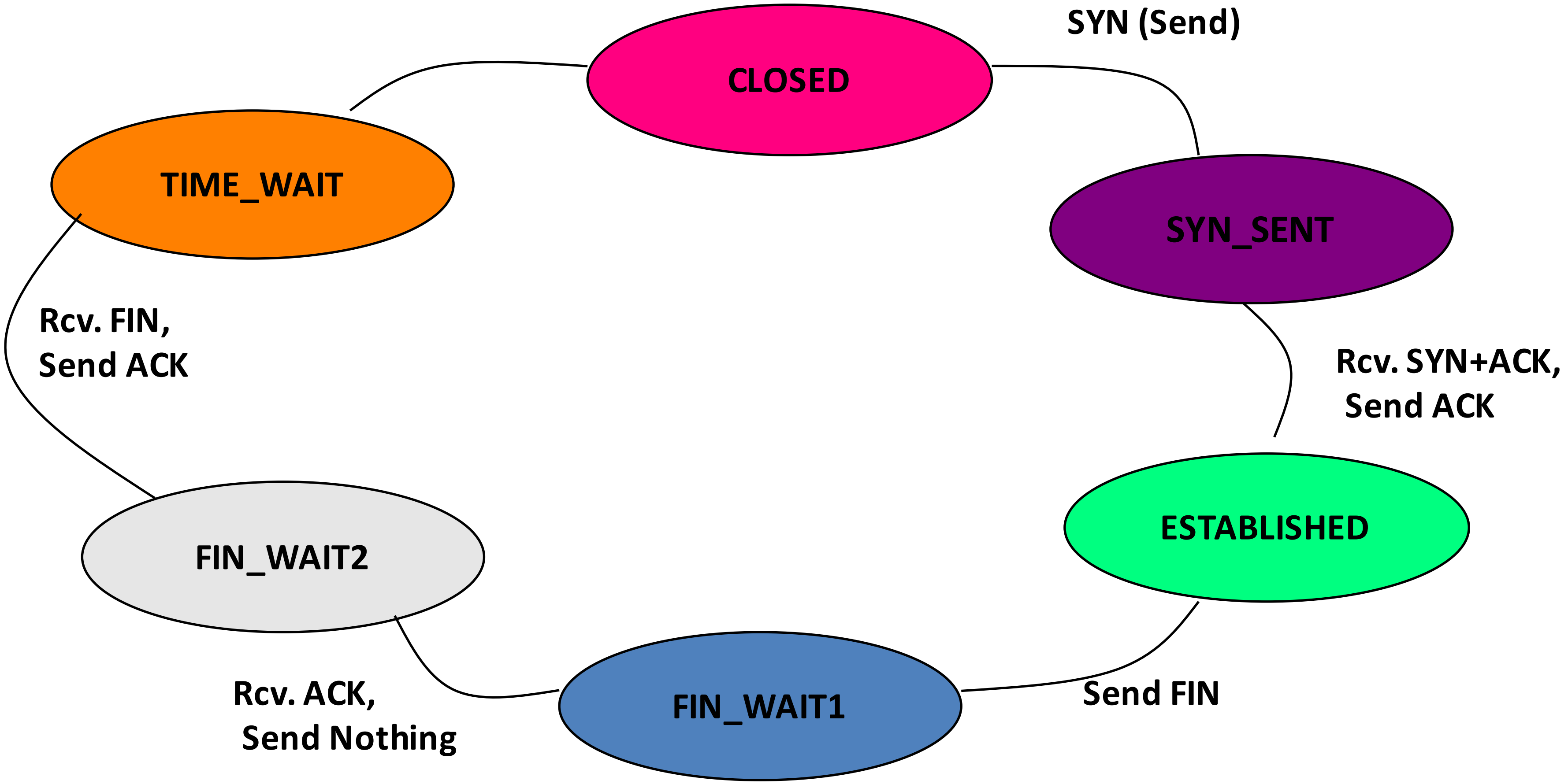
Data

TCP State Transitions

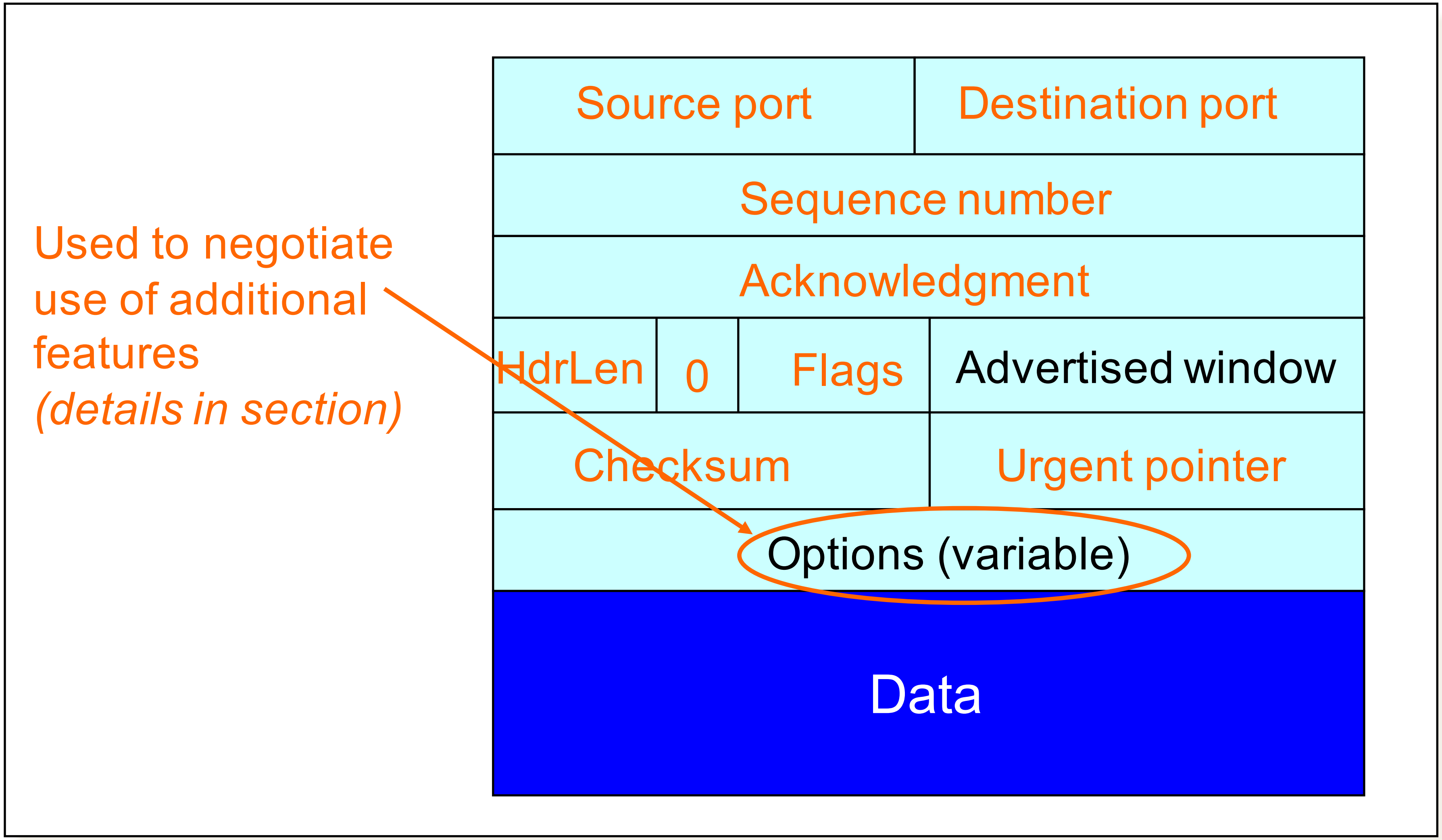


Data, ACK exchanges are in here

An Simpler View of the Client Side

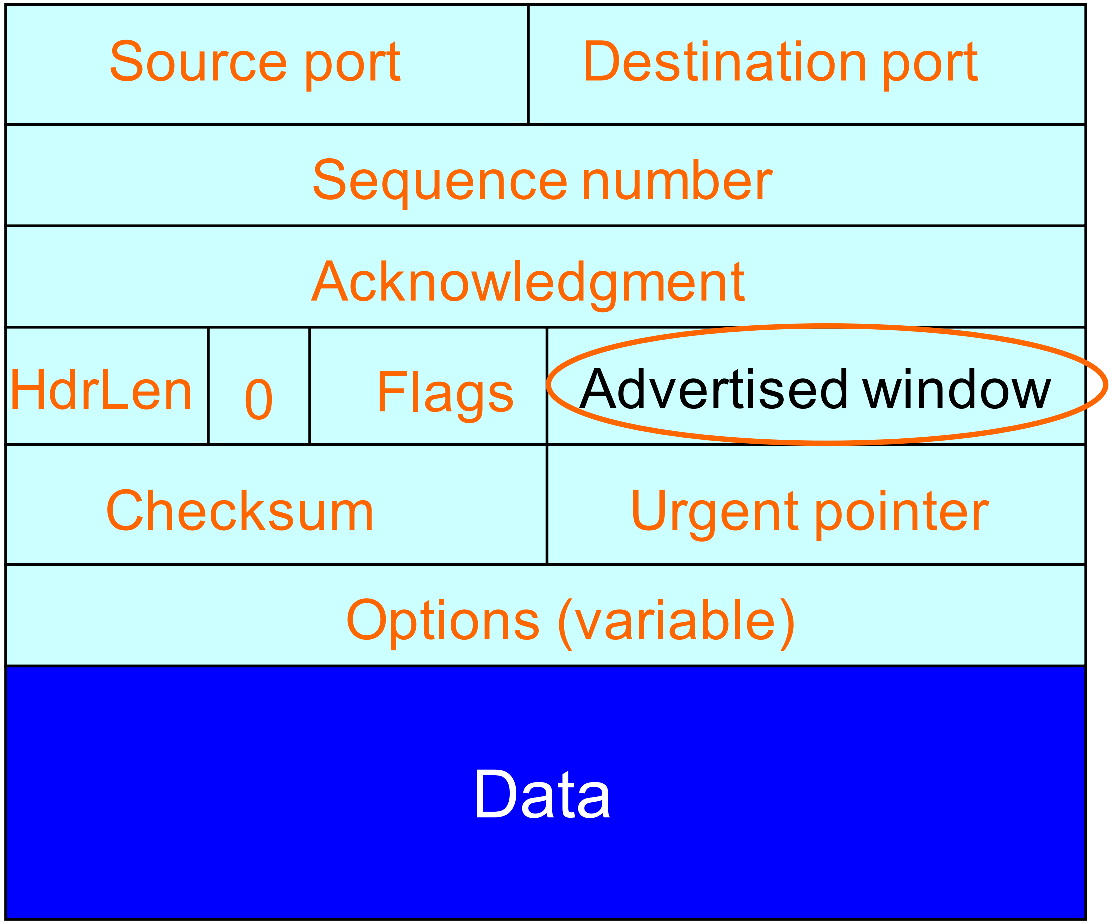


TCP Header



Used to negotiate use of additional features *(details in section)*

TCP Header



Source port

Destination port

Sequence number

Acknowledgment

HdrLen

0

Flags

Advertised window

Checksum

Urgent pointer

Options (variable)

Data

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP

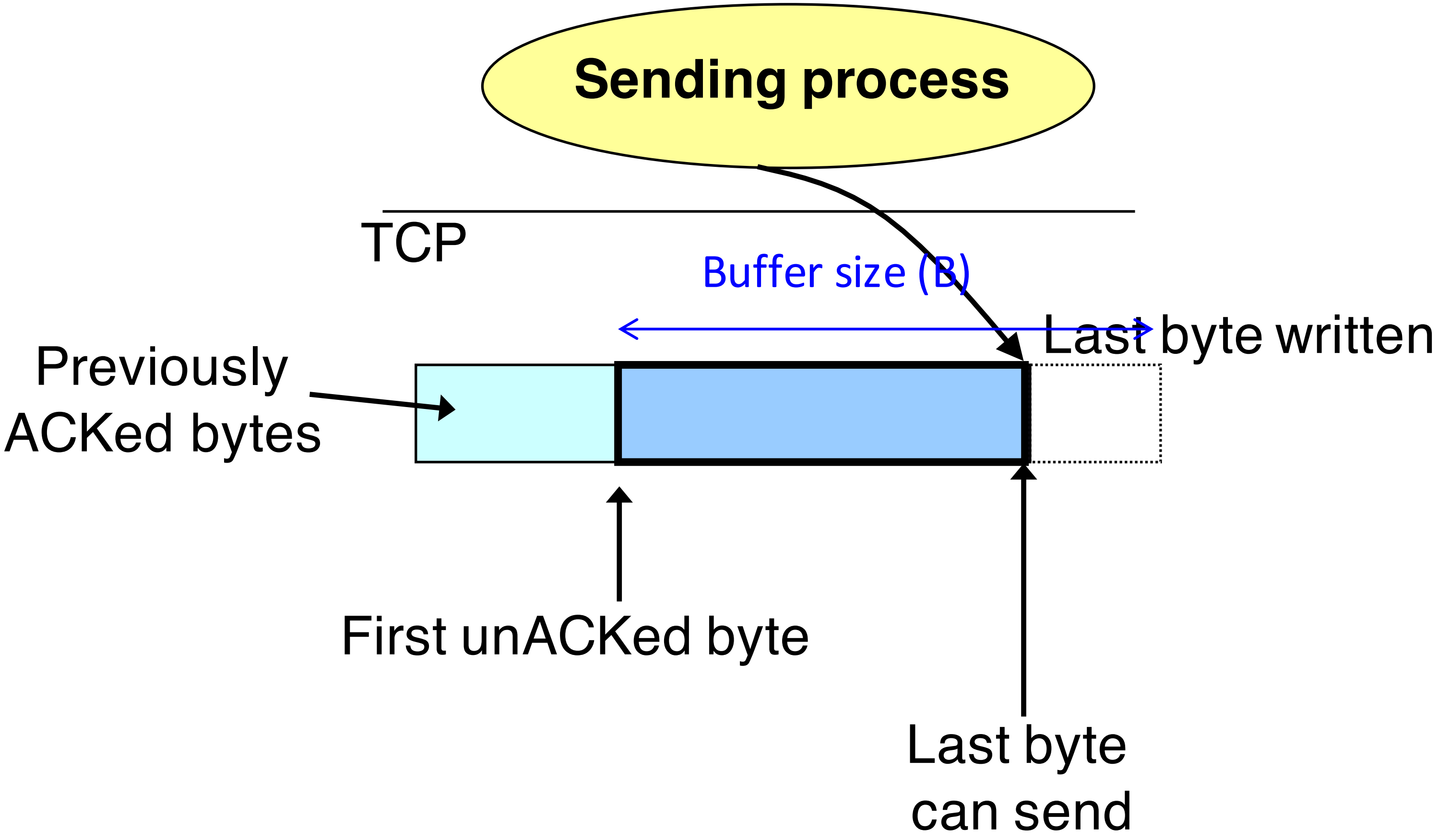
Recap: Sliding Window (so far)

- Both sender & receiver maintain a **window**

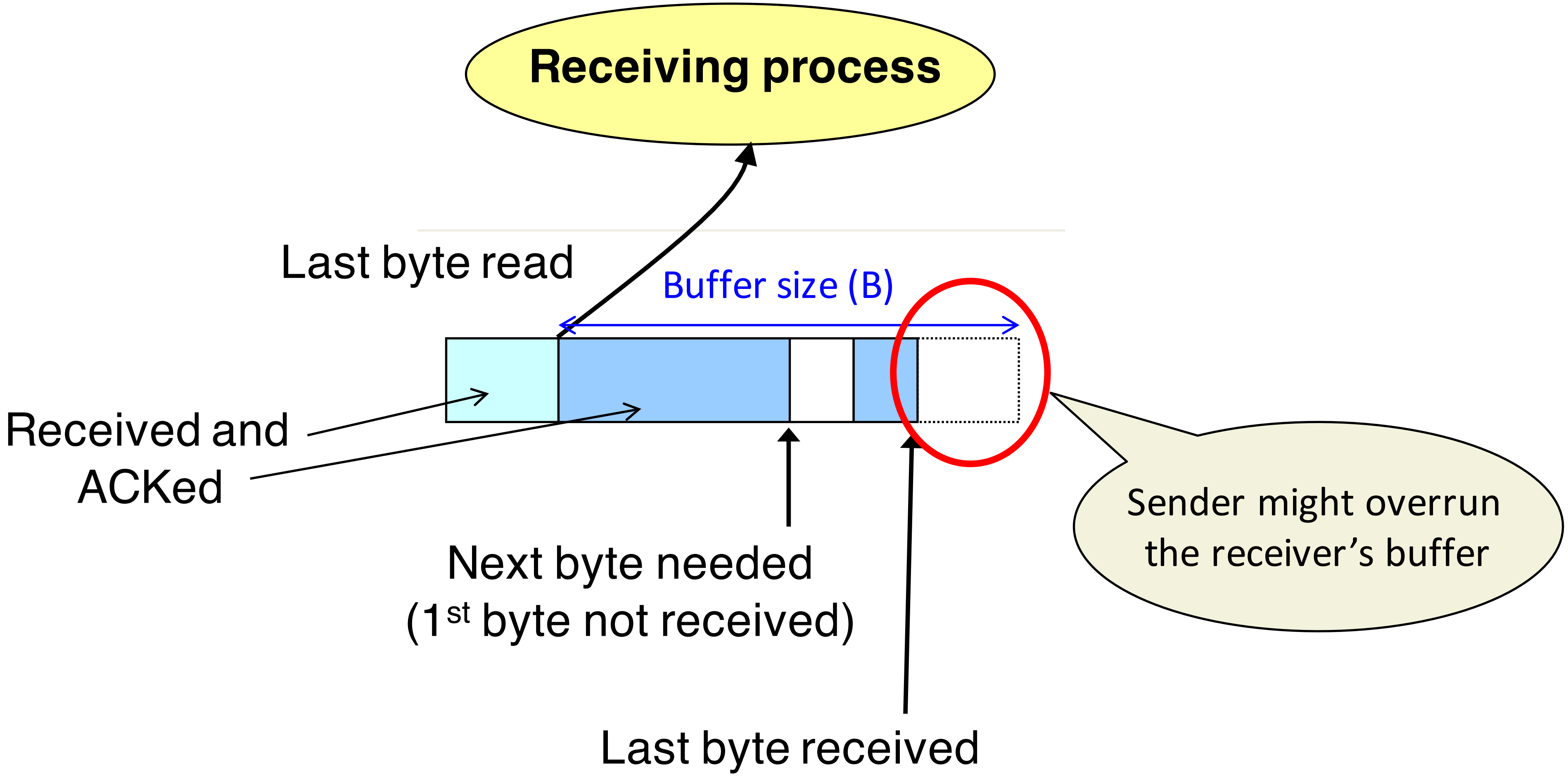
- Left edge of window:
  - Sender: beginning of unacknowledged data
  - Receiver: beginning of undelivered data

- Right edge: Left edge + *constant*
  - constant only limited by buffer size in the transport layer

Sliding Window at Sender (so far)



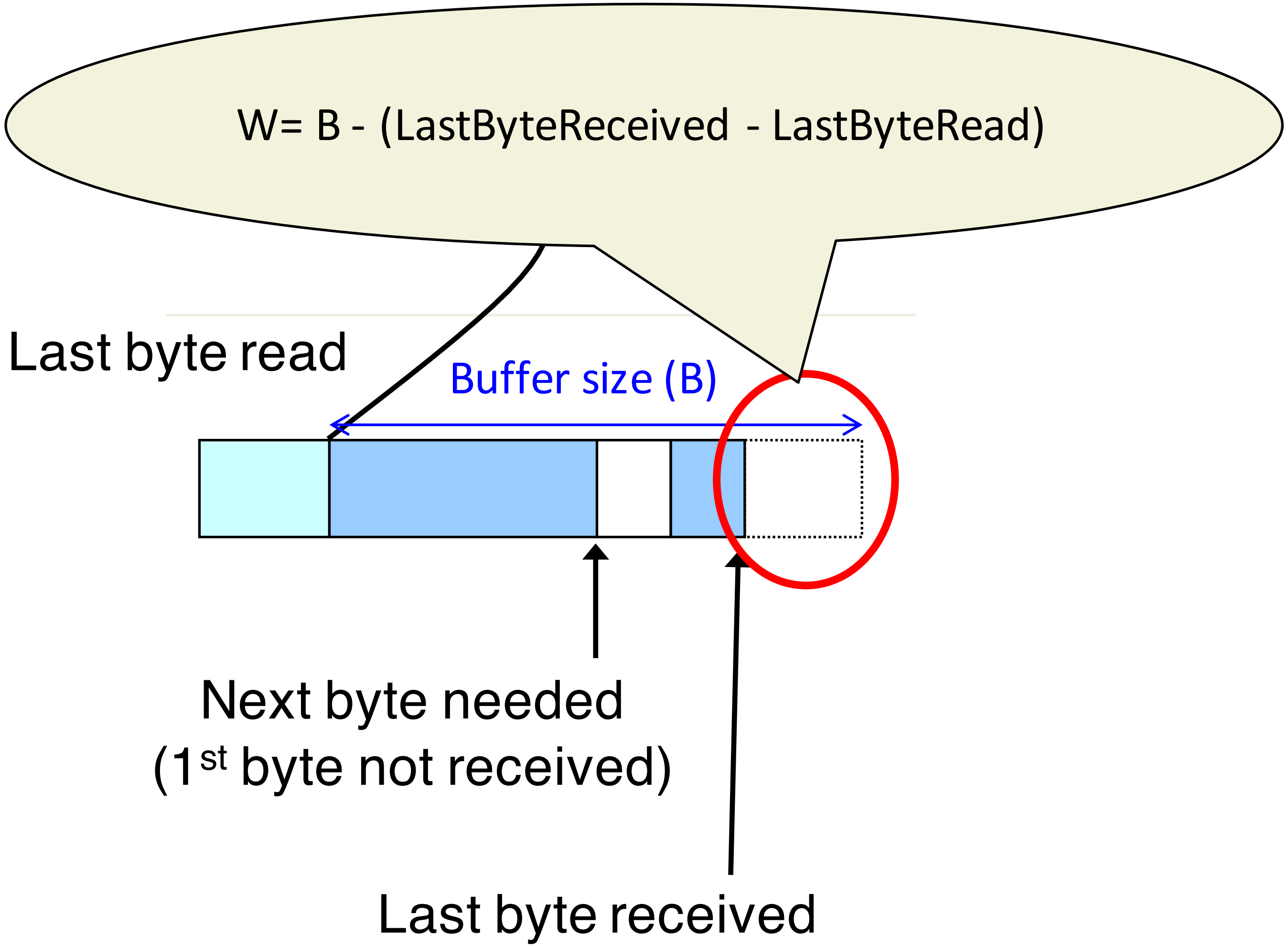
Sliding Window at Receiver (so far)


Receiving process
Last byte read
Buffer size (B)
Received and ACKed
Next byte needed (1st byte not received)
Last byte received
Sender might overrun the receiver's buffer

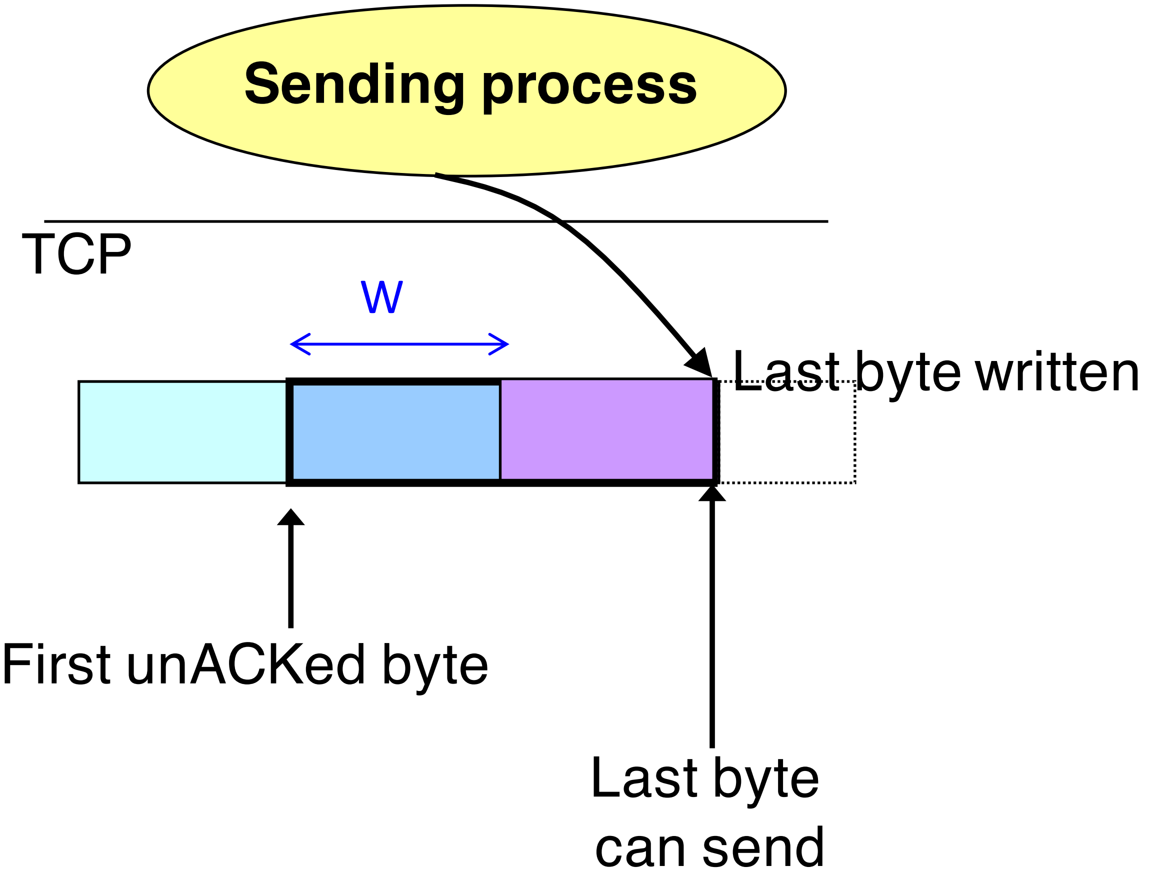
Solution: Advertised Window (Flow Control)

- Receiver uses an “Advertised Window” (W) to prevent sender from overflowing its window
  - Receiver indicates value of W in ACKs
  - Sender limits number of bytes it can have in flight <= W

Sliding Window at Receiver



Sliding Window at Sender (so far)



Sliding Window w/ Flow Control

- Sender: window advances when new data ack'd
- Receiver: window advances as receiving process consumes data
- Receiver advertises to the sender where the receiver window currently ends ("righthand edge")
  - Sender agrees not to exceed this amount

Advertised Window Limits Rate

- Sender can send no faster than W/RTT bytes/sec

- Receiver only advertises more space when it has consumed old arriving data

- In original TCP design, that was the **sole** protocol mechanism controlling sender's rate

- What's missing?

TCP

- The concepts underlying TCP are simple
  - acknowledgments (feedback)
  - timers
  - sliding windows
  - buffer management
  - sequence numbers

TCP

- The concepts underlying TCP are simple
- But tricky in the details
  - How do we set timers?
  - What is the seqno for an ACK-only packet?
  - What happens if advertised window = 0?
  - What if the advertised window is ½ an MSS?
  - Should receiver acknowledge packets right away?
  - What if the application generates data in units of 0.1 MSS?
  - What happens if I get a duplicate SYN? Or a RST while I'm in FIN_WAIT, *etc., etc., etc.*

TCP

- The concepts underlying TCP are simple
- But tricky in the details
- Do the details matter?

Sizing Windows for Congestion Control

- What are the problems?
- How might we address them?

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP

We have seen:

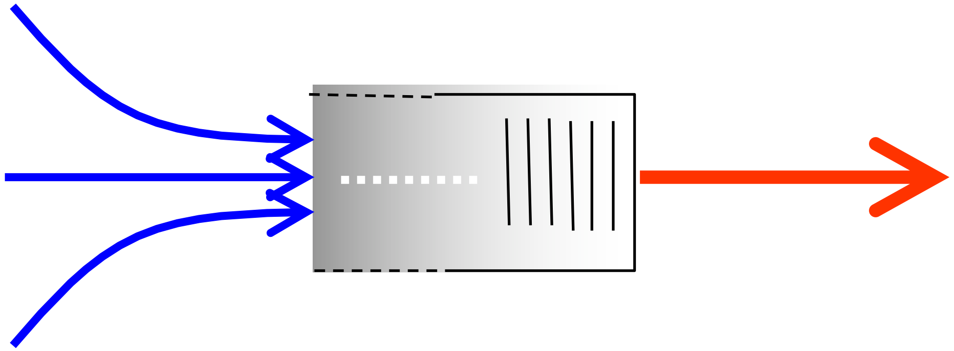
- Flow control: adjusting the sending rate to keep from overwhelming a slow *receiver*

Now lets attend…

- Congestion control: adjusting the sending rate to keep from overloading the *network*

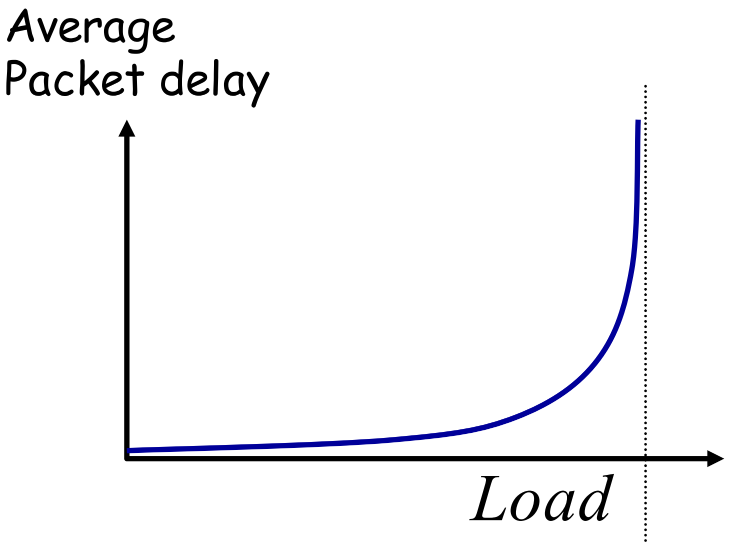
Statistical Multiplexing → Congestion

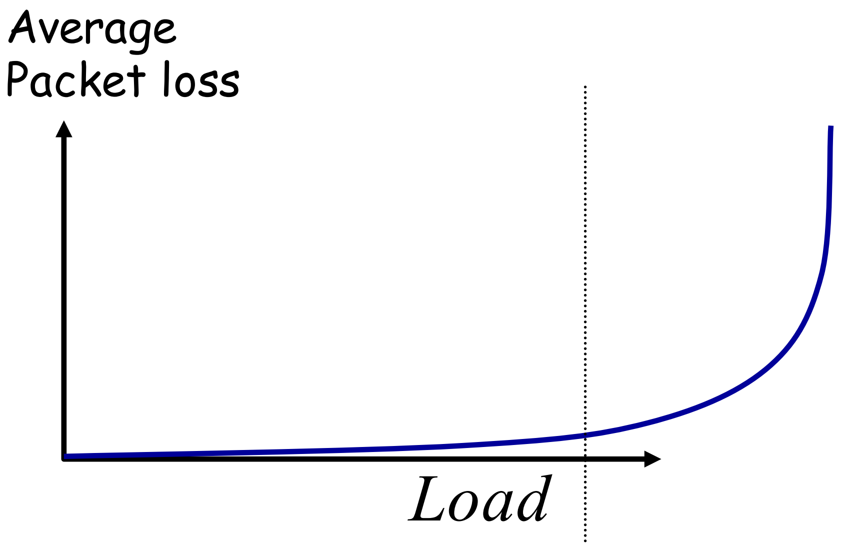
- If two packets arrive at the same time
  - A router can only transmit one
  - … and either buffers or drops the other
- If many packets arrive in a short period of time
  - The router cannot keep up with the arriving traffic
  - … delays traffic, and the buffer may eventually overflow
- Internet traffic is bursty



Congestion is undesirable

Typical queuing system with bursty arrivals





Must balance utilization versus delay and loss

Who Takes Care of Congestion?

- Network? End hosts? Both?

- TCP's approach:
  - **End hosts** adjust sending rate
  - Based on **implicit feedback** from network

- Not the only approach
  - A consequence of history rather than planning

Some History: TCP in the 1980s

- Sending rate only limited by flow control
  - Packet drops → senders (repeatedly!) retransmit a full window's worth of packets

- Led to "congestion collapse" starting Oct. 1986
  - Throughput on the NSF network dropped from 32Kbits/s to 40bits/sec

- "Fixed" by Van Jacobson's development of TCP's congestion control (CC) algorithms

Jacobson's Approach

- Extend TCP's existing window-based protocol but adapt the window size in response to congestion
  - required no upgrades to routers or applications!
  - patch of a few lines of code to TCP implementations
- A pragmatic and effective solution
  - but many other approaches exist
- Extensively improved on since
  - topic now sees less activity in ISP contexts
  - but is making a comeback in datacenter environments

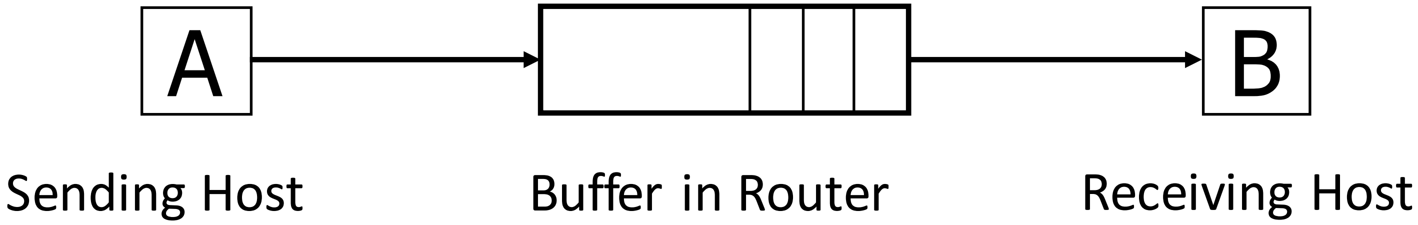
Three Issues to Consider

- Discovering the available (bottleneck) bandwidth

- Adjusting to variations in bandwidth

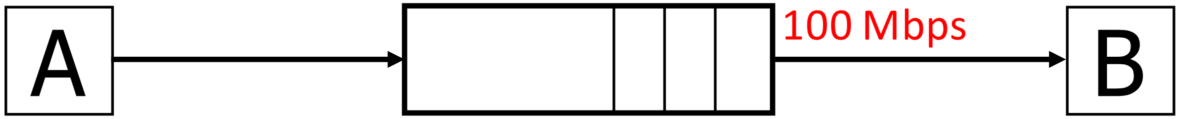
- Sharing bandwidth between flows

Abstract View



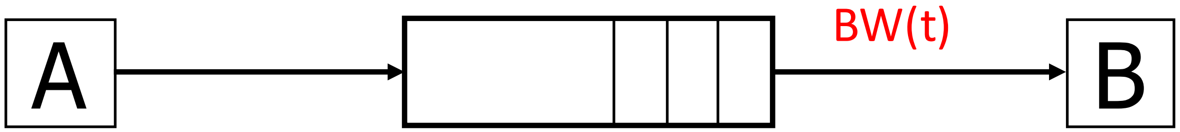
- Ignore internal structure of router and model it as having a single queue for a particular input-output pair

Discovering available bandwidth



- Pick sending rate to match bottleneck bandwidth
  - Without any *a priori* knowledge
  - Could be gigabit link, could be a modem

Adjusting to variations in bandwidth

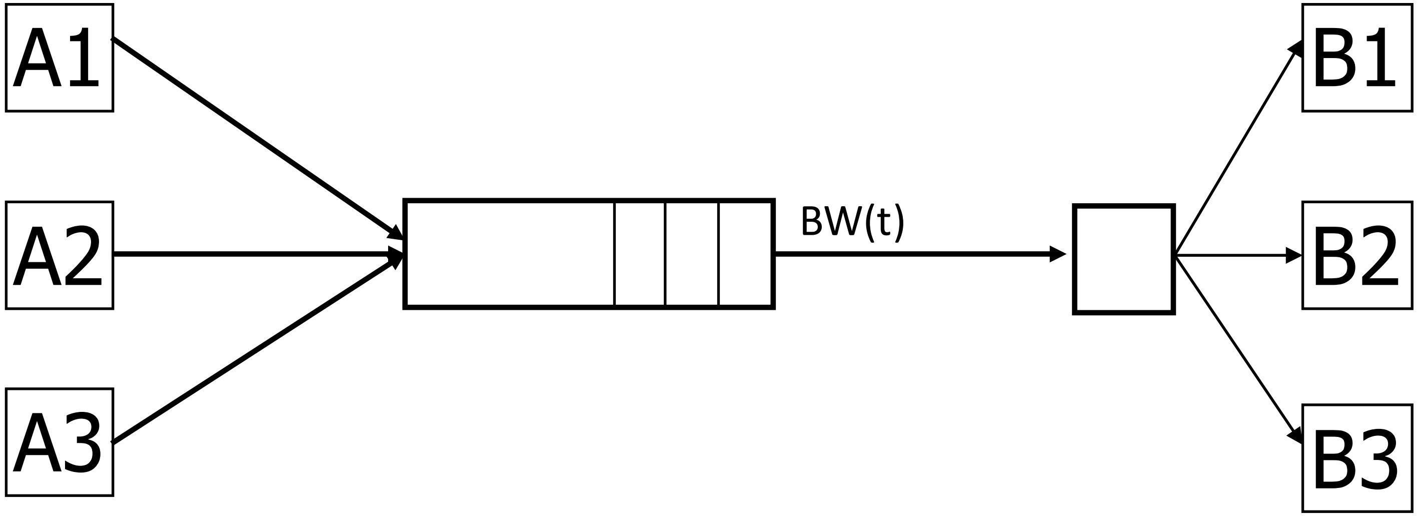


- Adjust rate to match instantaneous bandwidth
  - Assuming you have rough idea of bandwidth

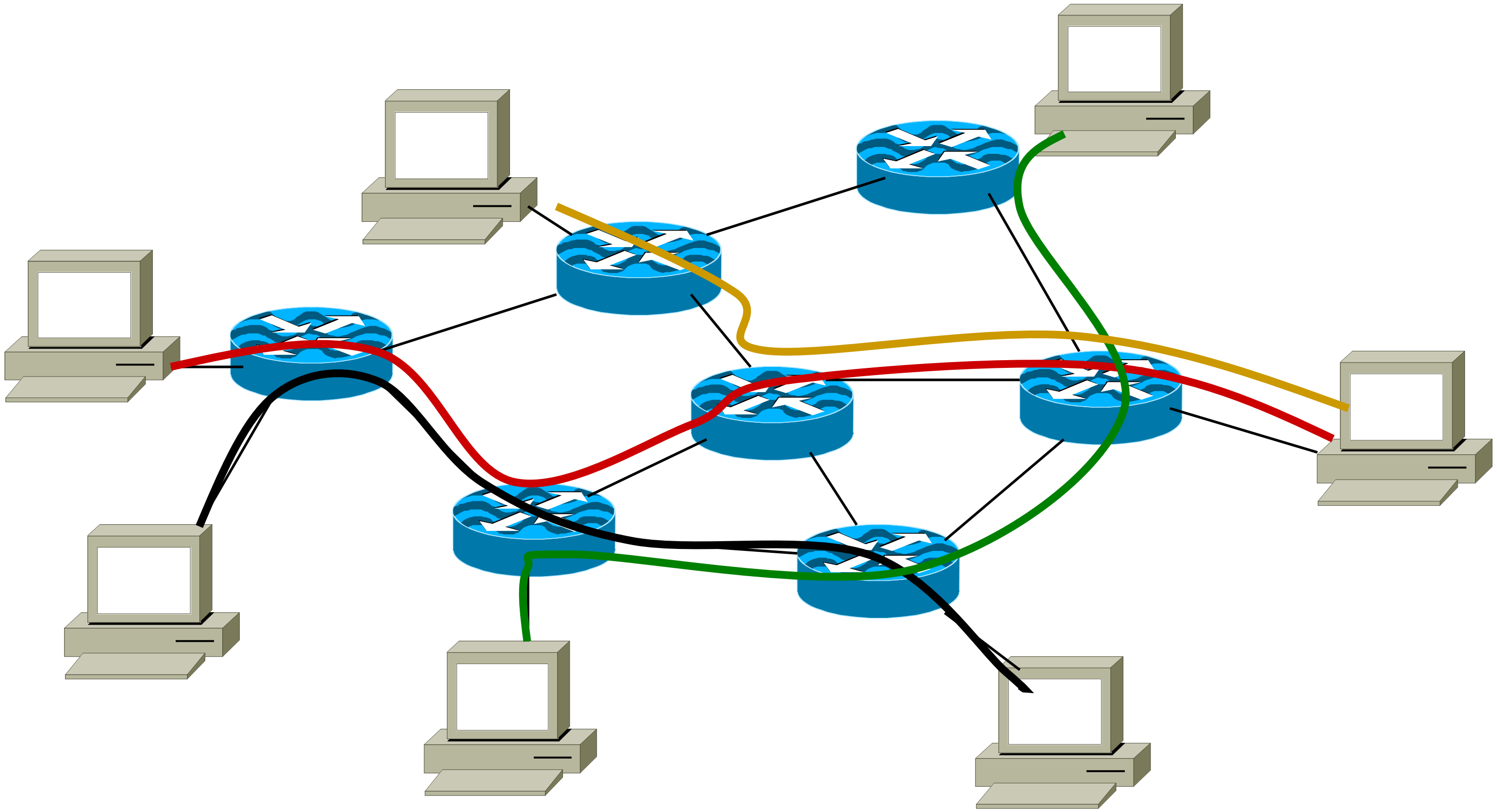
Multiple flows and sharing bandwidth

Two Issues:

- Adjust total sending rate to match bandwidth
- Allocation of bandwidth between flows



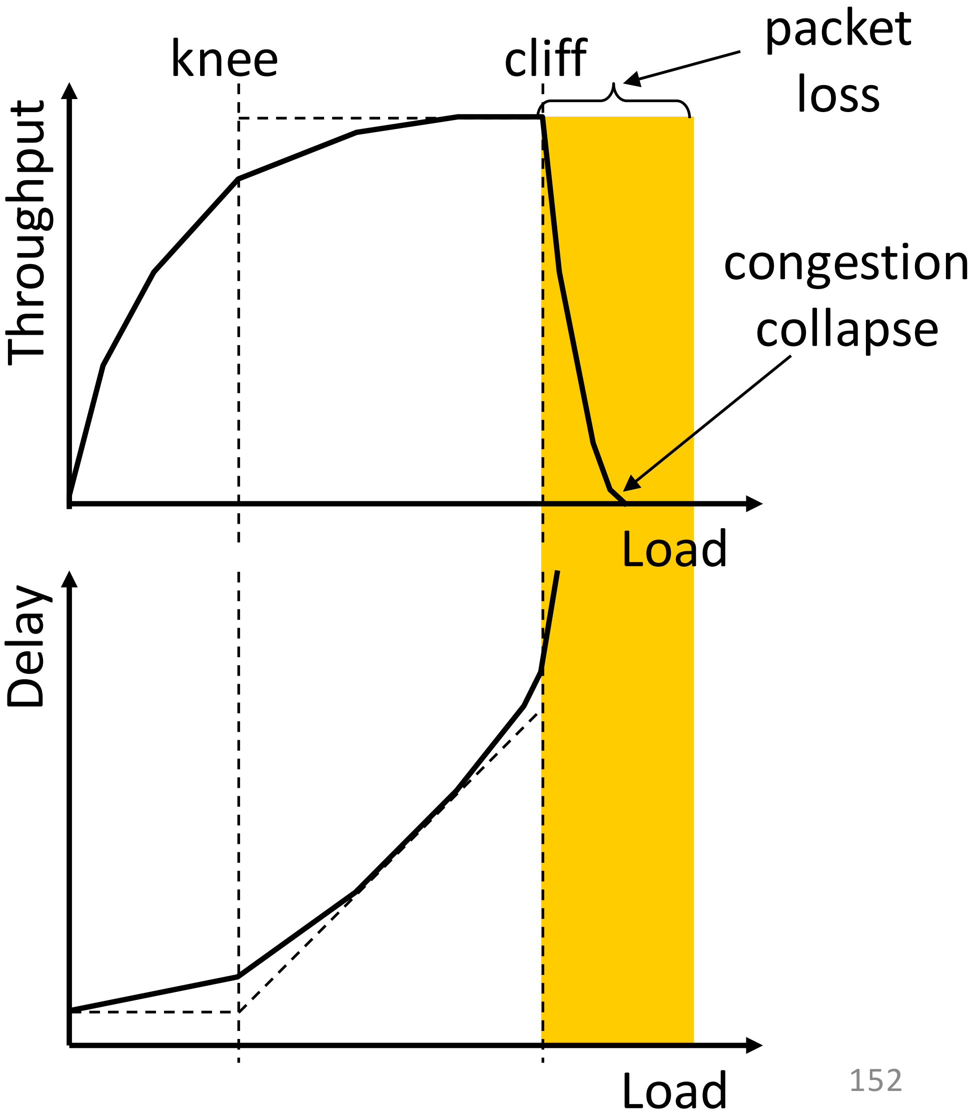
Reality



Congestion control is a resource allocation problem involving many flows, many links, and complicated global dynamics

View from a single flow

- Knee – point after which
  - Throughput increases slowly
  - Delay increases fast
- Cliff – point after which
  - Throughput starts to drop to zero (congestion collapse)
  - Delay approaches infinity



General Approaches

(0) Send without care

- Many packet drops

General Approaches

(0) Send without care

(1) Reservations

- Pre-arrange bandwidth allocations
- Requires negotiation before sending packets
- Low utilization

General Approaches

(0) Send without care

(1) Reservations

(2) Pricing

- Don't drop packets for the high-bidders
- Requires payment model

General Approaches

(0) Send without care

(1) Reservations

(2) Pricing

(3) Dynamic Adjustment

- Hosts probe network; infer level of congestion; adjust
- Network reports congestion level to hosts; hosts adjust
- Combinations of the above
- Simple to implement but suboptimal, messy dynamics

General Approaches

(0) Send without care

(1) Reservations

(2) Pricing

(3) Dynamic Adjustment

All three techniques have their place

- *Generality* of dynamic adjustment has proven powerful
- Doesn’t presume business model, traffic characteristics, application requirements; does assume good citizenship

TCP’s Approach in a Nutshell

- TCP connection has window
  - Controls number of packets in flight
- Sending rate: ~Window/RTT
- Vary window size to control sending rate

All These Windows…

- Congestion Window: CWND
  - How many bytes can be sent without overflowing routers
  - Computed by the sender using congestion control algorithm

- Flow control window: AdvertisedWindow (RWND)
  - How many bytes can be sent without overflowing receiver's buffers
  - Determined by the receiver and reported to the sender

- Sender-side window = minimum{CWND,RWND}
  - Assume for this material that RWND >> CWND

Note

- This lecture will talk about CWND in units of MSS
  - (Recall MSS: Maximum Segment Size, the amount of payload data in a TCP packet)
  - This is only for pedagogical purposes

- **In reality this is a LIE:** Real implementations maintain CWND in bytes

Two Basic Questions

- How does the sender detect congestion?
- How does the sender adjust its sending rate?
  - To address three issues
    - Finding available bottleneck bandwidth
    - Adjusting to bandwidth variations
    - Sharing bandwidth

Detecting Congestion

- Packet delays
  - Tricky: noisy signal (delay often varies considerably)

- Router tell endhosts they’re congested

- Packet loss
  - Fail-safe signal that TCP already has to detect
  - Complication: non-congestive loss (checksum errors)

- Two indicators of packet loss
  - No ACK after certain time interval: timeout
  - Multiple duplicate ACKs

Not All Losses the Same

- Duplicate ACKs: isolated loss
  - Still getting ACKs

- Timeout: much more serious
  - Not enough dupacks
  - Must have suffered several losses

- We will adjust rate differently for each case

Rate Adjustment

- Basic structure:
  - Upon receipt of ACK (of new data): increase rate
  - Upon detection of loss: decrease rate

- How we increase/decrease the rate depends on the phase of congestion control we're in:
  - Discovering available bottleneck bandwidth *vs.*
  - Adjusting to bandwidth variations

Bandwidth Discovery with Slow Start

- Goal: estimate available bandwidth
  - start slow (for safety)
  - but ramp up quickly (for efficiency)
- Consider
  - RTT = 100ms, MSS=1000bytes
  - Window size to fill 1Mbps of BW = 12.5 packets
  - Window size to fill 1Gbps = 12,500 packets
  - Either is possible!

“Slow Start” Phase

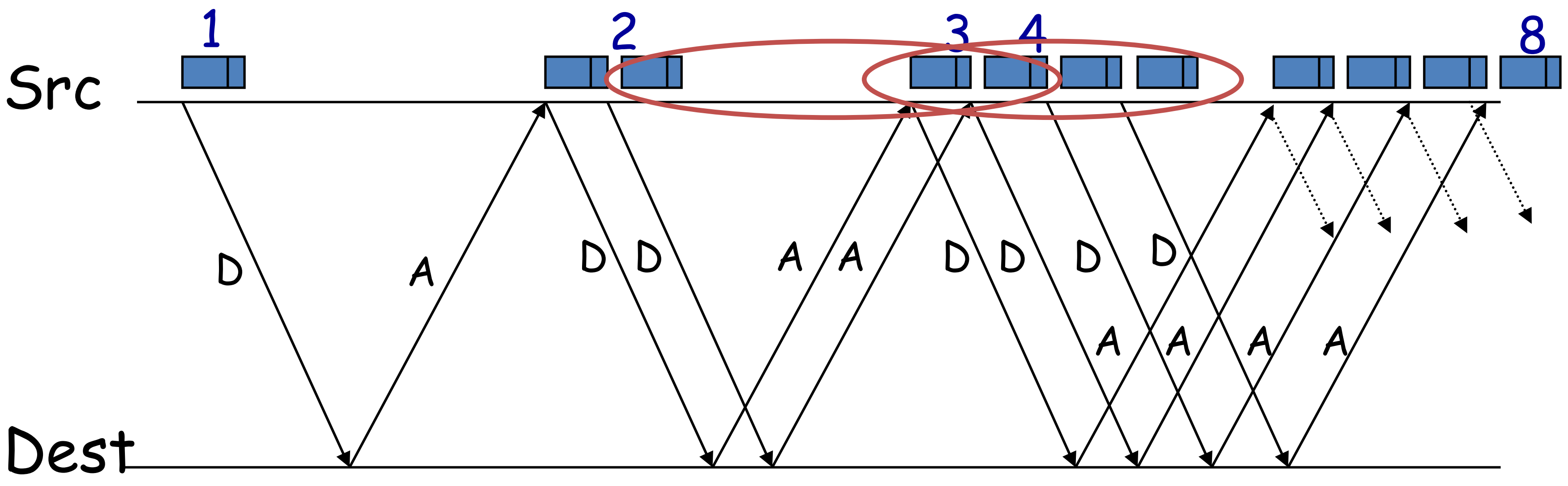
- Sender starts at a slow rate but increases **exponentially** until first loss

- Start with a small congestion window
  - Initially, CWND = 1
  - So, initial sending rate is MSS/RTT

- Double the CWND for each RTT with no loss

Slow Start in Action

- For each RTT: double CWND
- Simpler implementation: for each ACK, CWND += 1



Adjusting to Varying Bandwidth

- Slow start gave an estimate of available bandwidth

- Now, want to track variations in this available bandwidth, oscillating around its current value
  - Repeated probing (rate increase) and backoff (rate decrease)

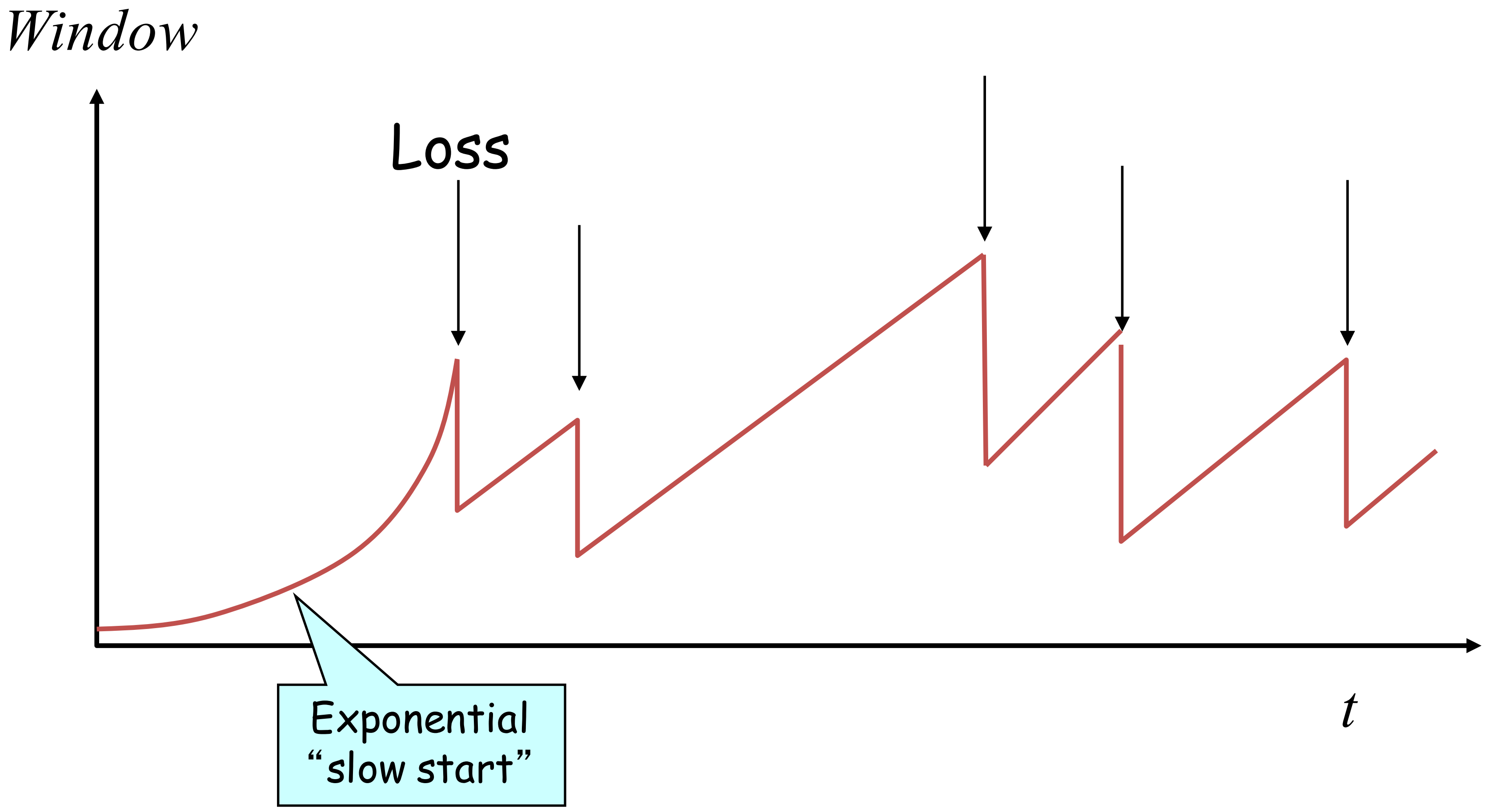
- TCP uses: "Additive Increase Multiplicative Decrease" (AIMD)
  - We'll see why shortly…

AIMD

- Additive increase
  - Window grows by one MSS for every RTT with no loss
  - For each successful RTT, CWND = CWND + 1
  - Simple implementation:
    - for each ACK, CWND = CWND+ 1/CWND

- Multiplicative decrease
  - On loss of packet, divide congestion window in **half**
  - On loss, CWND = CWND/2

Leads to the TCP "Sawtooth"



Slow-Start vs. AIMD

- When does a sender stop Slow-Start and start Additive Increase?
- Introduce a "slow start threshold" (ssthresh)
  - Initialized to a large value
  - On timeout, ssthresh = CWND/2
- When CWND = ssthresh, sender switches from slow-start to AIMD-style increase

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD

Why AIMD?

Recall: Three Issues

- Discovering the available (bottleneck) bandwidth
  - Slow Start
- Adjusting to variations in bandwidth
  - AIMD
- Sharing bandwidth between flows

Goals for bandwidth sharing

- Efficiency: High utilization of link bandwidth
- Fairness: Each flow gets equal share

Why AIMD?

- Some rate adjustment options: Every RTT, we can
  - Multiplicative increase or decrease: CWND→ a*CWND
  - Additive increase or decrease: CWND→ CWND + b

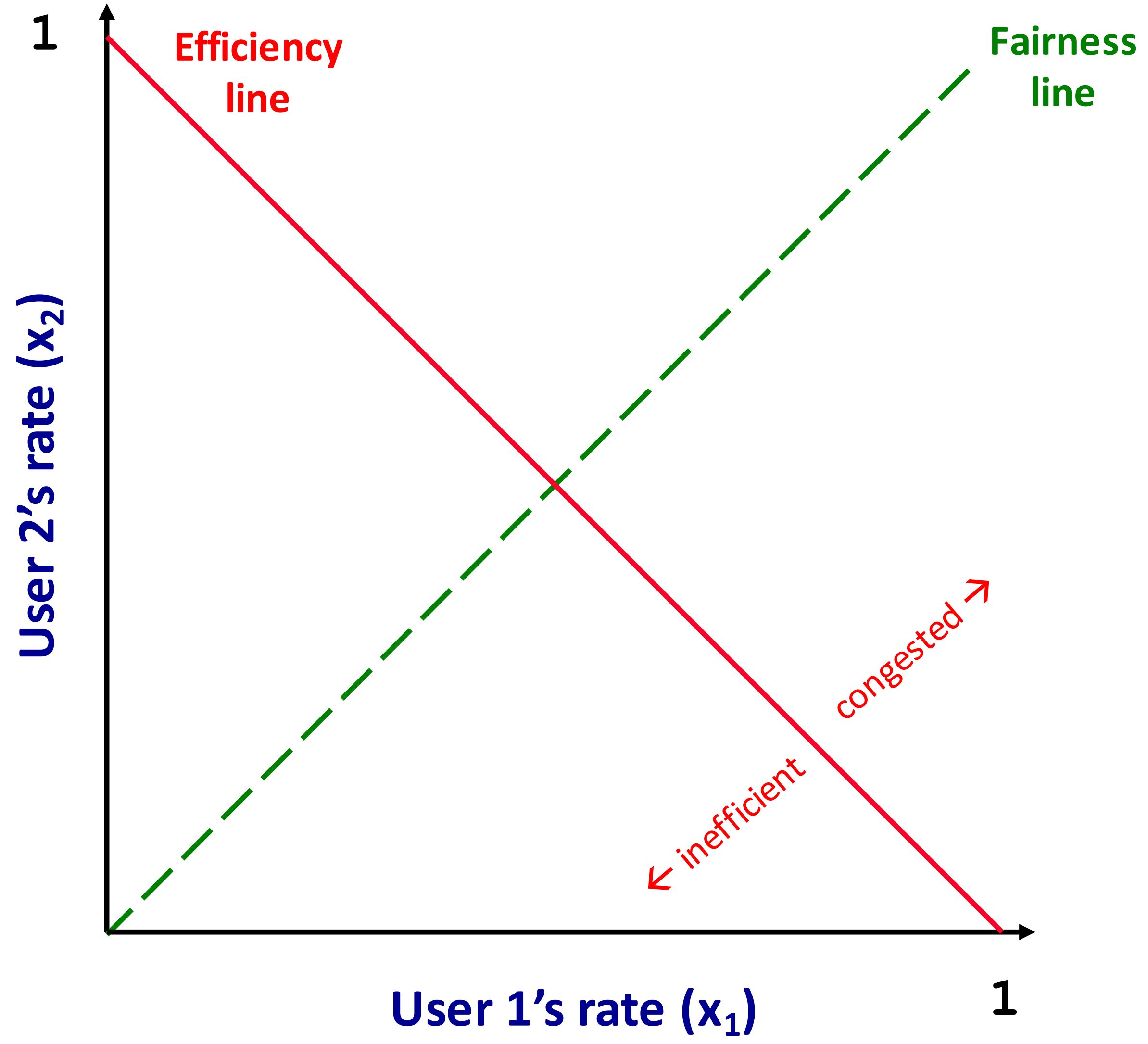
- Four alternatives:
  - AIAD: gentle increase, gentle decrease
  - AIMD: gentle increase, drastic decrease
  - MIAD: drastic increase, gentle decrease
  - MIMD: drastic increase and decrease

Simple Model of Congestion Control

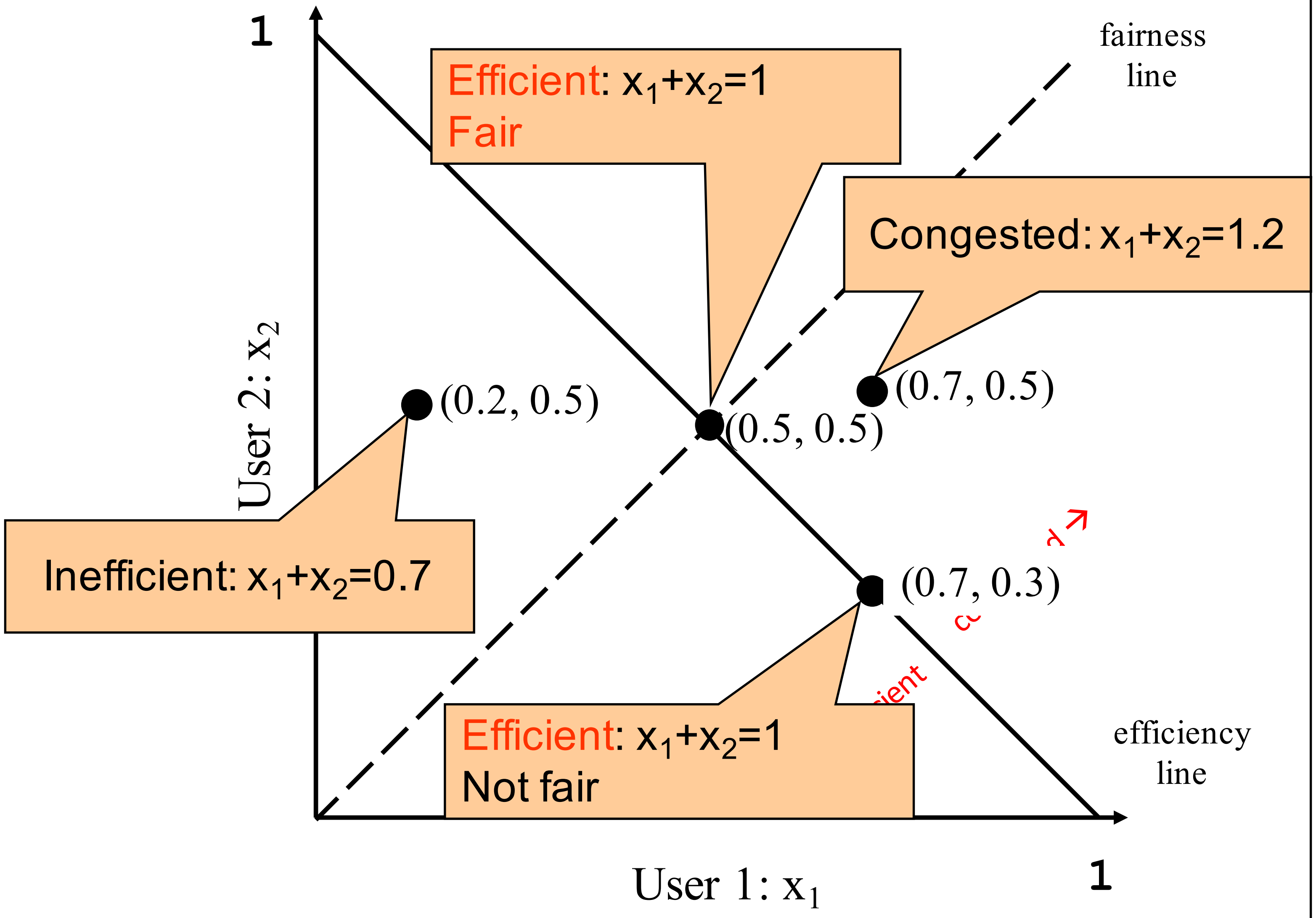
- Two users
  - rates x_1 and x_2

- Congestion when $x_1+x_2 > 1$
- Unused capacity when $x_1+x_2 < 1$

- Fair when $x_1 = x_2$



Example



Efficient: $x_1+x_2=1$
Fair

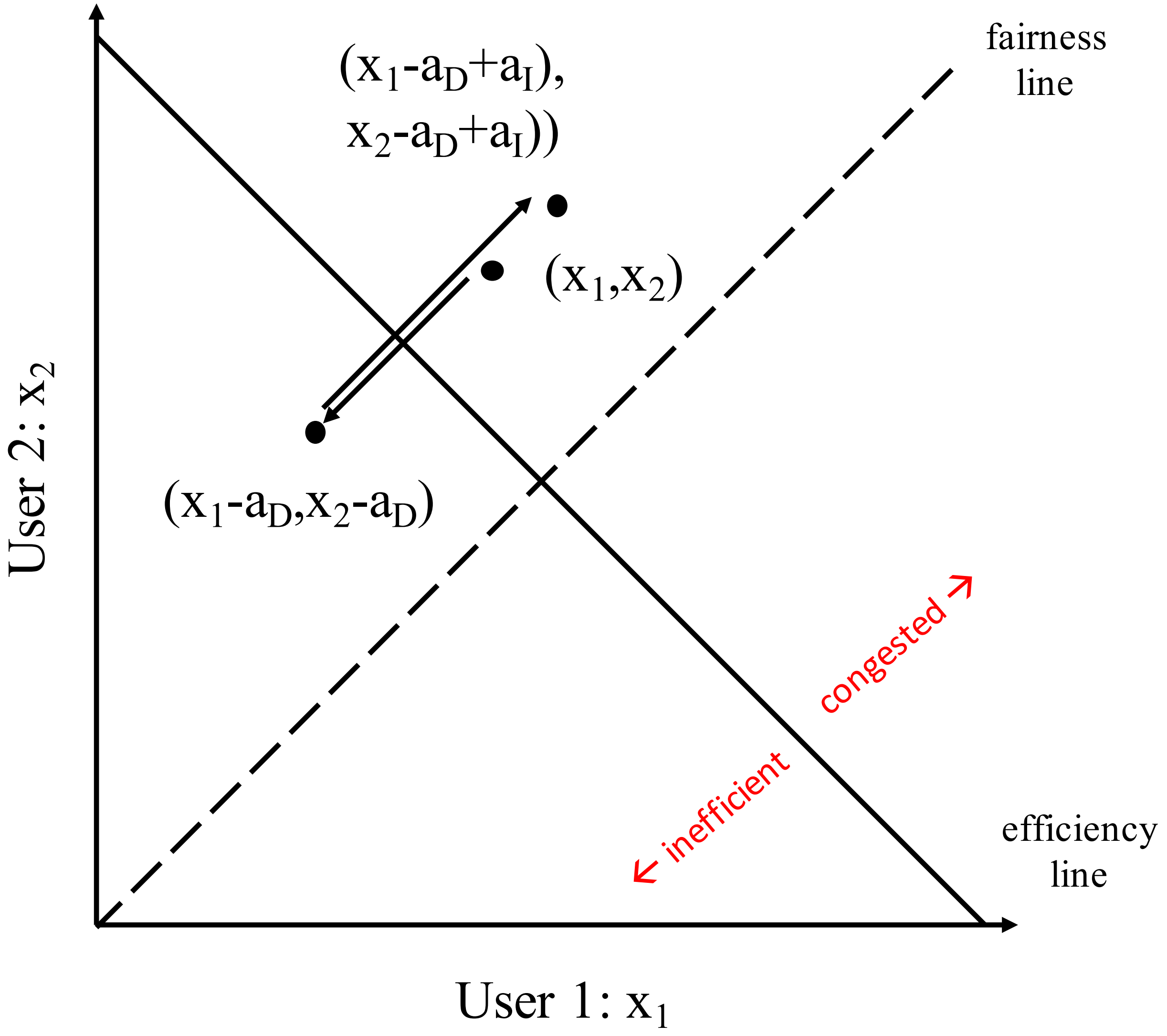
Congested: $x_1+x_2=1.2$

Inefficient: $x_1+x_2=0.7$

Efficient: $x_1+x_2=1$
Not fair

AIAD

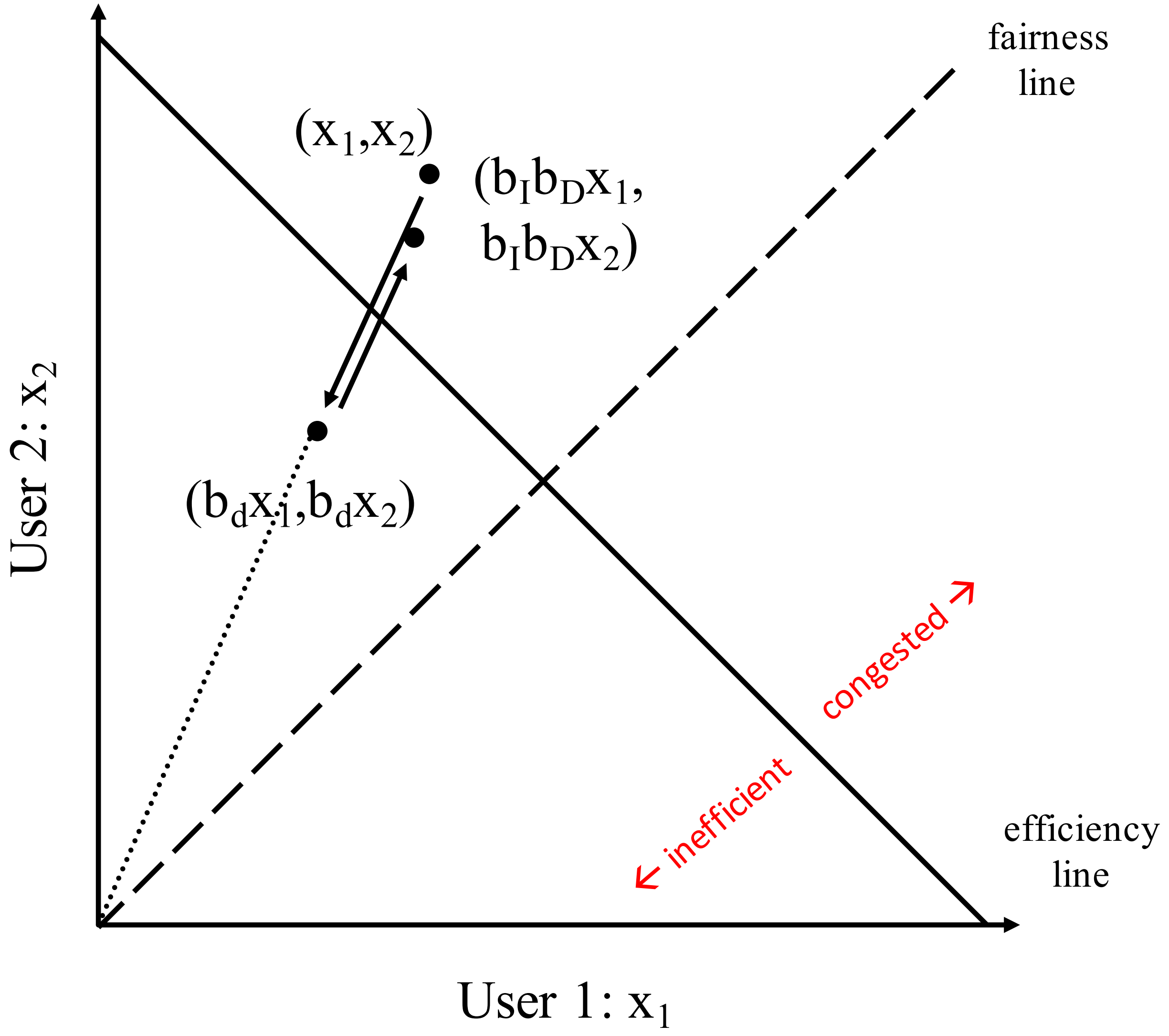
- Increase: $x + a_I$
- Decrease: $x - a_D$
- Does not converge to fairness



$(x_1\text{-}a_D\text{+}a_I),$
$x_2\text{-}a_D\text{+}a_I))$
fairness line
(x_1,x_2)
$(x_1\text{-}a_D,x_2\text{-}a_D)$
User 2: x_2
congested →
← inefficient
efficiency line
User 1: x_1

MIMD

- Increase: $x*b_I$
- Decrease: $x*b_D$
- Does not converge to fairness

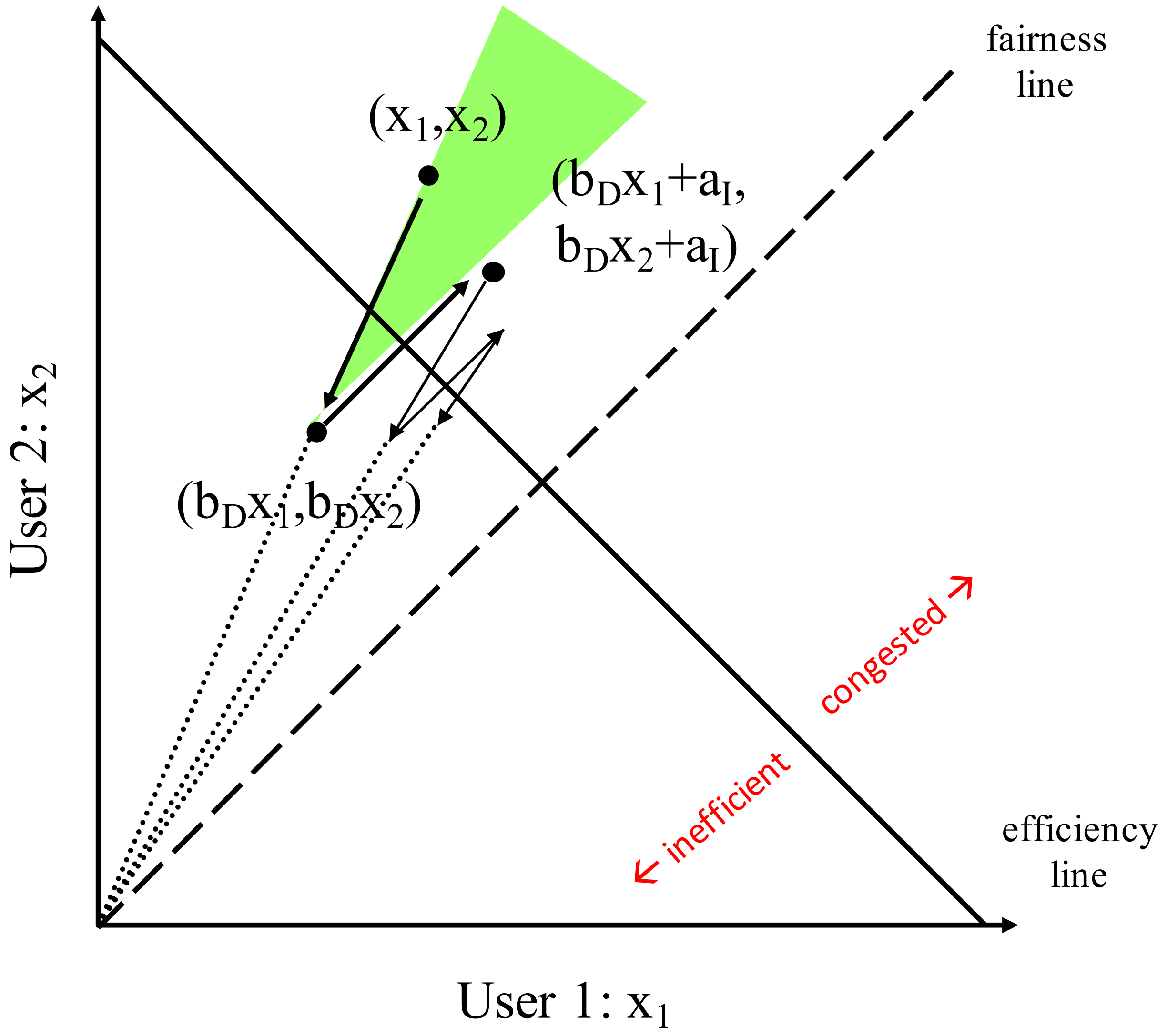


Recall: Three Issues

- Discovering the available (bottleneck) bandwidth
  - Slow Start
- Adjusting to variations in bandwidth
  - AIMD
- Sharing bandwidth between flows

AIMD

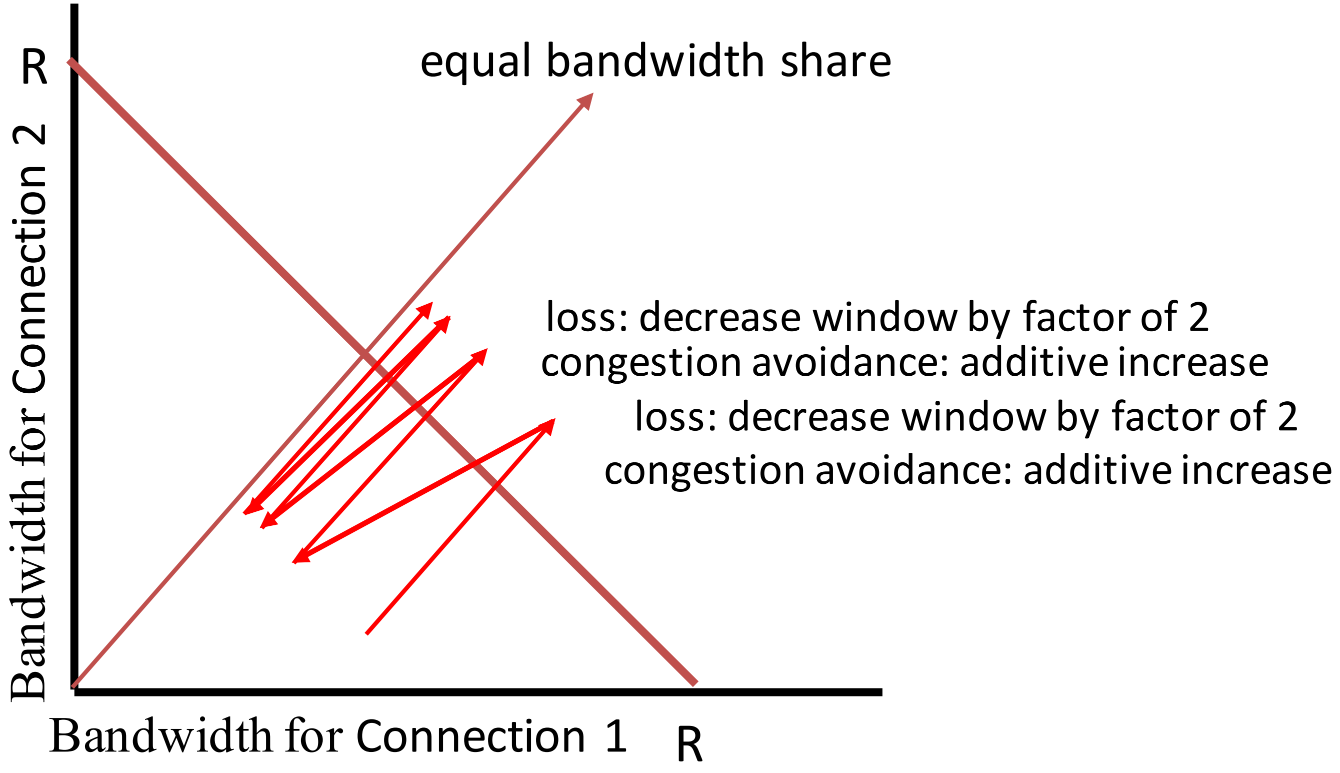
- Increase: $x+a_I$
- Decrease: $x*b_D$
- Converges to fairness



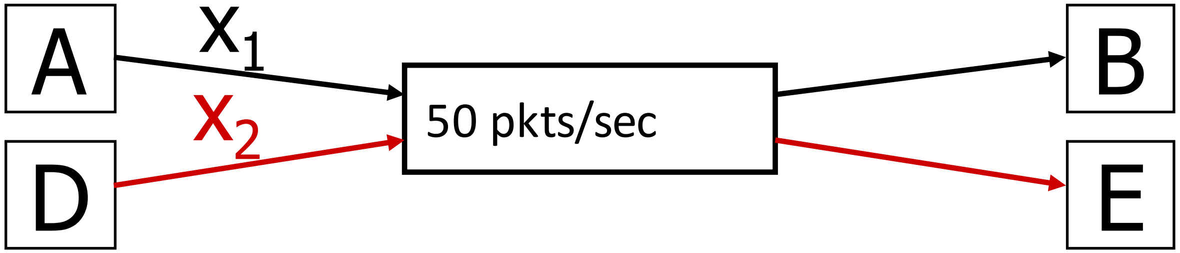
Why is AIMD fair?
(a pretty animation…)

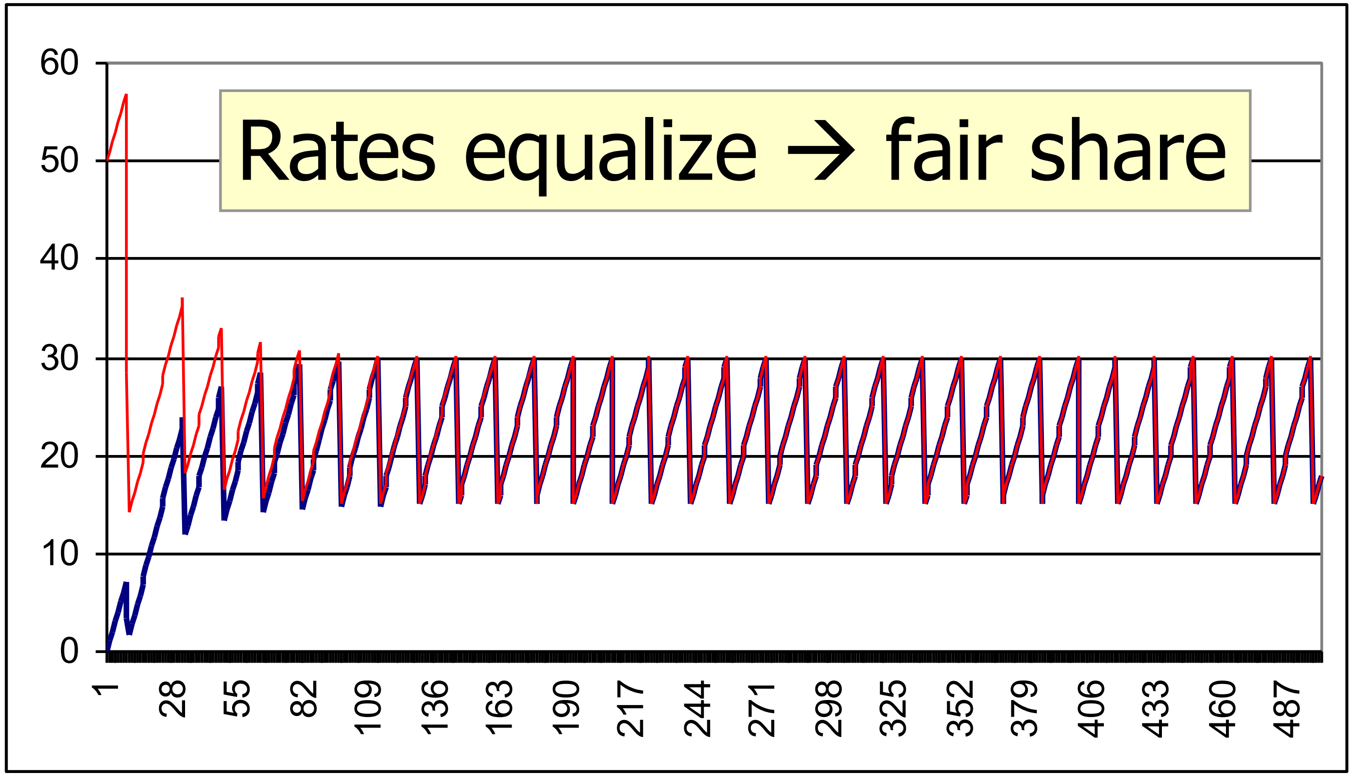
Two competing sessions:

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


R
equal bandwidth share
loss: decrease window by factor of 2
congestion avoidance: additive increase
loss: decrease window by factor of 2
congestion avoidance: additive increase
Bandwidth for Connection 2
Bandwidth for Connection 1
R

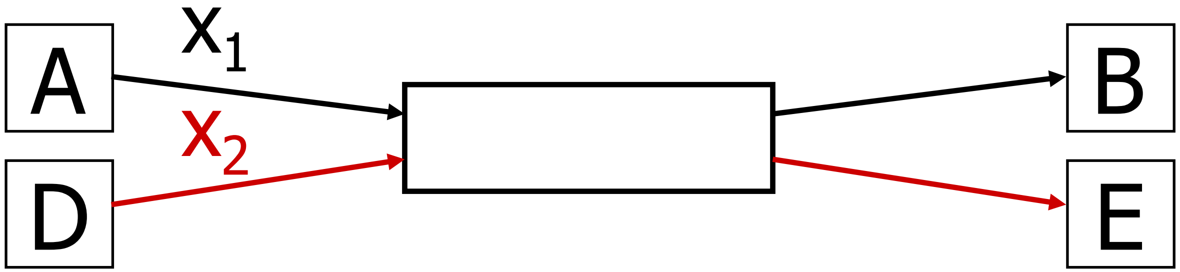
AIMD Sharing Dynamics

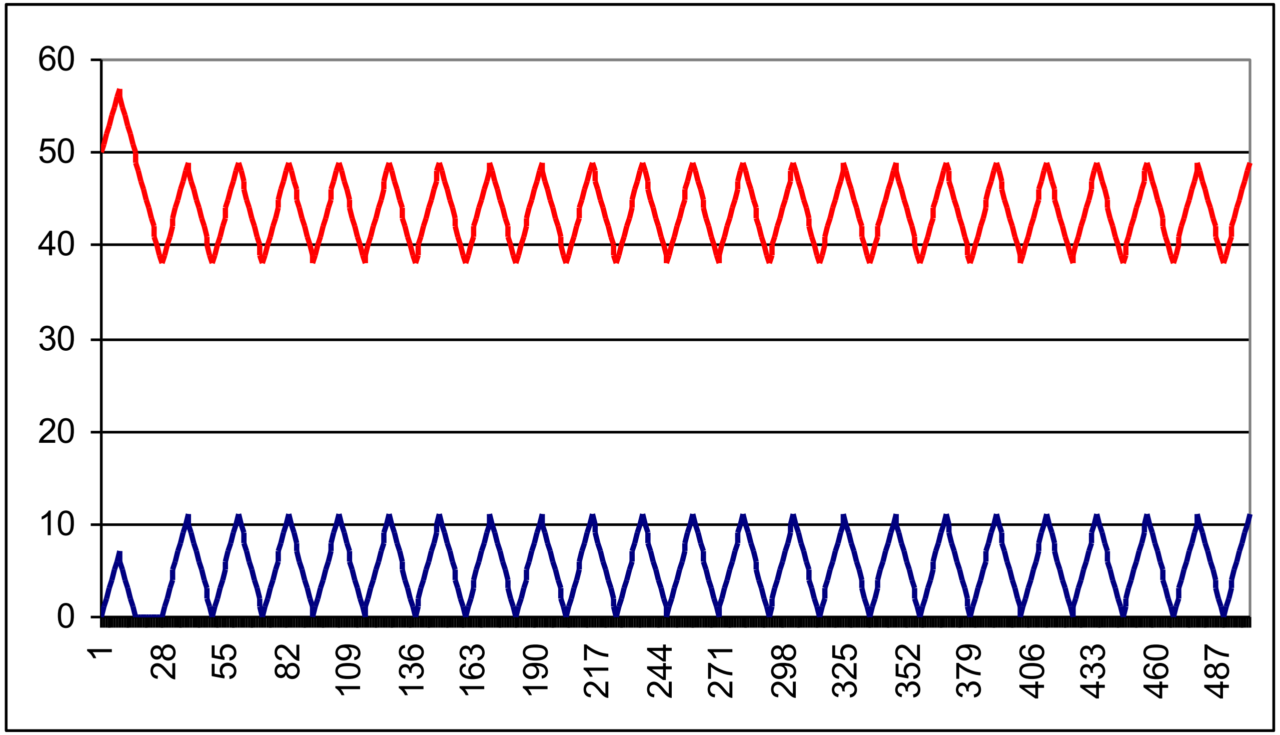




Rates equalize → fair share

AIAD Sharing Dynamics





TCP Congestion Control (Gruesome) Details

Implementation

- State at sender
  - CWND (initialized to a small constant)
  - ssthresh (initialized to a large constant)
  - [Also dupACKcount and timer, as before]

- Events
  - ACK (new data)
  - dupACK (duplicate ACK for old data)
  - Timeout

Event: ACK (new data)

- If CWND < ssthresh
  - CWND += 1

- *CWND packets per RTT*
- *Hence after one RTT with no drops:*
  CWND = 2xCWND

Event: ACK (new data)

- If CWND < ssthresh
  - CWND += 1

Slow start phase

- Else
  - CWND = CWND + 1/CWND

“Congestion Avoidance” phase (additive increase)

- *CWND packets per RTT*
- *Hence after one RTT with no drops:*
  CWND = CWND + 1

Event: TimeOut

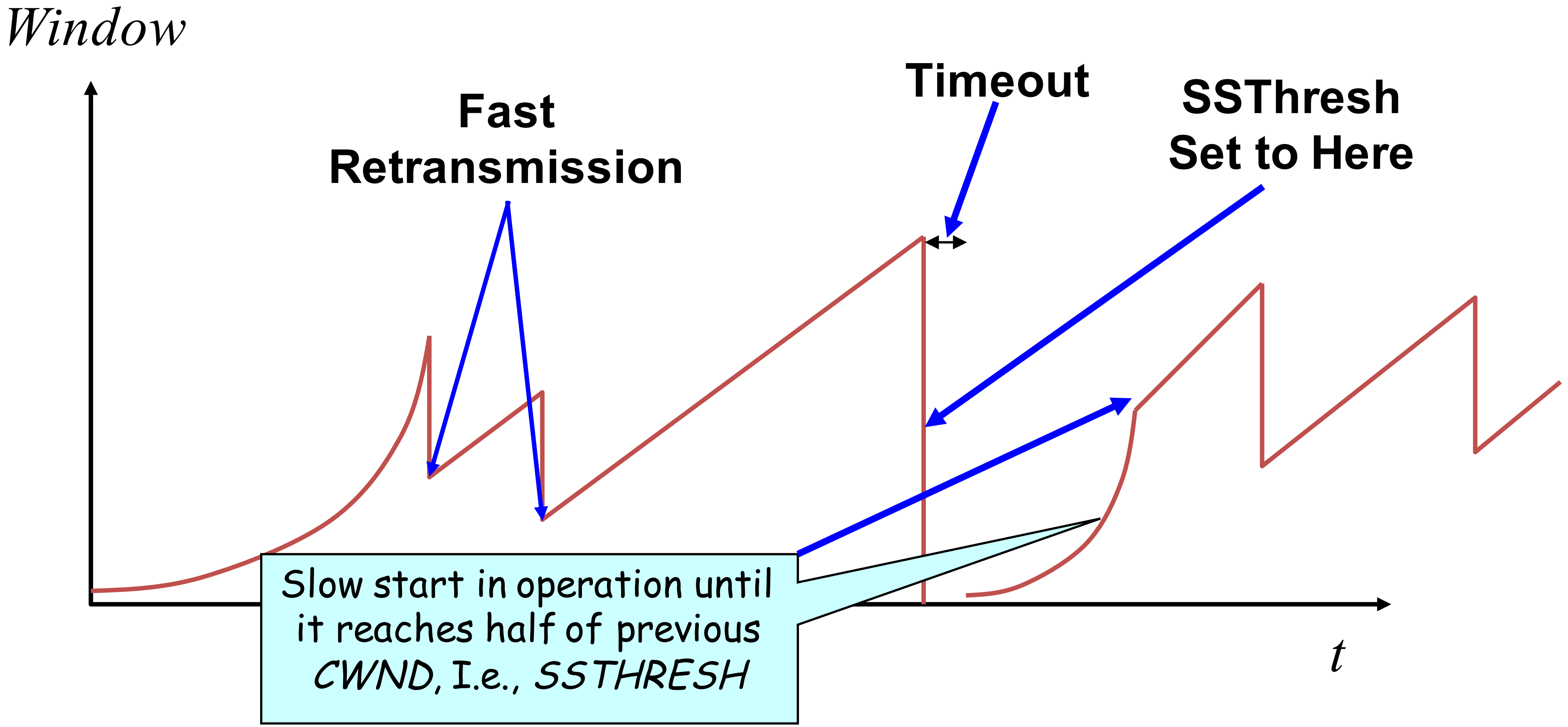
- On Timeout
  - ssthresh ← CWND/2
  - CWND ← 1

Event: dupACK

- dupACKcount ++

- If dupACKcount = 3 /* fast retransmit */
  - ssthresh = CWND/2
  - CWND = CWND/2

Example



Slow start in operation until it reaches half of previous *CWND*, I.e., *SSTHRESH*

Slow-start restart: Go back to CWND = 1 MSS, but take advantage of knowing the previous value of CWND

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD, Fast-Recovery

One Final Phase: Fast Recovery

- The problem: congestion avoidance too slow in recovering from an isolated loss

Example (in units of MSS, not bytes)

- Consider a TCP connection with:
  - CWND=10 packets
  - Last ACK was for packet # 101
    - i.e., receiver expecting next packet to have seq. no. 101

- 10 packets [101, 102, 103,…, 110] are in flight
  - Packet 101 is dropped
  - What ACKs do they generate?
  - And how does the sender respond?

Timeline

- ACK 101 (due to 102) cwnd=10 dupACK#1 (no xmit)
- ACK 101 (due to 103) cwnd=10 dupACK#2 (no xmit)
- ACK 101 (due to 104) cwnd=10 dupACK#3 (no xmit)
- RETRANSMIT 101 ssthresh=5 cwnd= 5
- ACK 101 (due to 105) cwnd=5 + 1/5 (no xmit)
- ACK 101 (due to 106) cwnd=5 + 2/5 (no xmit)
- ACK 101 (due to 107) cwnd=5 + 3/5 (no xmit)
- ACK 101 (due to 108) cwnd=5 + 4/5 (no xmit)
- ACK 101 (due to 109) cwnd=5 + 5/5 (no xmit)
- ACK 101 (due to 110) cwnd=6 + 1/5 (no xmit)
- ACK 111 (due to 101) ← only now can we transmit new packets
- Plus no packets in flight so ACK "clocking" (to increase CWND) stalls for another RTT

Solution: Fast Recovery

Idea: Grant the sender temporary “credit” for each dupACK so as to keep packets in flight

- If dupACKcount = 3
  - ssthresh = cwnd/2
  - cwnd = ssthresh + 3
- While in fast recovery
  - cwnd = cwnd + 1 for each additional duplicate ACK
- Exit fast recovery after receiving new ACK
  - set cwnd = ssthresh

Example

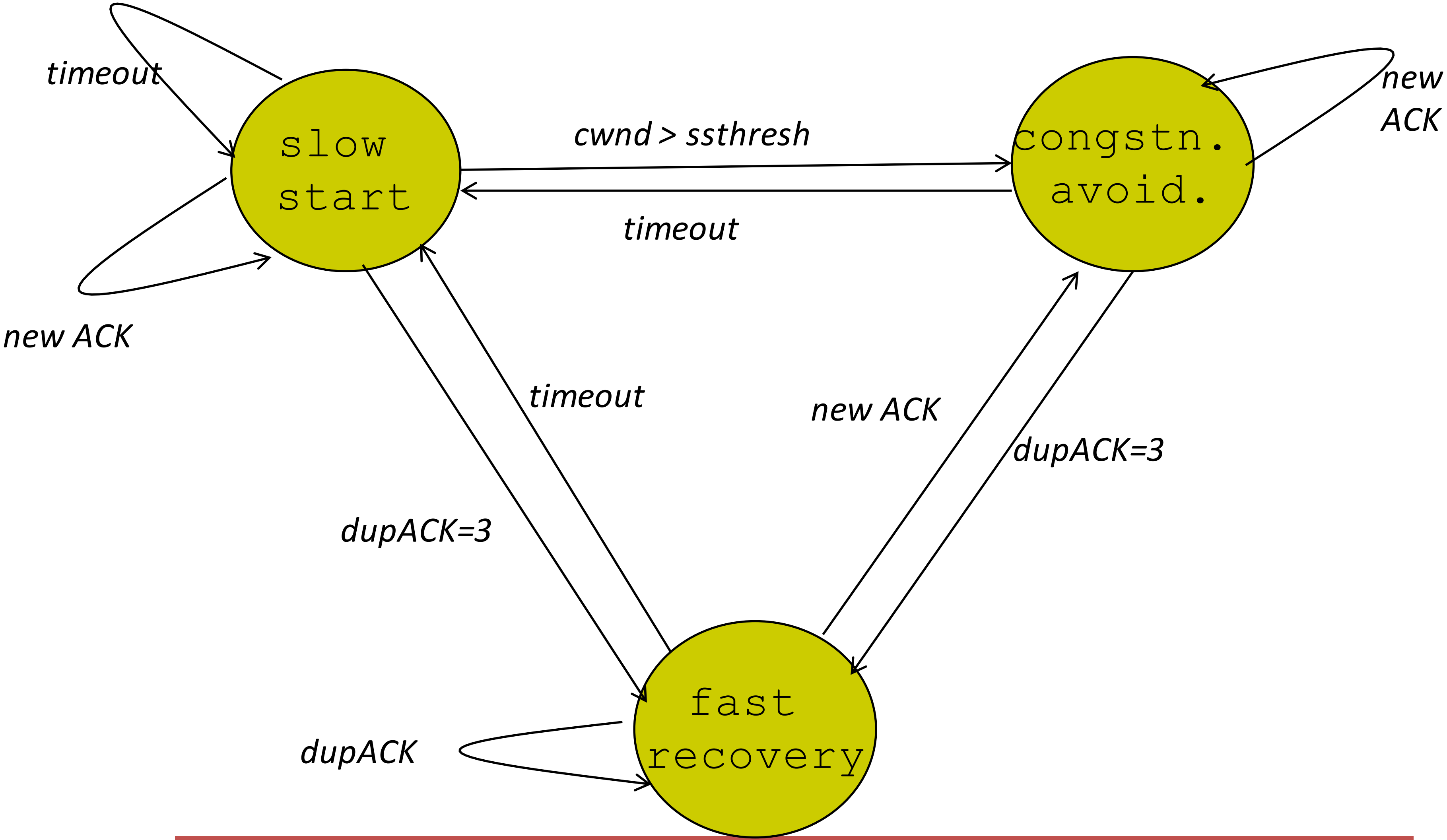
- Consider a TCP connection with:
  - CWND=10 packets
  - Last ACK was for packet # 101
    - i.e., receiver expecting next packet to have seq. no. 101

- 10 packets [101, 102, 103,…, 110] are in flight
  - Packet 101 is dropped

Timeline

- ACK 101 (due to 102) cwnd=10 dup#1
- ACK 101 (due to 103) cwnd=10 dup#2
- ACK 101 (due to 104) cwnd=10 dup#3
- REXMIT 101 ssthresh=5 cwnd= 8 (5+3)
- ACK 101 (due to 105) cwnd= 9 (no xmit)
- ACK 101 (due to 106) cwnd=10 (no xmit)
- ACK 101 (due to 107) cwnd=11 (xmit 111)
- ACK 101 (due to 108) cwnd=12 (xmit 112)
- ACK 101 (due to 109) cwnd=13 (xmit 113)
- ACK 101 (due to 110) cwnd=14 (xmit 114)
- ACK 111 (due to 101) cwnd = 5 (xmit 115) ← exiting fast recovery
- Packets 111-114 already in flight
- ACK 112 (due to 111) cwnd = 5 + 1/5 ← back in congestion avoidance

Putting it all together: The TCP State Machine (partial)



- How are ssthresh, CWND and dupACKcount updated for each event that causes a state transition?

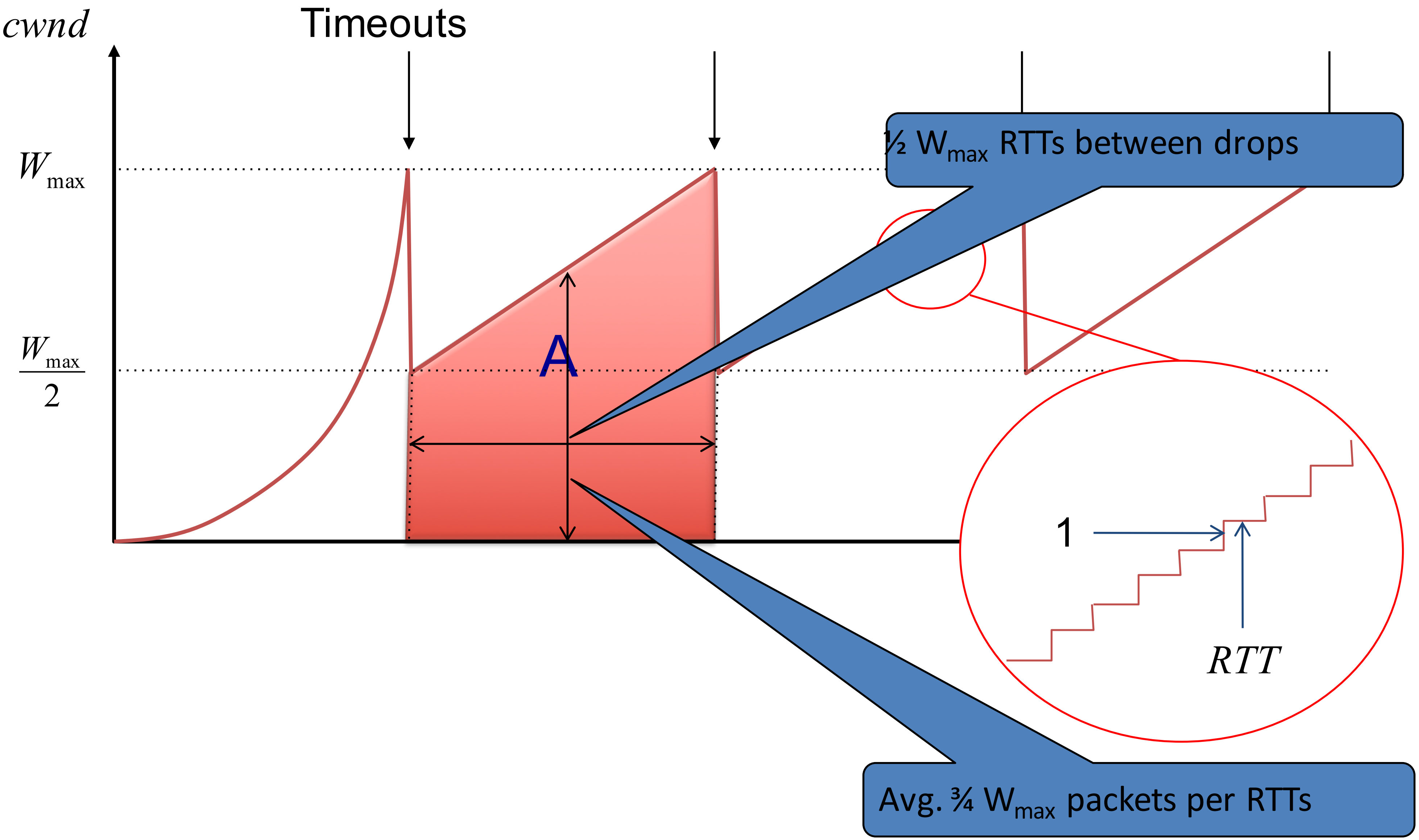
TCP Flavors

- TCP-Tahoe
  - cwnd =1 on triple dupACK
- TCP-Reno
  - cwnd =1 on timeout
  - cwnd = cwnd/2 on triple dupack
- TCP-newReno
  - TCP-Reno + improved fast recovery
- TCP-SACK
  - incorporates selective acknowledgements

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD, Fast-Recovery, Throughput

TCP Throughput Equation

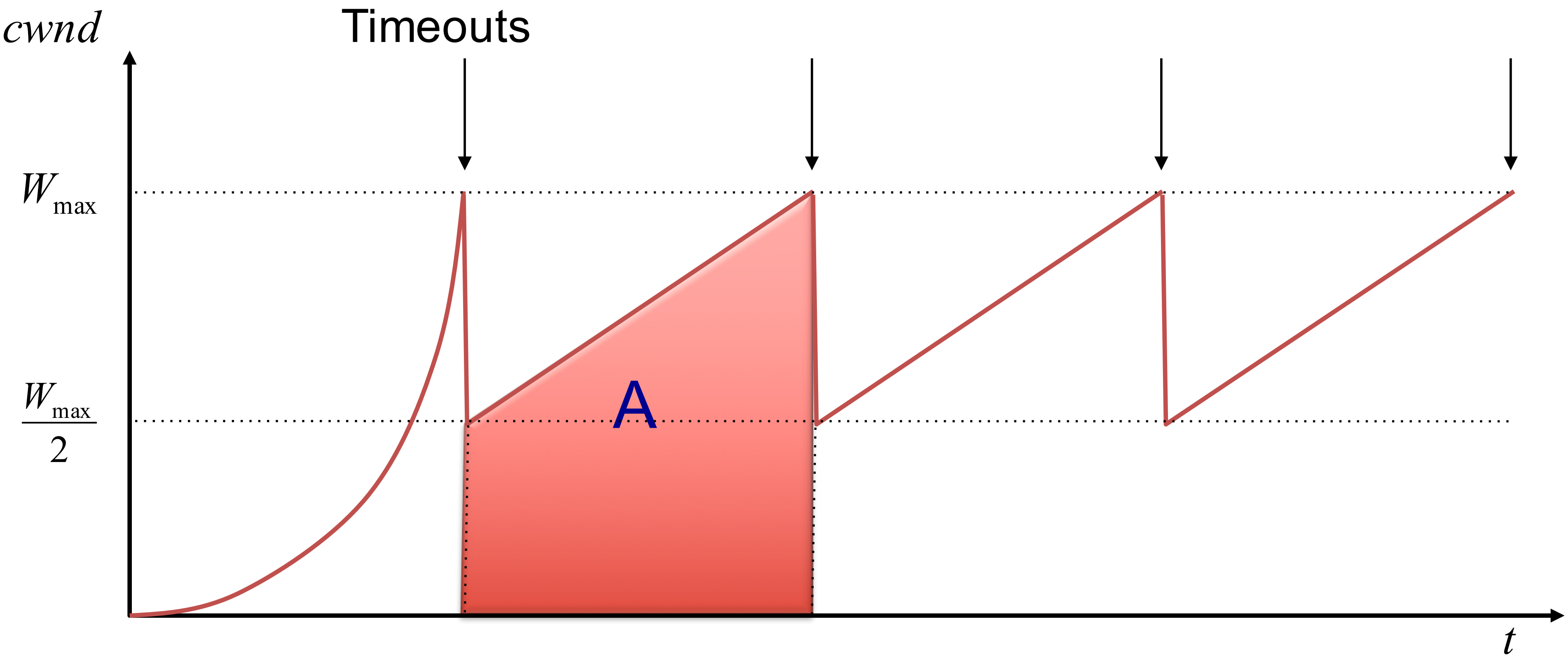
A Simple Model for TCP Throughput



½ W_{max} RTTs between drops

Avg. ¾ W_{max} packets per RTTs

A Simple Model for TCP Throughput



$$\text{Packet drop rate, } p = 1/A, \text{ where } A = \frac{3}{8}W_{\max}^2$$

$$\text{Throughput, } B = \frac{A}{\left(\frac{W_{\max}}{2}\right)RTT} = \sqrt{\frac{3}{2}}\frac{1}{RTT\sqrt{p}}$$

Some implications: (1) Fairness

$$\text{Throughput}, B = \sqrt{\frac{3}{2}} \frac{1}{RTT\sqrt{p}}$$

- Flows get throughput inversely proportional to RTT
  - Is this fair?

Some Implications:
(2) How does this look at high speed?

- Assume that RTT = 100ms, MSS=1500bytes

- What value of *p* is required to go 100Gbps?
  - Roughly 2 x 10^{-12}
- How long between drops?
  - Roughly 16.6 hours
- How much data has been sent in this time?
  - Roughly 6 petabits

- These are not practical numbers!

Some implications:
(3) Rate-based Congestion Control

$$\text{Throughput}, B = \sqrt{\frac{3}{2}}\frac{1}{RTT\sqrt{p}}$$

- One can dispense with TCP and just match eqtn:
  - Equation-based congestion control
  - Measure drop percentage *p*, and set rate accordingly
  - Useful for streaming applications

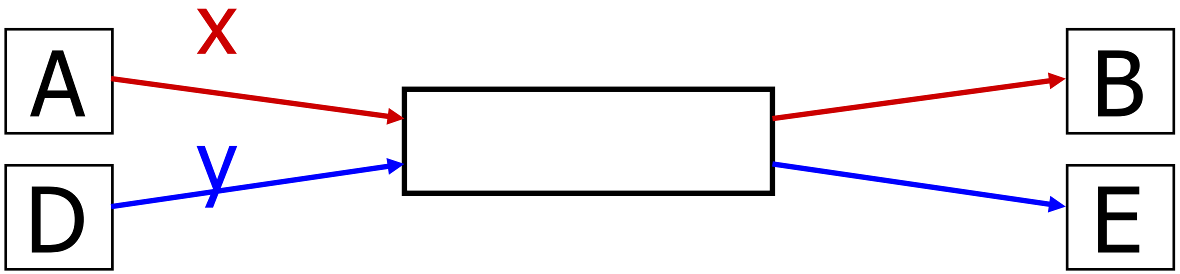
Some Implications: (4) Lossy Links

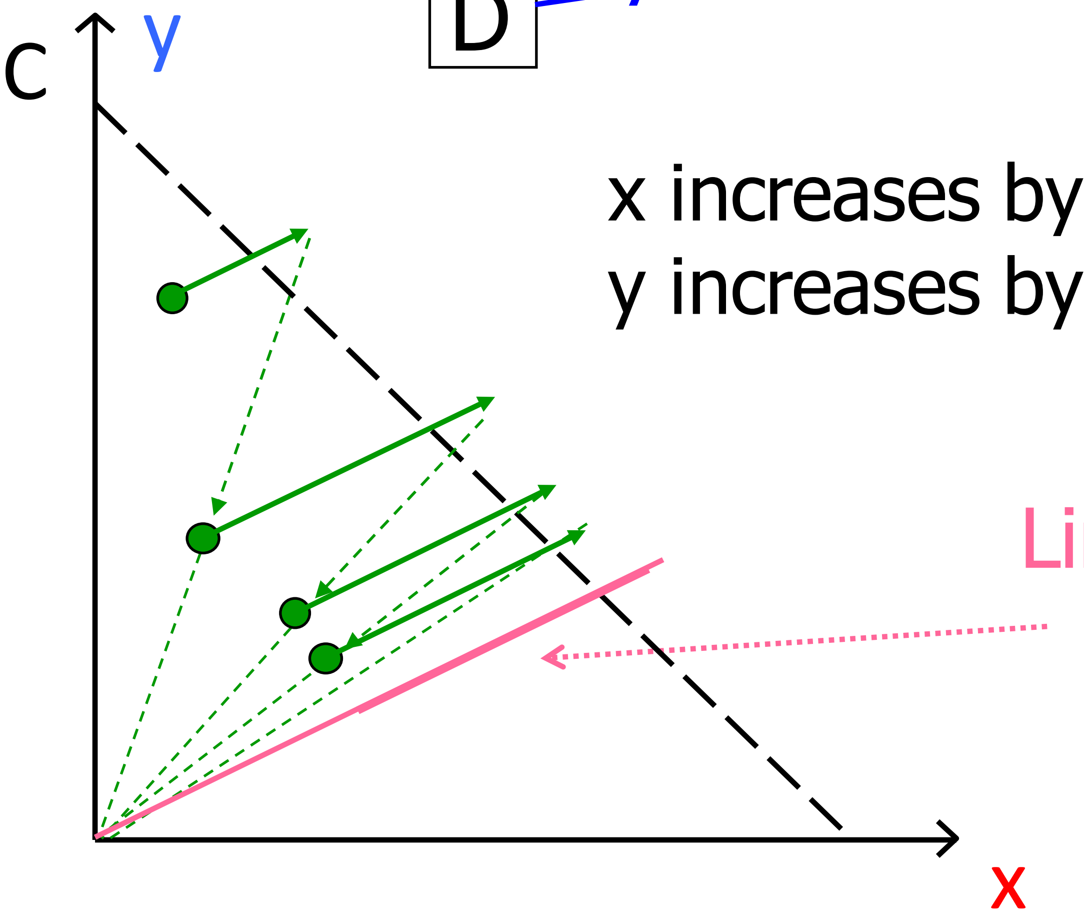
- TCP assumes all losses are due to congestion
- What happens when the link is lossy?
- Throughput ~ 1/sqrt(p) where p is loss prob.
- This applies even for non-congestion losses!

Other Issues: Cheating

- Cheating pays off
- Some favorite approaches to cheating:
  - Increasing CWND faster than 1 per RTT
  - Using large initial CWND
  - Opening many connections

Increasing CWND Faster





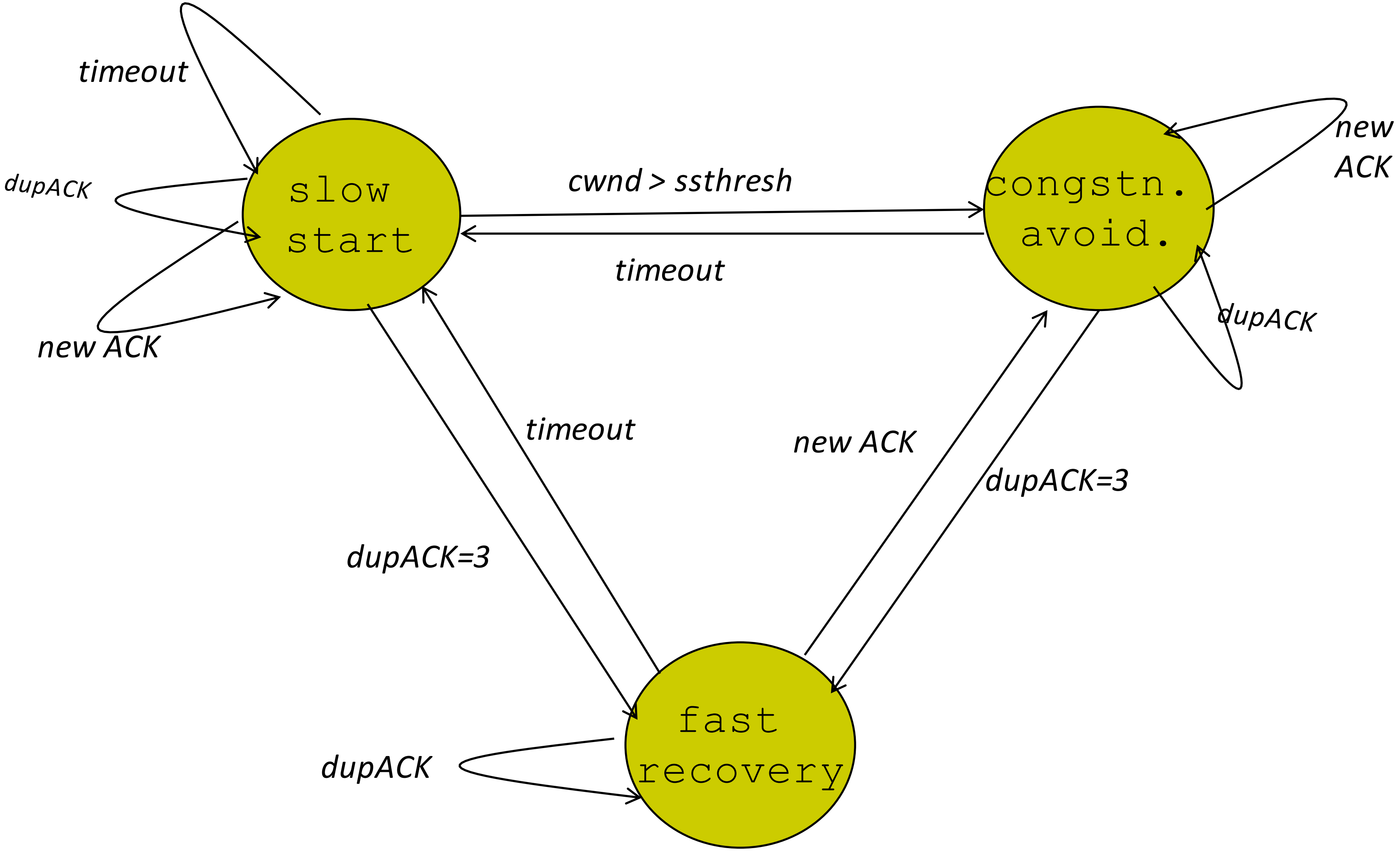
x increases by 2 per RTT
y increases by 1 per RTT

Limit rates:
x = 2y

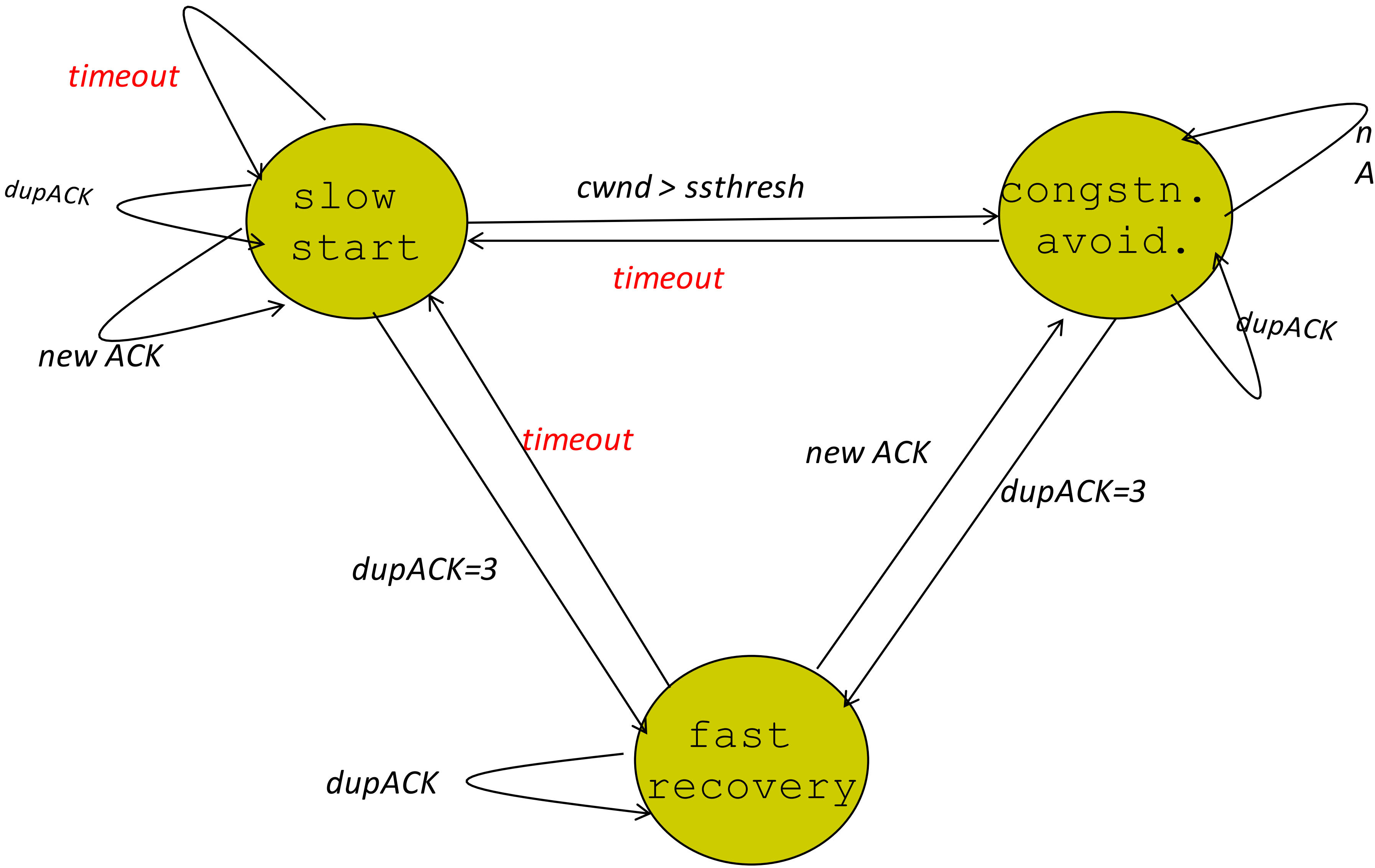
- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD, Fast-Recovery, Throughput
- Limitations of TCP Congestion Control

A Closer look at problems with TCP Congestion Control

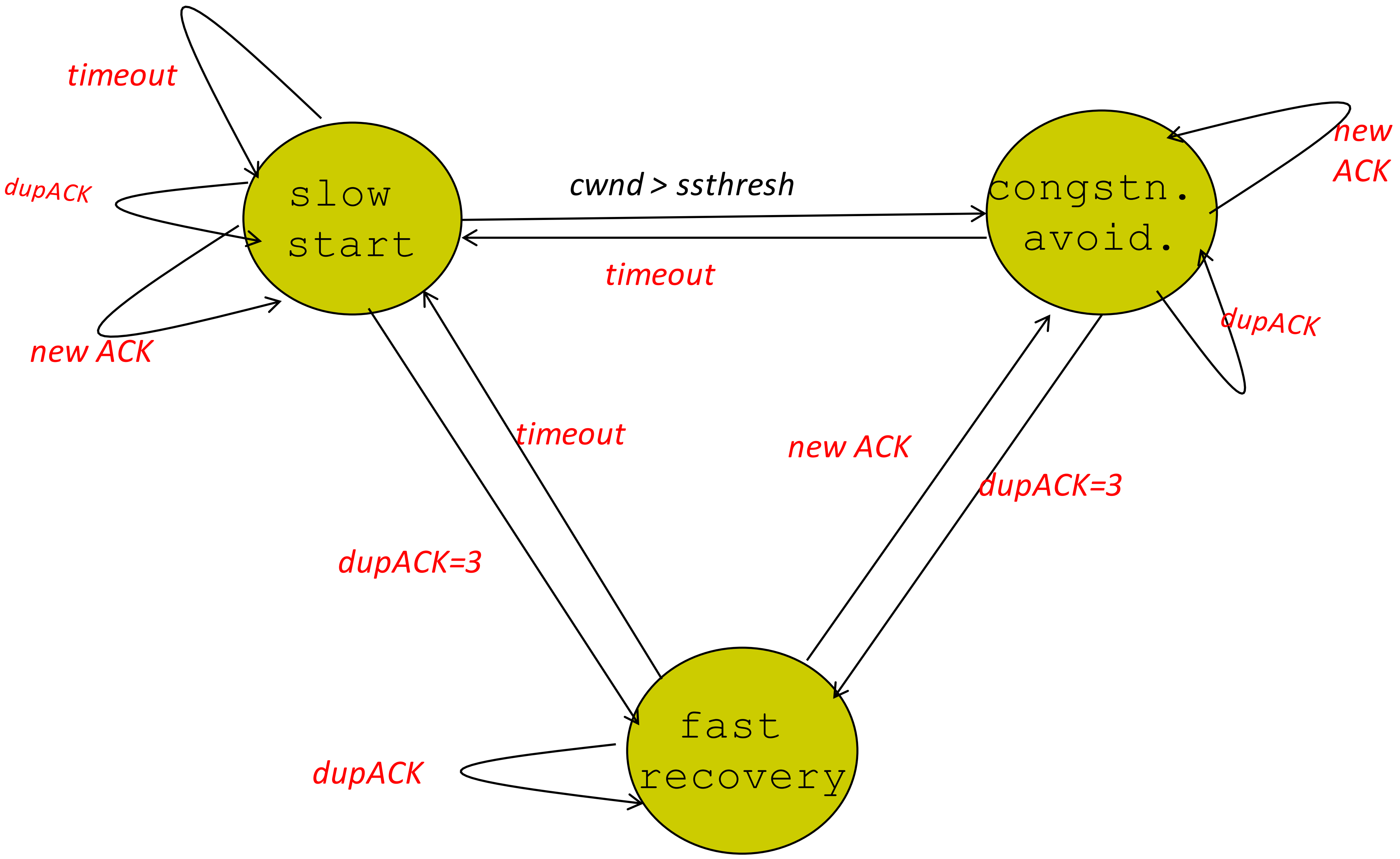
TCP State Machine



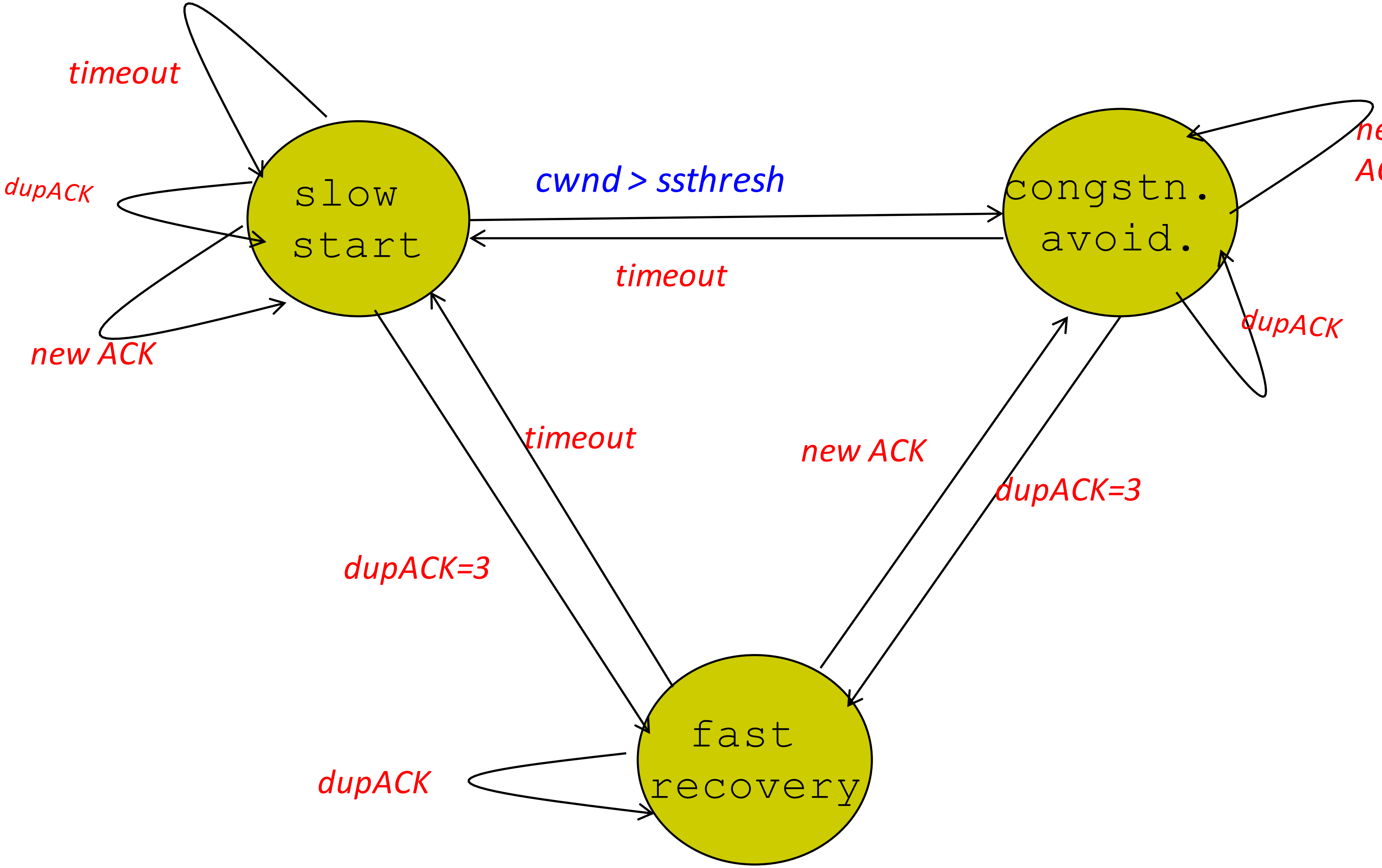
TCP State Machine



TCP State Machine



TCP State Machine



TCP Flavors

- TCP-Tahoe
  - CWND =1 on triple dupACK
- TCP-Reno
  - CWND =1 on timeout
  - CWND = CWND/2 on triple dupack
- TCP-newReno
  - TCP-Reno + improved fast recovery
- TCP-SACK
  - incorporates selective acknowledgements

Our default assumption

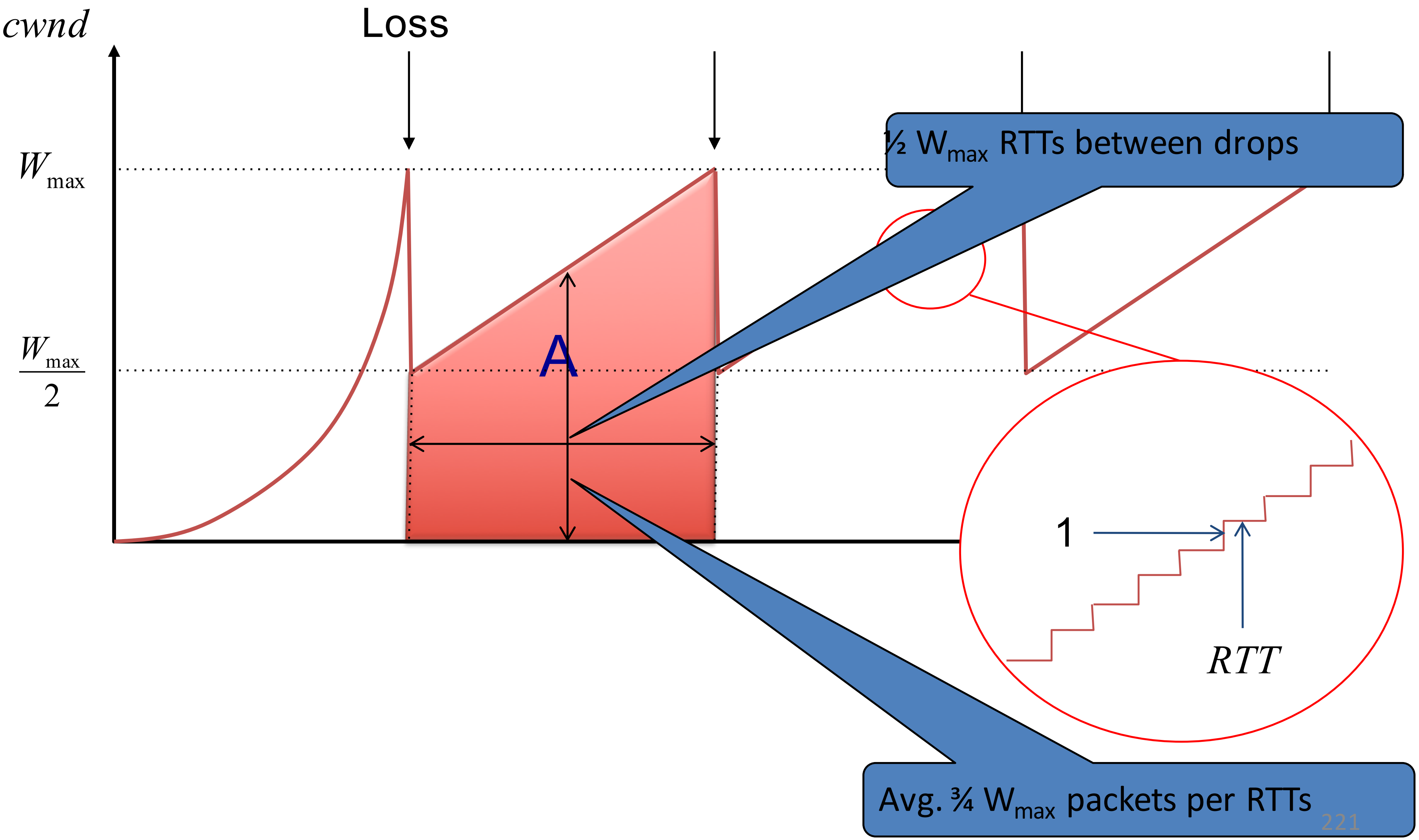
Interoperability

- How can all these algorithms coexist? Don't we need a single, uniform standard?

- What happens if I'm using Reno and you are using Tahoe, and we try to communicate?

TCP Throughput Equation

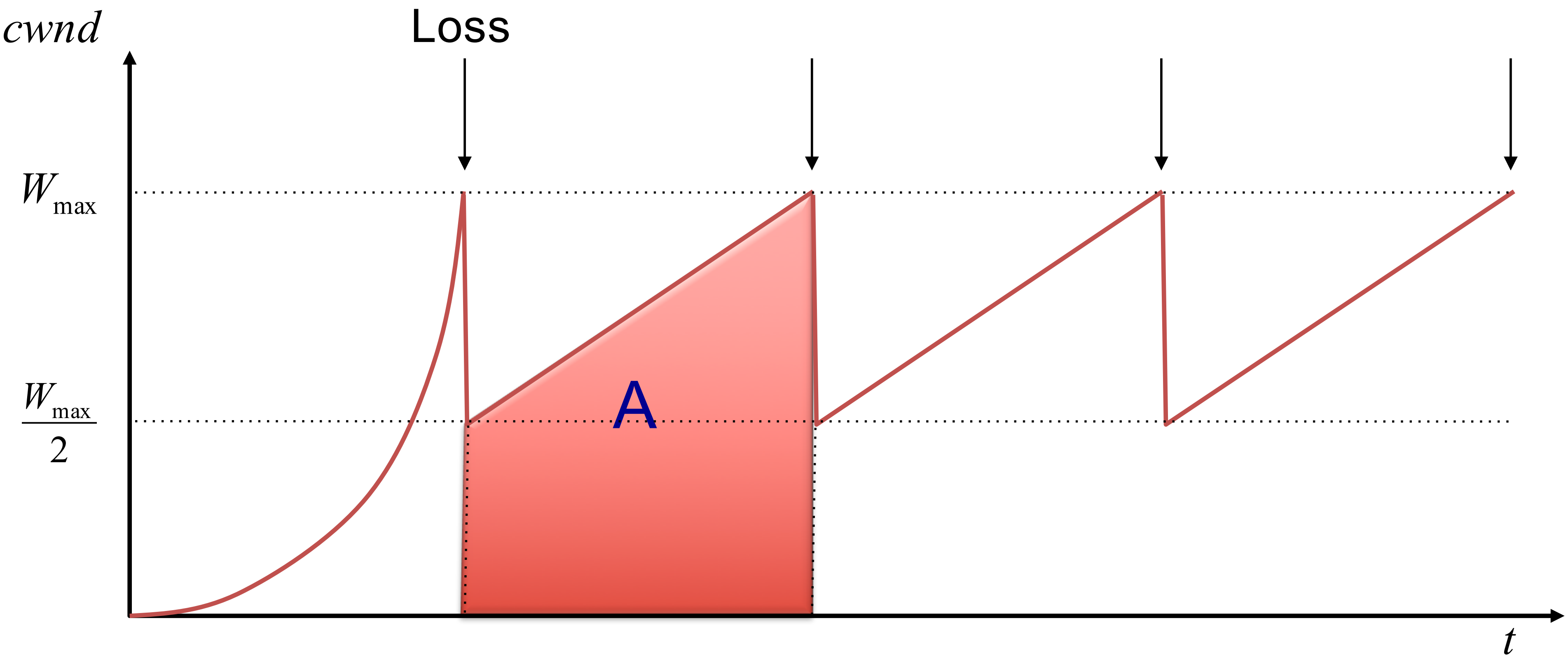
A Simple Model for TCP Throughput



½ W_{max} RTTs between drops

Avg. ¾ W_{max} packets per RTTs

A Simple Model for TCP Throughput



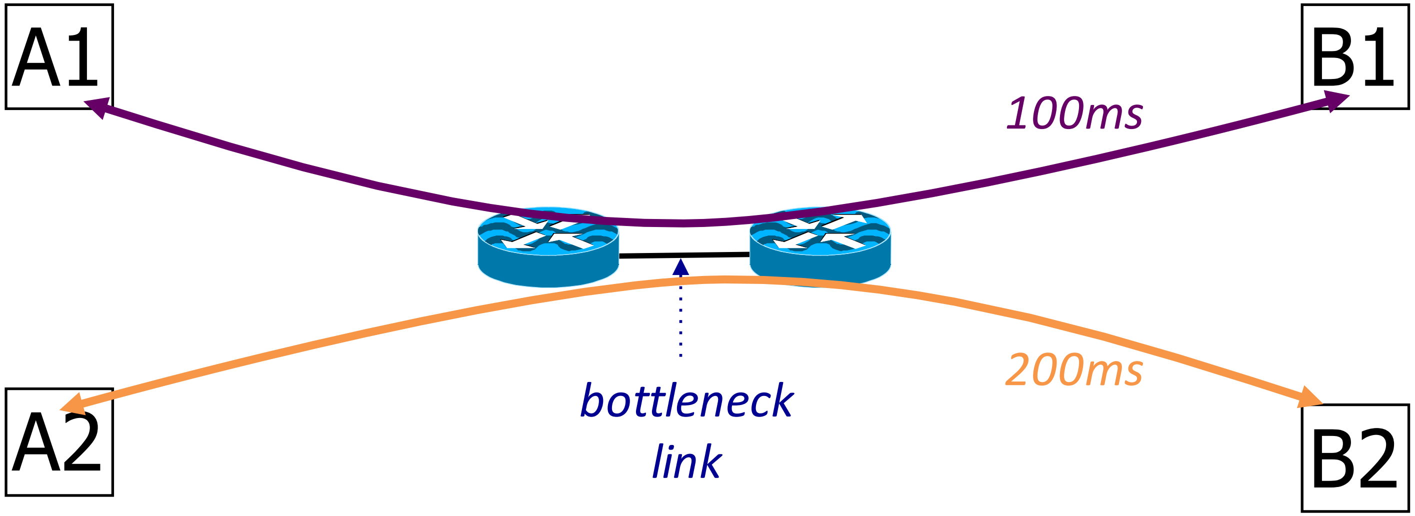
$$\text{Packet drop rate}, p = 1/A, \text{ where } A = \frac{3}{8}W_{\max}^2$$

$$\text{Throughput}, B = \frac{A}{\left(\frac{W_{\max}}{2}\right)RTT} = \sqrt{\frac{3}{2}}\frac{1}{RTT\sqrt{p}}$$

Implications (1): Different RTTs

$$\text{Throughput} = \sqrt{\frac{3}{2}}\frac{1}{RTT\sqrt{p}}$$

- Flows get throughput inversely proportional to RTT
- TCP unfair in the face of heterogeneous RTTs!



Implications (2): High Speed TCP

$$\text{Throughput} = \sqrt{\frac{3}{2}}\frac{1}{RTT\sqrt{p}}$$

- Assume RTT = 100ms, MSS=1500bytes
- What value of *p* is required to reach 100Gbps throughput
  - ~ 2 x 10^{-12}
- How long between drops?
  - ~ 16.6 hours
- How much data has been sent in this time?
  - ~ 6 petabits
- These are not practical numbers!

Adapting TCP to High Speed

- Once past a threshold speed, increase CWND faster
  - A proposed standard [Floyd'03]: once speed is past some threshold, change equation to $p^{-.8}$ rather than $p^{-.5}$
  - Let the additive constant in AIMD depend on CWND

- Other approaches?
  - Multiple simultaneous connections (hack but works today)
  - Router-assisted approaches (will see shortly)

Implications (3): *Rate*-based CC

$$\text{Throughput} = \sqrt{\frac{3}{2}}\frac{1}{RTT\sqrt{p}}$$

- TCP throughput is "choppy"
  - repeated swings between W/2 to W
- Some apps would prefer sending at a steady rate
  - e.g., streaming apps
- A solution: "Equation-Based Congestion Control"
  - ditch TCP's increase/decrease rules and just follow the equation
  - measure drop percentage *p*, and set rate accordingly
- Following the TCP equation ensures we're "TCP friendly"
  - i.e., use no more than TCP does in similar setting

Other Limitations of TCP Congestion Control

(4) Loss not due to congestion?

- TCP will confuse **any loss event** with congestion
- Flow will cut its rate
  - Throughput ~ 1/sqrt(p) where p is loss prob.
  - Applies even for non-congestion losses!
- We'll look at proposed solutions shortly…

(5) How do short flows fare?

- 50% of flows have < 1500B to send; 80% < 100KB

- Implication (1): short flows never leave slow start!
  - short flows never attain their fair share

- Implication (2): too few packets to trigger dupACKs
  - Isolated loss may lead to timeouts
  - At typical timeout values of ~500ms, might severely impact flow completion time

(6) TCP fills up queues → long delays

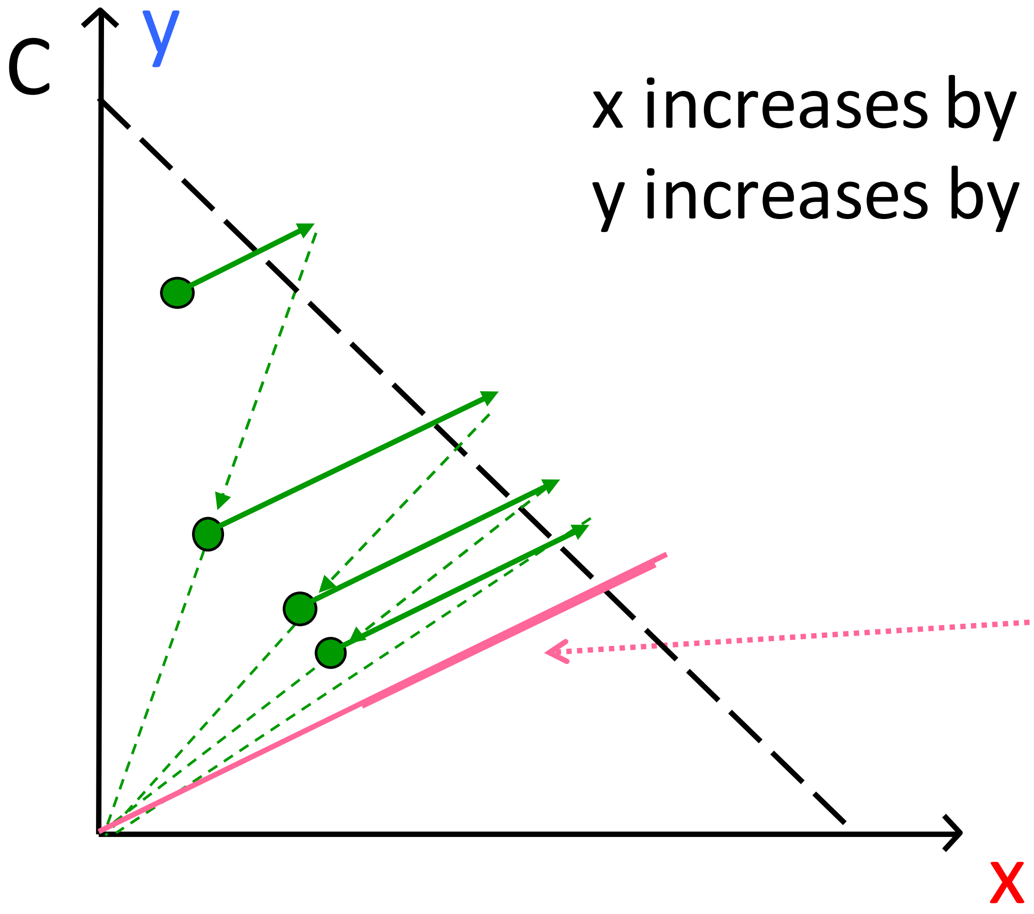
- A flow deliberately overshoots capacity, until it experiences a drop

- Means that delays are large for *everyone*
  - Consider a flow transferring a 10GB file sharing a bottleneck link with 10 flows transferring 100B

(7) Cheating

- Three easy ways to cheat
  - Increasing CWND faster than +1 MSS per RTT

Increasing CWND Faster



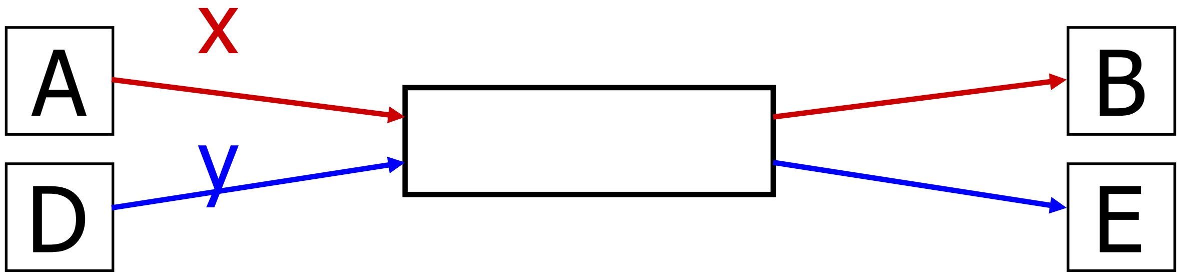
x increases by 2 per RTT
y increases by 1 per RTT

Limit rates:
x = 2y

(7) Cheating

- Three easy ways to cheat
  - Increasing CWND faster than +1 MSS per RTT
  - Opening many connections

Open Many Connections



Assume

- A starts 10 connections to B
- D starts 1 connection to E
- Each connection gets about the same throughput

Then A gets 10 times more throughput than D

(7) Cheating

- Three easy ways to cheat
  - Increasing CWND faster than +1 MSS per RTT
  - Opening many connections
  - Using large initial CWND

- Why hasn’t the Internet suffered a congestion collapse yet?

(8) CC intertwined with reliability

- Mechanisms for CC and reliability are tightly coupled
  - CWND adjusted based on ACKs and timeouts
  - Cumulative ACKs and fast retransmit/recovery rules
- Complicates evolution
  - Consider changing from cumulative to selective ACKs
  - A failure of modularity, not layering
- Sometimes we want CC but not reliability
  - e.g., real-time applications
- Sometimes we want reliability but not CC (?)

236

Recap: TCP problems

- Misled by non-congestion losses
- Fills up queues leading to high delays
- Short flows complete before discovering available capacity
- AIMD impractical for high speed links
- Sawtooth discovery too choppy for some apps
- Unfair under heterogeneous RTTs
- Tight coupling with reliability mechanisms
- Endhosts can cheat

Routers tell endpoints if they're congested

Routers tell endpoints what rate to send at

Routers enforce fair sharing

Could fix many of these with some help from routers!

- What does TCP do?
  - ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - AIMD, Fast-Recovery, Throughput
- Limitations of TCP Congestion Control
- Router-assisted Congestion Control

238

Router-Assisted Congestion Control

- Three tasks for CC:
  - Isolation/fairness
  - Adjustment
  - Detecting congestion

How can routers ensure each flow gets its “fair share”?

Fairness: General Approach

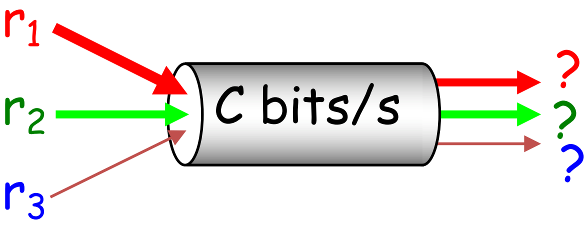
- Routers classify packets into “flows”
  - (For now) flows are packets between same source/destination
- Each flow has its own FIFO queue in router
- Router services flows in a fair fashion
  - When line becomes free, take packet from next flow in a fair order
- What does “fair” mean exactly?

Max-Min Fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:

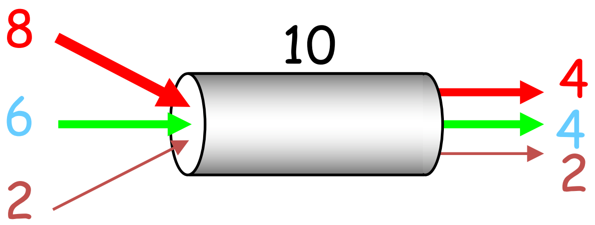
$$a_i = \min(f, r_i)$$

where f is the unique value such that Sum(a_i) = C



Example

- $C = 10$; $r_1 = 8$, $r_2 = 6$, $r_3 = 2$; $N = 3$
- $C/3 = 3.33 \rightarrow$
  - Can service all of r_3
  - Remove r_3 from the accounting: $C = C - r_3 = 8$; $N = 2$
- $C/2 = 4 \rightarrow$
  - Can't service all of r_1 or r_2
  - So hold them to the remaining fair share: $f = 4$



$f = 4$:
min(8, 4) = 4
min(6, 4) = 4
min(2, 4) = 2

Max-Min Fairness

- Given set of bandwidth demands r_i and total bandwidth C, max-min bandwidth allocations are:

$$a_i = \min(f, r_i)$$

- where f is the unique value such that Sum(a_i) = C

- Property:
  - If you don't get full demand, no one gets more than you

- This is what round-robin service gives if all packets are the same size

How do we deal with packets of different sizes?

- Mental model: Bit-by-bit round robin (“fluid flow”)

- Can you do this in practice?

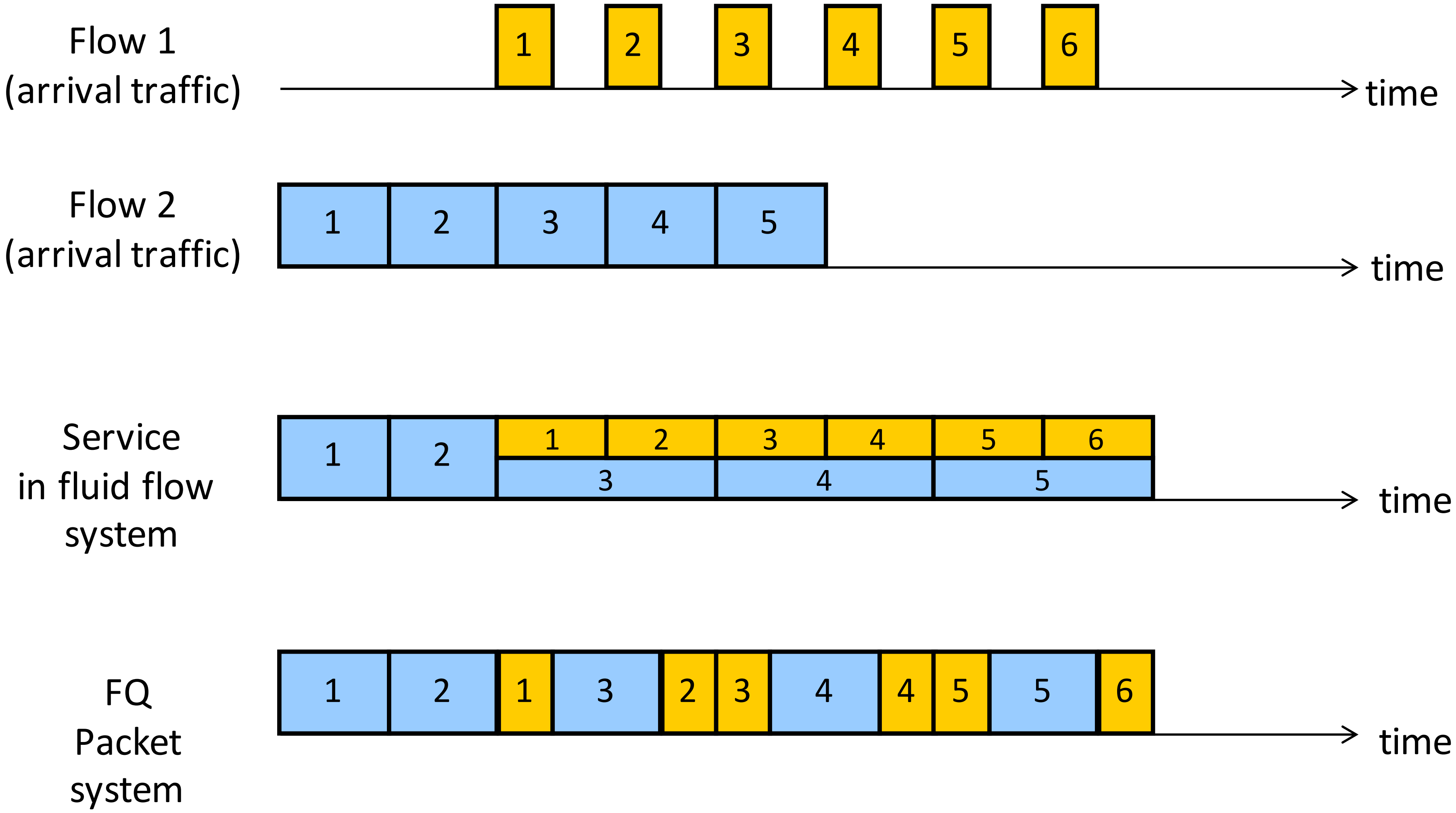
- No, packets cannot be preempted

- But we can approximate it
  - This is what “fair queuing” routers do

Fair Queuing (FQ)

- For each packet, compute the time at which the last bit of a packet would have left the router *if* flows are served bit-by-bit
- Then serve packets in the increasing order of their deadlines

Example


Flow 1
(arrival traffic)
1
2
3
4
5
6
time
Flow 2
(arrival traffic)
1
2
3
4
5
time
Service
in fluid flow
system
1
2
1
2
3
4
5
6
3
4
5
time
FQ
Packet
system
1
2
1
3
2
3
4
4
5
5
6
time

Fair Queuing (FQ)

- Think of it as an implementation of round-robin generalized to the case where not all packets are equal sized

- Weighted fair queuing (WFQ): assign different flows different shares

- Today, some form of WFQ implemented in almost all routers
  - Not the case in the 1980-90s, when CC was being developed
  - Mostly used to isolate traffic at larger granularities (e.g., per-prefix)

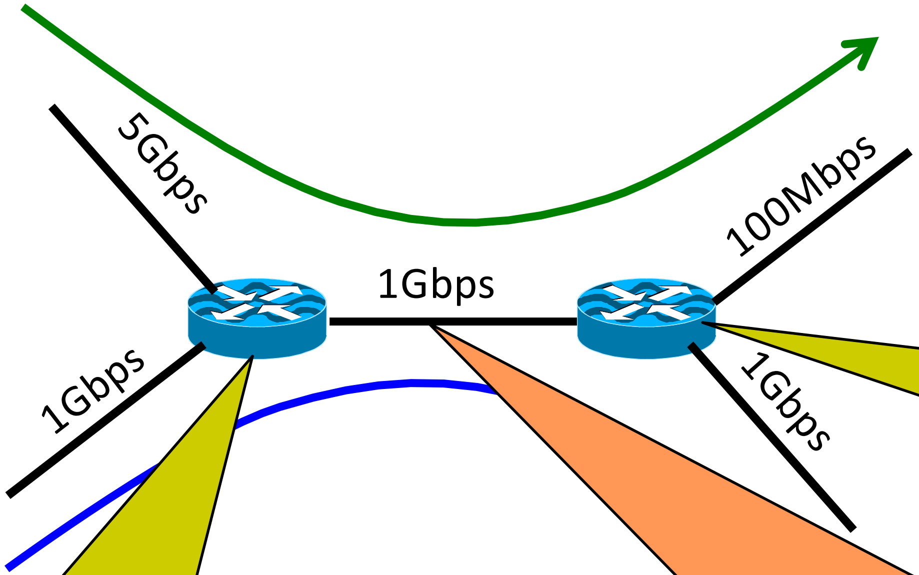
FQ vs. FIFO

- FQ advantages:
  - Isolation: cheating flows don’t benefit
  - Bandwidth share does not depend on RTT
  - Flows can pick any rate adjustment scheme they want

- Disadvantages:
  - More complex than FIFO: per flow queue/state, additional per-packet book-keeping

FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion



Will drop an additional 400Mbps from the green flow

Blue and Green get 0.5Gbps; any excess will be dropped

If the green flow doesn't drop its sending rate to 100Mbps, we're wasting 400Mbps that could be usefully given to the blue flow

FQ in the big picture

- FQ does not eliminate congestion → it just manages the congestion
  - robust to cheating, variations in RTT, details of delay, reordering, retransmission, *etc.*
- But congestion (and packet drops) still occurs
- And we still want end-hosts to discover/adapt to their fair share!
- What would the end-to-end argument say w.r.t. congestion control?

Fairness is a controversial goal

- What if you have 8 flows, and I have 4?
  - Why should you get twice the bandwidth
- What if your flow goes over 4 congested hops, and mine only goes over 1?
  - Why shouldn’t you be penalized for using more scarce bandwidth?
- And what is a flow anyway?
  - TCP connection
  - Source-Destination pair?
  - Source?

Router-Assisted Congestion Control

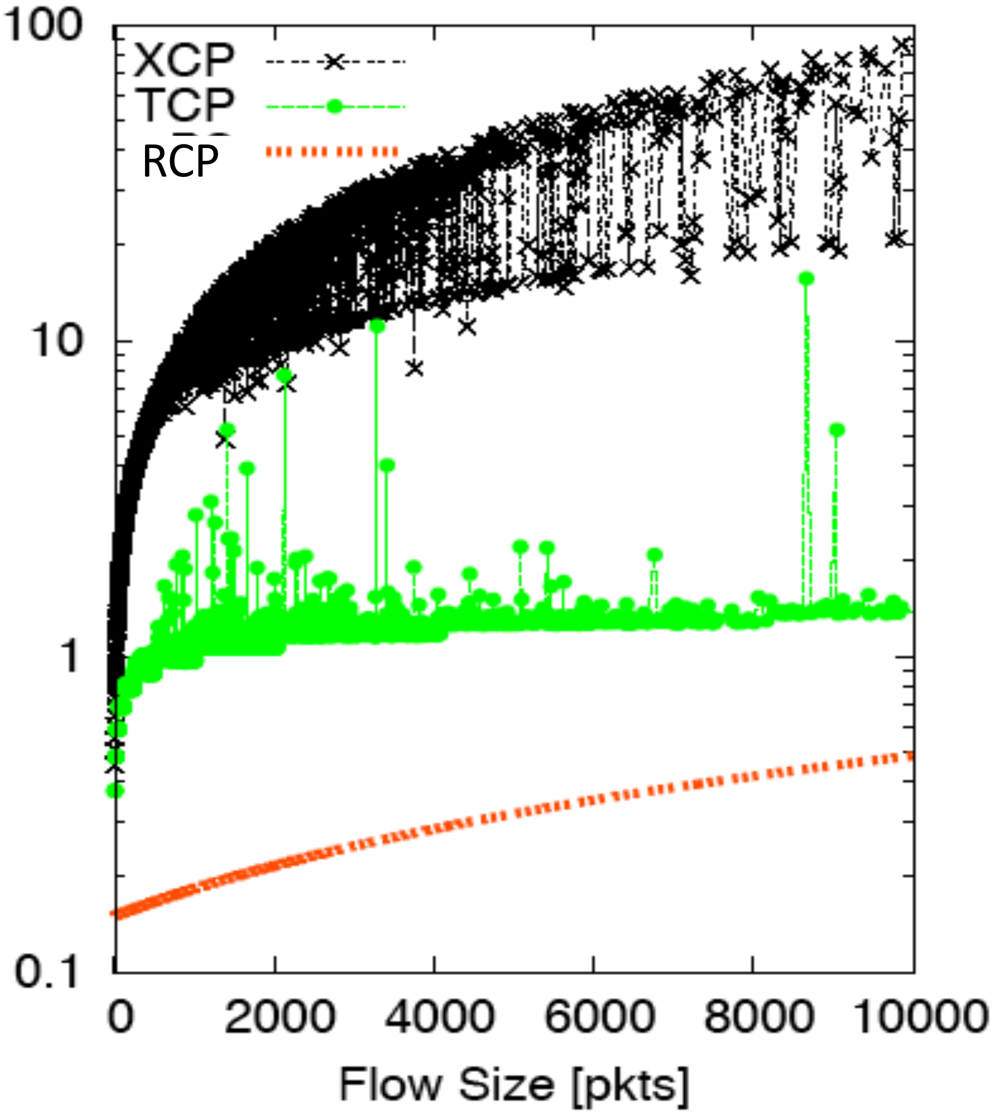
- CC has three different tasks:
  - Isolation/fairness
  - Rate adjustment
  - Detecting congestion

Why not just let routers tell endhosts what rate they should use?

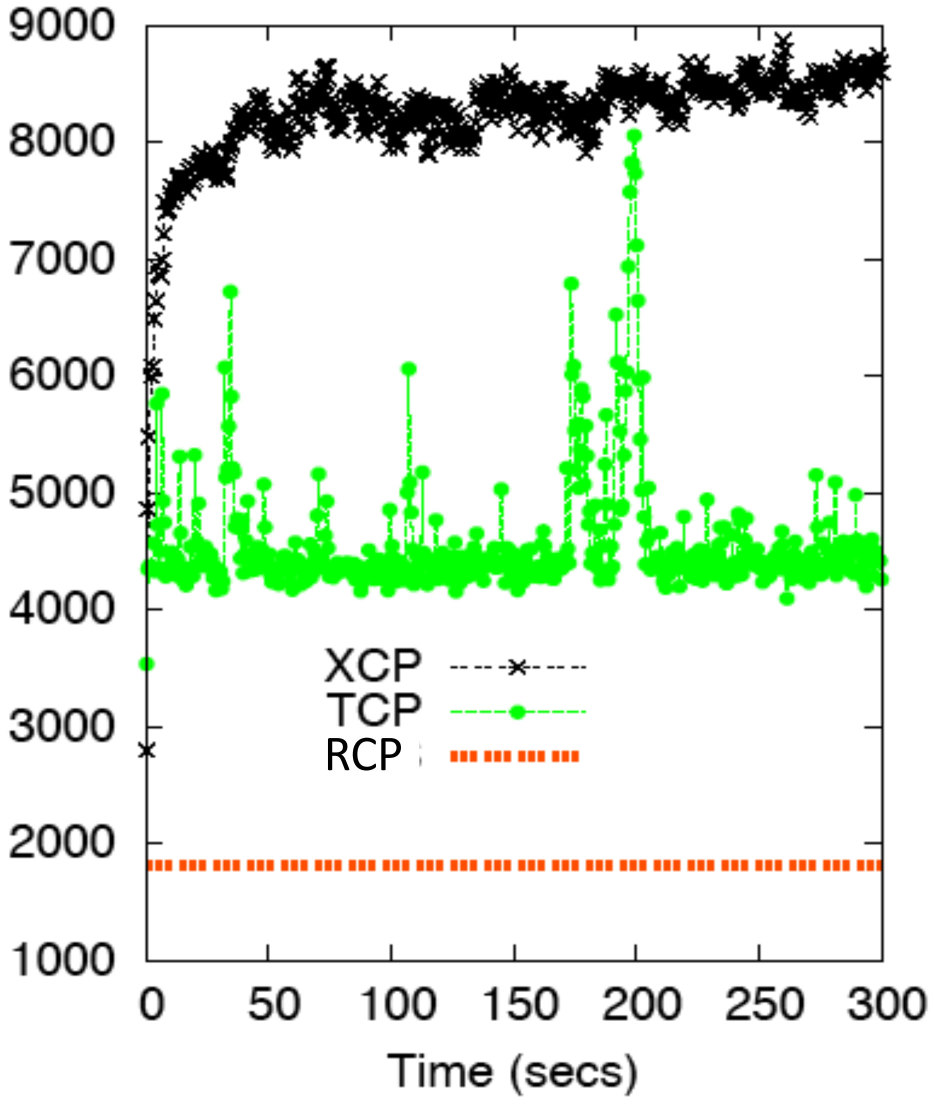
- Packets carry “rate field”
- Routers insert “fair share” *f* in packet header
  - Calculated as with FQ
- End-hosts set sending rate (or window size) to *f*
  - hopefully (still need some policing of endhosts!)
- This is the basic idea behind the “Rate Control Protocol” (RCP) from Dukkipati *et al.* ’07

Flow Completion Time: TCP vs. RCP (Ignore XCP)

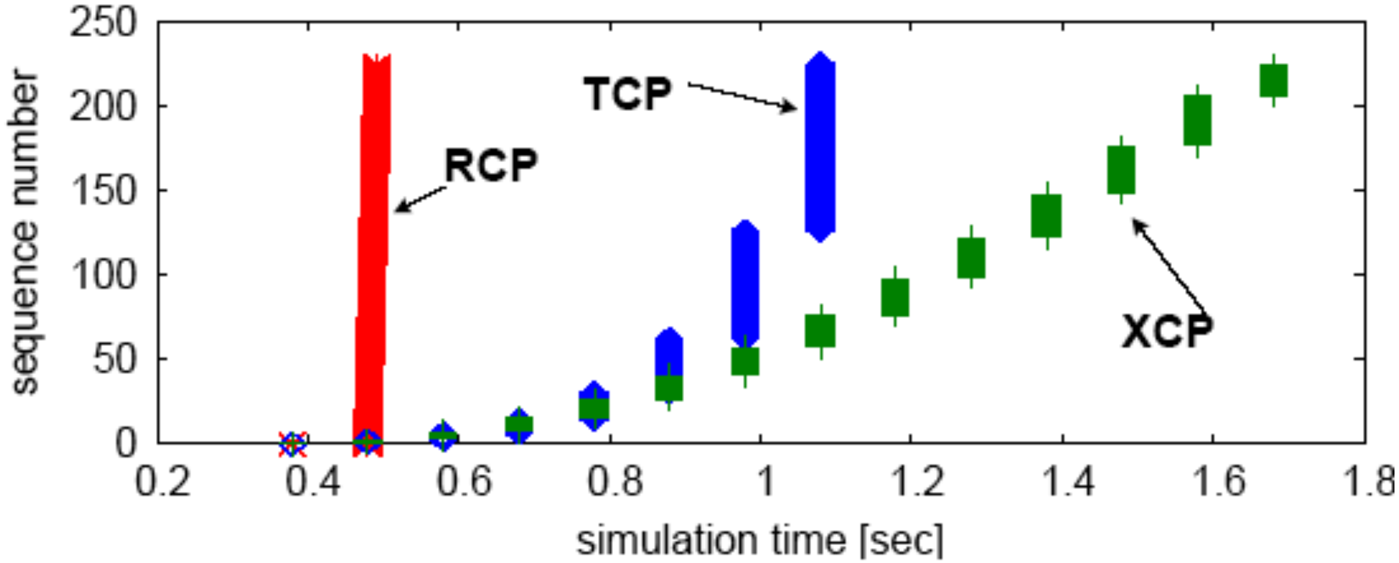
Flow Duration (secs) vs. Flow Size

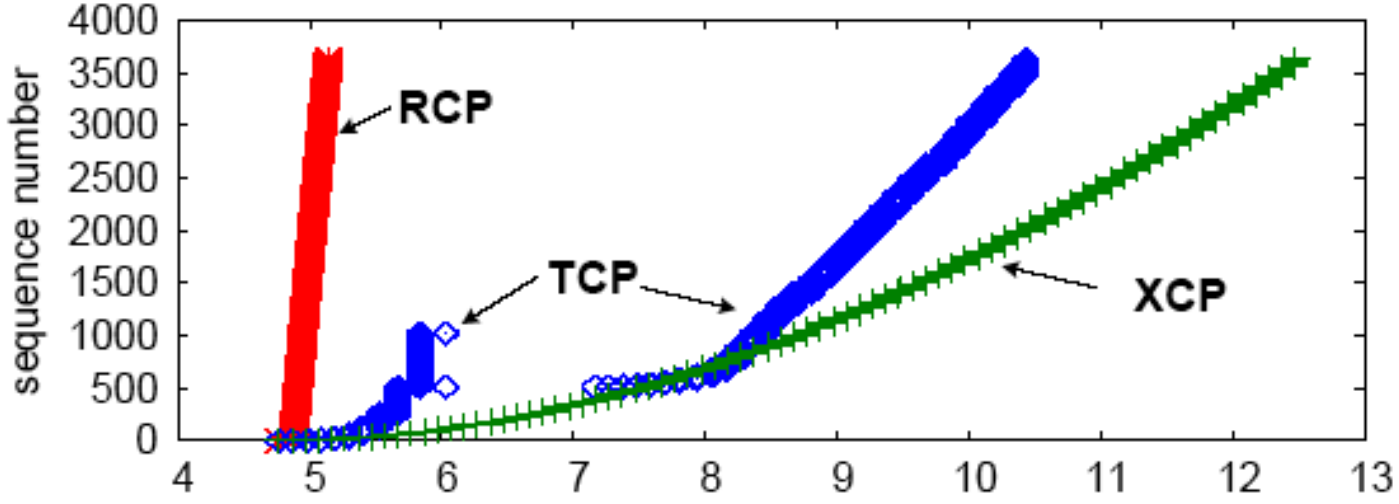


Active Flows vs. time



Why the improvement?





Router-Assisted Congestion Control

- CC has three different tasks:
  - Isolation/fairness
  - Rate adjustment
  - Detecting congestion

Explicit Congestion Notification (ECN)

- Single bit in packet header; set by congested routers
  - If data packet has bit set, then ACK has ECN bit set
- Many options for when routers set the bit
  - tradeoff between (link) utilization and (packet) delay
- Congestion semantics can be exactly like that of drop
  - I.e., endhost reacts as though it saw a drop

- Advantages:
  - Don’t confuse corruption with congestion; recovery w/ rate adjustment
  - Can serve as an early indicator of congestion to avoid delays
  - Easy (easier) to incrementally deploy
    - defined as extension to TCP/IP in RFC 3168 (uses diffserv bits in the IP header)

One final proposal: Charge people for congestion!

- Use ECN as congestion markers
- Whenever I get an ECN bit set, I have to pay $$
- Now, there's no debate over what a flow is, or what fair is…
- Idea started by Frank Kelly here in Cambridge
  - "optimal" solution, backed by much math
  - Great idea: simple, elegant, effective
  - Unclear that it will impact practice – although London congestion works

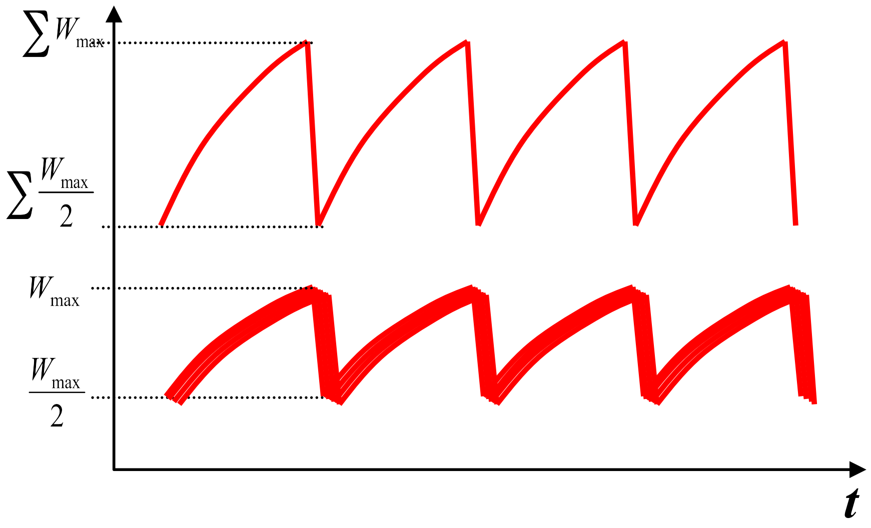
Some TCP issues outstanding…

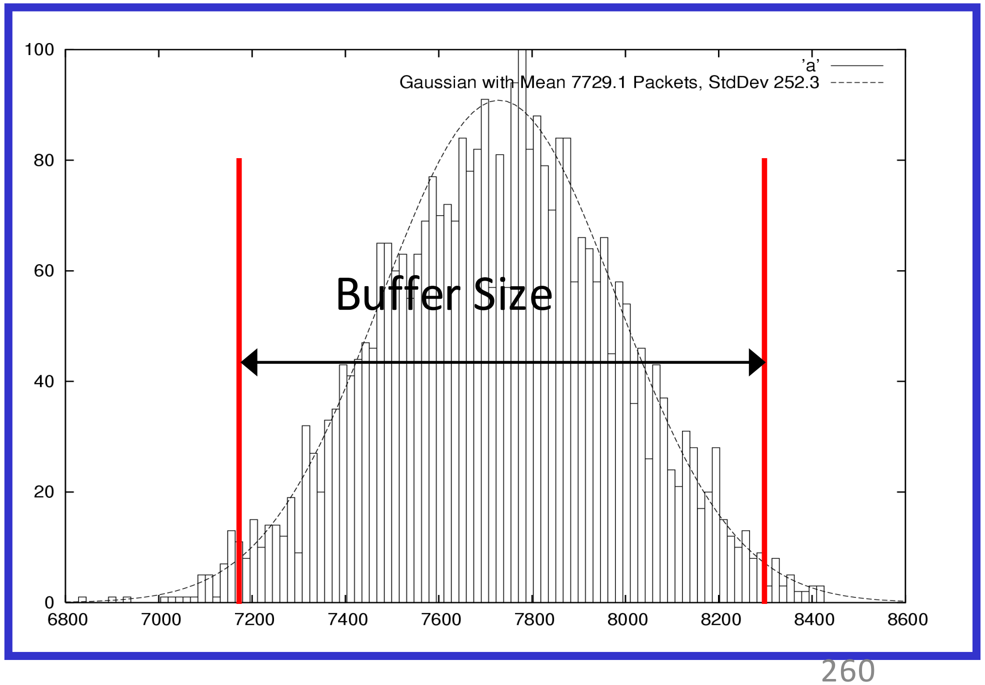
Synchronized Flows

- **Aggregate window has same dynamics**
- **Therefore buffer occupancy has same dynamics**
- **Rule-of-thumb still holds.**

Many TCP Flows

- **Independent, desynchronized**
- **Central limit theorem says the aggregate becomes Gaussian**
- **Variance (buffer size) decreases as *N* increases**





TCP in detail

- What does TCP do?
  - – ARQ windowing, set-up, tear-down
- Flow Control in TCP
- Congestion Control in TCP
  - – AIMD, Fast-Recovery, Throughput
- Limitations of TCP Congestion Control
- Router-assisted Congestion Control

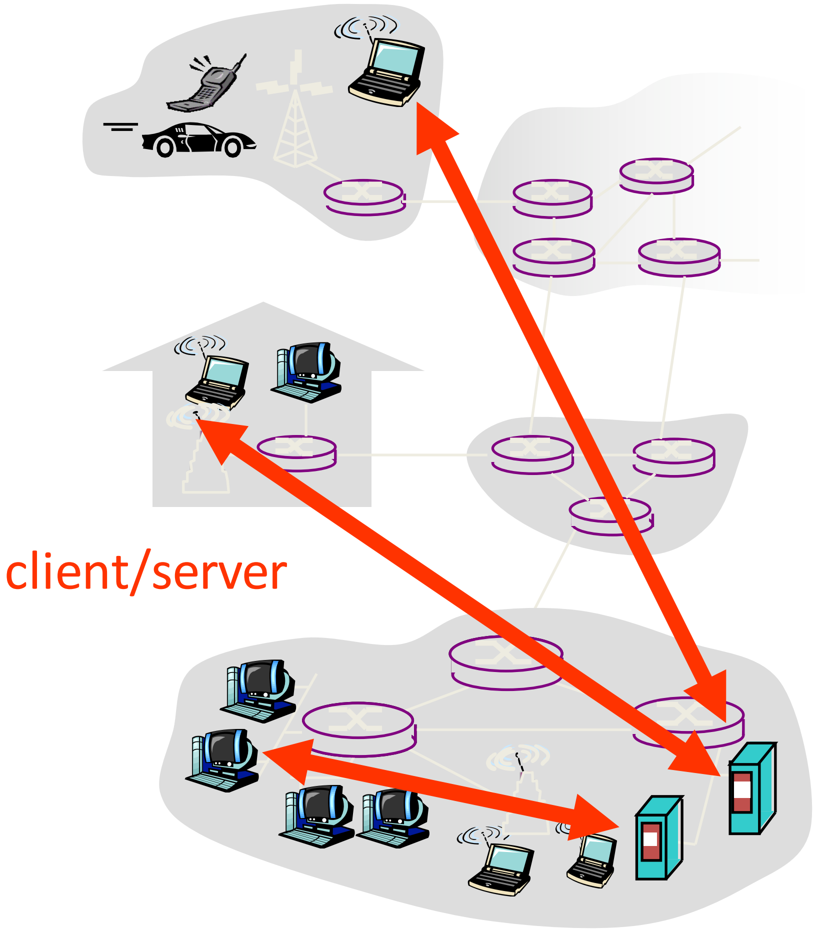
Recap

- TCP:
  - somewhat hacky
  - but practical/deployable
  - good enough to have raised the bar for the deployment of new, more optimal, approaches
  - though the needs of datacenters might change the status quos

Topic 6 – Applications

- Overview
- Traditional Applications (web)
- Infrastructure Services (DNS)
- Multimedia Applications (SIP)
- P2P Networks

Client-server architecture



server:

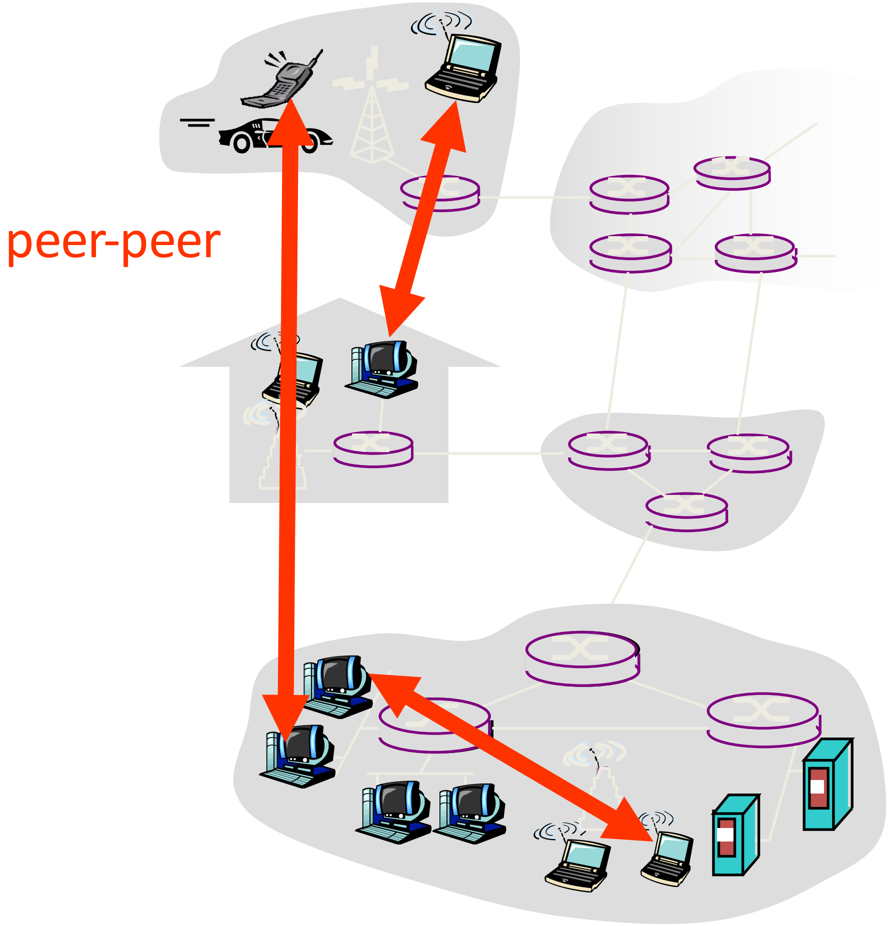
- always-on host
- permanent IP address
- server farms for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses



Highly scalable but difficult to manage

Hybrid of client-server and P2P

Skype

- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging

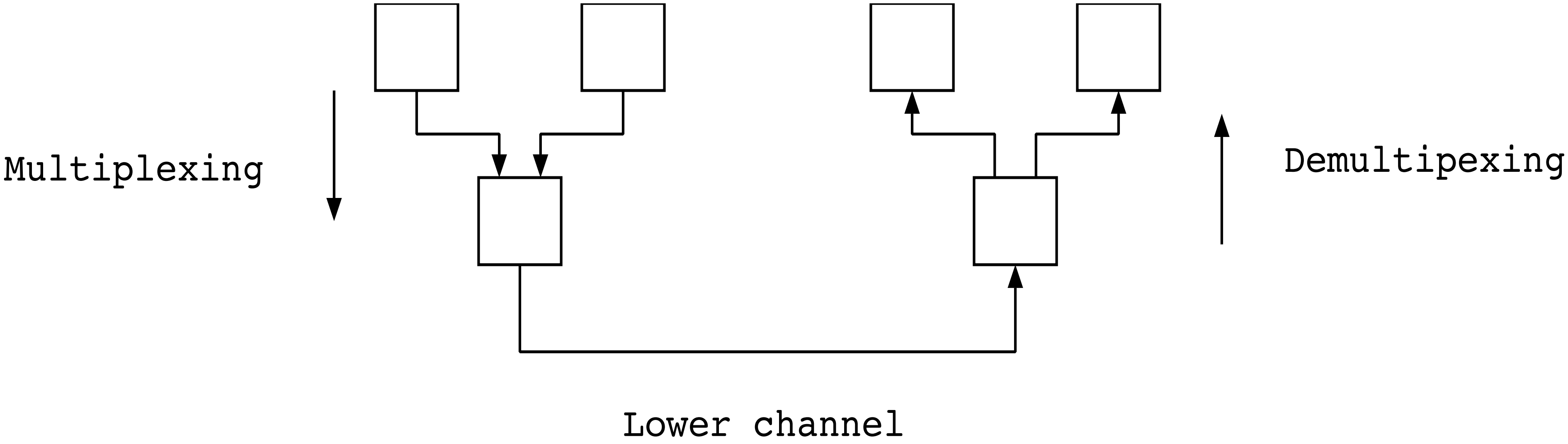
- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* No, *many* processes can be running on same host

- *identifier* includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to yuba.stanford.edu web server:
  - IP address: 171.64.74.58
  - Port number: 80
- more shortly…

Recall: Multiplexing is a service provided by (each) layer too!



Application: one web-server multiple sets of content

Host: one machine multiple services

Network: one physical box multiple addresses (like vns.cl.cam.ac.uk)

….

UNIX: /etc/protocols = examples of different transport-protocols on top of IP

UNIX: /etc/services = examples of different (TCP/UDP) services – by port

(These files are an example of a (static)

App-layer protocol defines

- Types of messages exchanged,
  - e.g., request, response
- Message syntax:
  - what fields in messages & how fields are delineated
- Message semantics
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:

- e.g., Skype

What transport service does an app need?

Data loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Throughput

❒ some apps (e.g., multimedia) require minimum amount of throughput to be "effective"

❒ other apps ("elastic apps") make use of whatever throughput they get

Security

❒ Encryption, data integrity, …

Mysterious secret of *Transport*

- There is more than sort of *transport* layer

Shocked?

I seriously doubt it…

Recall the two most common TCP and UDP

Naming

- Internet has one global system of addressing: IP
  - By explicit design
- And one global system of naming: DNS
  - Almost by accident
- At the time, only items worth naming were hosts
  - A mistake that causes many painful workarounds
- Everything is now named relative to a host
  - Content is most notable example (URL structure)

Logical Steps in Using Internet

- Human has name of entity she wants to access
  - Content, host, etc.
- Invokes an application to perform relevant task
  - Using that name
- App invokes DNS to translate name to address
- App invokes transport protocol to contact host
  - Using address as destination

Addresses vs Names

- Scope of relevance:
  - App/user is primarily concerned with names
  - Network is primarily concerned with addresses
- Timescales:
  - Name lookup once (or get from cache)
  - Address lookup on each packet
- When moving a host to a different subnet:
  - The address changes
  - The name does not change
- When moving content to a differently named host
  - Name and address both change!

Relationship Between Names&Addresses

- Addresses can change underneath
  - Move www.bbc.co.uk to 212.58.246.92
  - Humans/Apps should be unaffected

- Name could map to multiple IP addresses
  - www.bbc.co.uk to multiple replicas of the Web site
  - Enables
    - Load-balancing
    - Reducing latency by picking nearby servers

- Multiple names for the same address
  - E.g., aliases like www.bbc.co.uk and bbc.co.uk
  - Mnemonic stable name, and dynamic canonical name
    - Canonical name = actual name of host

Mapping from Names to Addresses

- Originally: per-host file /etc/hosts
  - SRI (Menlo Park) kept master copy
  - Downloaded regularly
  - Flat namespace

- Single server not resilient, doesn’t scale
  - Adopted a distributed hierarchical system

- Two intertwined hierarchies:
  - Infrastructure: hierarchy of DNS servers
  - Naming structure: www.bbc.co.uk

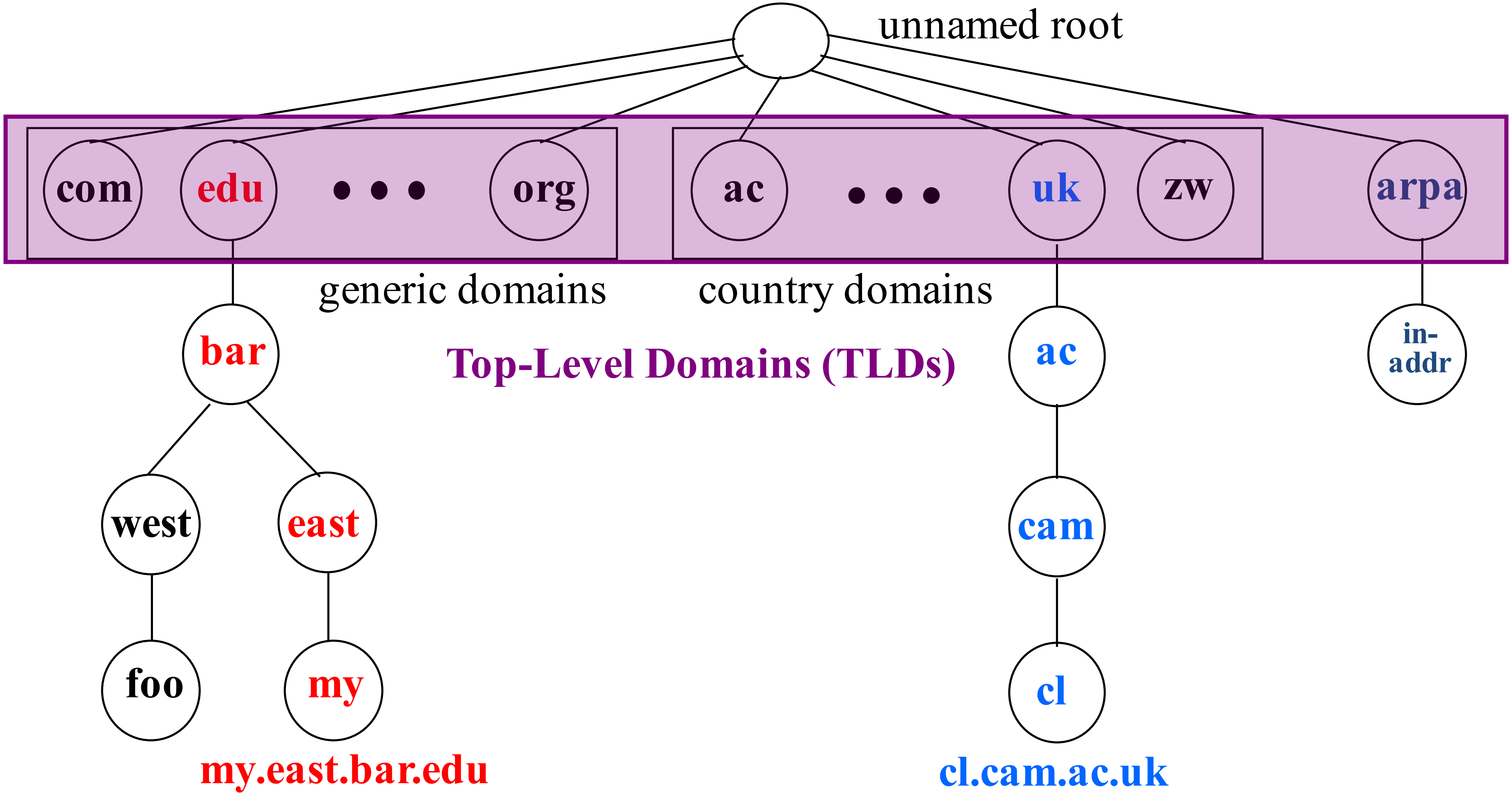
Domain Name System (DNS)

- Top of hierarchy: Root
  - Location hardwired into other servers

- Next Level: Top-level domain (TLD) servers
  - .com, .edu, etc.
  - .uk, .au, .to, etc.
  - Managed professionally

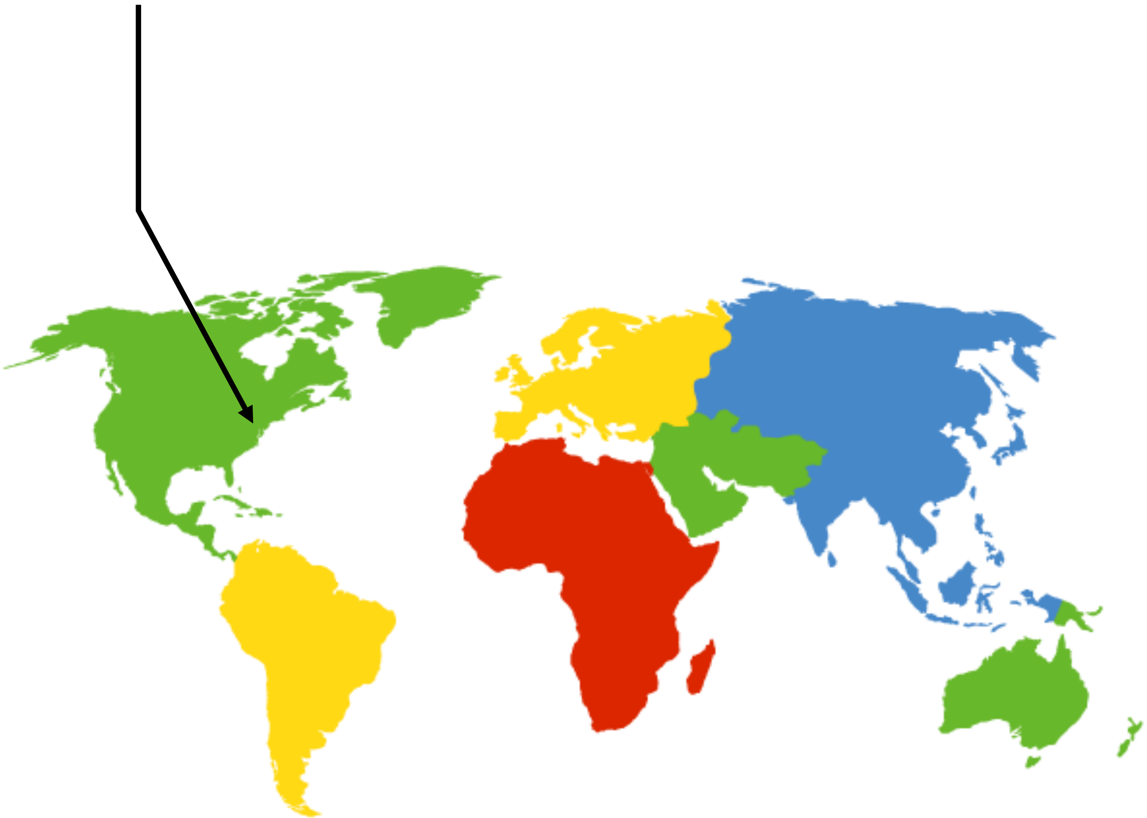
- Bottom Level: Authoritative DNS servers
  - Actually do the mapping
  - Can be maintained locally or by a service provider

Distributed Hierarchical Database



DNS Root

- Located in Virginia, USA
- How do we make the root scale?



DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
  - Labeled A through M
- Does this scale?

A Verisign, Dulles, VA
C Cogent, Herndon, VA
D U Maryland College Park, MD
G US DoD Vienna, VA
H ARL Aberdeen, MD
J Verisign

K RIPE London

I Autonomica, Stockholm

E NASA Mt View, CA
F Internet Software
Consortium
Palo Alto, CA

M WIDE Tokyo

B USC-ISI Marina del Rey, CA
L ICANN Los Angeles, CA

DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
  - Labeled A through M
- Replication via any-casting (localized routing for addresses)

A Verisign, Dulles, VA
C Cogent, Herndon, VA (also Los Angeles, NY, Chicago)
D U Maryland College Park, MD
G US DoD Vienna, VA
H ARL Aberdeen, MD
J Verisign (21 locations)

K RIPE London (plus 16 other locations)

I Autonomica, Stockholm (plus 29 other locations)

E NASA Mt View, CA
F Internet Software Consortium, Palo Alto, CA
(and 37 other locations)

M WIDE Tokyo plus Seoul, Paris, San Francisco

B USC-ISI Marina del Rey, CA
L ICANN Los Angeles, CA

Using DNS

- Two components
  - Local DNS servers
  - Resolver software on hosts
- Local DNS server (“default name server”)
  - Usually near the endhosts that use it
  - Local hosts configured with local server (e.g., /etc/resolv.conf) or learn server via DHCP
- Client application
  - Extract server name (e.g., from the URL)
  - Do gethostbyname() to trigger resolver code

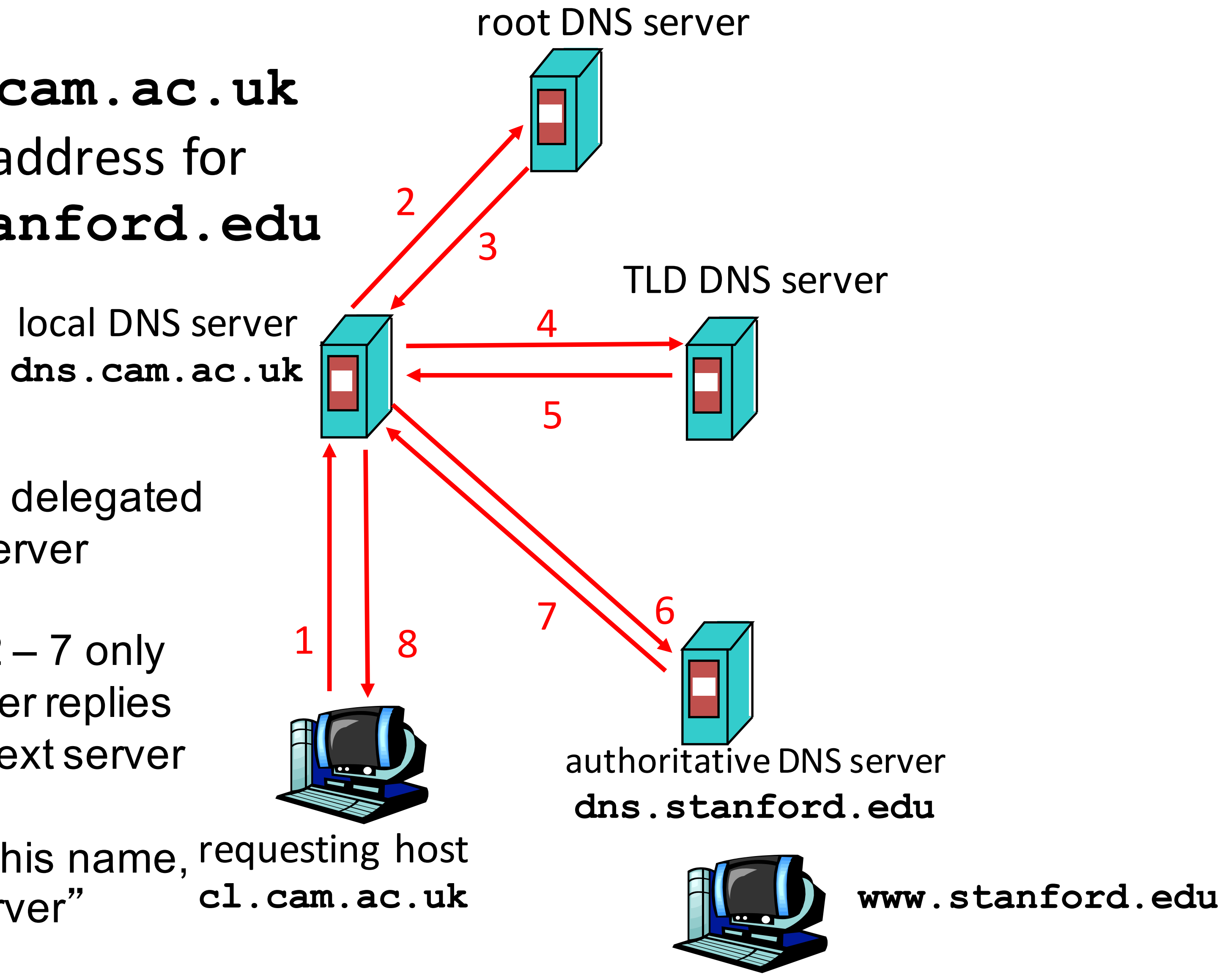
How Does Resolution Happen?

(**Iterative** example)

Host at **`cl.cam.ac.uk`**

wants IP address for

`www.stanford.edu`



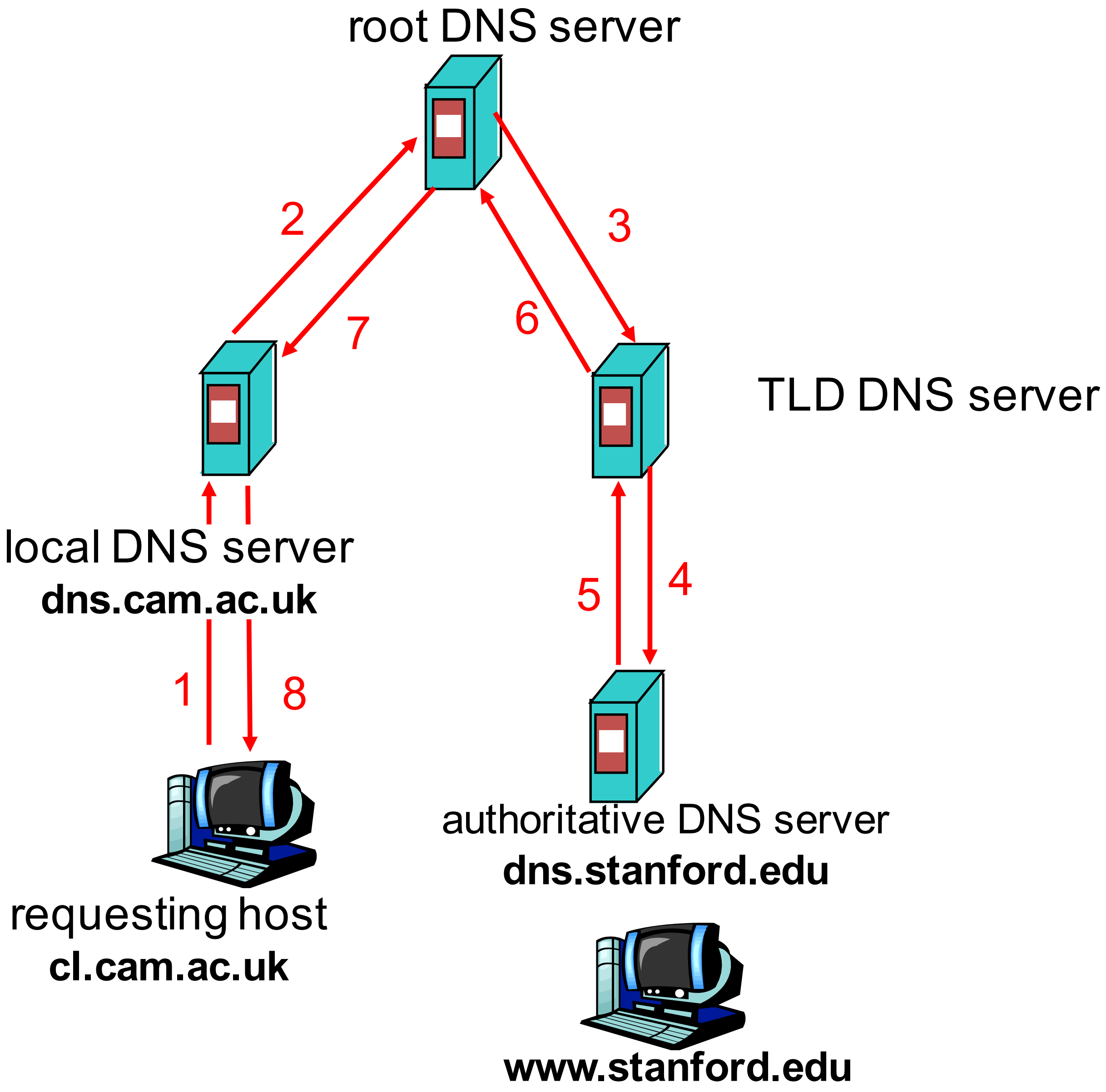
iterated query:

- Host enquiry is delegated to local DNS server
- Consider transactions 2 – 7 only
- contacted server replies with name of next server to contact
- “I don’t know this name, but ask this server”

DNS name resolution **recursive** example

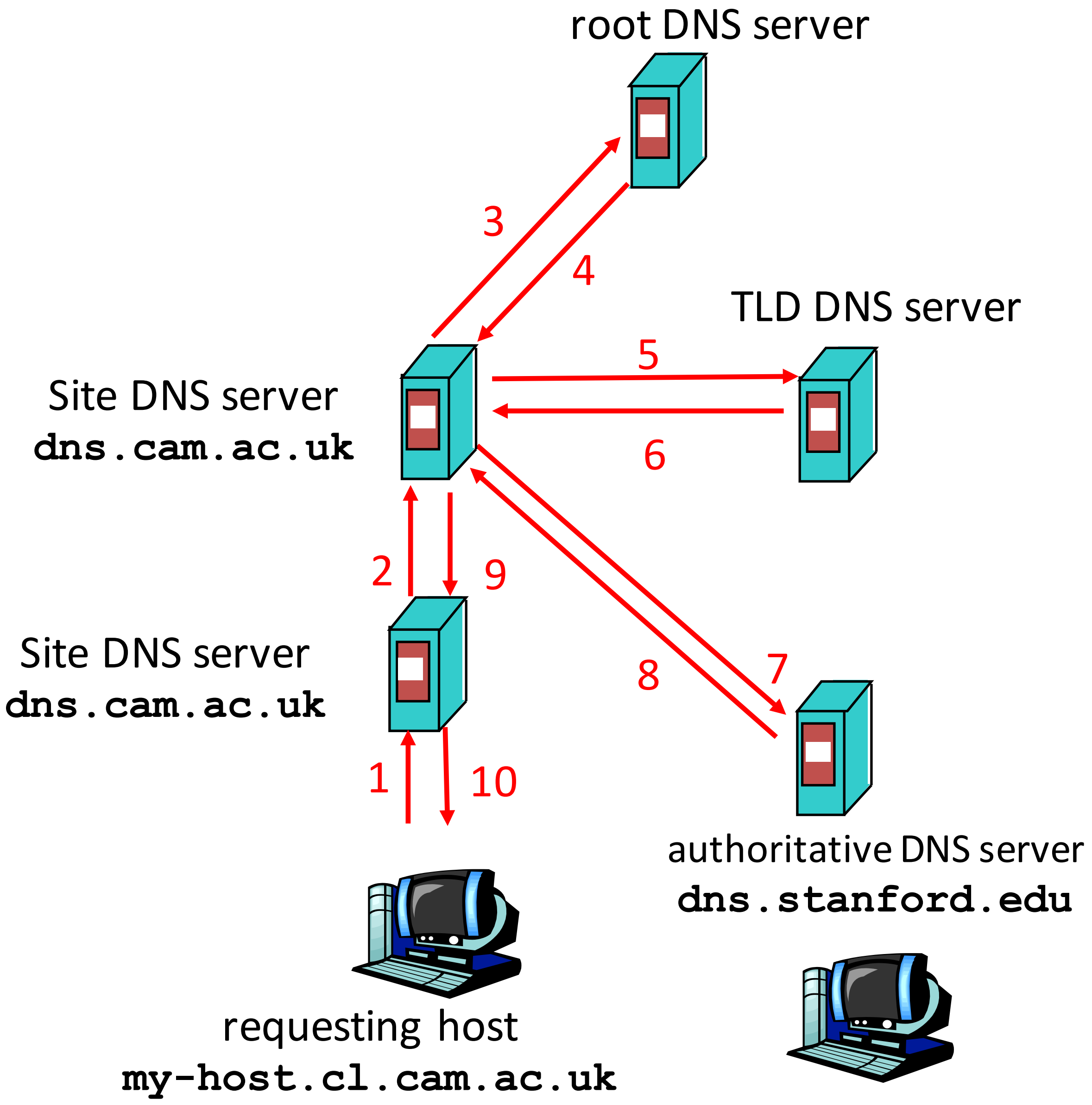
recursive query:

- ❒ puts burden of name resolution on contacted name server
- ❒ heavy load?



Recursive and Iterative Queries - **Hybrid** case

- Recursive query
  - Ask server to get answer for you
  - E.g., requests 1,2 and responses 9,10
- Iterative query
  - Ask server who to ask next
  - E.g., all other request-response pairs



DNS Caching

- Performing all these queries takes time
  - And all this before actual communication takes place
  - E.g., 1-second latency before starting Web download
- Caching can greatly reduce overhead
  - The top-level servers very rarely change
  - Popular sites (e.g., www.bbc.co.uk) visited often
  - Local DNS server often has the information cached
- How DNS caching works
  - DNS servers cache responses to queries
  - Responses include a “time to live” (TTL) field
  - Server deletes cached entry after TTL expires

Negative Caching

- Remember things that don' t work
  - Misspellings like *bbcc.co.uk* and *www.bbc.com.uk*
  - These can take a long time to fail the first time
  - Good to remember that they don’t work
  - … so the failure takes less time the next time around

- But: negative caching is optional
  - And not widely implemented

Reliability

- DNS servers are replicated (primary/secondary)
  - Name service available if at least one replica is up
  - Queries can be load-balanced between replicas
- Usually, UDP used for queries
  - Need reliability: must implement this on top of UDP
  - Spec supports TCP too, but not always implemented
- Try alternate servers on timeout
  - Exponential backoff when retrying same server
- Same identifier for all queries
  - Don’t care which server responds

DNS Measurements (MIT data from 2000)

- What is being looked up?
  - ~60% requests for A records
  - ~25% for PTR records
  - ~5% for MX records
  - ~6% for ANY records
- How long does it take?
  - Median ~100msec (but 90th percentile ~500msec)
  - 80% have no referrals; 99.9% have fewer than four
- Query packets per lookup: ~2.4
  - But this is misleading….

DNS Measurements (MIT data from 2000)

- Does DNS give answers?
  - ~23% of lookups fail to elicit an answer!
  - ~13% of lookups result in NXDOMAIN (or similar)
    - Mostly reverse lookups
  - Only ~64% of queries are successful!
    - *How come the web seems to work so well?*

- ~ 63% of DNS packets in unanswered queries!
  - Failing queries are frequently retransmitted
  - 99.9% successful queries have ≤2 retransmissions

DNS Measurements (MIT data from 2000)

- Top 10% of names accounted for ~70% of lookups
  - Caching should really help!
- 9% of lookups are unique
  - Cache hit rate can never exceed 91%
- Cache hit rates ~ 75%
  - But caching for more than 10 hosts doesn’t add much

A Common Pattern…..

- Distributions of various metrics (file lengths, access patterns, etc.) often have two properties:
  - Large fraction of total metric in the top 10%
  - Sizable fraction (~10%) of total fraction in low values
- Not an exponential distribution
  - Large fraction is in top 10%
  - But low values have very little of overall total
- Lesson: have to pay attention to both ends of dist.
- Here: caching helps, but not a panacea

Moral of the Story

- If you design a highly resilient system, many things can be going wrong without you noticing it!

and this is a **good** thing

Cache Poisoning, an **old** *badness* example

- Suppose you are a Bad Guy and you control the name server for foobar.com. You receive a request to resolve www.foobar.com and reply:

```
;; QUESTION SECTION:
;www.foobar.com.              IN     A

;; ANSWER SECTION:
www.foobar.com.       300     IN     A      212.44.9.144

;; AUTHORITY SECTION:
foobar.com.           600     IN     NS     dns1.foobar.com.
foobar.com.           600     IN     NS     google.com.

;; ADDITIONAL SECTION:
google.com.             5     IN     A      212.44.9.155
```

Evidence of the attack disappears 5 seconds later!

A foobar.com machine, *not* google.com

DNS and Security

- No way to verify answers
  - Opens up DNS to many potential attacks
  - DNSSEC fixes this
- Most obvious vulnerability: recursive resolution
  - Using recursive resolution, host must trust DNS server
  - When at Starbucks, server is under their control
  - And can return whatever values it wants
- More subtle attack: Cache poisoning
  - Those “additional” records can be anything!

Why is the web so successful?

- What do the web, youtube, facebook, tumblr, twitter, flickr, ….. have in common?
  - The ability to self-publish
- Self-publishing that is easy, independent, *free*
- No interest in collaborative and idealistic endeavor
  - People aren't looking for Nirvana (or even Xanadu)
  - People also aren't looking for technical perfection
- Want to make their mark, and find something neat
  - Two sides of the same coin, creates synergy
  - "Performance" more important than dialogue….

Web Components

- Infrastructure:
  - Clients
  - Servers
  - Proxies

- Content:
  - Individual objects (files, etc.)
  - Web sites (coherent collection of objects)

- Implementation
  - HTML: formatting content
  - URL: naming content
  - HTTP: protocol for exchanging content

    Any content not just HTML!

HTML: HyperText Markup Language

- A *Web page* has:
  - Base HTML file
  - Referenced objects (*e.g.*, images)

- HTML has several functions:
  - Format text
  - Reference images
  - Embed *hyperlinks* (HREF)

URL Syntax

protocol : //hostname[:port]/directorypath /resource

protocol	http, ftp, https, smtp, rtsp, *etc*.
hostname	DNS name, IP address
port	Defaults to protocol's standard port *e.g.* http: 80 https: 443
directory path	Hierarchical, reflecting file system
resource	Identifies the desired resource Can also extend to program executions: `http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b`

HyperText Transfer Protocol (HTTP)

- Request-response protocol
- Reliance on a global namespace
- Resource *metadata*
- *Stateless*
- ASCII format

```
$ telnet www.cl.cam.ac.uk 80
GET /~awm22/win HTTP/1.0
<blank line, i.e., CRLF>
```

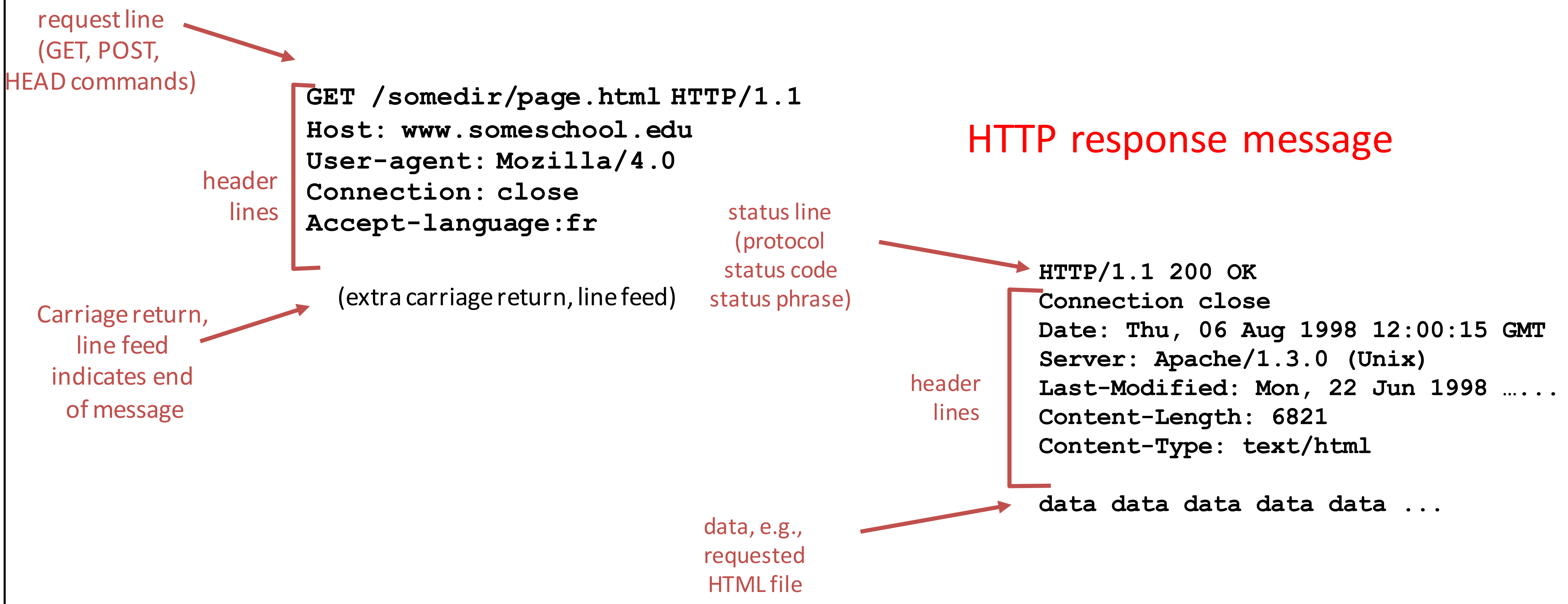
Steps in HTTP Request

- HTTP Client initiates TCP connection to server
  - SYN
  - SYNACK
  - ACK
- Client sends HTTP request to server
  - Can be piggybacked on TCP's ACK
- HTTP Server responds to request
- Client receives the request, terminates connection
- TCP connection termination exchange

How many RTTs for a single request?

Client-Server Communication

- two types of HTTP messages: *request, response*
- HTTP request message: (GET POST HEAD ….)



request line (GET, POST, HEAD commands)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

HTTP response message

status line (protocol status code status phrase)

```
HTTP/1.1 200 OK
Connection close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …....
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

header lines

data, e.g., requested HTML file

Different Forms of Server Response

- Return a file
  - URL matches a file (*e.g.,* `/www/index.html`)
  - Server returns file as the response
  - Server generates appropriate response header
- Generate response dynamically
  - URL triggers a program on the server
  - Server runs program and sends output to client
- Return meta-data with no body

HTTP Resource Meta-Data

- Meta-data
  - Info *about* a resource, stored as a separate entity
- Examples:
  - Size of resource, last modification time, type of content
- Usage example: Conditional GET Request
  - Client requests object "**`If-modified-since`**"
  - If unchanged, "**`HTTP/1.1 304 Not Modified`**"
  - No body in the server's response, only a header

HTTP is *Stateless*

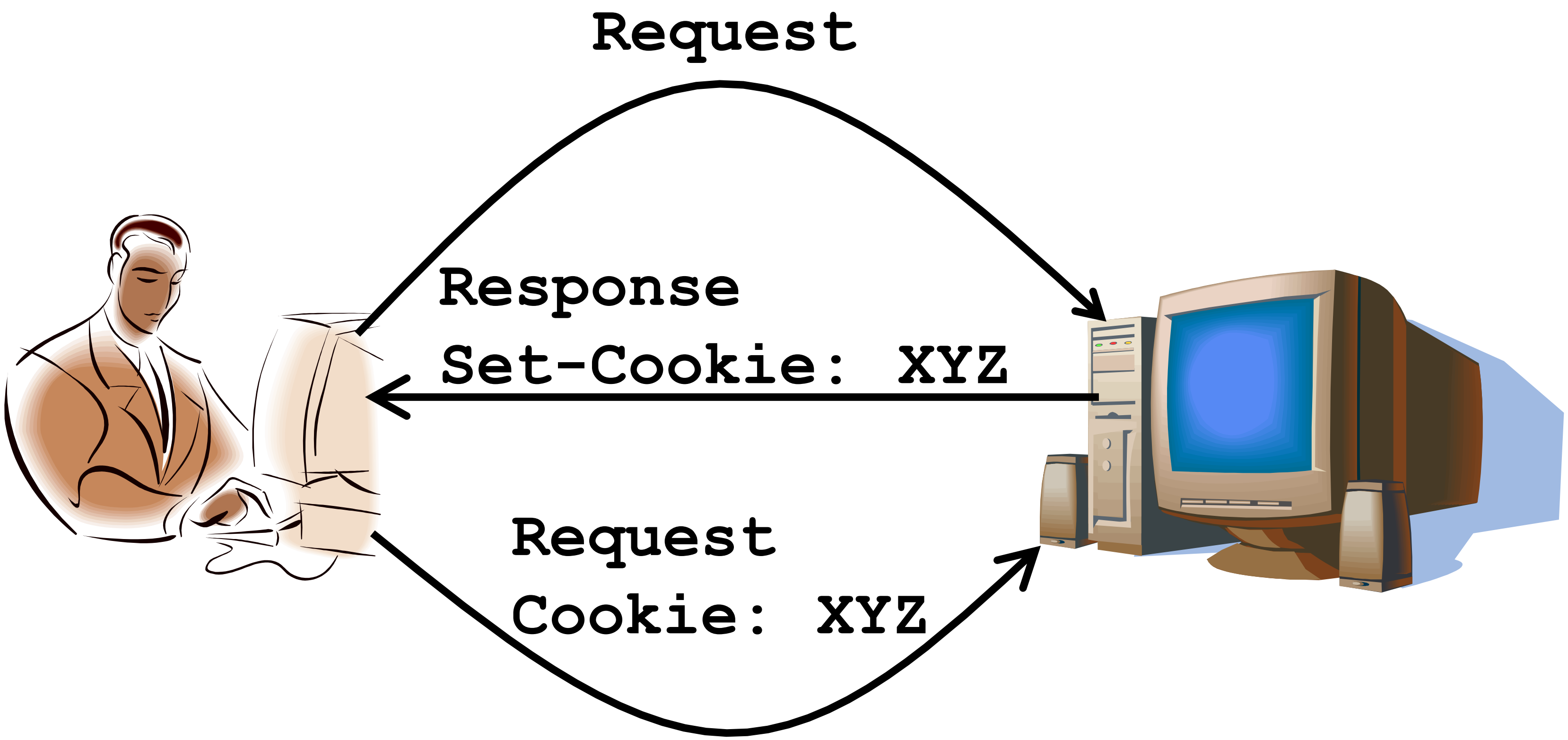
- Each request-response treated independently
  - Servers *not* required to retain state

- **Good**: Improves scalability on the server-side
  - Failure handling is easier
  - Can handle higher rate of requests
  - Order of requests doesn‘t matter

- **Bad**: Some applications need persistent state
  - Need to uniquely identify user or store temporary info
  - *e.g.,* Shopping cart, user profiles, usage tracking, …

State in a Stateless Protocol: Cookies

- *Client-side* state maintenance
  - Client stores small(?) state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication



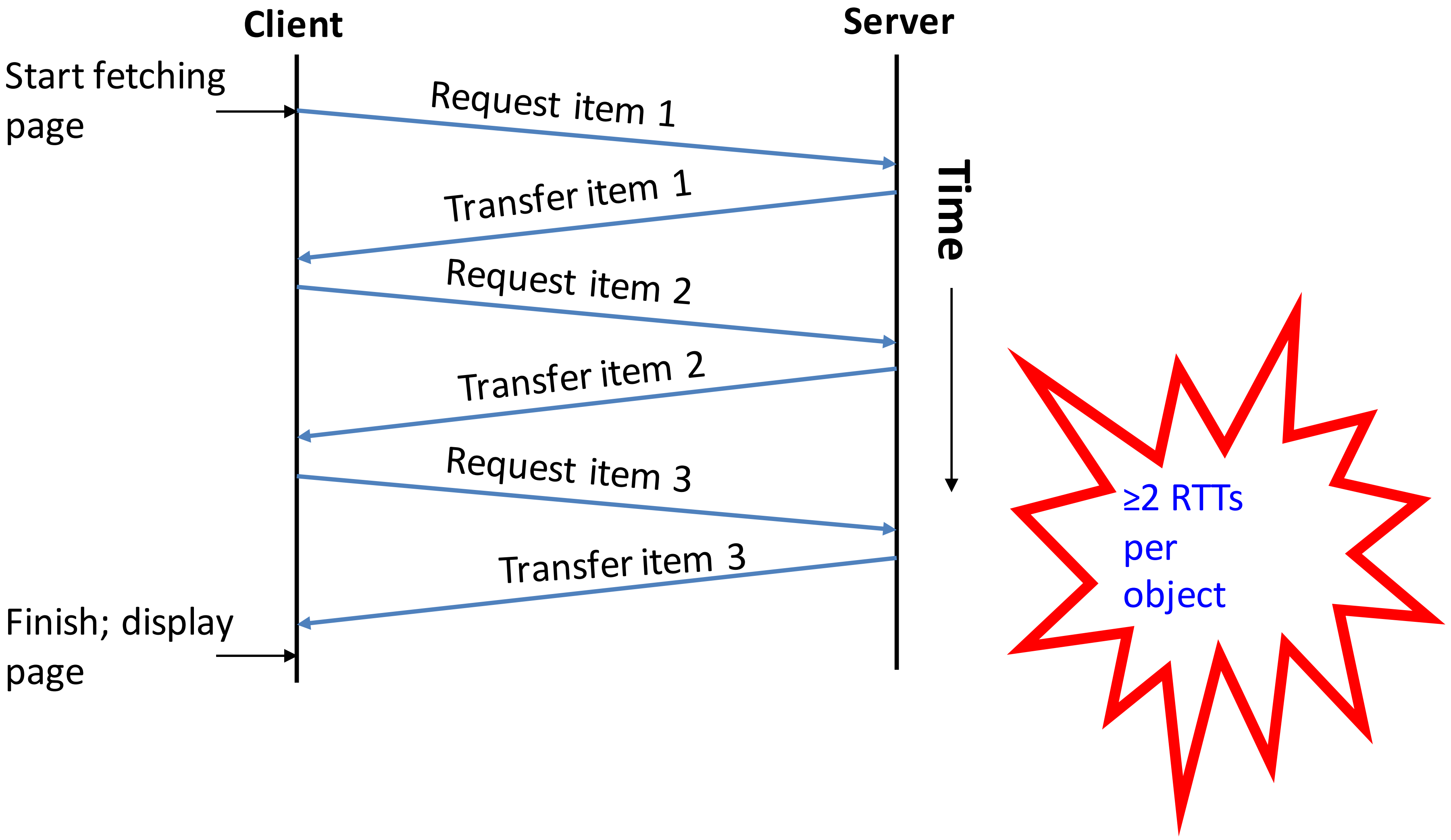
HTTP Performance

- Most Web pages have multiple objects
  - *e.g.,* HTML file and a bunch of embedded images

- How do you retrieve those objects (naively)?
  - *One item at a time*

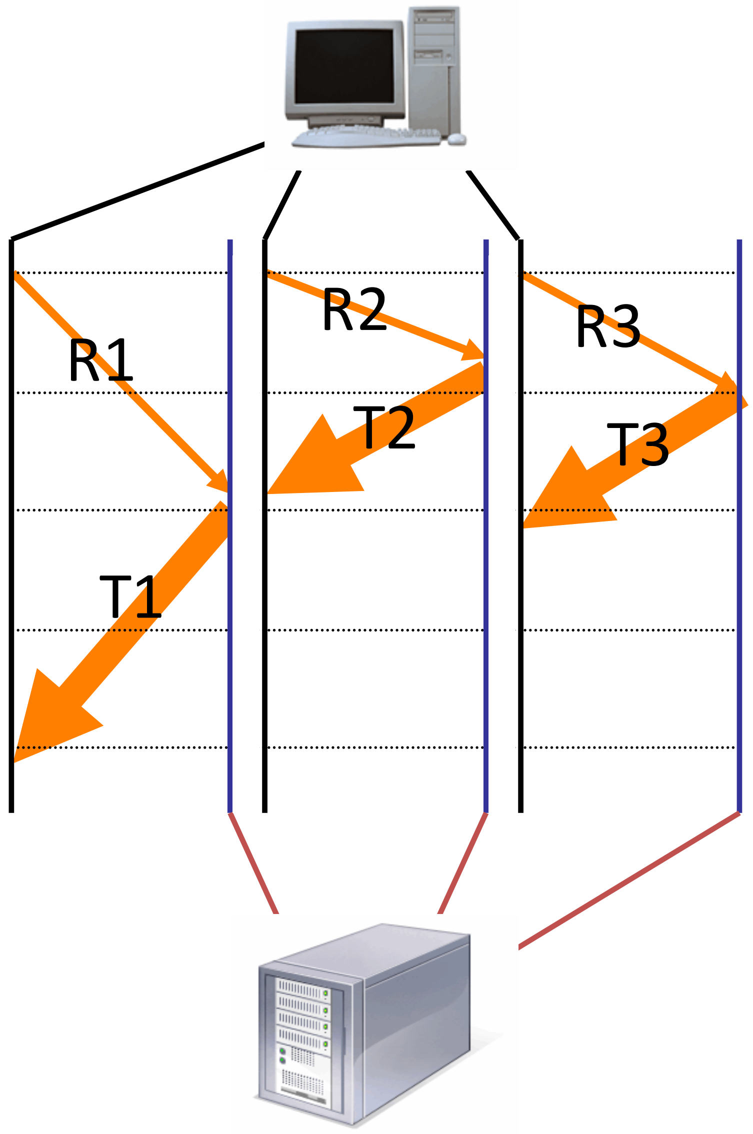
- Put stuff in the optimal place?
  - *Where is that precisely?*
    - ***Enter the Web cache and the CDN***

Fetch HTTP Items: Stop & Wait



Improving HTTP Performance:
Concurrent Requests & Responses

- Use multiple connections *in parallel*
- Does not necessarily maintain order of responses

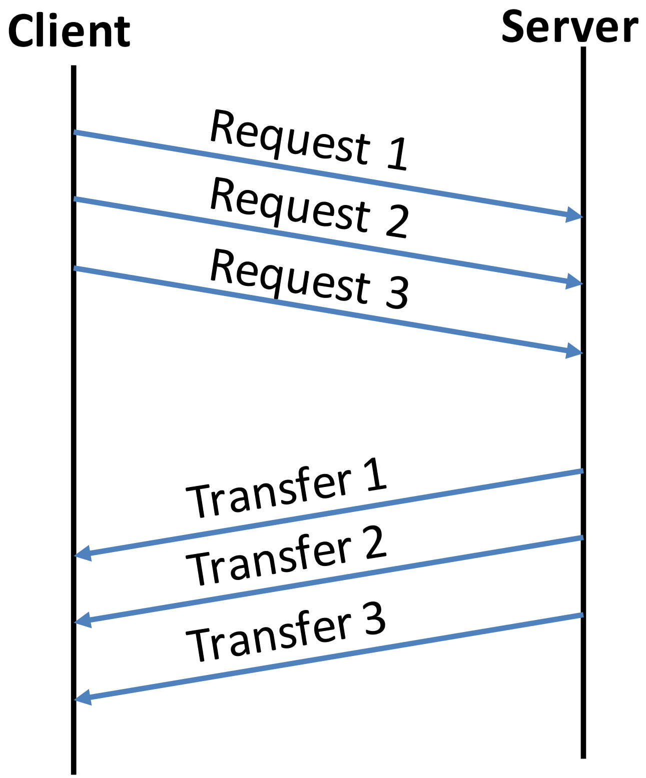


- Client = ☺
- Server = ☺
- Network = ☹ Why?

Improving HTTP Performance:

Pipelined Requests & Responses

- *Batch* requests and responses
  - Reduce connection overhead
  - Multiple requests sent in a single batch
  - Maintains order of responses
  - Item 1 always arrives before item 2
- How is this different from concurrent requests/responses?
  - Single TCP connection



Improving HTTP Performance:
Persistent Connections

- Enables multiple transfers per connection
  - Maintain TCP connection across multiple requests
  - Including transfers subsequent to current page
  - Client or server can tear down connection
- Performance advantages:
  - Avoid overhead of connection set-up and tear-down
  - Allow TCP to learn more accurate RTT estimate
  - Allow TCP congestion window to increase
  - i.e., leverage previously discovered bandwidth
- Default in HTTP/1.1

HTTP *evolution*

- 1.0 – one object per TCP: simple but slow
- Parallel connections - multiple TCP, one object each: wastes b/w, may be svr limited, out of order
- 1.1 pipelining – aggregate retrieval time: ordered, multiple objects sharing single TCP
- 1.1 persistent – aggregate TCP overhead: lower overhead in time, increase overhead at ends (e.g., when should/do you close the connection?)

Scorecard: Getting n Small Objects

Time dominated by latency

- One-at-a-time: ~2n RTT
- Persistent: ~ (n+1)RTT
- M concurrent: ~2[n/m] RTT
- Pipelined: ~2 RTT
- Pipelined/Persistent: ~2 RTT first time, RTT later

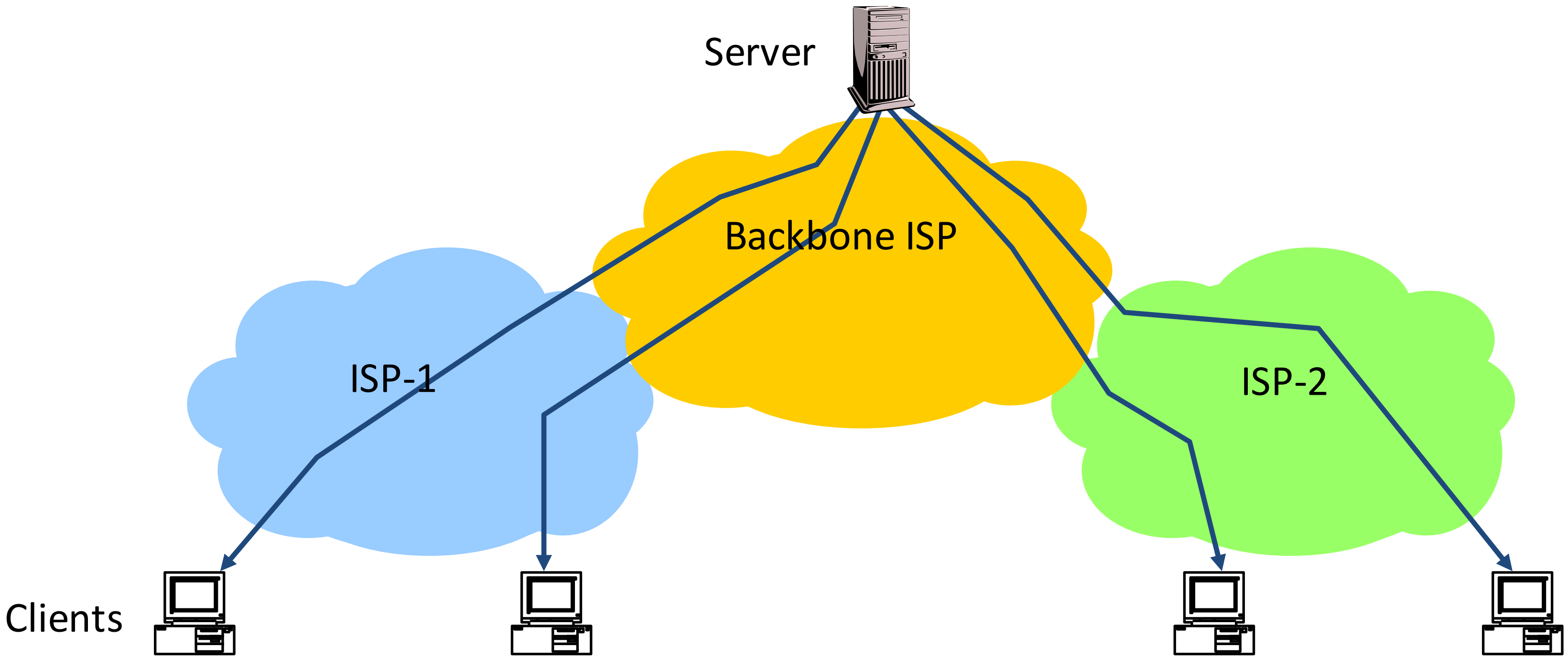
Scorecard: Getting n Large Objects

Time dominated by bandwidth

- One-at-a-time: ~ nF/B
- M concurrent: ~ [n/m] F/B
  - assuming shared with large population of users
- Pipelined and/or persistent: ~ nF/B
  - The only thing that helps is getting more bandwidth..

Improving HTTP Performance: Caching

- Many clients transfer same information
  - Generates redundant server and network load
  - Clients experience unnecessary latency



Improving HTTP Performance:
Caching: How

- Modifier to GET requests:
  - `If-modified-since` – returns "not modified" if resource not modified since specified time
- Response header:
  - `Expires` – how long it's safe to cache the resource
  - `No-cache` – ignore all caches; always get resource directly from server

Improving HTTP Performance:
Caching: Why

- Motive for placing content closer to client:
  - User gets better response time
  - Content providers get happier users
    - Time is money, really!
  - Network gets reduced load

- Why does caching work?
  - Exploits *locality of reference*

- How well does caching work?
  - Very well, up to a limit
  - Large overlap in content
  - But many unique requests

Improving HTTP Performance:

Caching on the Client

Example: Conditional GET Request

- Return resource only if it has changed at the server
  - Save server resources!

Request from client to server:

```
GET /~awm22/win HTTP/1.1
Host: www.cl.cam.ac.uk
User-Agent: Mozilla/4.03
If-Modified-Since: Sun, 27 Aug 2006 22:25:50 GMT
<CRLF>
```

- How?
  - Client specifies "if-modified-since" time in request
  - Server compares this against "last modified" time of desired resource
  - Server returns "304 Not Modified" if resource has not changed
  - …. or a "200 OK" with the latest version otherwise

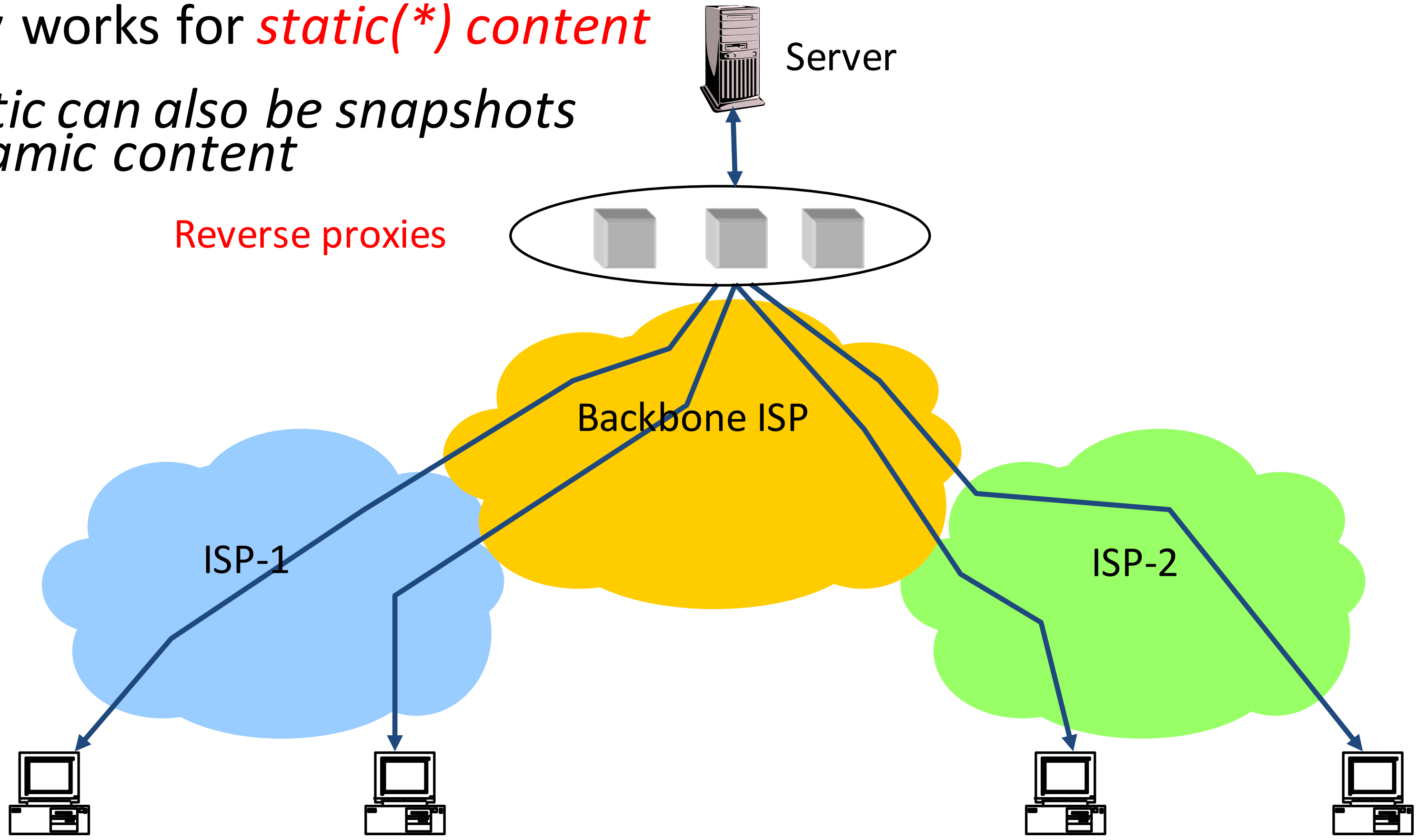
Improving HTTP Performance:
Caching with Reverse Proxies

Cache documents close to **server**

→ decrease server load

- Typically done by content providers
- Only works for *static(*) content*

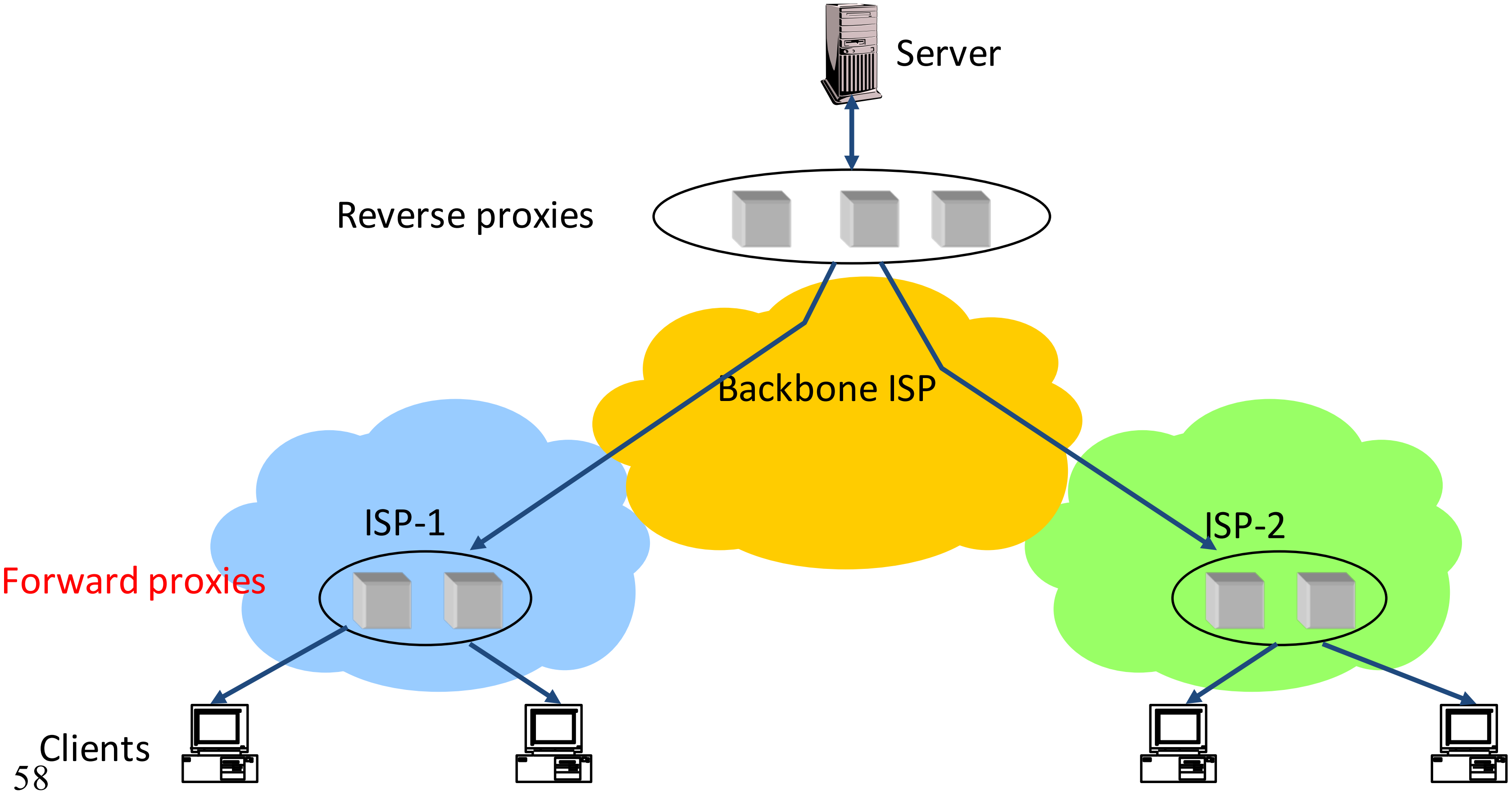
(*) static can also be snapshots of dynamic content



Improving HTTP Performance:
Caching with Forward Proxies

Cache documents close to **clients**
→ reduce network traffic and decrease latency

- Typically done by ISPs or corporate LANs

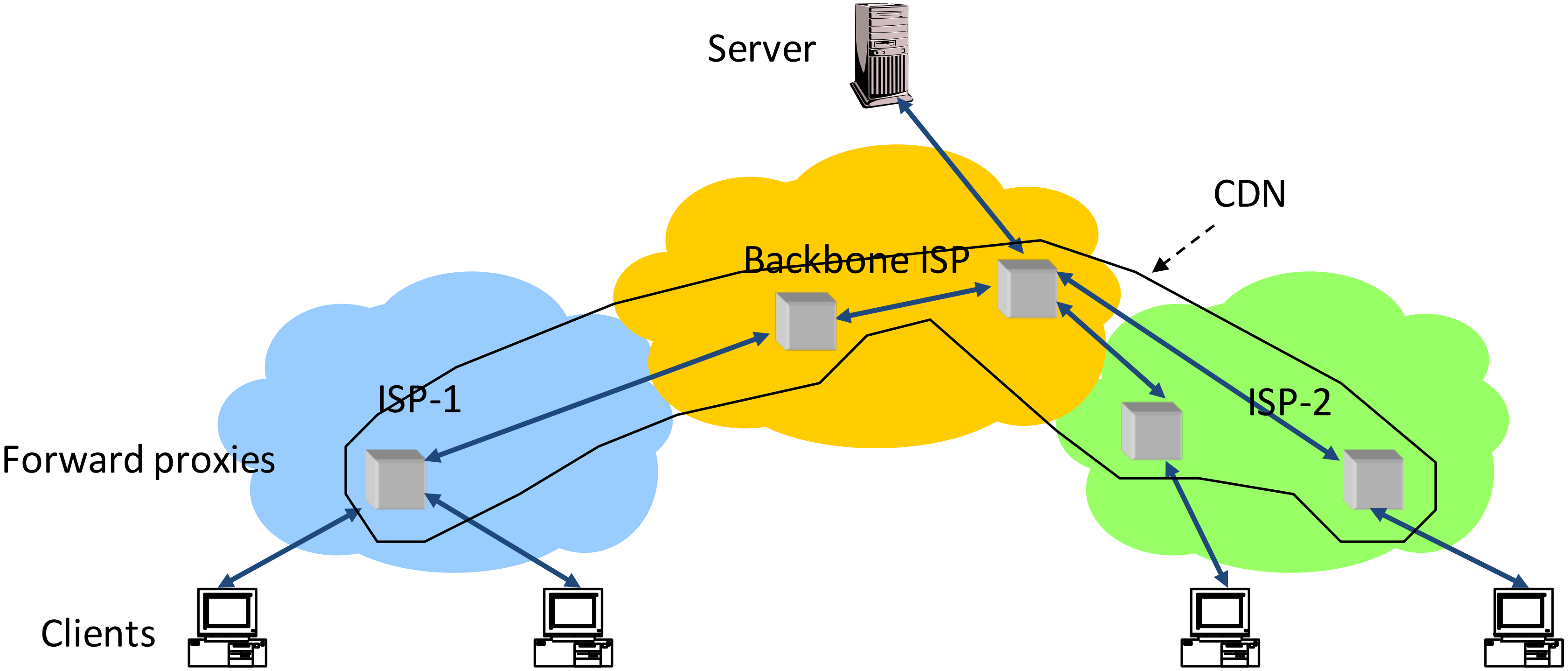


Improving HTTP Performance:

Caching w/ Content Distribution Networks

- Integrate forward and reverse caching functionality
  - One overlay network (usually) administered by one entity
  - *e.g.,* Akamai
- Provide document caching
  - **Pull:** Direct result of clients' requests
  - **Push:** Expectation of high access rate
- Also do some processing
  - Handle *dynamic* web pages
  - *Transcoding*
  - *Maybe do some security function – watermark IP*

Improving HTTP Performance:
Caching with CDNs (cont.)



Improving HTTP Performance: CDN Example – Akamai

- Akamai creates new domain names for each client content provider.
  - e.g., *a128.g.akamai.net*
- The CDN's DNS servers are authoritative for the new domains
- The client content provider modifies its content so that embedded URLs reference the new domains.
  - "Akamaize" content
  - e.g.: *http://www.bbc.co.uk/popular-image.jpg* becomes *http://a128.g.akamai.net/popular-image.jpg*
- *Requests now sent to CDN's infrastructure…*

Hosting: Multiple Sites Per Machine

- Multiple Web sites on a single machine
  - Hosting company runs the Web server on behalf of multiple sites (*e.g.*, www.foo.com and www.bar.com)
- Problem: `GET /index.html`
  - `www.foo.com/index.html` or `www.bar.com/index.html`?
- Solutions:
  - Multiple server processes on the same machine
    - Have a separate IP address (or port) for each server
  - Include site name in HTTP request
    - Single Web server process with a single IP address
    - Client includes "Host" header (*e.g.,* `Host: www.foo.com`)
    - *Required header* with HTTP/1.1

Hosting: Multiple Machines Per Site

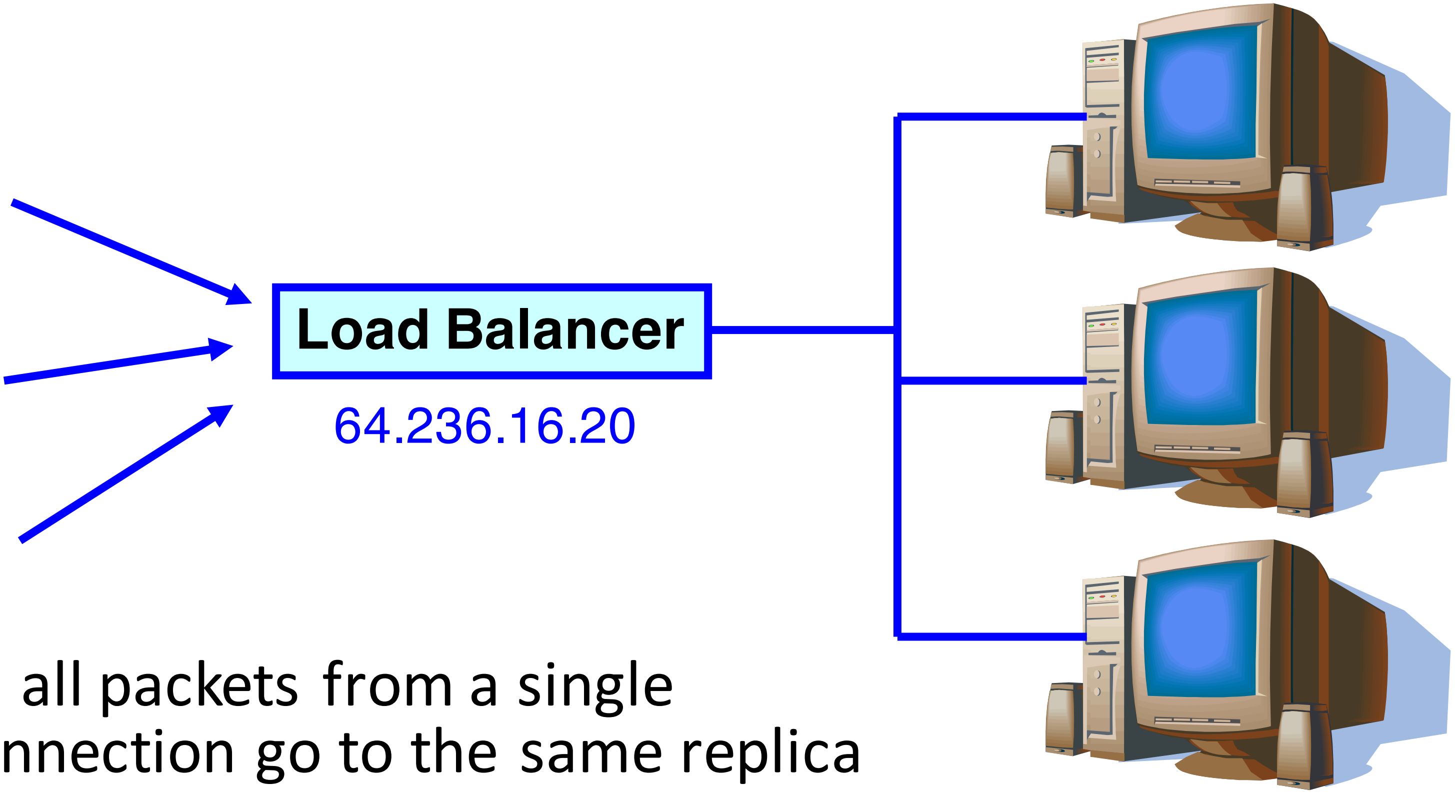
- Replicate popular Web site across many machines
  - Helps to handle the load
  - Places content closer to clients

- Helps when content isn't cacheable

- Problem: Want to direct client to particular replica
  - Balance load across server replicas
  - Pair clients with nearby servers

Multi-Hosting at Single Location

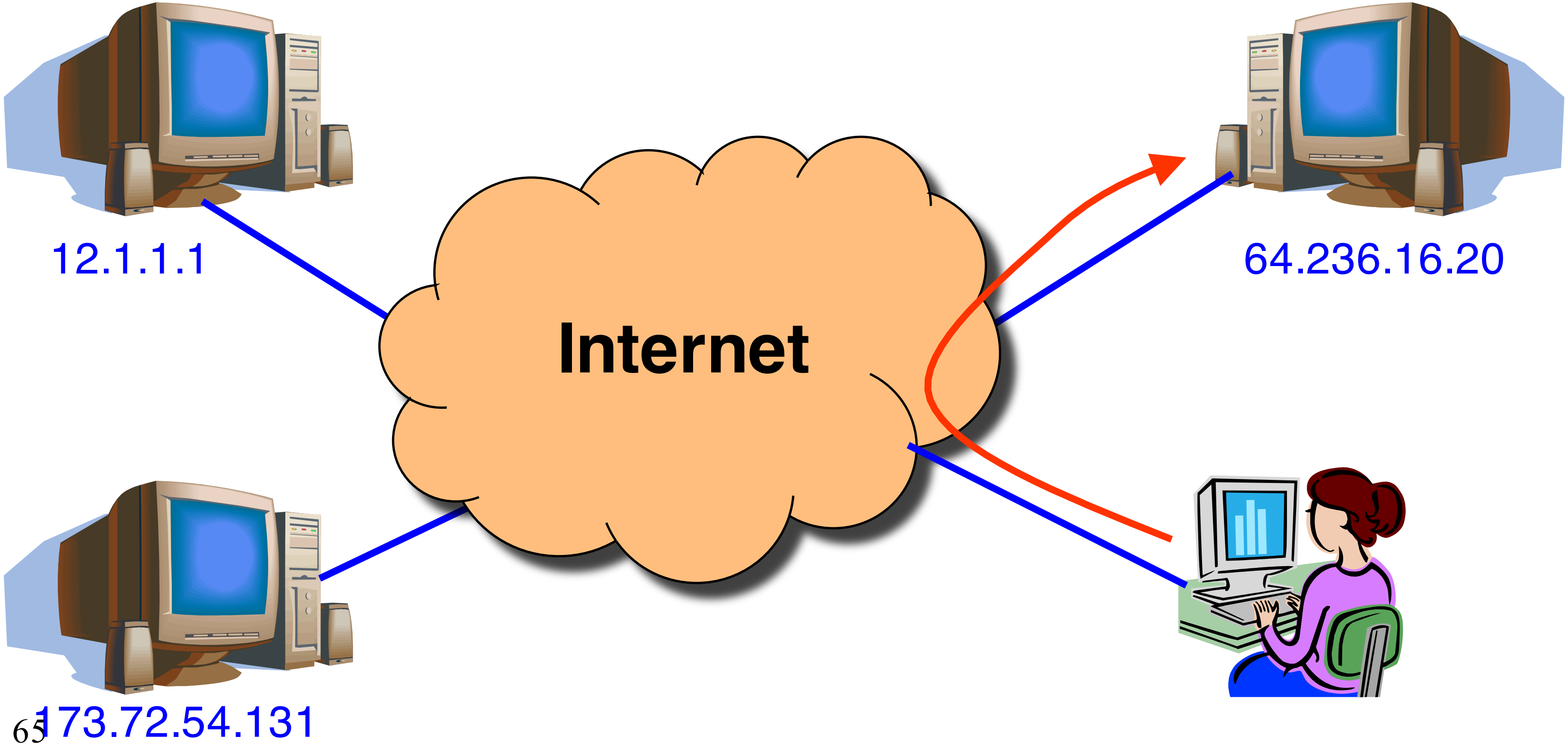
- Single IP address, multiple machines
  - Run multiple machines behind a single IP address



  - Ensure all packets from a single TCP connection go to the same replica

Multi-Hosting at Several Locations

- Multiple addresses, multiple machines
  - Same name but different addresses for all of the replicas
  - Configure DNS server to return *closest* address



CDN examples round-up

- CDN using DNS

  DNS has information on loading/distribution/location

- CDN using anycast

  same address from DNS name but local routes

- CDN based on rewriting HTML URLs

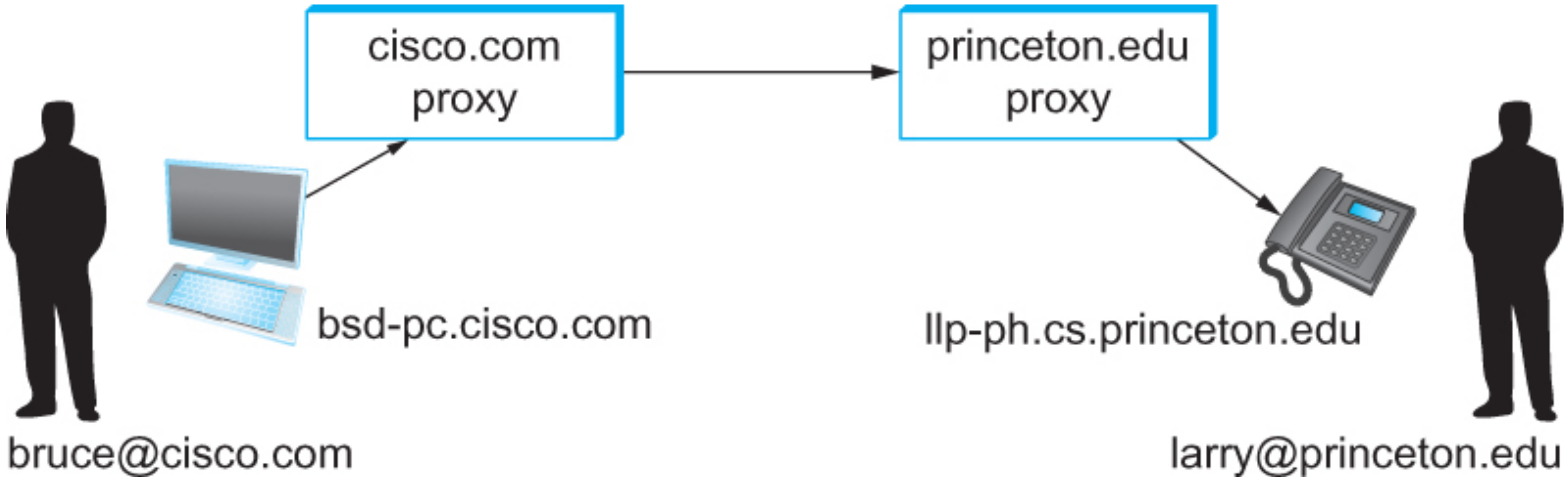
  (akami example just covered – akami uses DNS too)

SIP – Session Initiation Protocol

Session?

Anyone smell an OSI / ISO standards document burning?

SIP - VoIP



Establishing communication through SIP proxies.

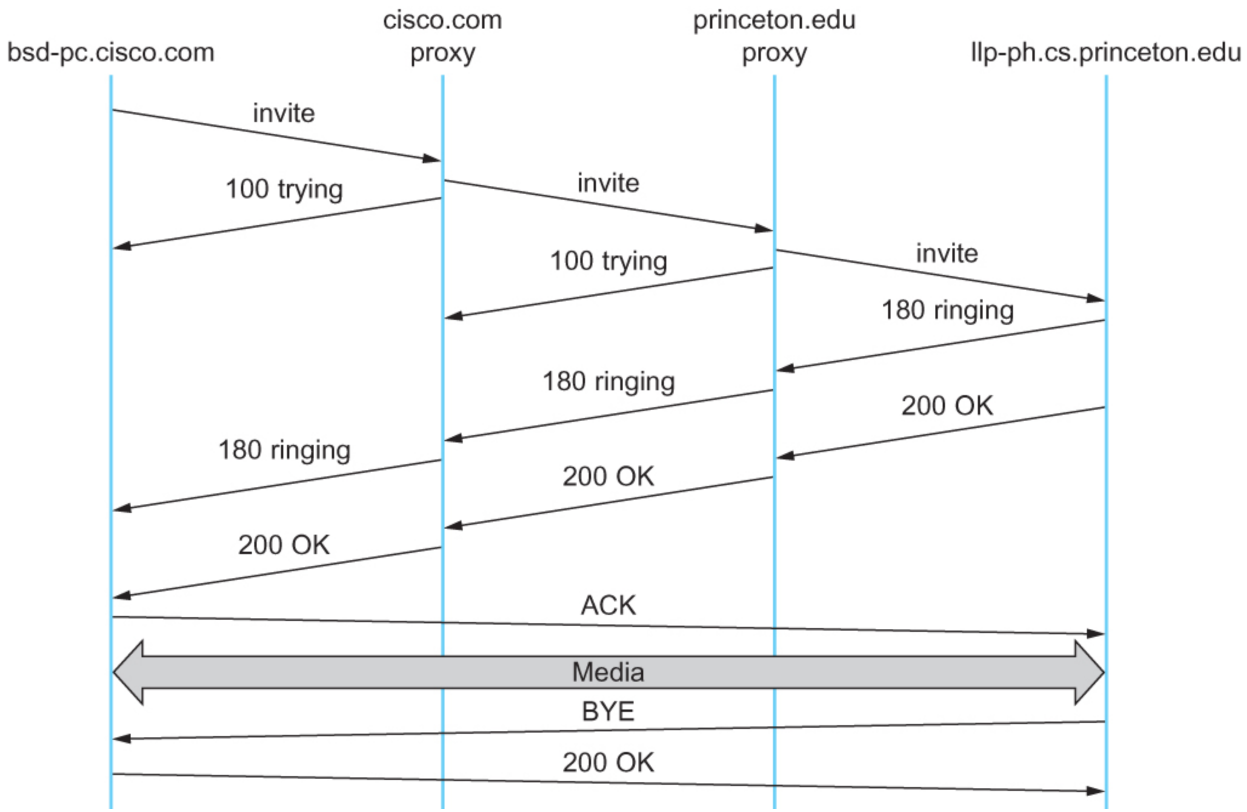
SIP?

- SIP – bringing the fun/complexity of telephony to the Internet
  - User location
  - User availability
  - User capabilities
  - Session setup
  - Session management
    - (e.g. “call forwarding”)

H.323 – ITU

- Why have one standard when there are at least two….
- The full H.323 is hundreds of pages
  - The protocol is known for its complexity – an ITU hallmark
- SIP is not much better
  - IETF grew up and became the ITU….

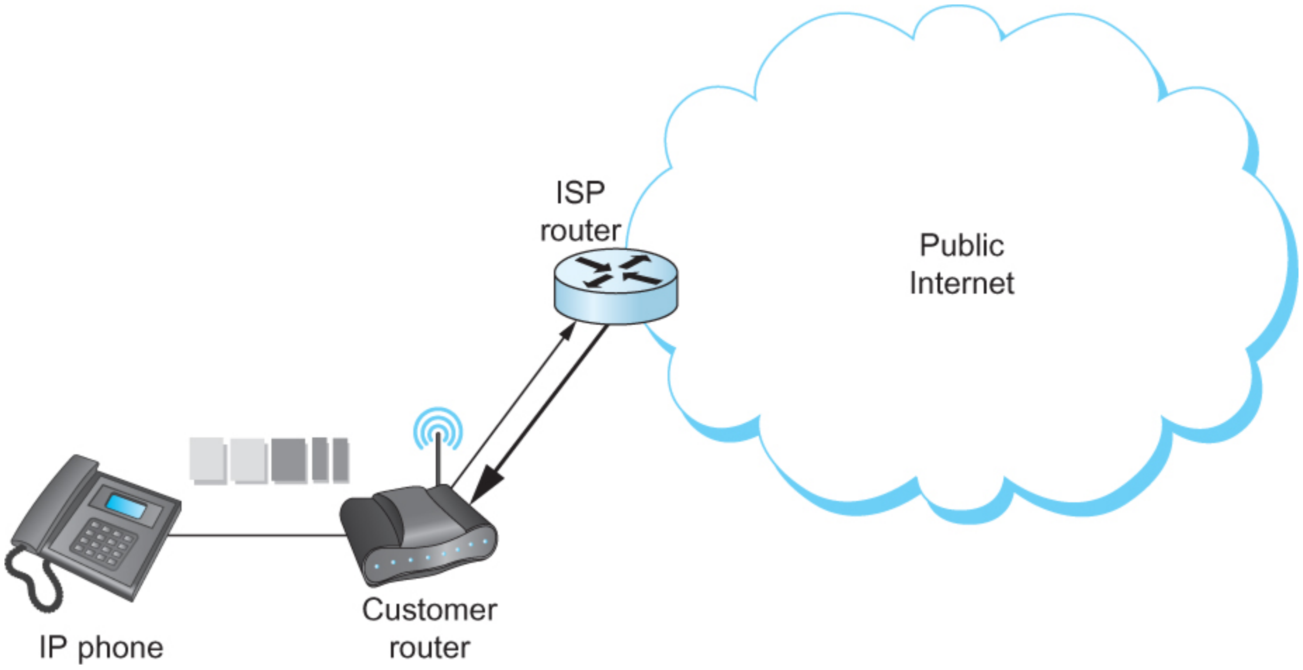
Multimedia Applications



Message flow for a basic SIP session

The (still?) missing piece:

Resource Allocation for Multimedia Applications

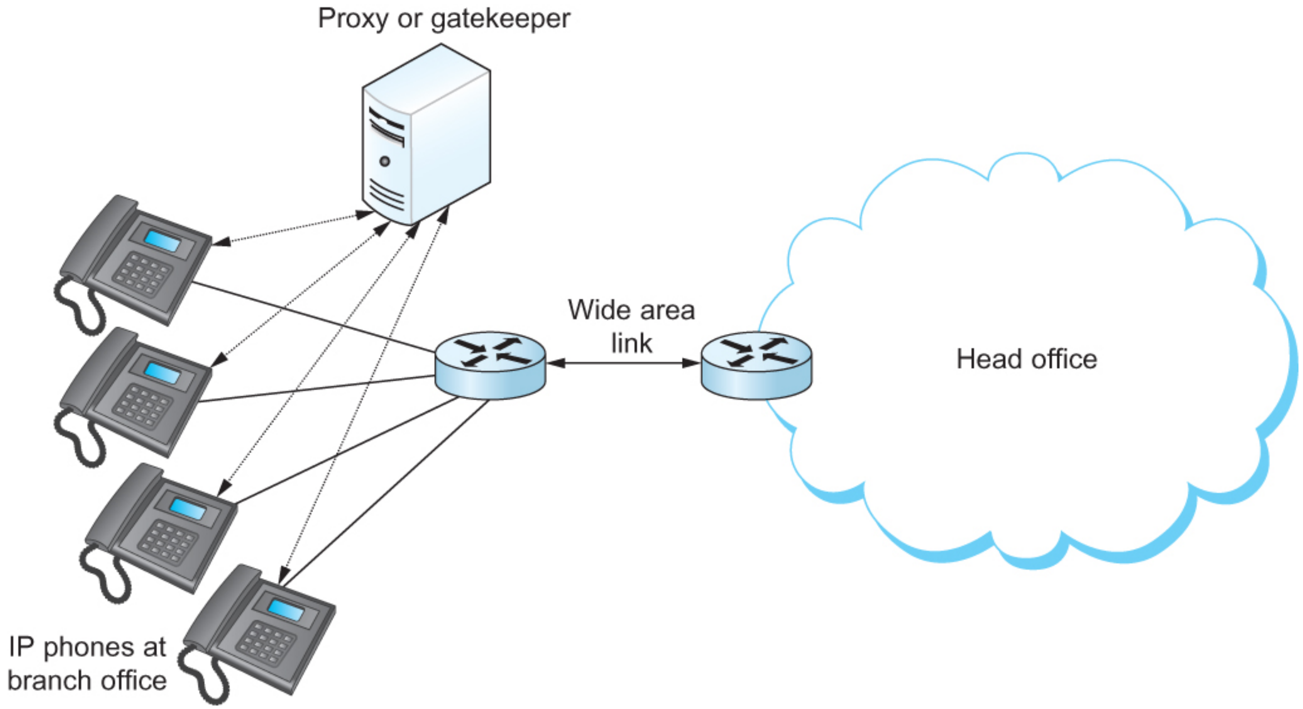


I can ‘differentiate’ VoIP from data but…

I can only control data going into the Internet

Multimedia Applications

- Resource Allocation for Multimedia Applications

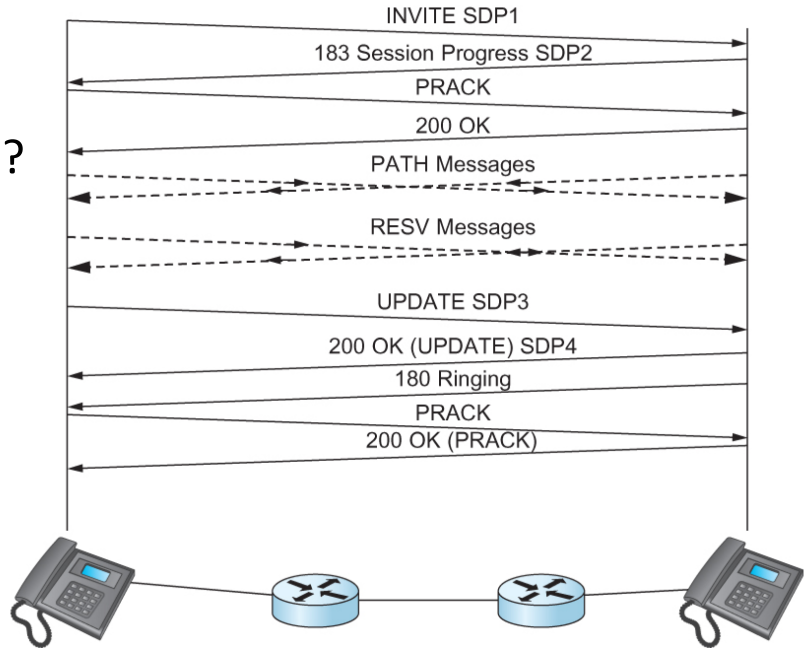


Admission control using session control protocol.

Resource Allocation for Multimedia Applications

Coming soon… ~~1995~~
~~2000~~
~~2010~~
who are we kidding??

Co-ordination of SIP signaling and resource reservation.



So where does it happen?

Inside single institutions or *domains of control…..*

(Universities, Hospitals, big corp…)

What about my aDSL/CABLE/etc it combines voice and data?

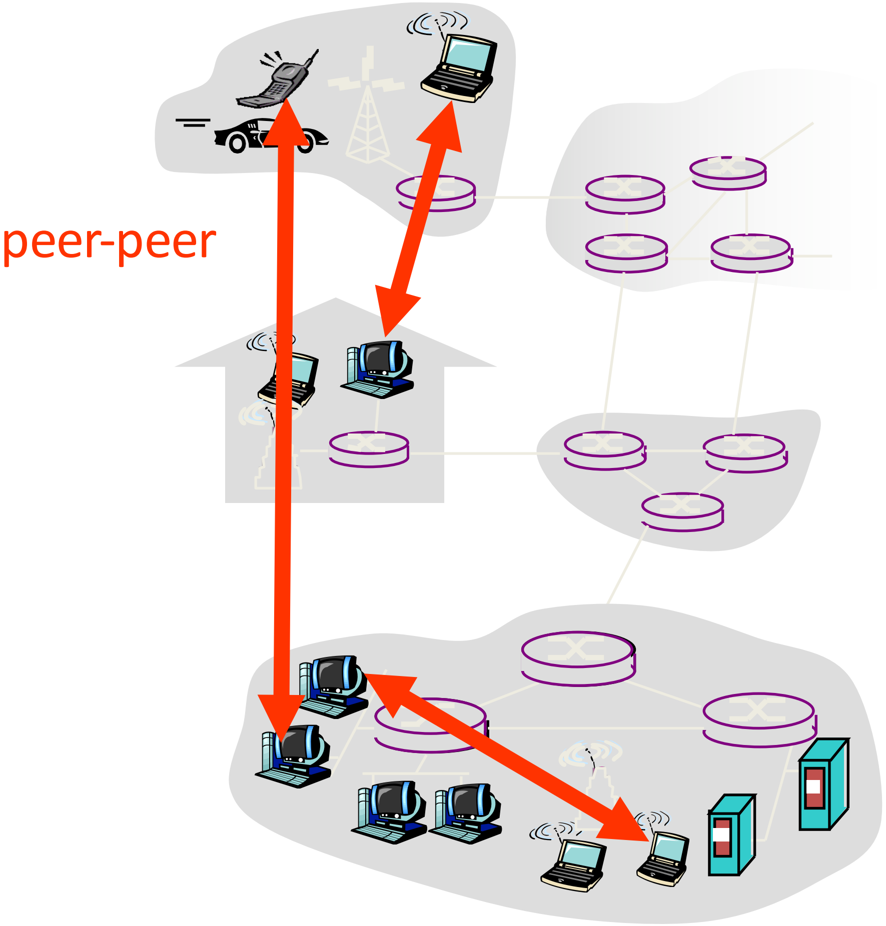
Phone company **controls** the multiplexing on the line and throughout their own network too……

P2P – efficient network use that annoys the ISP

Pure P2P architecture

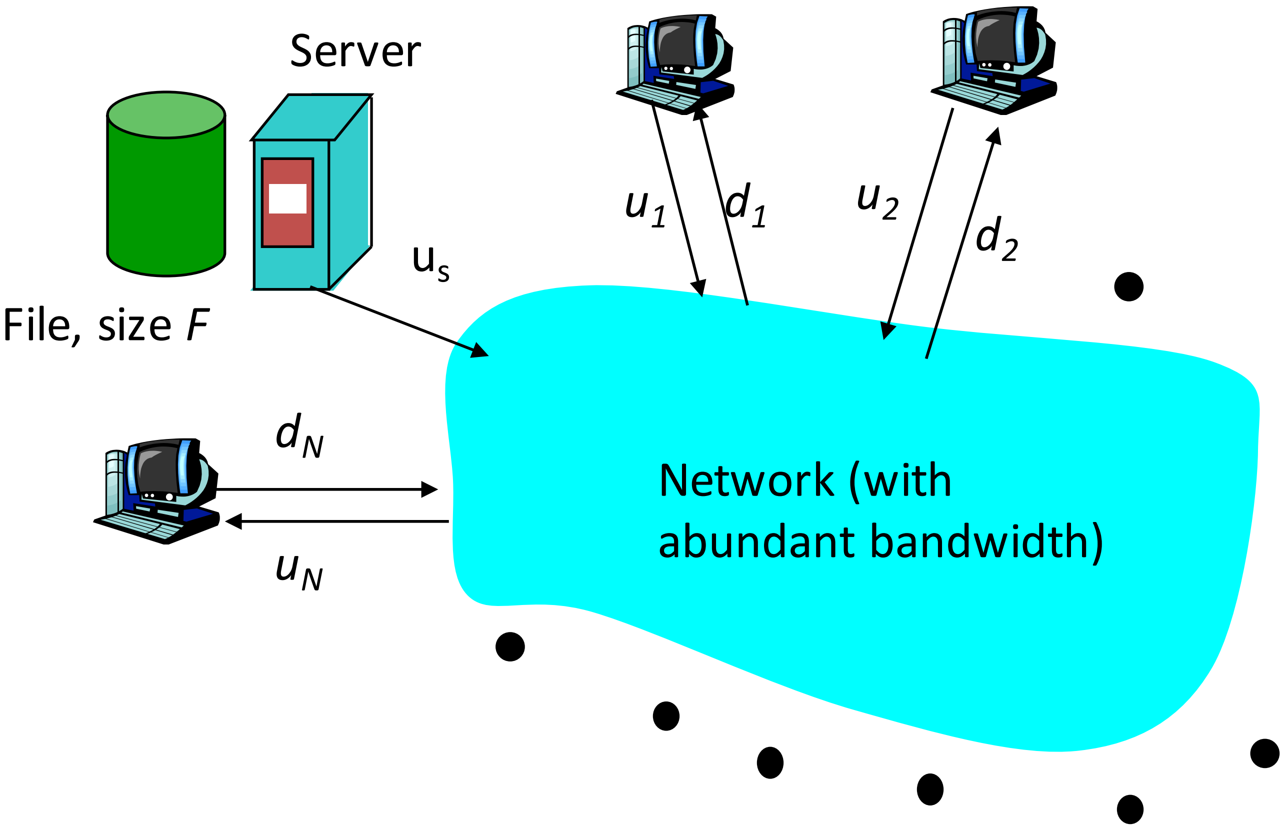
- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

- Three topics:
  - File distribution
  - Searching for information
  - Case Study: Skype



File Distribution: Server-Client vs P2P

Question : How much time to distribute file from one server to *N peers*?



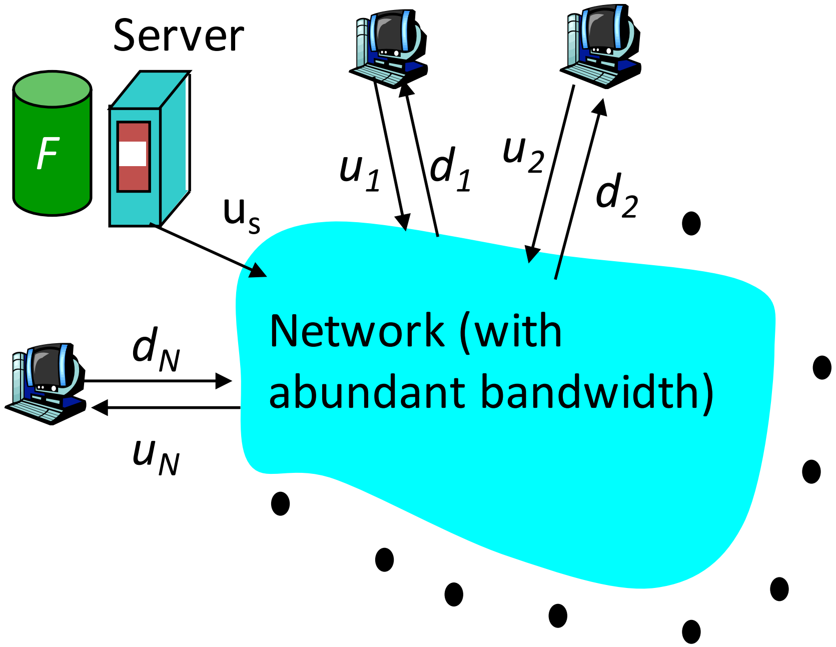
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File distribution time: server-client

- server sequentially sends N copies:
  - NF/u_s time
- client i takes F/d_i time to download

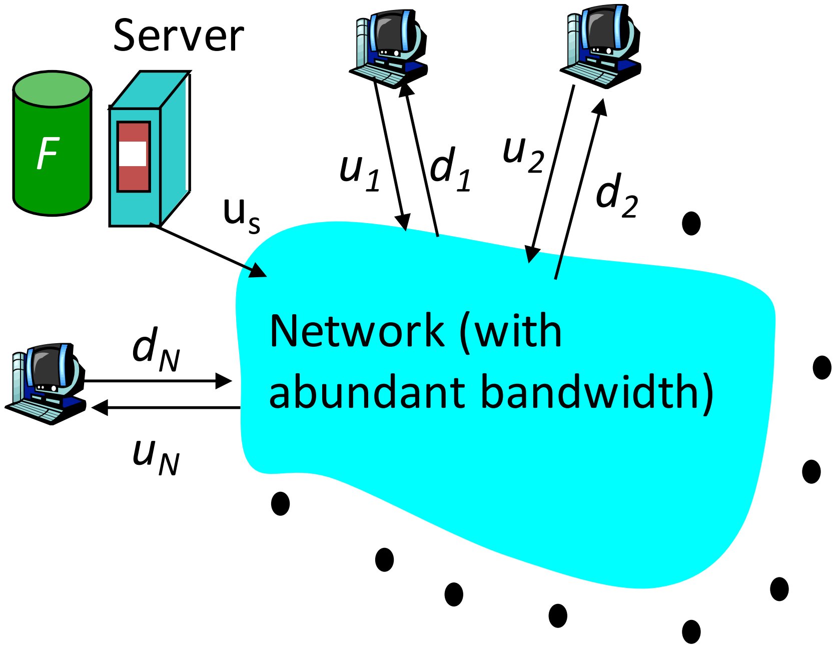


Time to distribute *F* to *N* clients using client/server approach $= d_{cs} = \max\{NF/u_s, F/\min_i(d_i)\}$

increases linearly in N (for large N)

File distribution time: P2P

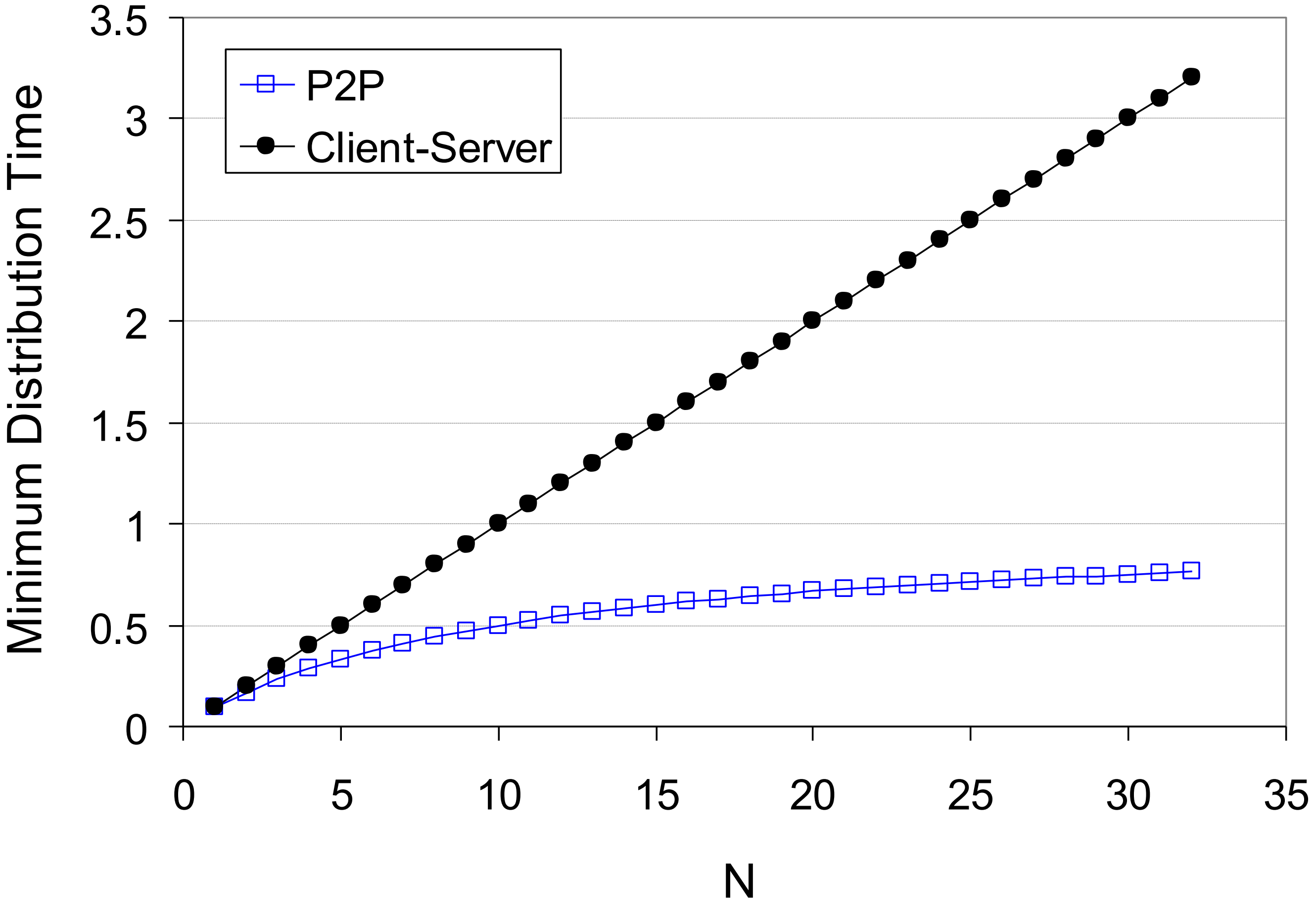
- server must send one copy: F/u_s time
- client i takes F/d_i time to download
- NF bits must be downloaded (aggregate)
  - ❒ fastest possible upload rate: $u_s + \sum u_i$



$$d_{P2P} = \max \{ \mathit{F/u_s}, \mathit{F/min(d_i)}, NF\underset{i}{/}(u_s + \sum u_i) \}$$

Server-client vs. P2P: example

Client upload rate = u, F/u = 1 hour, u_s = 10u, $d_{min} \geq u_s$



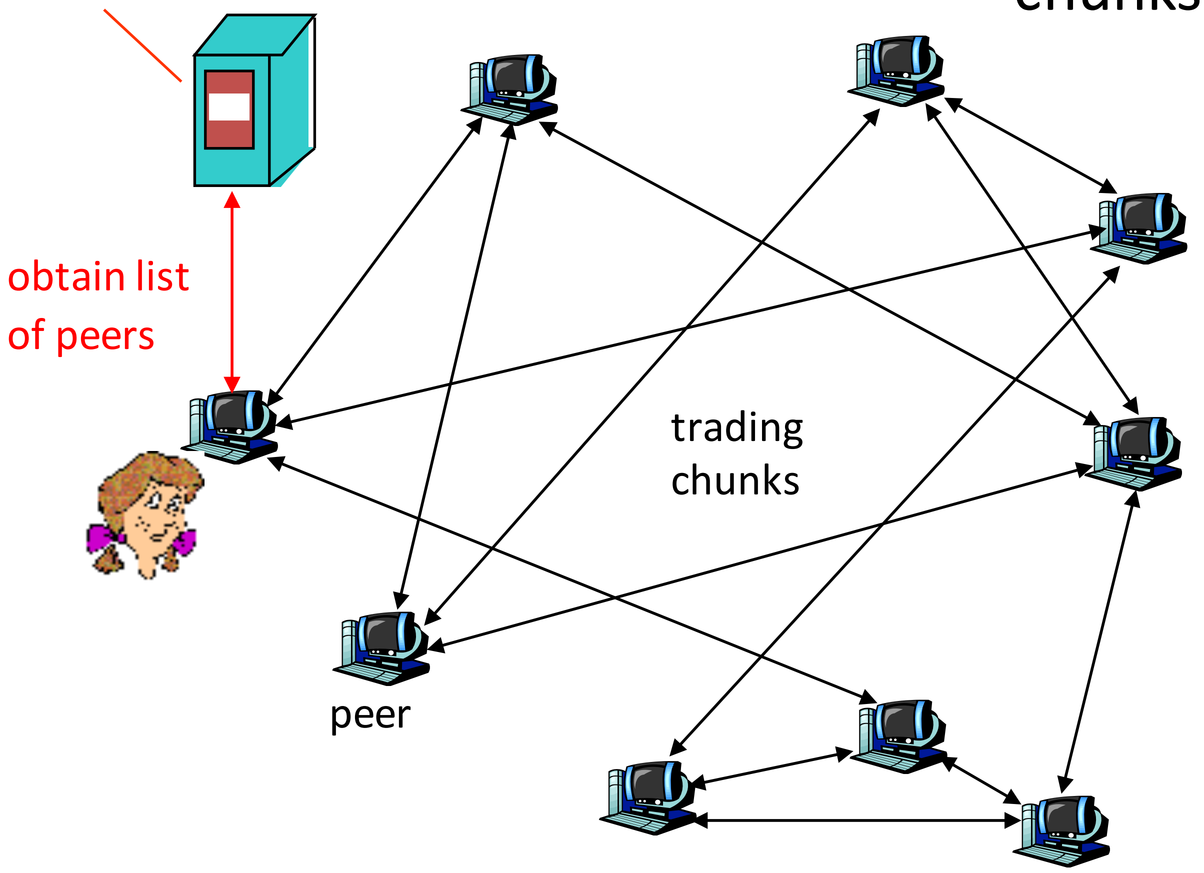
File distribution: BitTorrent*

*rather old BitTorrent

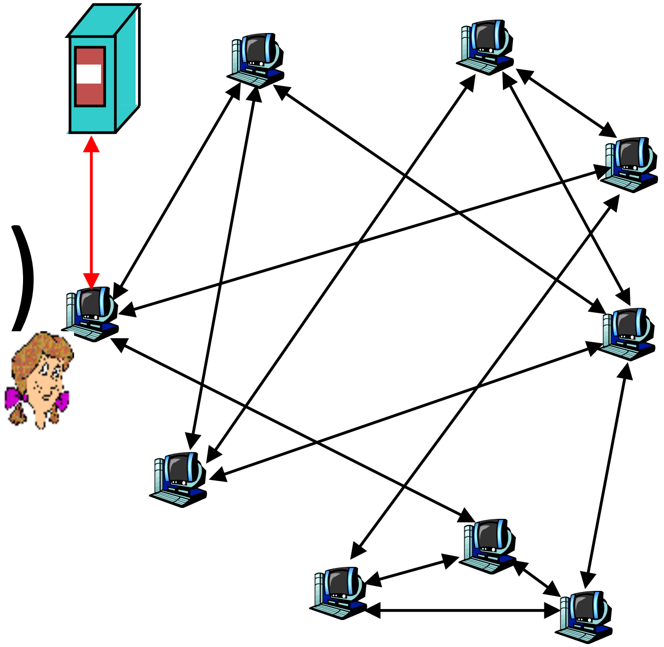
❒ P2P file distribution

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file



BitTorrent (1)



- file divided into 256KB *chunks*.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers (“neighbors”)
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent (2)

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

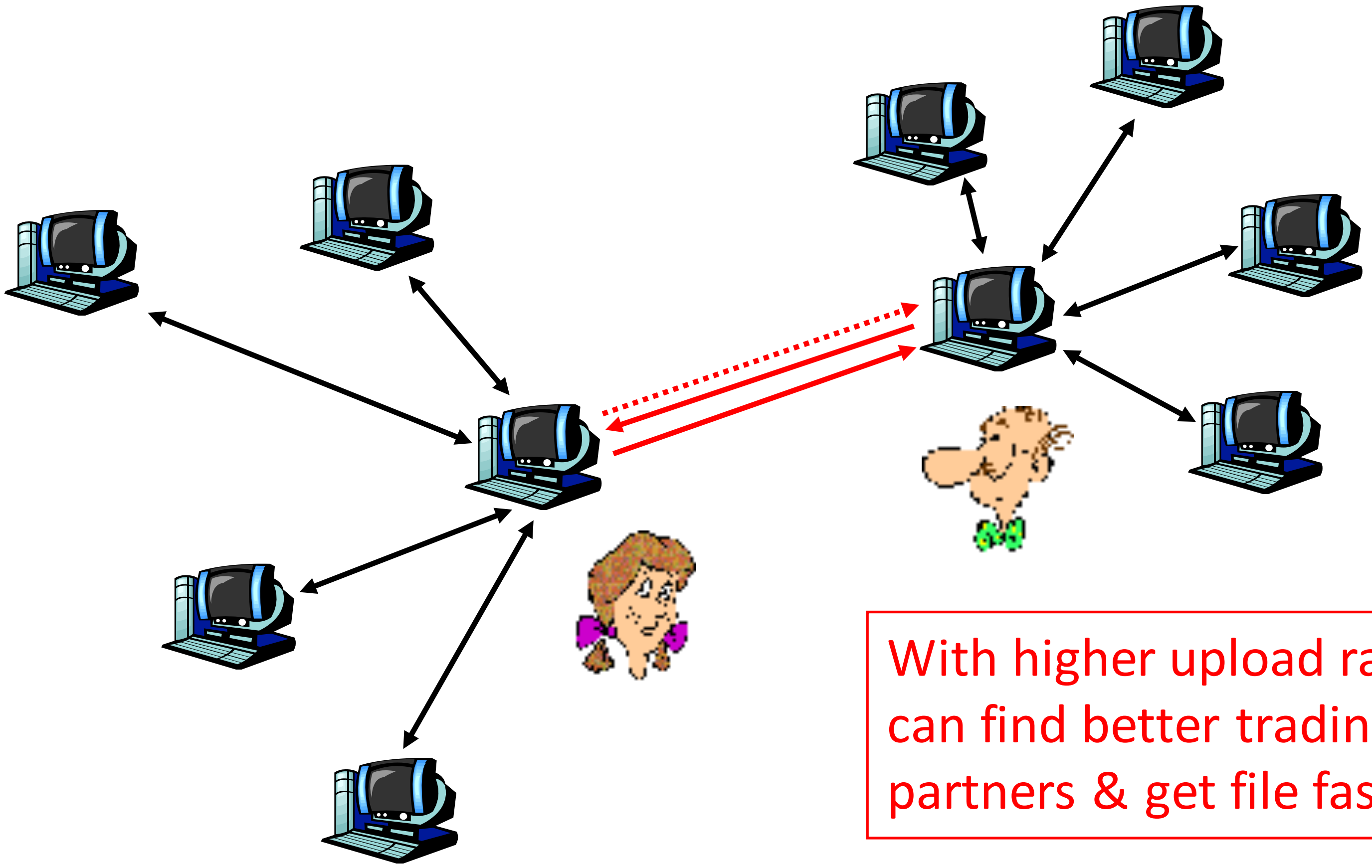
- ❒ Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - ❖ re-evaluate top 4 every 10 secs
- ❒ every 30 secs: randomly select another peer, starts sending chunks
  - ❖ newly chosen peer may join top 4
  - ❖ "optimistically unchoke"

BitTorrent: Tit-for-tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



With higher upload rate, can find better trading partners & get file faster!

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Database has (key, value) pairs;
  - key: ss number; value: human name
  - key: content type; value: IP address
- Peers query DB with key
  - DB returns values that match the key
- Peers can also insert (key, value) peers

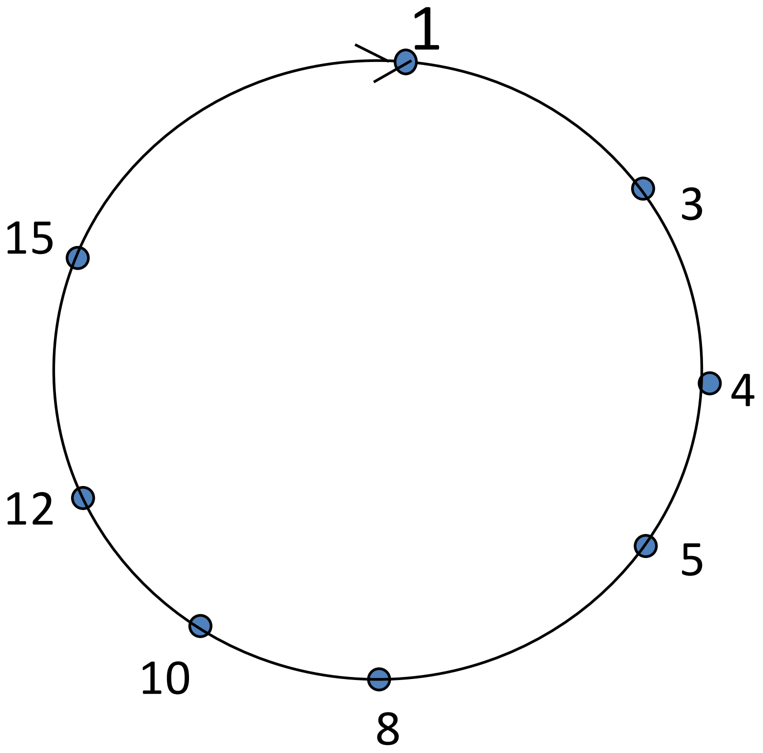
DHT Identifiers

- Assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in same range.
- To get integer keys, hash original key.
  - eg, key = h("Game of Thrones season 4")
  - This is why they call it a distributed "hash" table

How to assign keys to peers?

- Central issue:
  - Assigning (key, value) pairs to peers.
- Rule: assign key to the peer that has the closest ID.
- Convention in lecture: closest is the immediate successor of the key.
- Ex: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

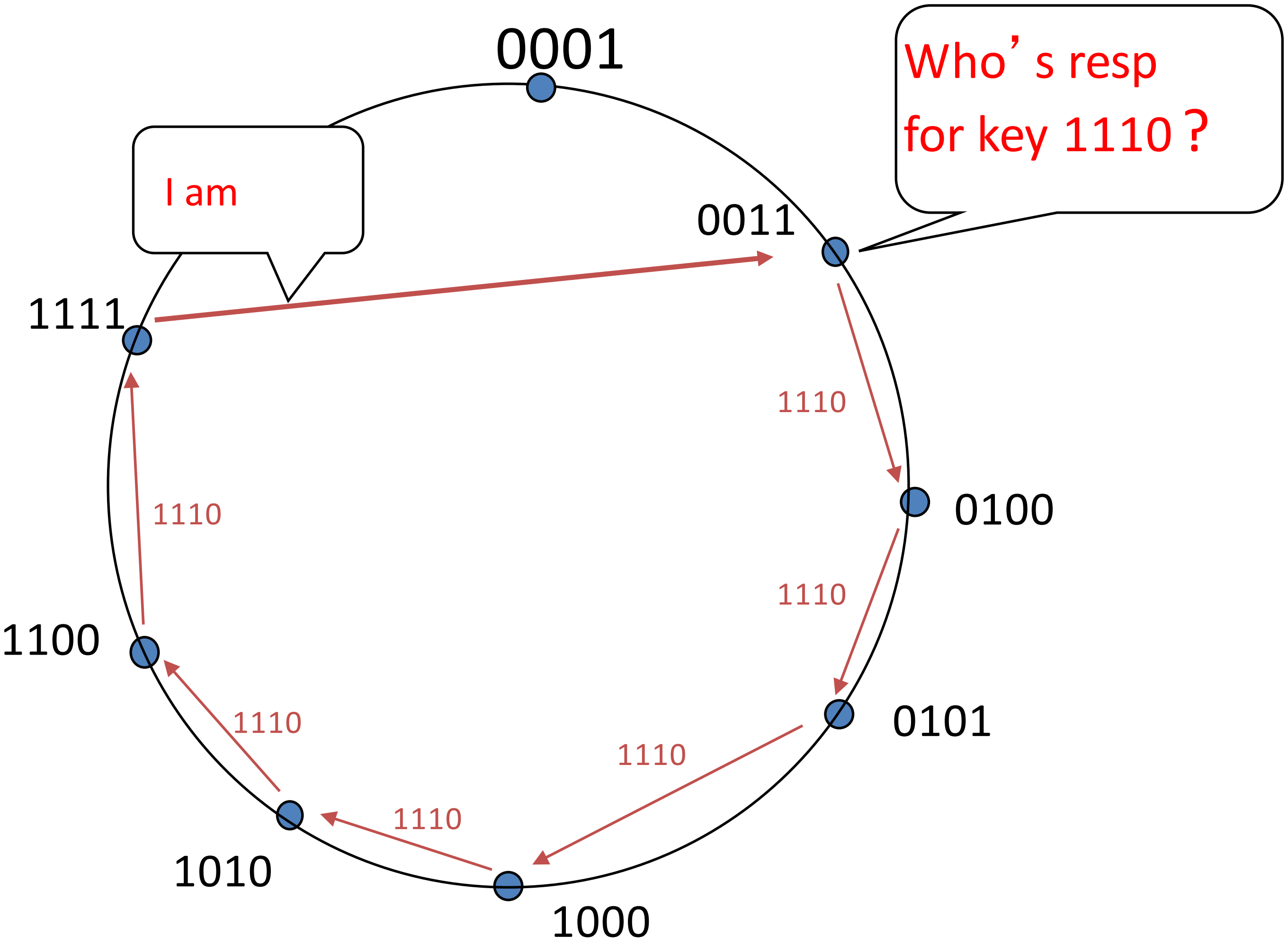
Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- “Overlay network”

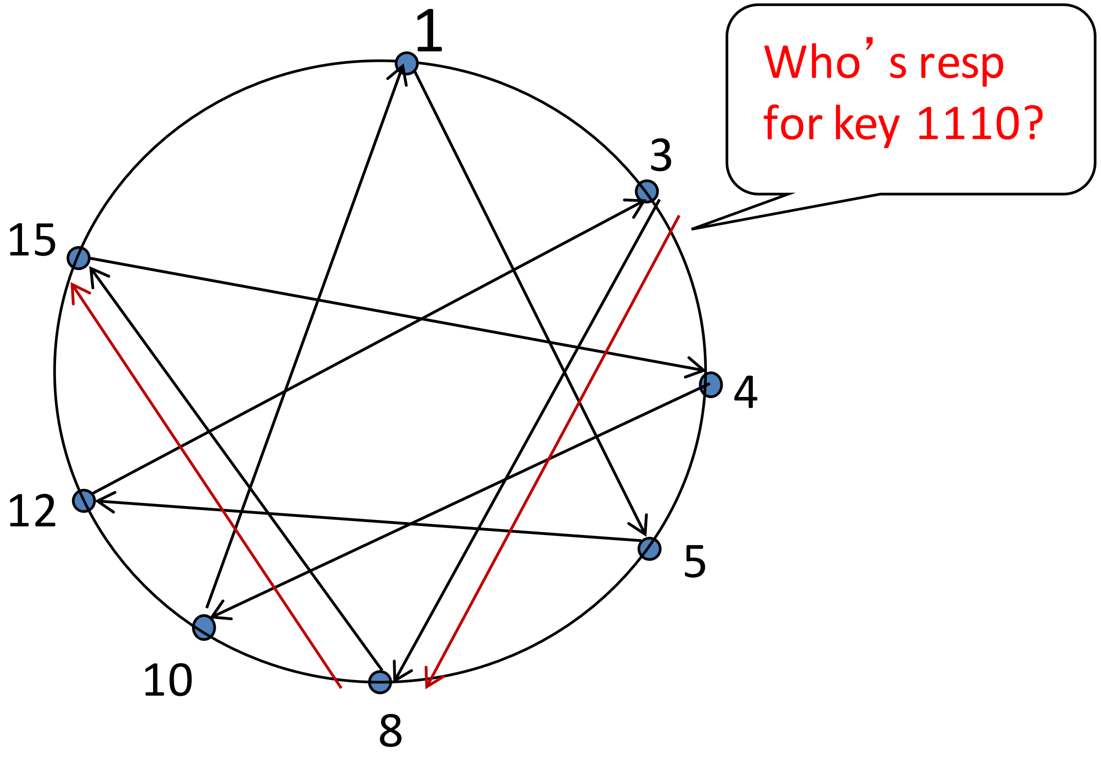
Circle DHT (2)

O(N) messages on avg to resolve query, when there are N peers



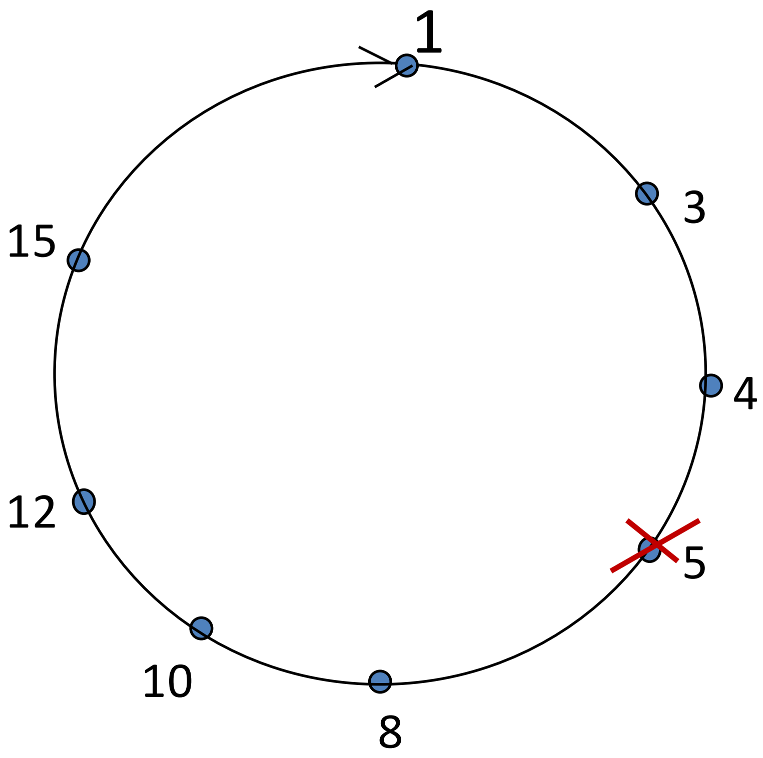
Define closest as closest successor

Circular DHT with Shortcuts



- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Reduced from 6 to 2 messages.
- Possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

Peer Churn



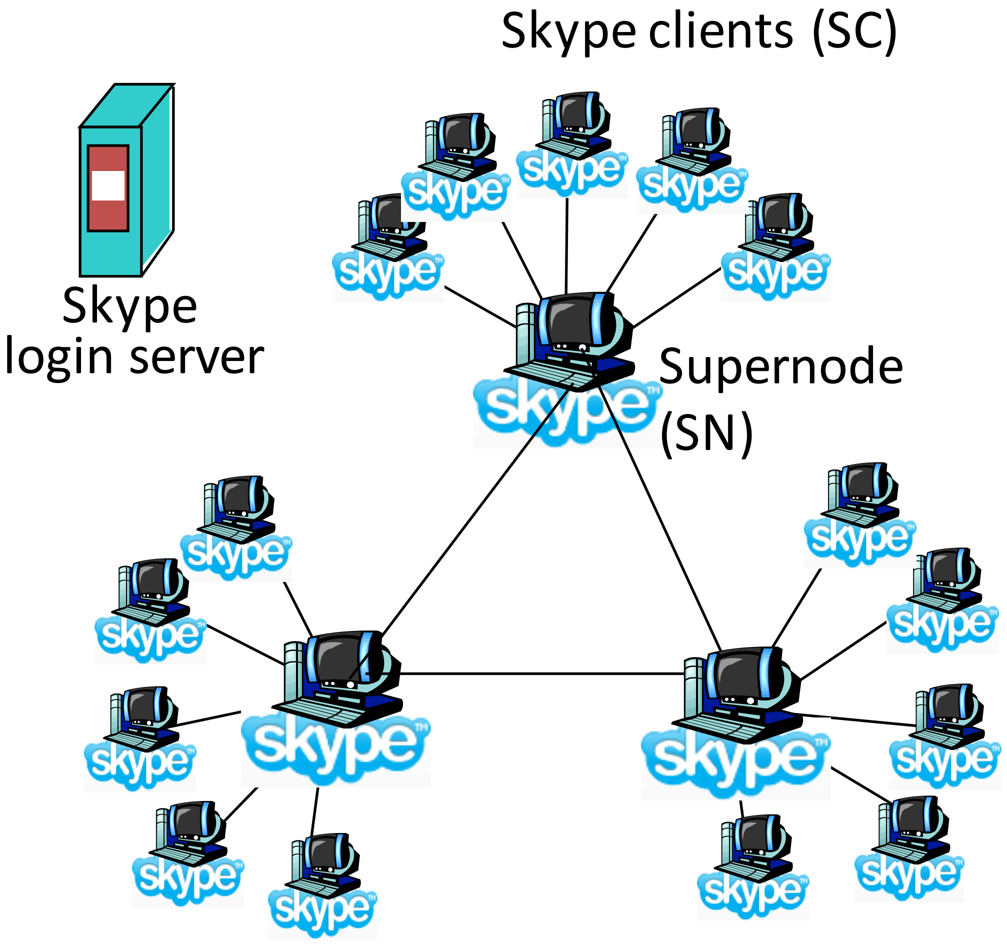
•To handle peer churn, require each peer to know the IP address of its two successors.

• Each peer periodically pings its two successors to see if they are still alive.

- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8' s immediate successor its second successor.
- What if peer 13 wants to join?

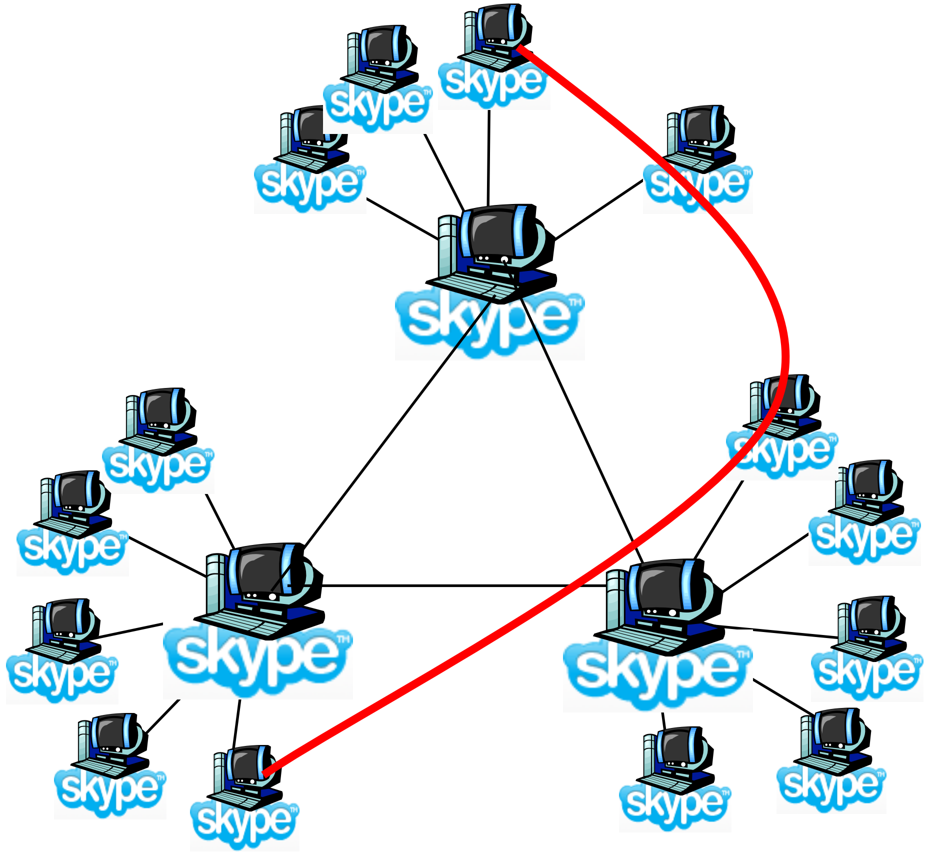
P2P Case study: Skype (pre-Microsoft)

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SNs
- Index maps usernames to IP addresses; distributed over SNs



Peers as relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents an outside peer from initiating a call to insider peer
- Solution:
  - Using Alice's and Bob's SNs, Relay is chosen
  - Each peer initiates session with relay.
  - Peers can now communicate through NATs via relay



Summary.

- Apps need protocols too

- We covered examples from
  - Traditional Applications (web)
  - Scaling and Speeding the web (CDN/Cache tricks)

- Infrastructure Services (DNS)
  - Cache and Hierarchy

- Multimedia Applications (SIP)
  - Extremely hard to do better than worst-effort

- P2P Network examples

Topic 7: Datacenters

What we will cover

- Characteristics of a datacenter environment
  - goals, constraints, workloads, *etc.*
- How and why DC networks are different (*vs.* WAN)
  - e.g., latency, geo, autonomy, …
- How traditional solutions fare in this environment
  - e.g., IP, Ethernet, TCP, ARP, DHCP
- Not details of *how* datacenter networks operate

Disclaimer

- Material is emerging (not established) wisdom

- Material is incomplete
  - many details on how and why datacenter networks operate aren’t public

Why Datacenters?

Your <public-life, private-life, banks, government> live in my datacenter.

Security, Privacy, Control, Cost, Energy, (breaking) received wisdom; all this and more come together into sharp focus in datacenters.

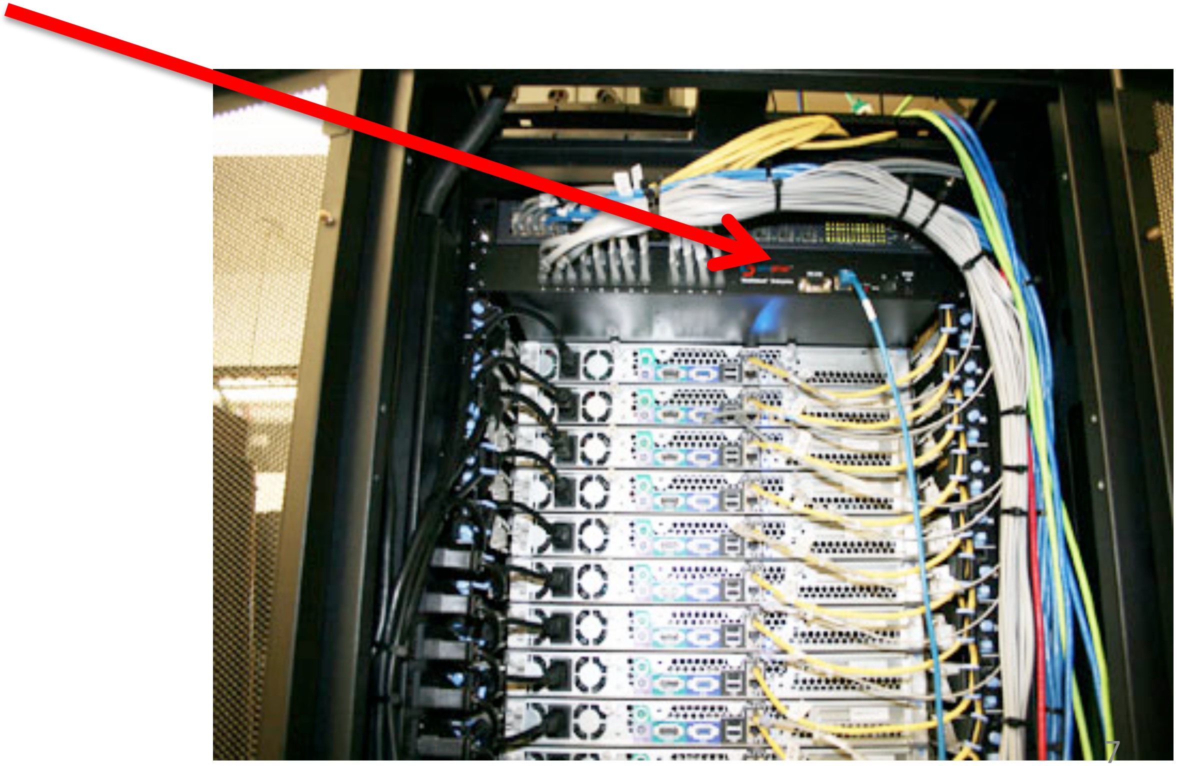
Do I need to labor the point?

What goes into a datacenter (network)?

- Servers organized in racks

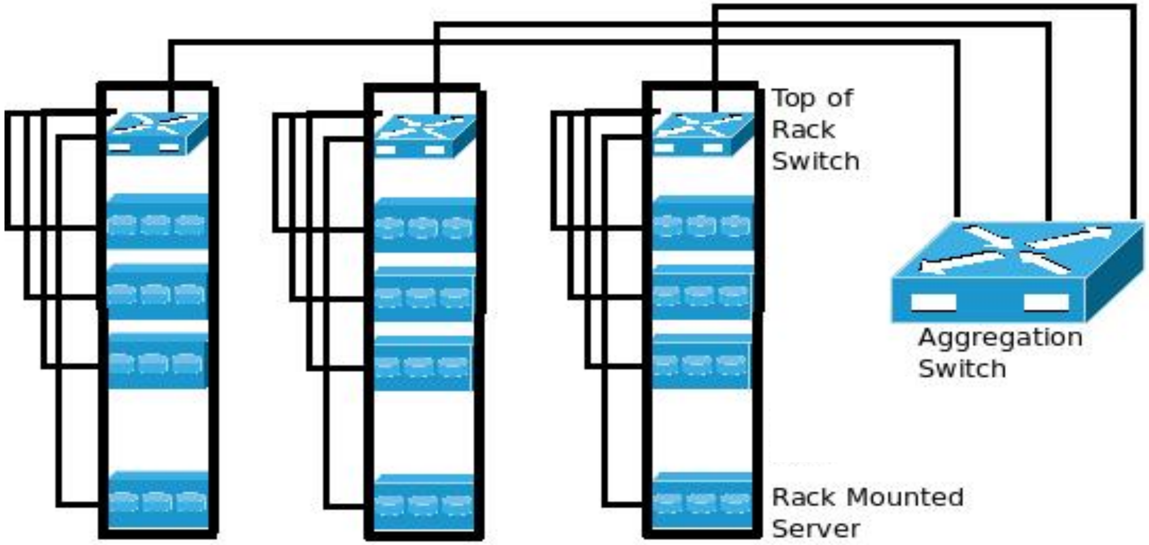
What goes into a datacenter (network)?

- Servers organized in racks
- Each rack has a `Top of Rack’ (ToR) switch



What goes into a datacenter (network)?

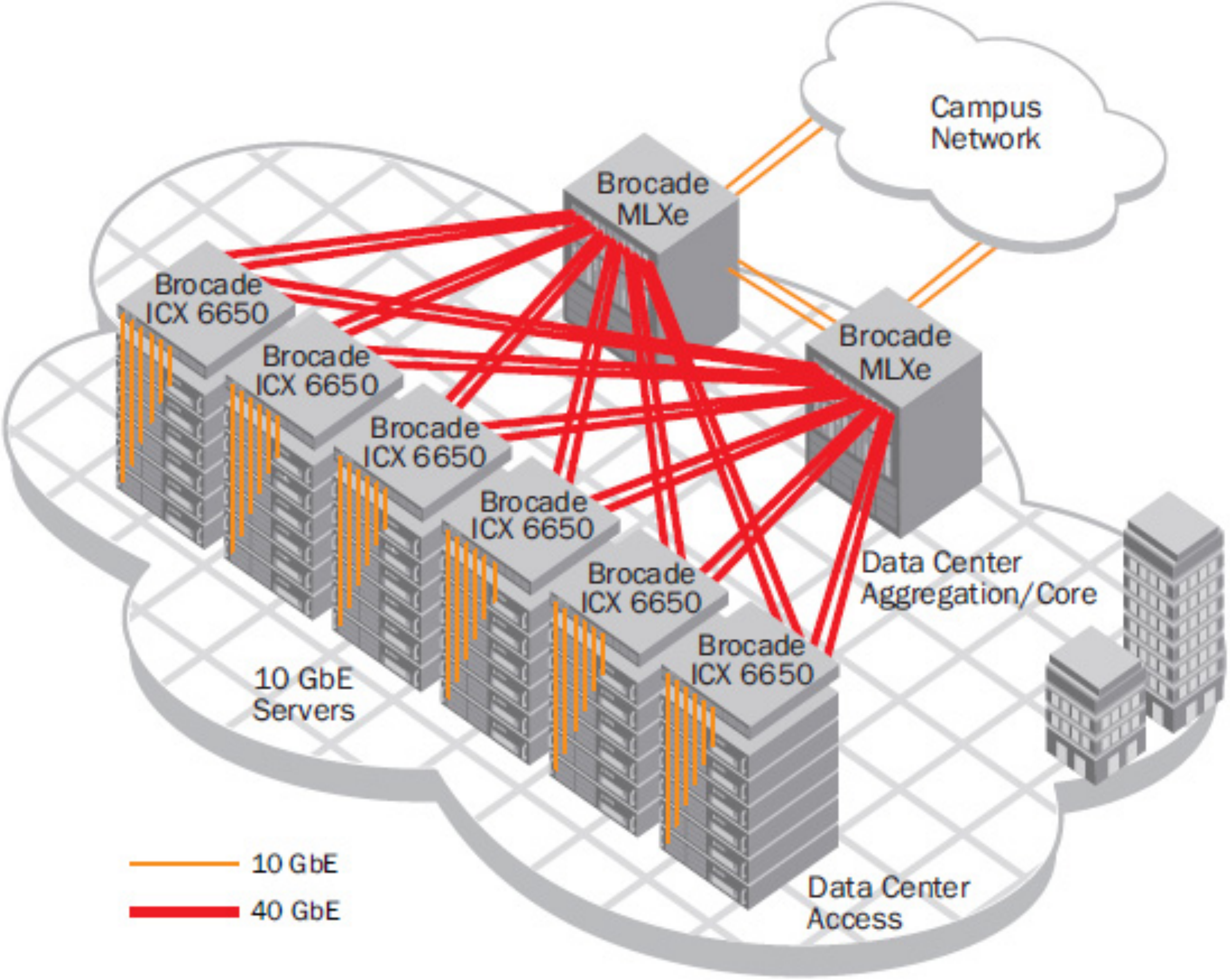
- Servers organized in racks
- Each rack has a `Top of Rack’ (ToR) switch
- An `aggregation fabric’ interconnects ToR switches



What goes into a datacenter (network)?

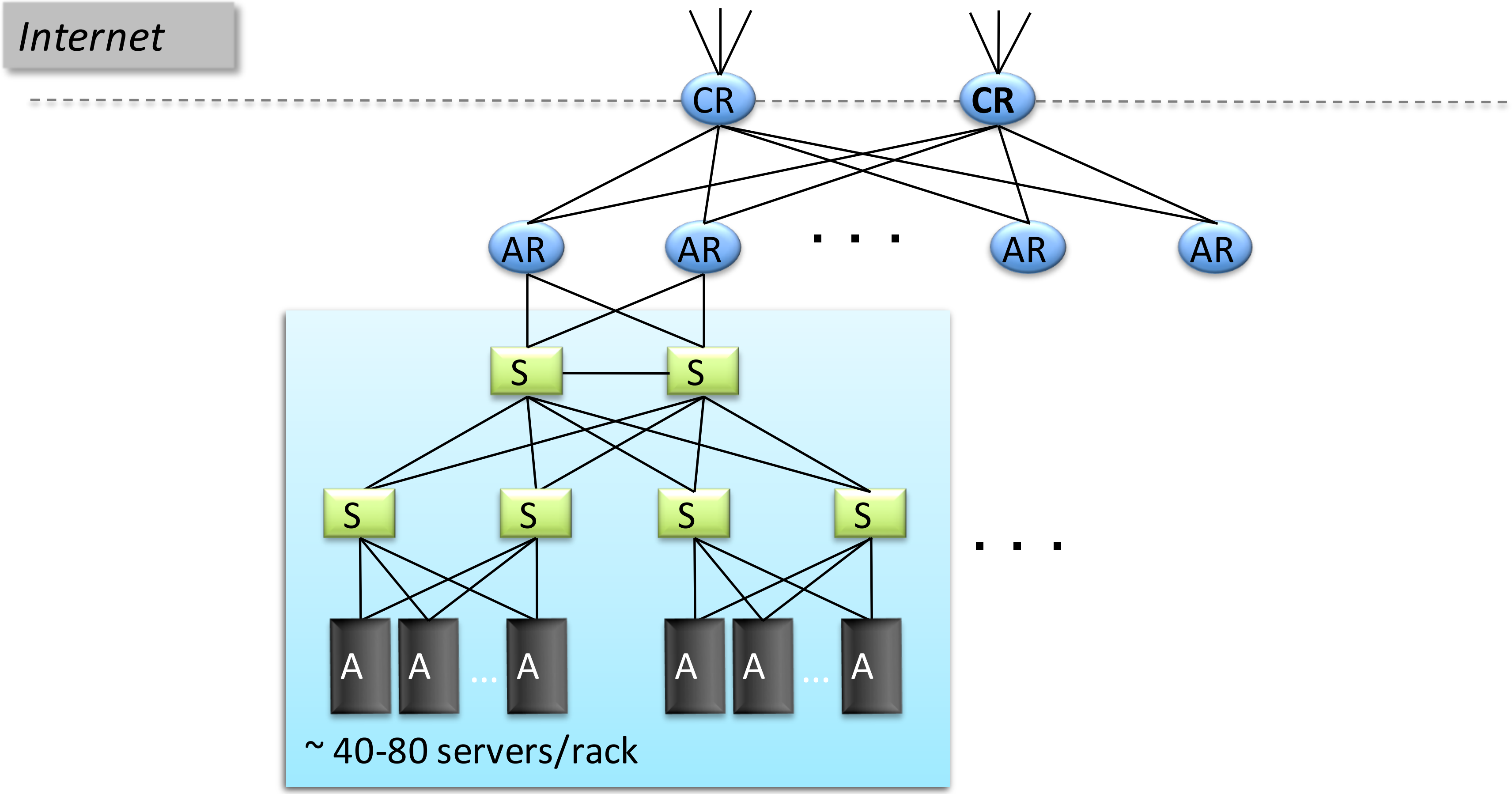
- Servers organized in racks
- Each rack has a `Top of Rack’ (ToR) switch
- An `aggregation fabric’ interconnects ToR switches
- Connected to the outside via `core’ switches
  – note: blurry line between aggregation and core
- With network redundancy of ~2x for robustness

Example 1



Brocade reference design

Example 2



Cisco reference design

Observations on DC architecture

- Regular, well-defined arrangement
- Hierarchical structure with rack/aggr/core layers
- Mostly homogenous within a layer
- Supports communication between servers and between servers and the external world

Contrast: ad-hoc structure, heterogeneity of WANs

What’s new?

SCALE!

How big exactly?

- 1M servers [Microsoft]
  - – less than google, more than amazon

- > $1B to build one site [Facebook]

- >$20M/month/site operational costs [Microsoft '09]

But only O(10-100) sites

What's new?

- Scale
- Service model
  - – user-facing, revenue generating services
  - – multi-tenancy
  - – jargon: SaaS, PaaS, DaaS, IaaS, …

Implications

- Scale
  - need scalable solutions (duh)
  - improving efficiency, lowering cost is critical

  → *`scale out' solutions w/ commodity technologies*

- Service model
  - performance means $$
  - *virtualization* for isolation and portability

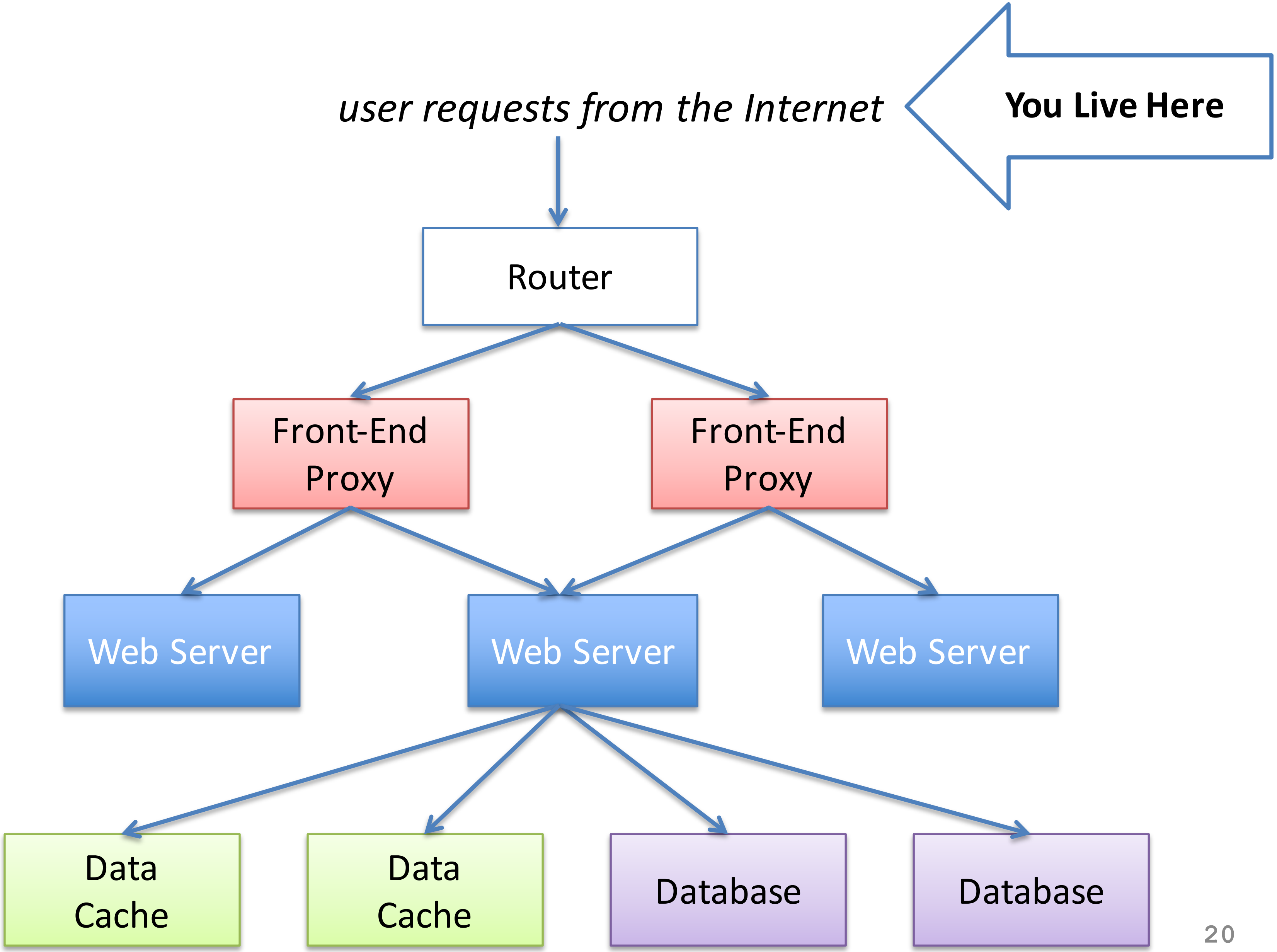
Multi-Tier Applications

- Applications decomposed into tasks
  - Many separate components
  - Running in parallel on different machines

Componentization leads to different types of network traffic

- "North-South traffic"
  - Traffic between external clients and the datacenter
  - Handled by front-end (web) servers, mid-tier application servers, and back-end databases
  - Traffic patterns fairly stable, though diurnal variations

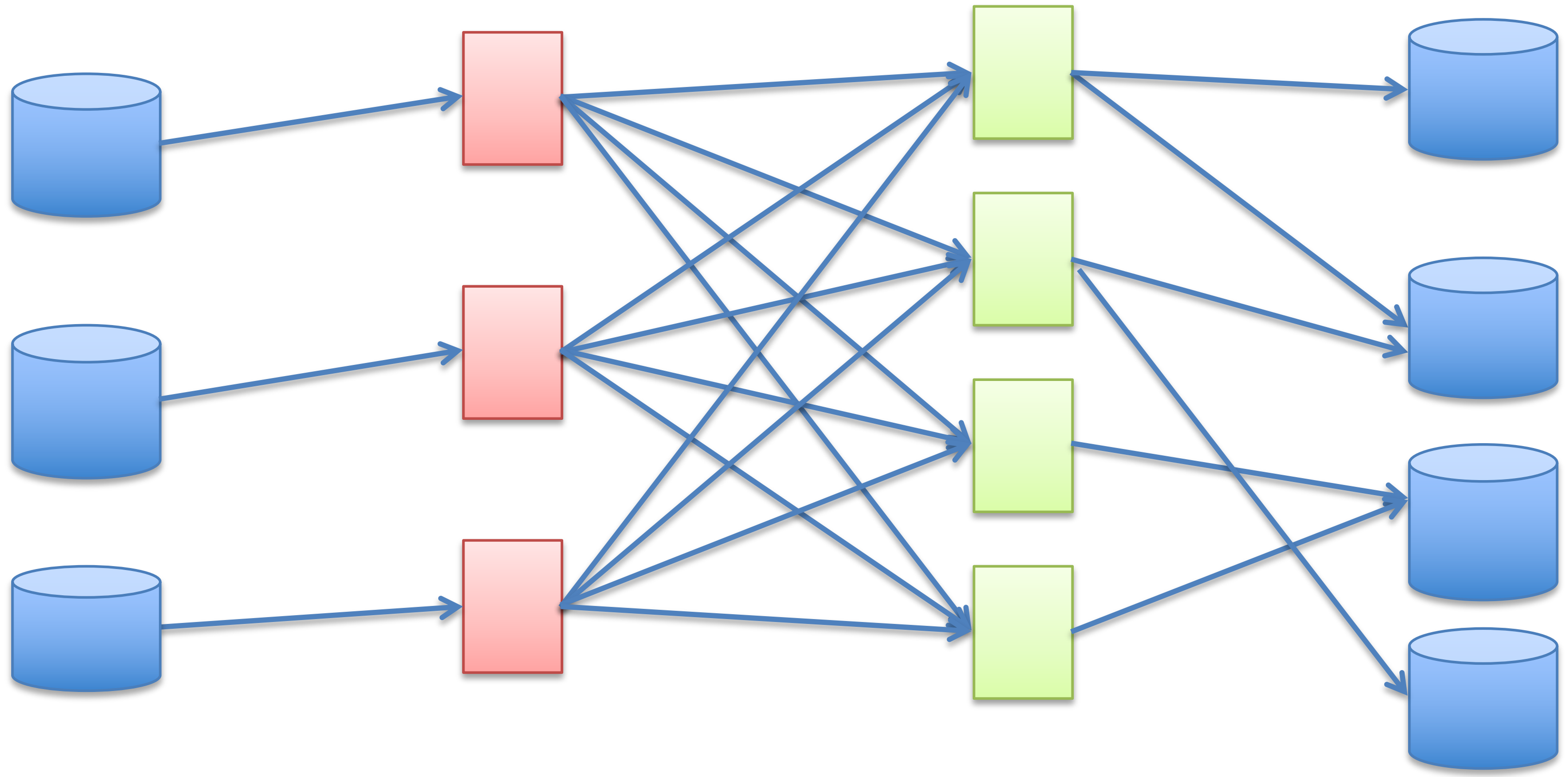
North-South Traffic



Componentization leads to different types of network traffic

- "North-South traffic"
  - Traffic between external clients and the datacenter
  - Handled by front-end (web) servers, mid-tier application servers, and back-end databases
  - Traffic patterns fairly stable, though diurnal variations
- "East-West traffic"
  - Traffic between machines in the datacenter
  - Comm *within* "big data" computations (e.g. Map Reduce)
  - Traffic may shift on small timescales (e.g., minutes)

East-West Traffic



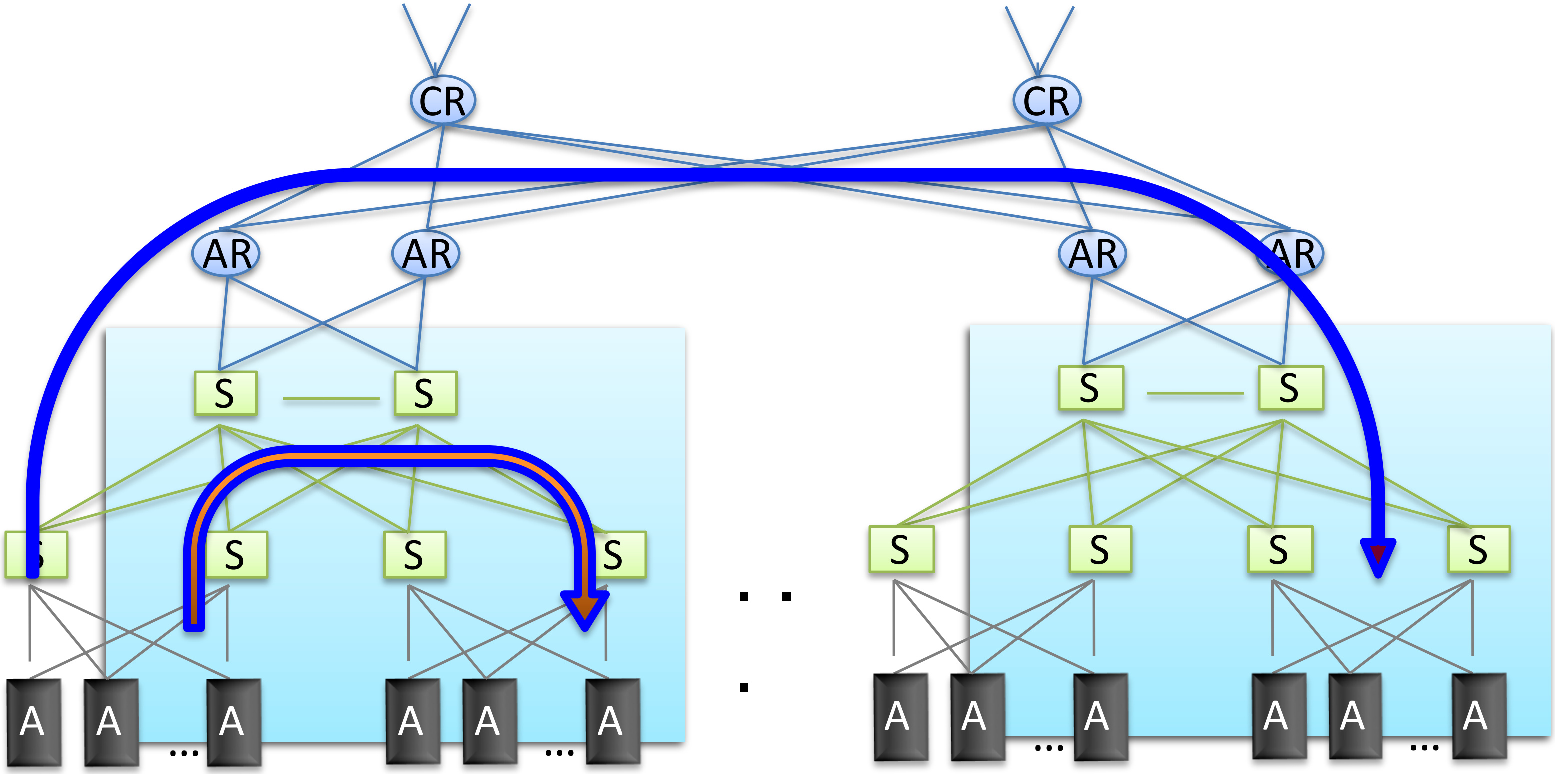
Distributed Storage

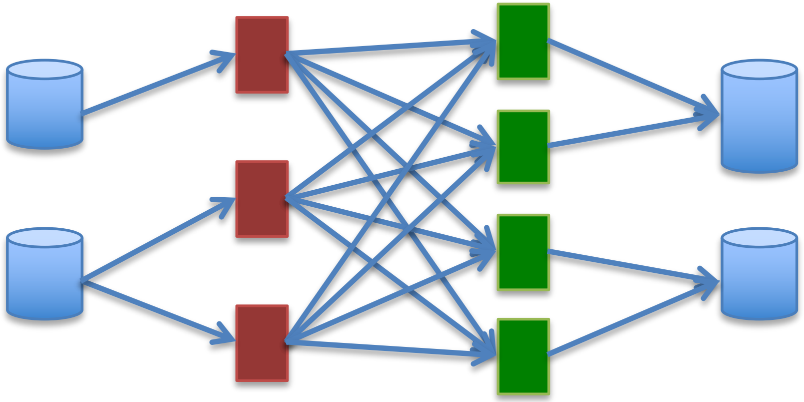
Map Tasks

Reduce Tasks

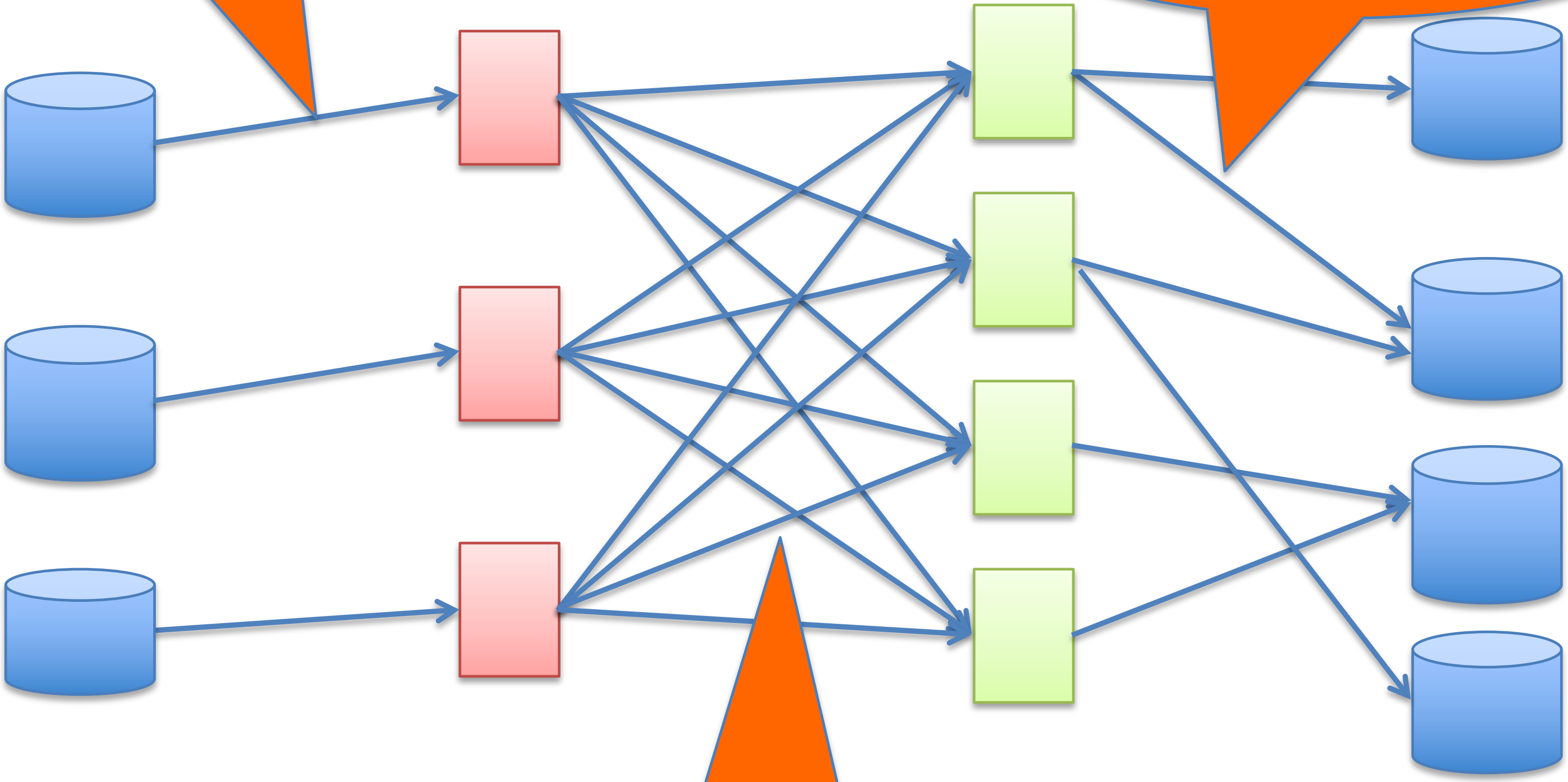
Distributed Storage

East-West Traffic





st-West Tr



Often doesn't cross the network

Some fraction (typically 2/3) crosses the network

Always goes over the network

Distributed Storage

Ma

duce

Distributed Storage

What’s different about DC networks?

Characteristics

- Huge scale:
  - ~20,000 switches/routers
  - *contrast: AT&T ~500 routers*

What's different about DC networks?

Characteristics

- Huge scale:
- Limited geographic scope:
  - High bandwidth: 10/40/100G
  - *Contrast: Cable/aDSL/WiFi*
  - Very low RTT: 10s of microseconds
  - *Contrast: 100s of milliseconds in the WAN*

What’s different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
  - Can deviate from standards, invent your own, *etc.*
  - “Green field” deployment is still feasible

What’s different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
  - can change (say) addressing, congestion control, *etc.*
  - can add mechanisms for security/policy/etc. at the endpoints (typically in the hypervisor)

What’s different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
- Control over the *placement* of traffic source/sink
  - e.g., map-reduce scheduler chooses where tasks run
  - alters traffic pattern (what traffic crosses which links)

What’s different about DC networks?

Characteristics

- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
- Control over the *placement* of traffic source/sink
- Regular/planned topologies (e.g., trees/fat-trees)
  - Contrast: ad-hoc WAN topologies (dictated by real-world geography and facilities)

What’s different about DC networks?

Characteristics

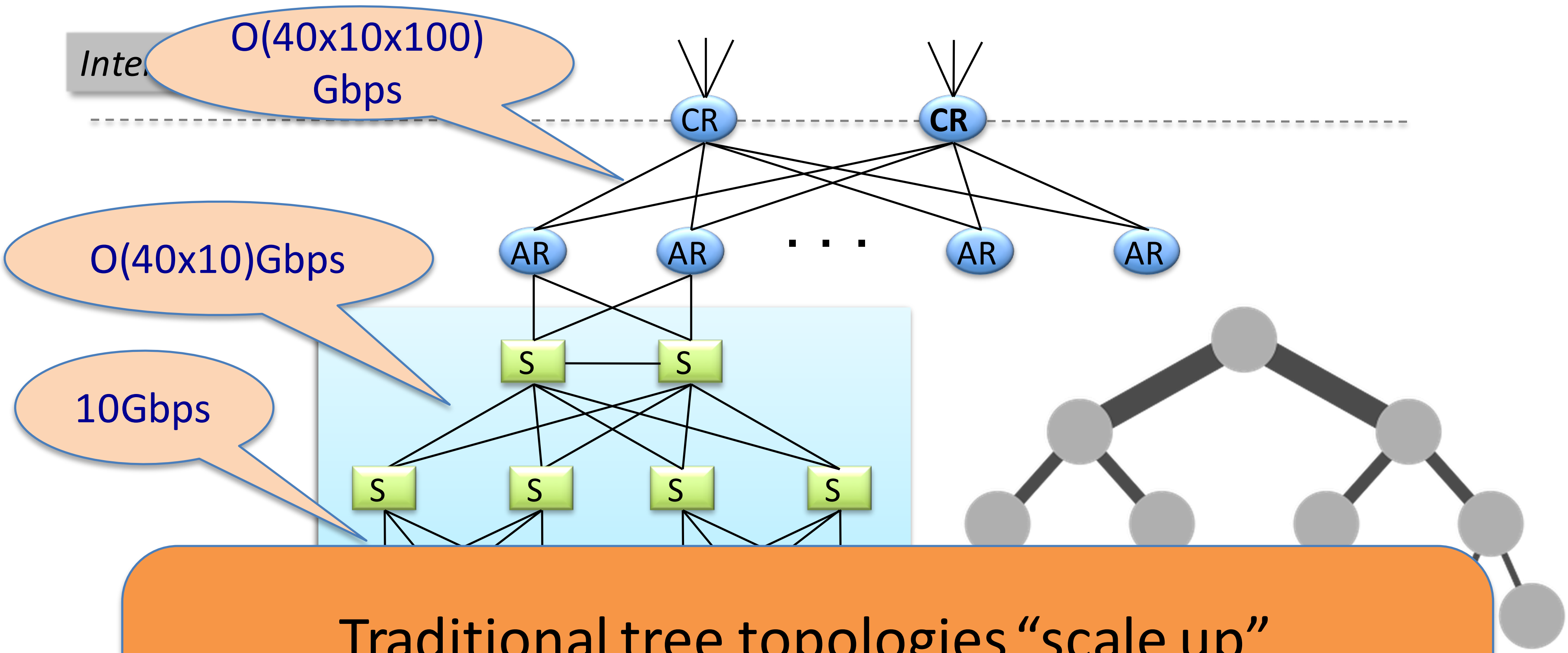
- Huge scale
- Limited geographic scope
- Single administrative domain
- Control over one/both endpoints
- Control over the *placement* of traffic source/sink
- Regular/planned topologies (e.g., trees/fat-trees)
- Limited heterogeneity
  - link speeds, technologies, latencies, …

What's different about DC networks?

Goals

- Extreme bisection bandwidth requirements
  - recall: all that east-west traffic
  - target: any server can communicate at its full link speed
  - problem: server's access link is 10Gbps!

Full Bisection Bandwidth



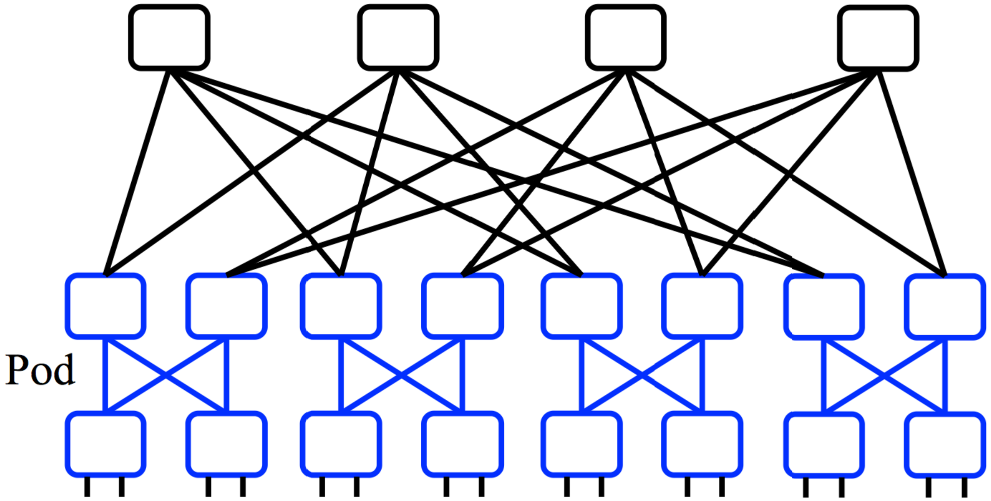
Traditional tree topologies “scale up”

- full bisection bandwidth is expensive
- typically, tree topologies “oversubscribed”

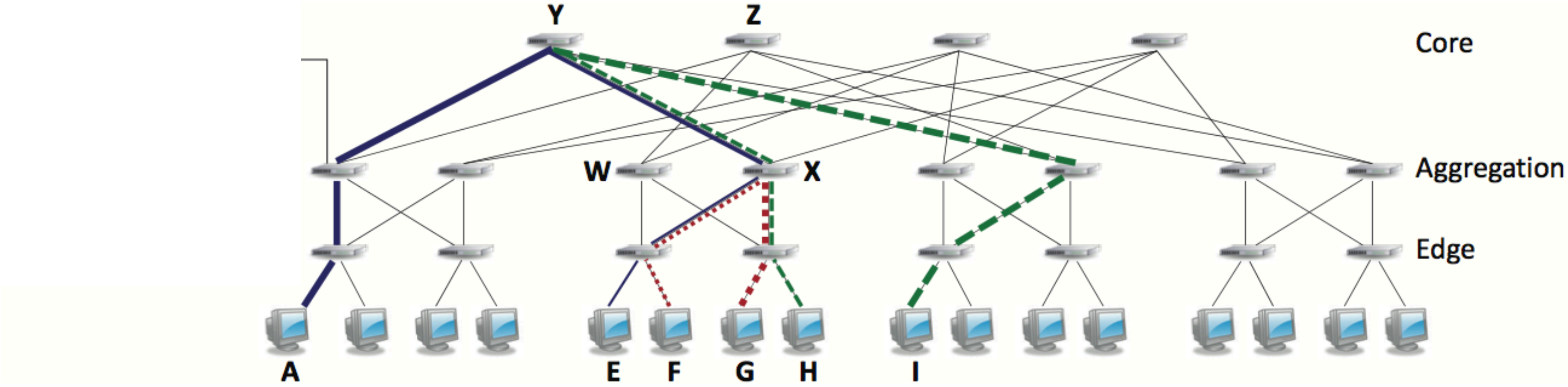
A “Scale Out” Design

- Build multi-stage `Fat Trees’ out of k-port switches
  - – k/2 ports up, k/2 down
  - – Supports k^3/4 hosts:
    - 48 ports, 27,648 hosts

All links are the same speed (e.g. 10Gps)



Full Bisection Bandwidth Not Sufficient



- To realize full bisectional throughput, routing must spread traffic across paths
- Enter load-balanced routing
  - How? (1) Let the network split traffic/flows at random (e.g., ECMP protocol -- RFC 2991/2992)
  - How? (2) Centralized flow scheduling?
  - Many more research proposals

What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
  - real money on the line
  - current target: 1μs RTTs
  - how? cut-through switches making a comeback
    - reduces switching time

What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
  - real money on the line
  - current target: 1μs RTTs
  - how? cut-through switches making a comeback
  - how? avoid congestion
    - reduces queuing delay

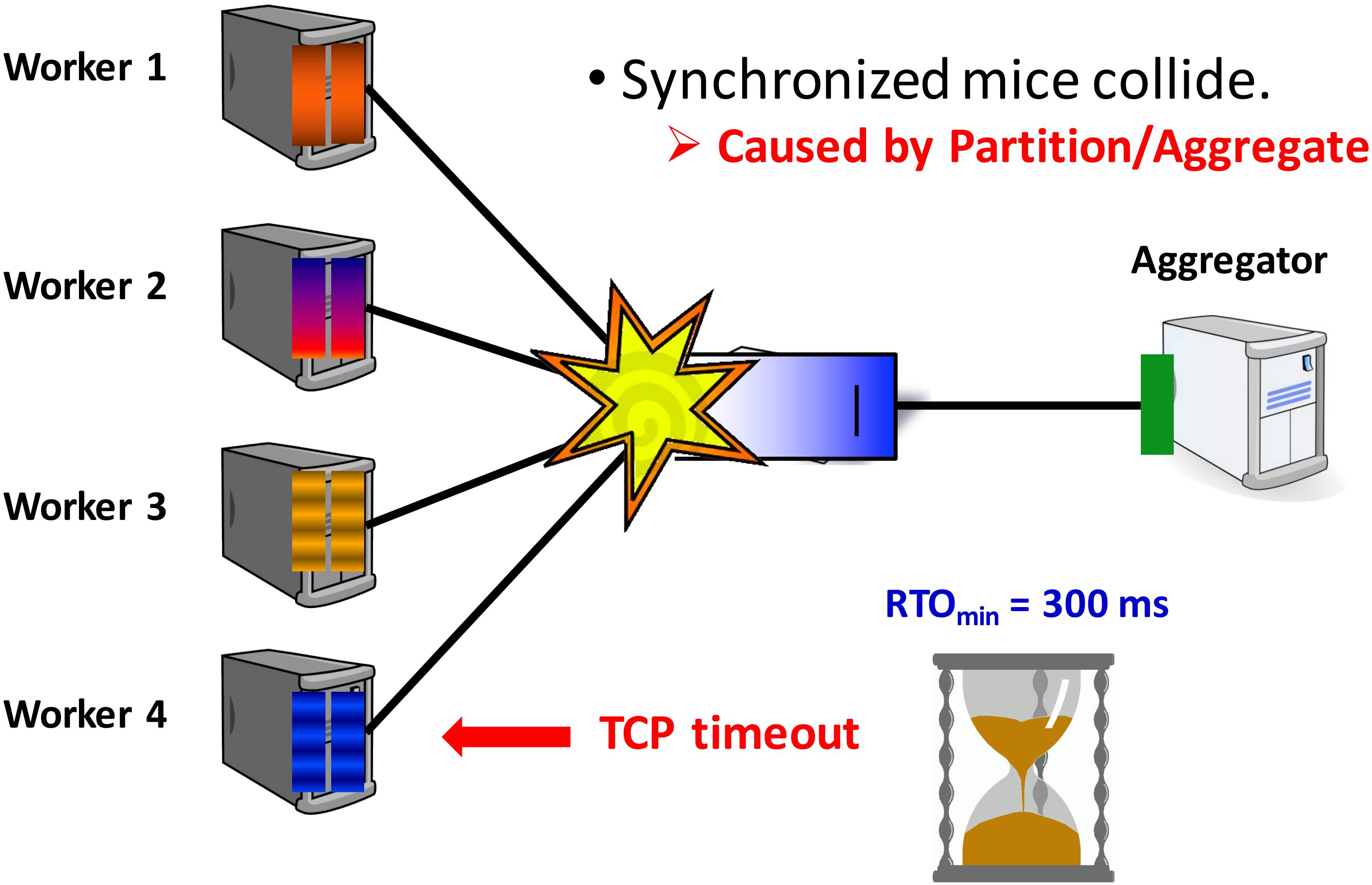
What's different about DC networks?

Goals

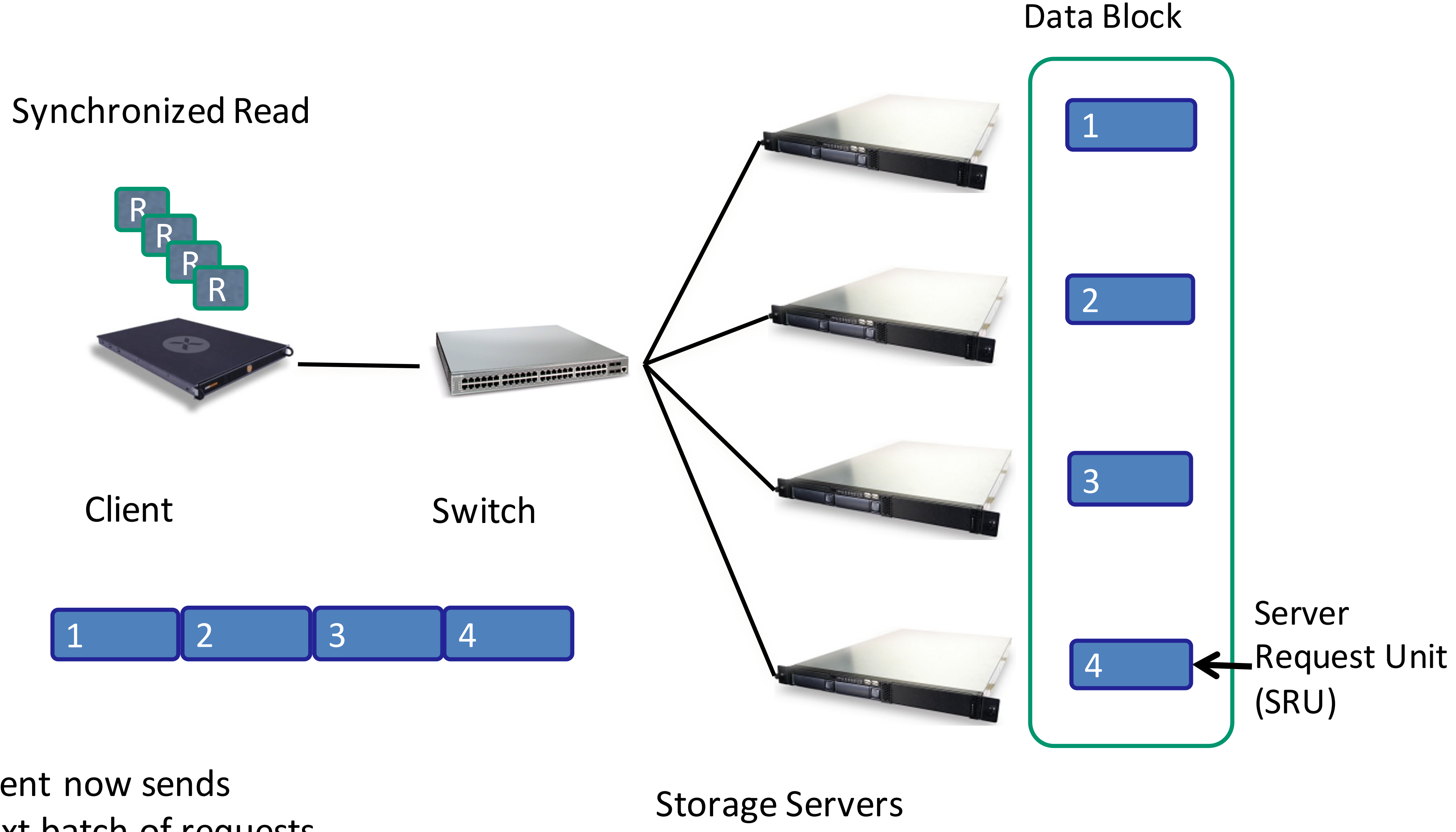
- Extreme bisection bandwidth requirements
- Extreme latency requirements
  - real money on the line
  - current target: 1μs RTTs
  - how? cut-through switches making a comeback (lec. 2!)
  - how? avoid congestion
  - how? fix TCP timers (e.g., default timeout is 500ms!)
  - how? fix/replace TCP to more rapidly fill the pipe

An example problem at scale - INCAST

- Synchronized mice collide.
  - **Caused by Partition/Aggregate.**

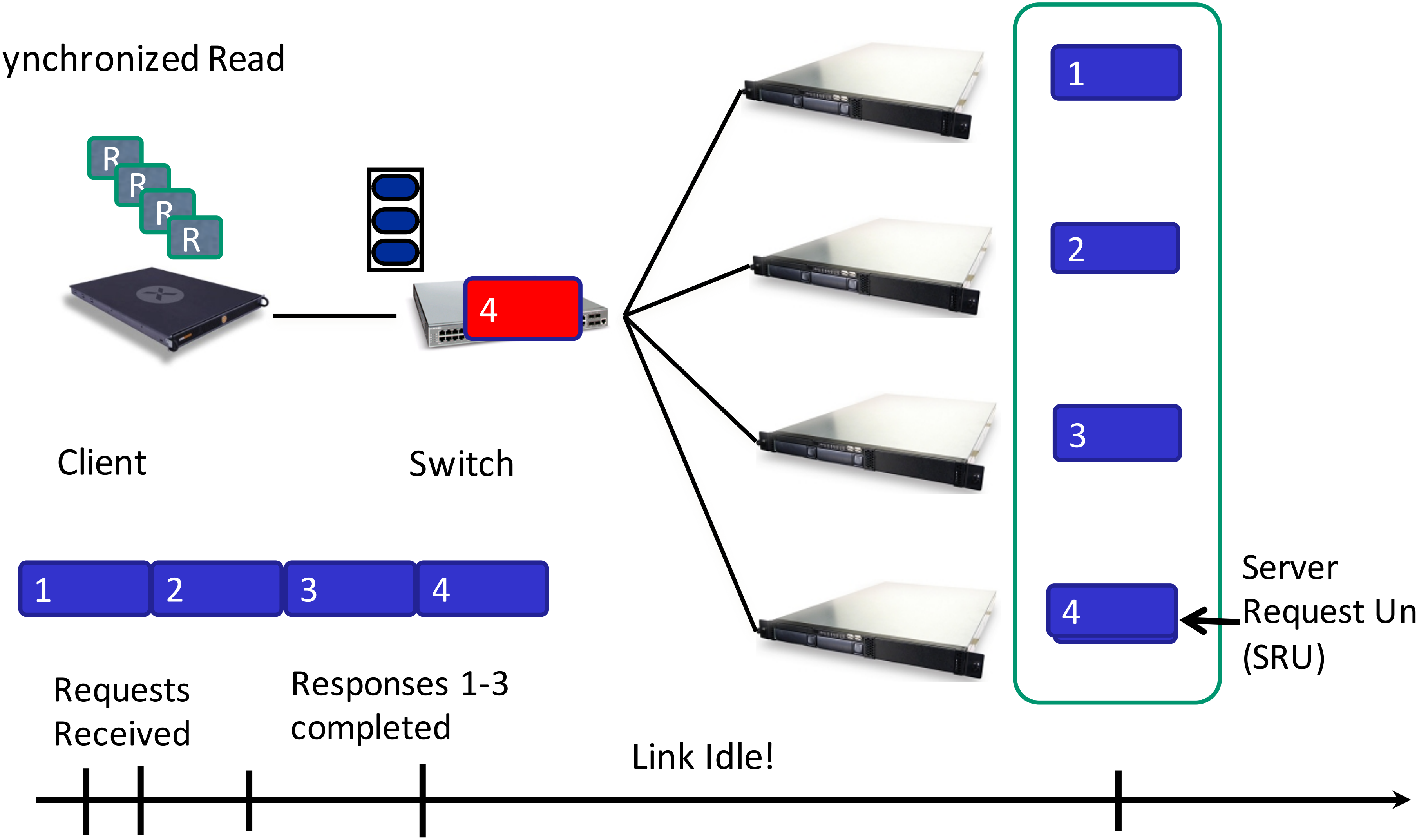


The Incast Workload



Client now sends next batch of requests

Incast Workload Overfills Buffers



Synchronized Read

Client

Switch

Server Request Unit (SRU)

Requests Received

Responses 1-3 completed

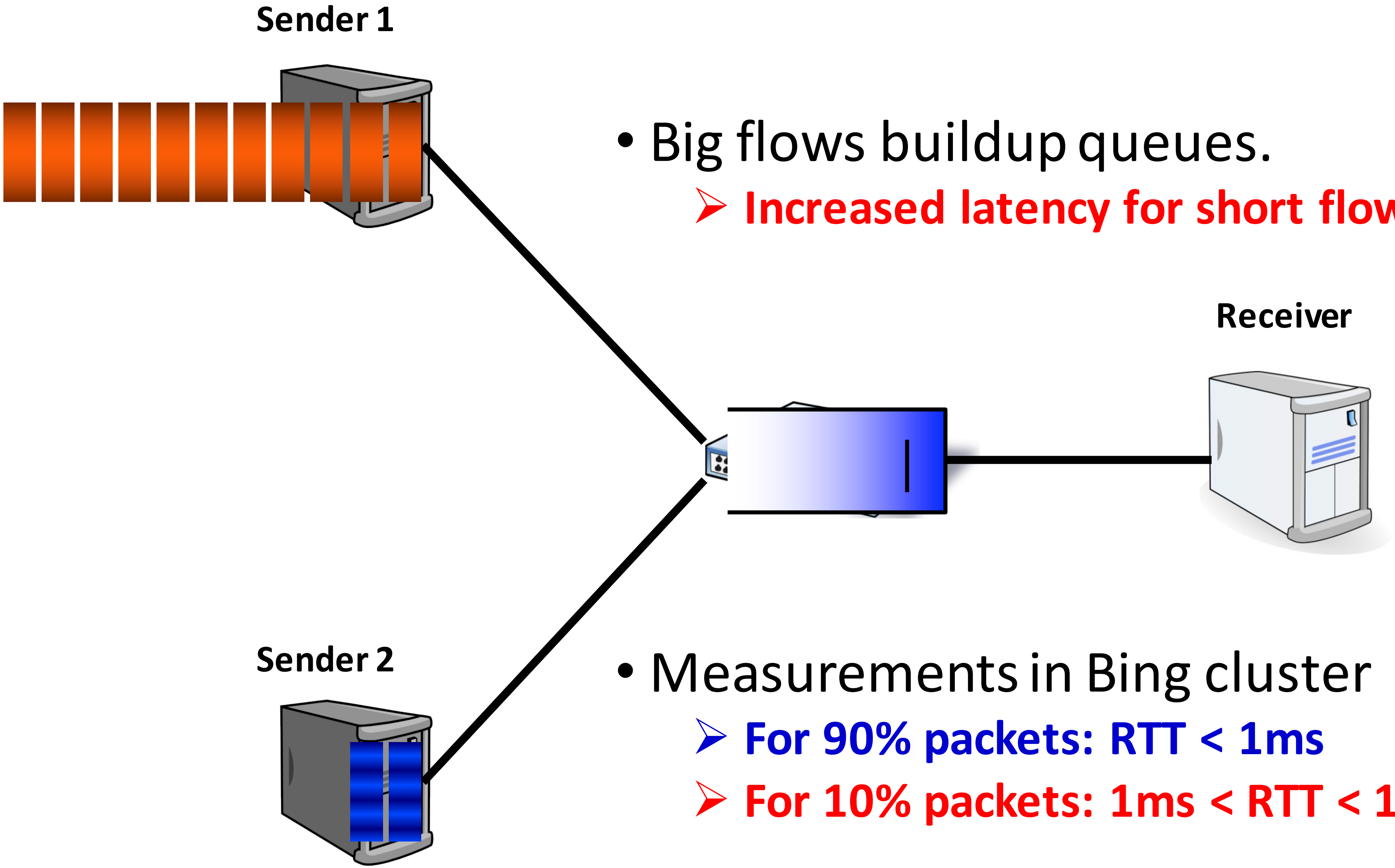
Link Idle!

Requests Sent

Response 4 dropped

Response 4 Resent

Queue Buildup

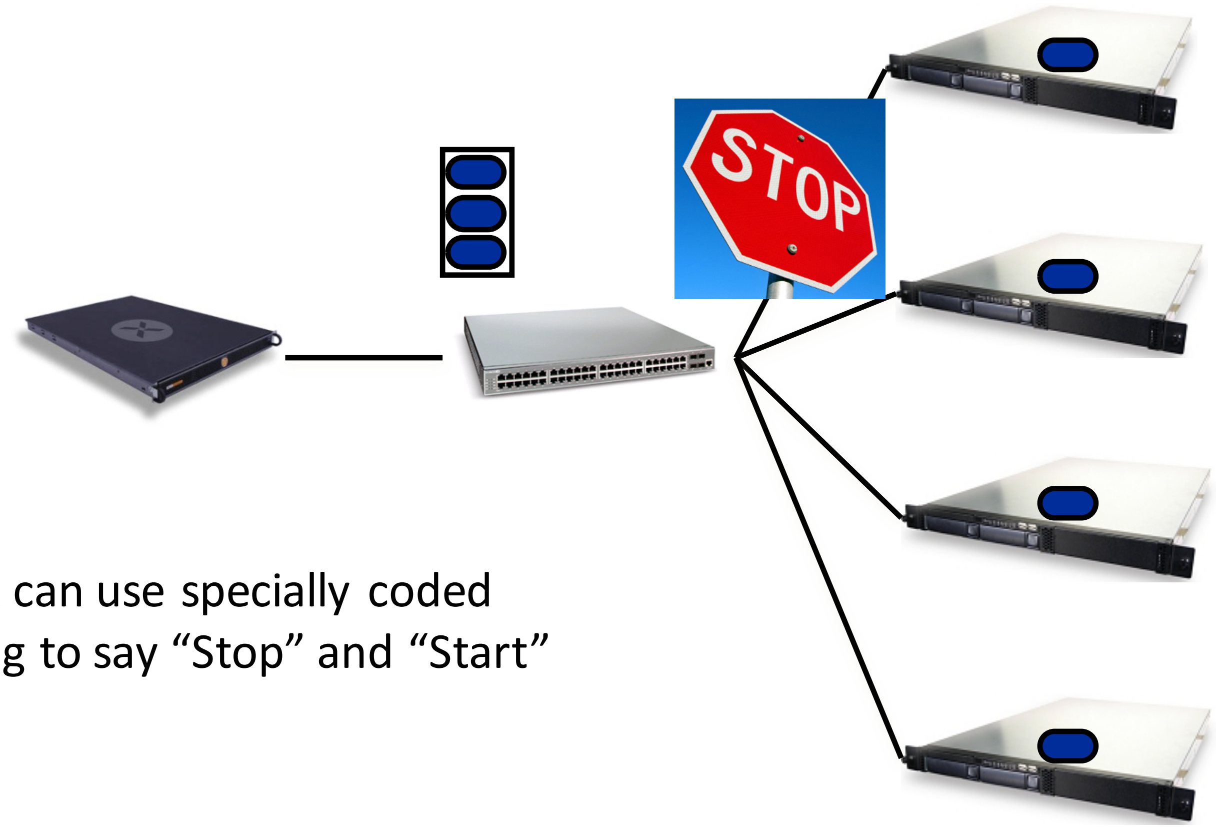


- Big flows buildup queues.
  - ➢ **Increased latency for short flows.**
- Measurements in Bing cluster
  - ➢ **For 90% packets: RTT < 1ms**
  - ➢ **For 10% packets: 1ms < RTT < 15ms**

Link-Layer Flow Control

Common between switches but this is flow-control to the end host too…

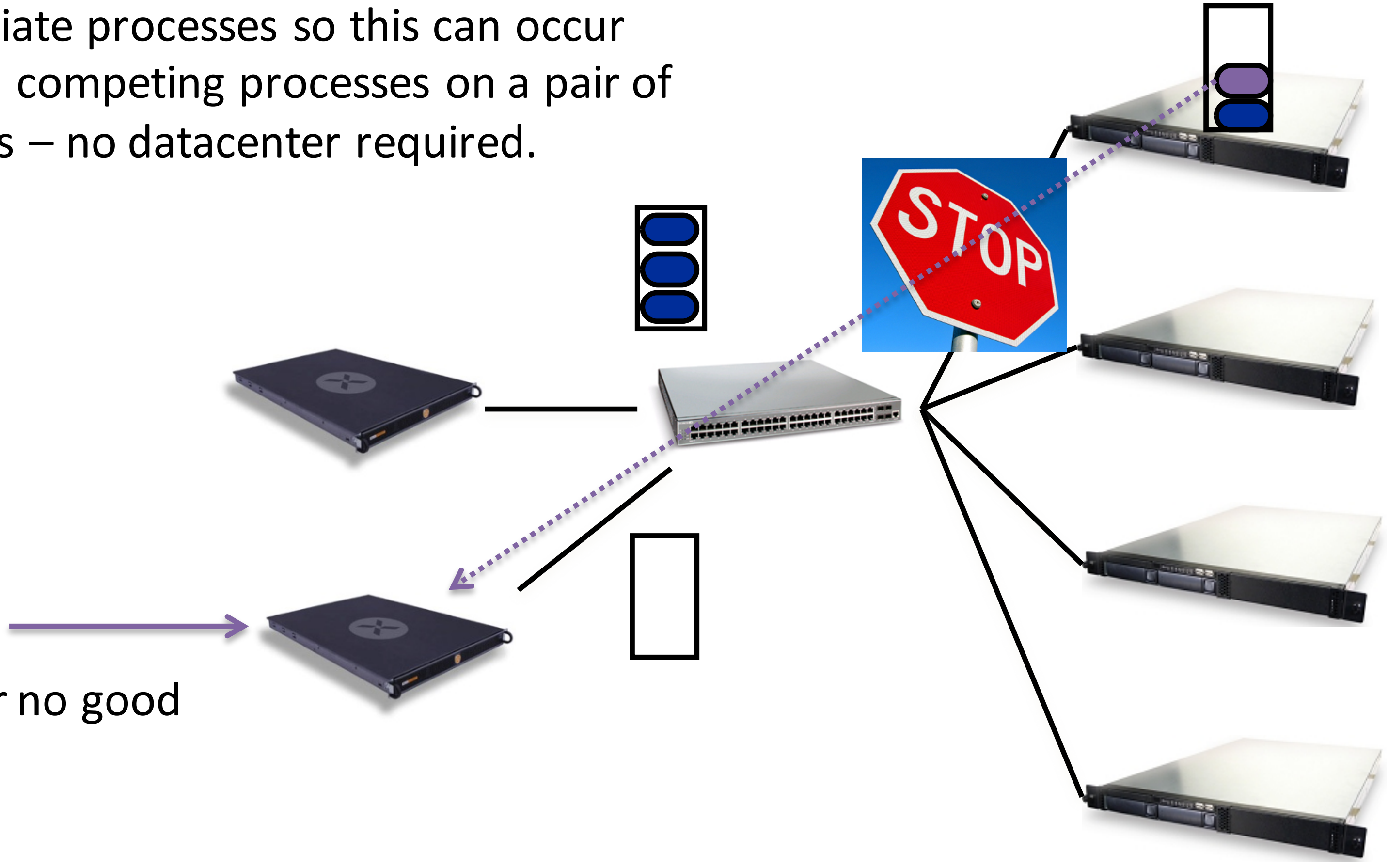
- Another idea to reduce incast is to employ Link-Layer Flow Control…..



Recall: the Data-Link can use specially coded symbols in the coding to say “Stop” and “Start”

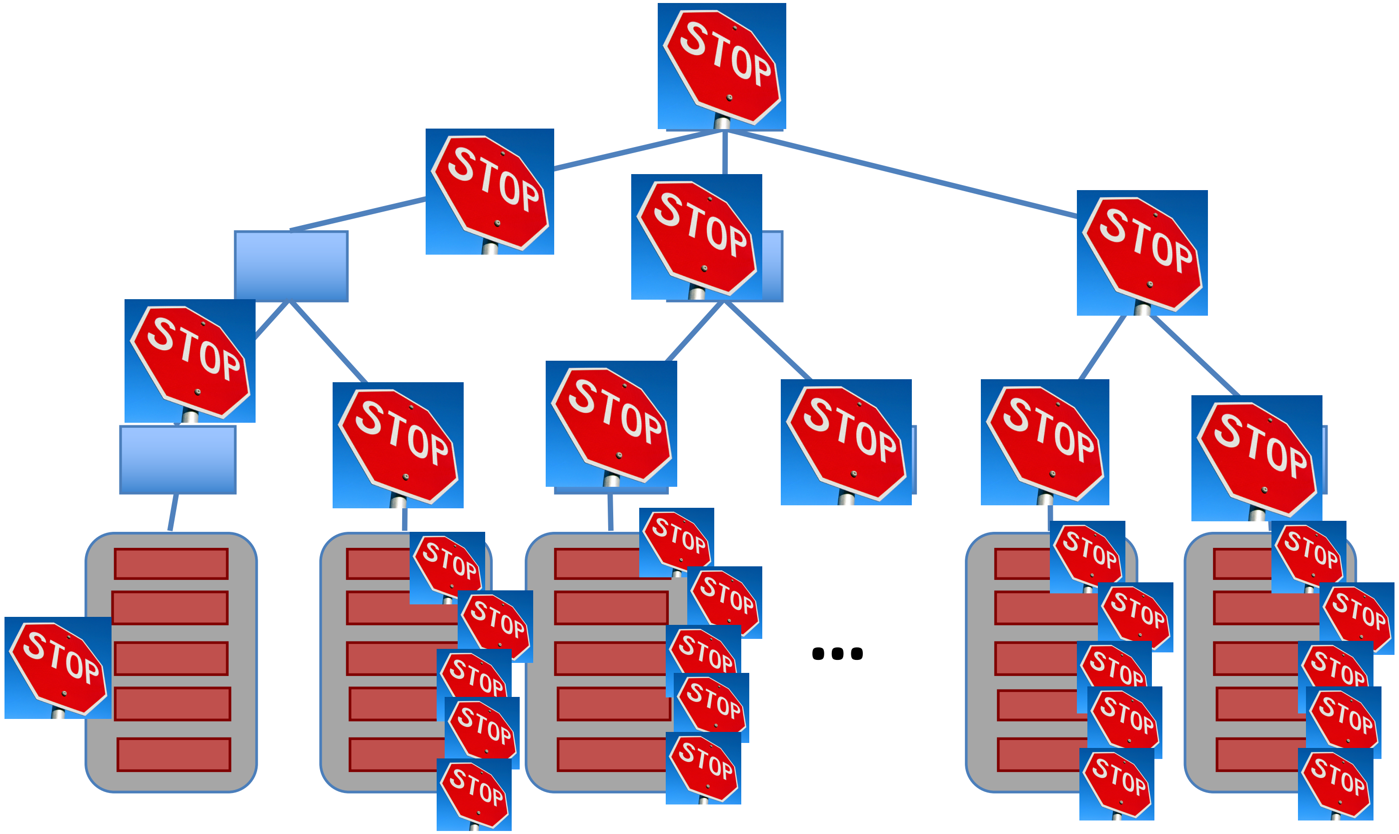
Link Layer Flow Control – The Dark side Head of Line Blocking….

Such HOL blocking does not even differentiate processes so this can occur between competing processes on a pair of machines – no datacenter required.



Waiting for no good reason….

Link Layer Flow Control
But its worse that you imagine….



Double down on trouble….

Did I mention this is Link-Layer!

That means no (IP) control traffic, no routing messages….

a whole system waiting for one machine

Incast is very unpleasant.

Reducing the impact of HOL in Link Layer Flow Control can be done through priority queues and *overtaking*….

What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
  - “your packet will reach in Xms, or not at all”
  - “your VM will always see at least YGbps throughput”
  - Resurrecting `best effort’ vs. `Quality of Service’ debates
  - How is still an open question

What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
- Differentiating between tenants is key
  - e.g., “No traffic between VMs of tenant A and tenant B”
  - “Tenant X cannot consume more than XGbps”
  - “Tenant Y’s traffic is low priority”

What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
- Differentiating between tenants is key
- Scalability (of course)
  - Q: How’s that Ethernet spanning tree looking?

What’s different about DC networks?

Goals

- Extreme bisection bandwidth requirements
- Extreme latency requirements
- *Predictable, deterministic* performance
- Differentiating between tenants is key
- Scalability (of course)
- Cost/efficiency
  - focus on commodity solutions, ease of management
  - some debate over the importance in the network case

Summary

- new characteristics and goals
- some liberating, some constraining
- scalability is the baseline requirement
- more emphasis on performance
- less emphasis on heterogeneity
- less emphasis on interoperability

Computer Networking UROP

- Assessed Practicals for Computer Networking.
  - so supervisors can set/use work
  - so we can have a Computer Networking *tick*

running over summer 2016

Talk to me.

Part 2 projects for 16-17

- Fancy doing something at scale or speed?

Talk to me.