

CS168

How the Internet Works:

A bottom-up view

Sylvia Ratnasamy

Fall 2026

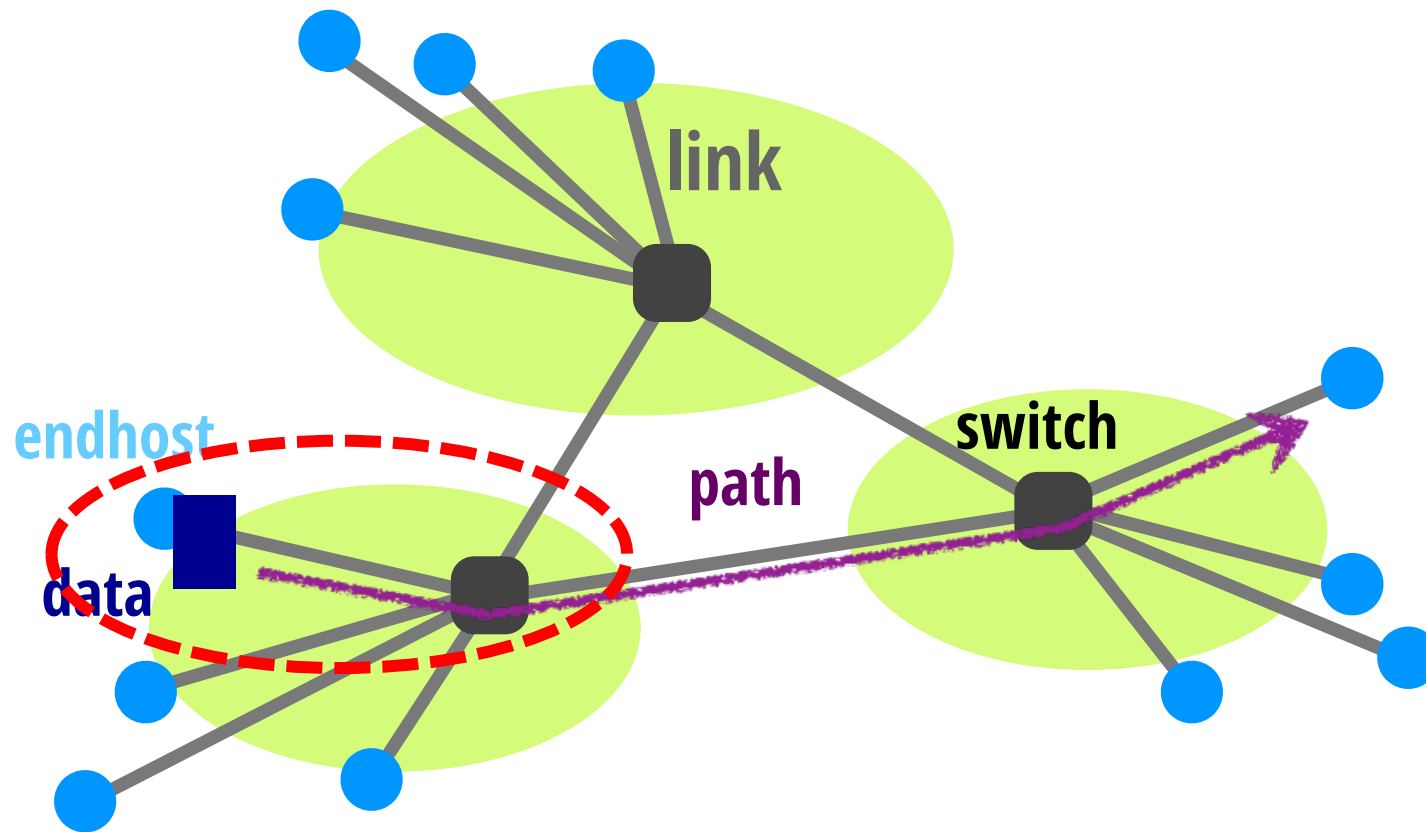
Goal for the next few lectures is to give you a broad overview of how the Internet works

- This lecture: bottom-up
 - Identify the fundamental pieces that make up the overall picture
- Next lecture: top-down
 - Identify the important architectural choices involved in the picture together

Today

- How is data transferred across the Internet?
- How are network resources shared?
- Start understanding of the “life of a packet” through the network
- Along the way: identify the key topics we’ll be studying this semester

Recall, from last lecture



The goal of the Internet is to transfer data between end hosts

How is data organized (in the network)?



Application data



Recap: packets

- Packets are a chunk of bits with:
 - **Payload:** meaningful only to the endpoints
 - Bits from a file, video, etc.
 - **Header:** meaningful to the network and endpoint
 - What information must a header contain? **The destination address!**
- In practice, a packet has multiple headers (next lecture)
- And communication between a pair of endhosts involves multiple packets
 - “Flow”: stream of packets exchanged between two endpoints (more on this later)

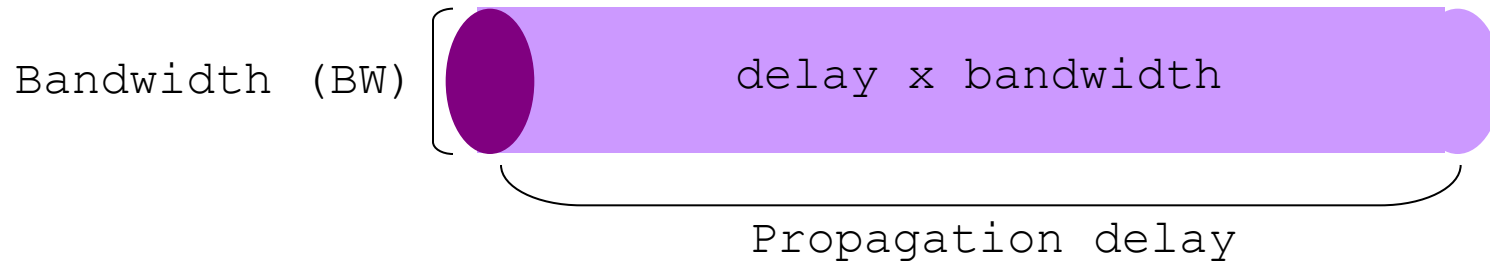
Packets on a link



Application data

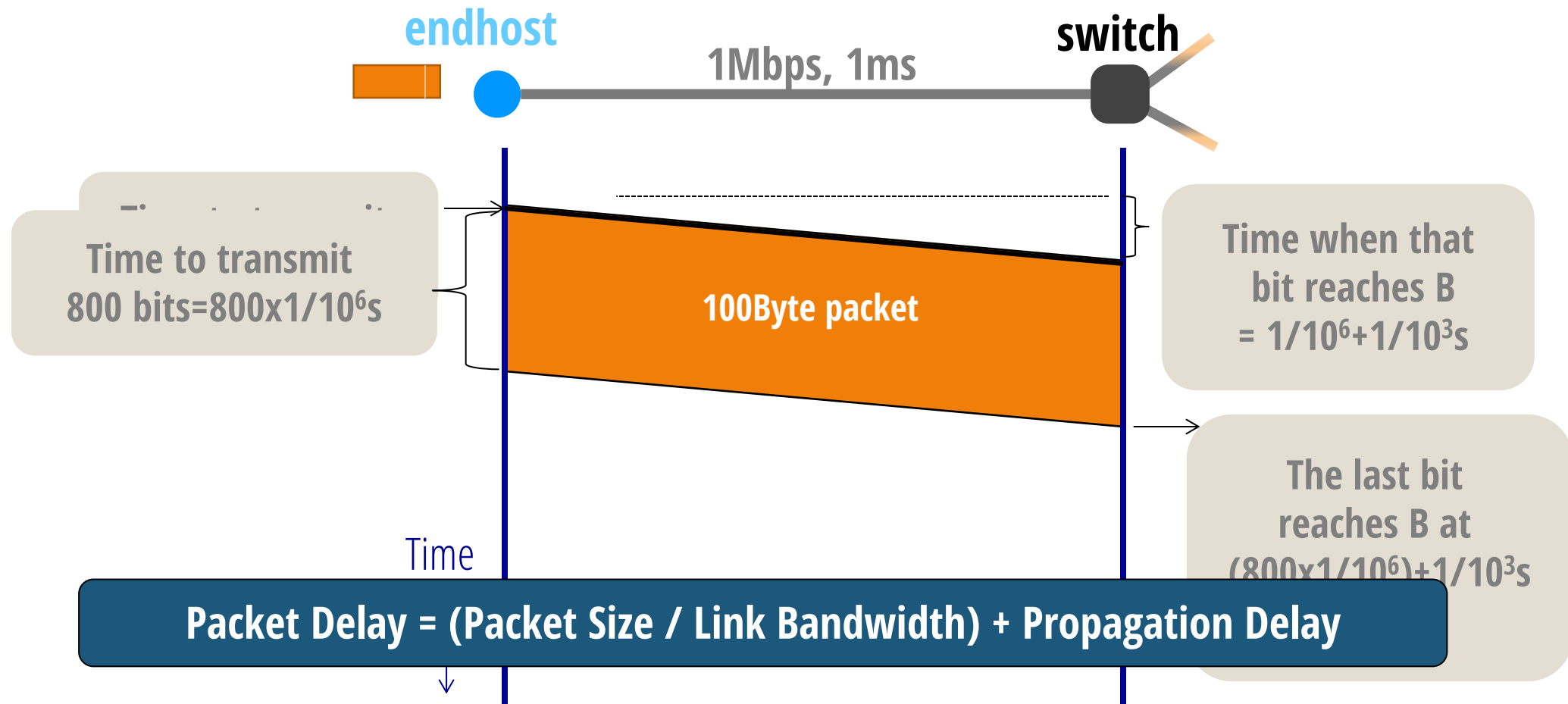


Properties of links



- **Bandwidth:** number of bits sent (or received) per unit time (bits/second or bps)
 - “width” of the link
- **Propagation delay:** time it takes a bit to travel along the link (seconds)
 - “length” of the link
- **Bandwidth-Delay Product (BDP):** bits/time $\times$ propagation delay (bits)
 - “capacity” of the link

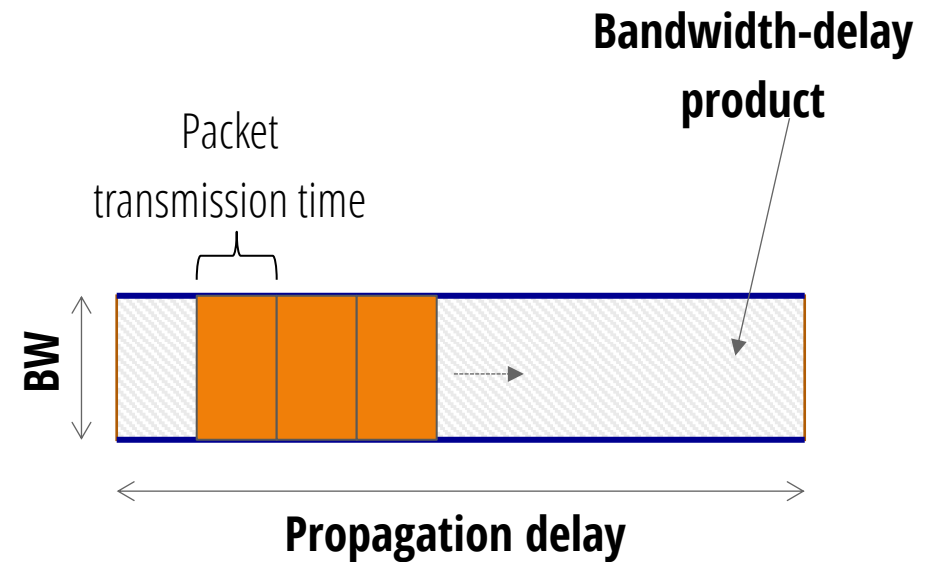
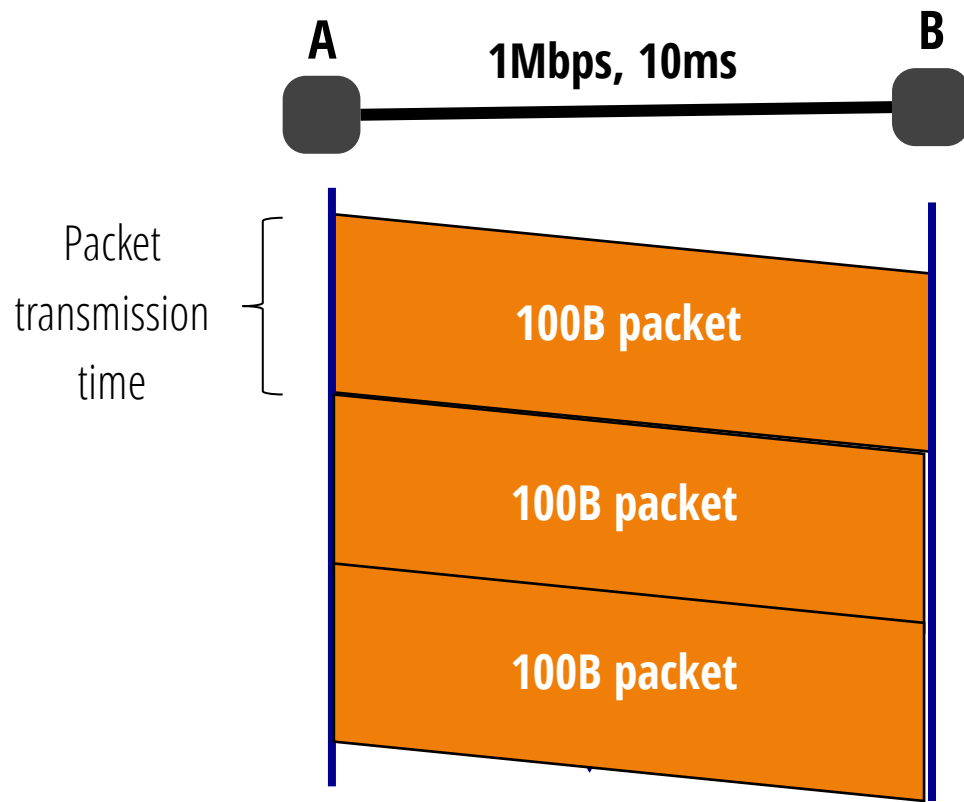
Packets on a link: sending a 100B packet



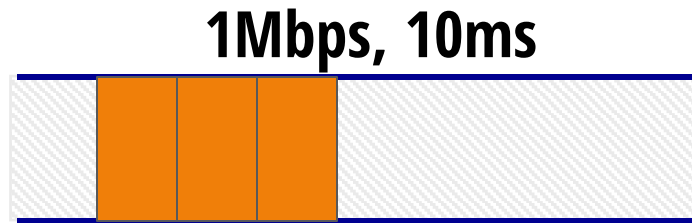
Question: which link is better?

- Link-1: bandwidth=10Mbps and propagation delay = 10ms
- Link-2: bandwidth=1Mbps and propagation delay = 1ms
- Packet delay for a **10B** packet:
 - With link 1: ~10ms
 - With link 2: ~1ms
- For a **10,000B** packet:
 - Link 1: ~18ms
 - Link 2: ~81ms

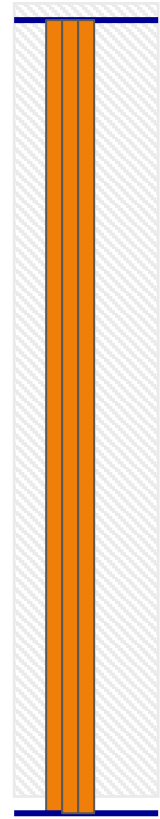
Packets on a link: an alternate “pipe” view



Packets on a link: an alternate “pipe” view



1Mbps, 5ms ?

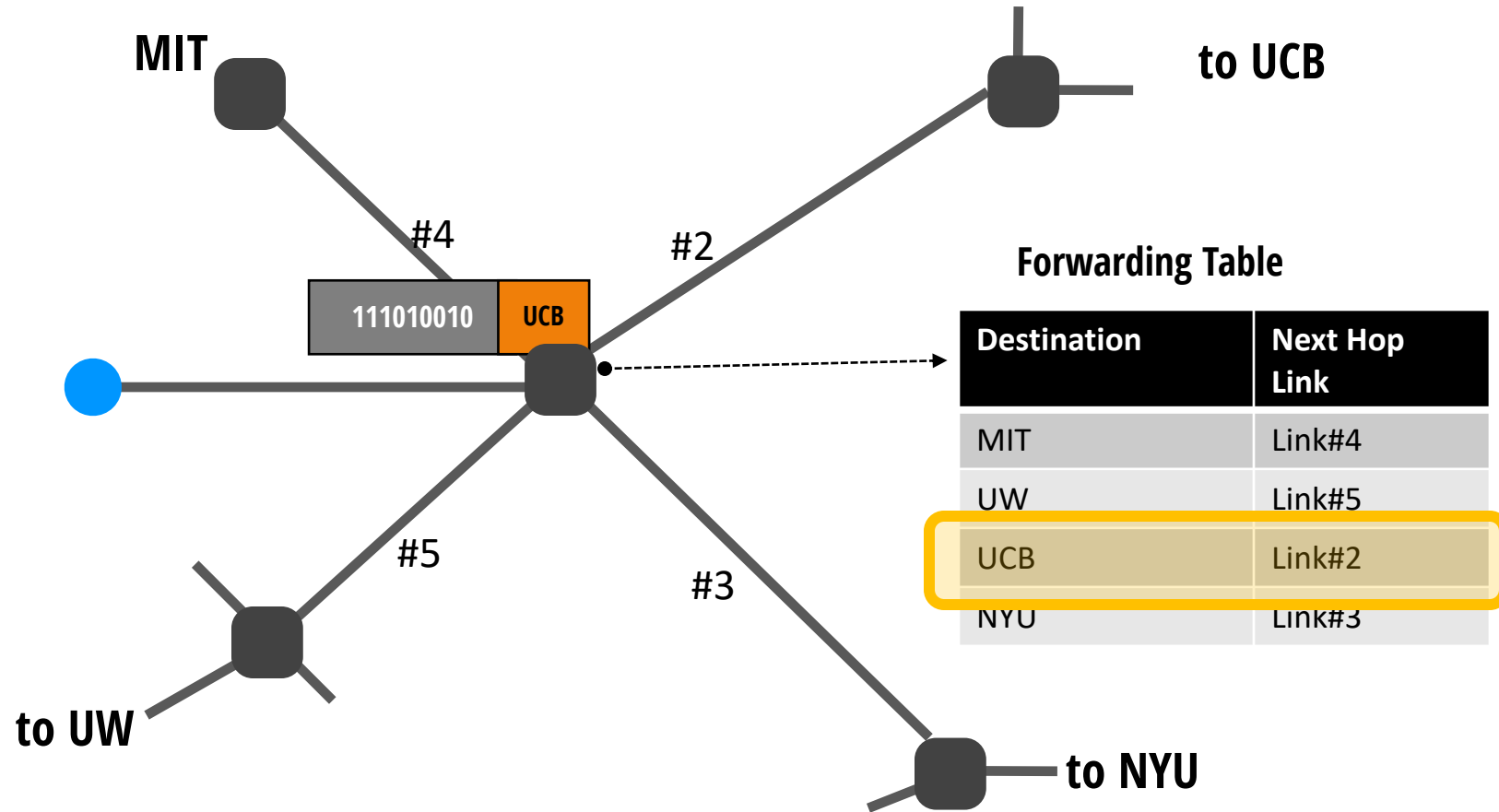


10Mbps, 1ms ?

Recap: packet on a link



Switches “forward” packets

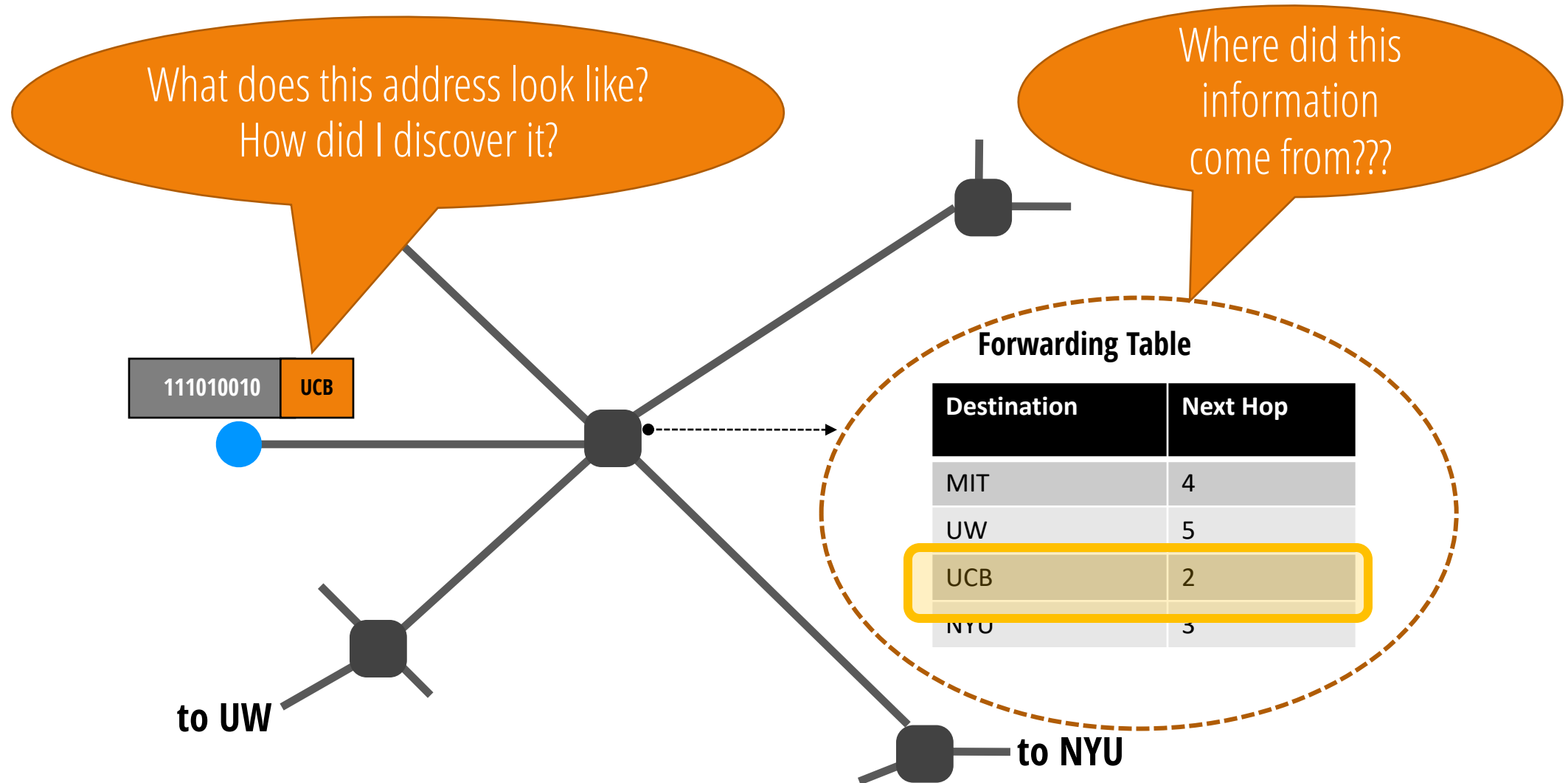


Recap: life of a packet so far...

- Source has some data to send to a destination
- Chunks it up into packets: each packet has a payload and a header
- Packet travels along a link
- Arrives at a switch; switch forwards the packet to its next hop

And the last two steps repeat until we reach the destination...

What are the fundamental challenges in this?



What are the fundamental challenges in this?

Challenge: addressing and naming

- In the real world, we have *names* and *addresses*
 - E.g., my name is Sylvia; my address ~~is~~ was 413 Soda Hall
 - When I moved to the Gateway building: my name didn't change but my address did
- Network **address**: where host is located
- Network **name**: which host it is
- Need an addressing and naming scheme that works at Internet scale!

Will discuss IP addressing a few lectures from now

Challenge: mapping names to addresses

- Consider when you access a web page
 - Insert URL into browser (e.g., *eeecs.berkeley.edu*)
 - You want to communicate with the server hosting *eeecs.berkeley.edu* content
- How do you get to the server?
 - URL is the **name** (e.g., *eeecs.berkeley.edu*)
 - Network needs **address** (e.g., where is *eeecs.berkeley.edu*)
- Must map – or “resolve” -- host names to addresses
- Done by the **Domain Name System (DNS)**

Will cover DNS in a later lecture (second half of semester)

Challenge: Routing

- When a packet arrives at a router, the **forwarding table** determines which outgoing link the packet is sent on
- How do you compute the forwarding tables necessary to deliver packets?

Will devote multiple lectures (and one project) to this question!

Routing (Conceptually)

- Distributed **routing algorithm** run between switches/routers
- Gather information about the network topology
- Compute paths through that topology
- Store forwarding information in each router:
 - If packet is destined for X, send it on this link
 - If packet is destined for Y, send it on that link
 - ...
- This is the **forwarding table**

Control Plane *vs* Data Plane

- **Control plane:** mechanisms used to compute forwarding tables
 - Inherently **global**: must know topology to compute
 - *Routing algorithm is part of the control plane*
 - Time scale: per network event
- **Data plane:** using those tables to actually forward packets
 - Inherently **local**: depends only on arriving packet and local forwarding table
 - *Forwarding mechanism ("lookup" algorithm) is part of data plane*
 - Time scale: per packet arrival

Control Plane: Challenge

- Computing good routes/paths at scale in the face of network failures and topology changes
(Will study routing algorithms starting lecture#5)
- While respecting ISPs' need for autonomy
(Will study inter-domain routing in depth later in the semester)

Data Plane: Challenge

- Consider a 1 Tbps link (10^{12}) receiving 10,000 bit packets
 - New packet arrives every 10 nanoseconds (10^{-8})
- The following operations must be done after packet arrives (in ~10 nanoseconds or less)
 - Parse packet (extract address, etc.)
 - Look up address in forwarding table
 - Update other fields in packet header (if needed)
 - Update relevant internal counters, etc.
 - Send packet to appropriate output link

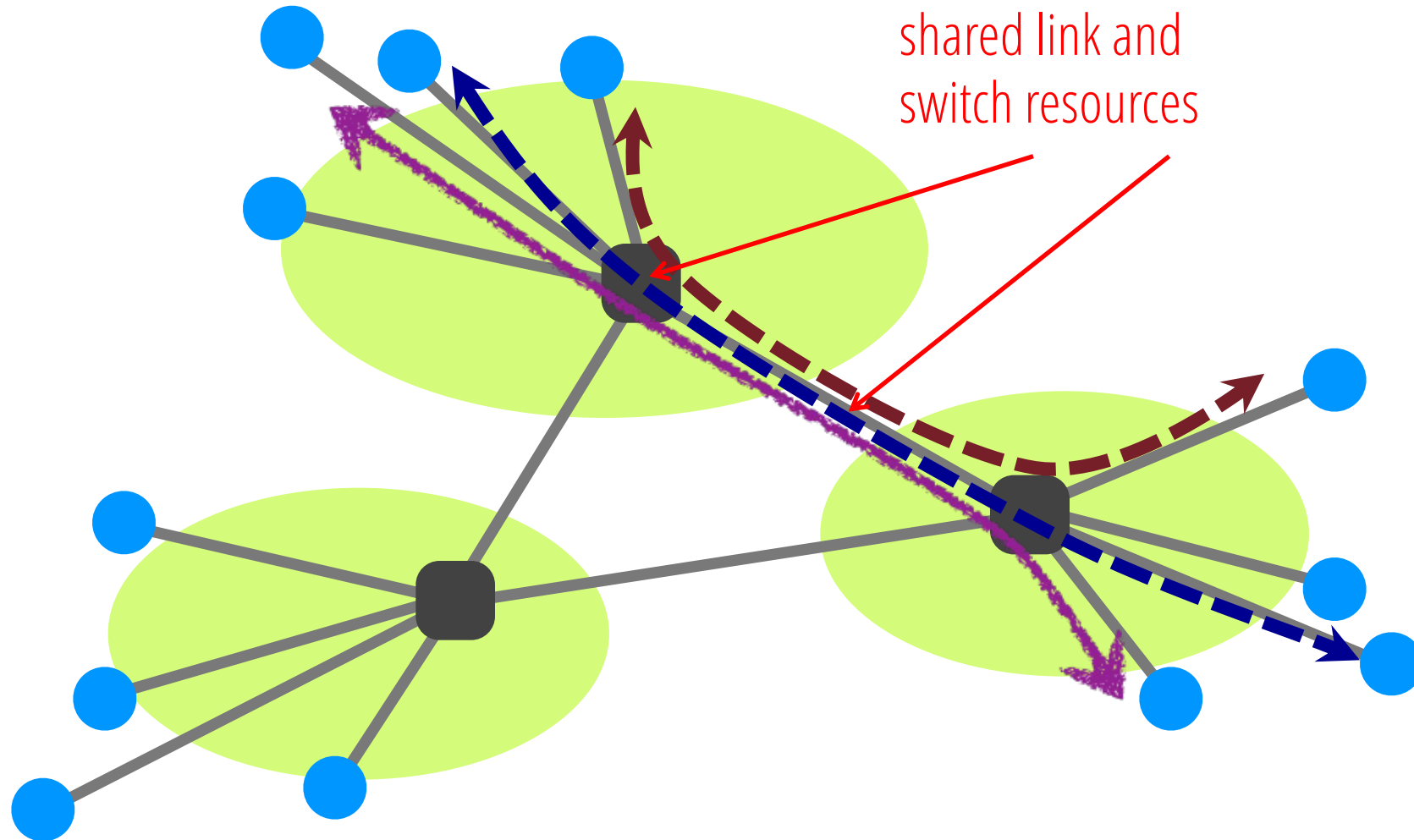
(Will study router designs and IP forwarding lookup algorithms.)

Hence, our important topics (so far)

- How do we name endhosts on the Internet? (naming)
- How do we address endhosts? (addressing)
- How do we map names to addresses? (mapping names to addresses)
- How do we compute forwarding tables? (routing control plane)
- How do we forward packets? (routing data plane)

Questions??

Let's back up a level...



Fundamental Fact About Networks

- Network must support many simultaneous flows at the same time
 - Recall, flow = stream of packets sent between two end hosts
- Which means network resources (links and switches) are **shared** between end hosts

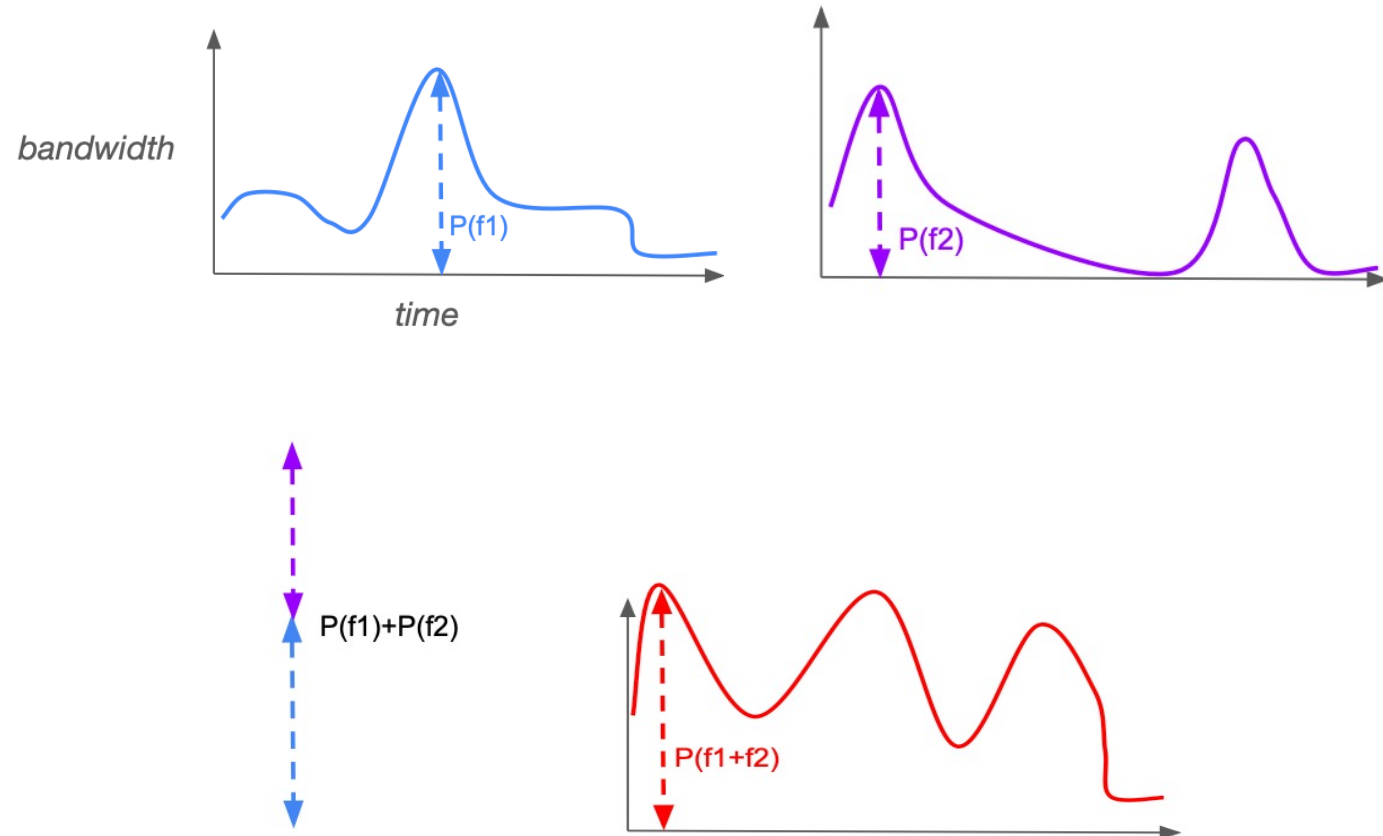
Network resources (i.e., bandwidth) are statistically multiplexed

Statistical Multiplexing

- Combining demands to **share** resources efficiently
 - *vs.* dedicated resources
- Long history in computer science
 - Processes on an OS (*vs.* every process has a dedicated core)
 - Cloud computing (*vs.* every user has a dedicated datacenter)
- Based on premise: peak of aggregate demand is $\ll$ aggregate of peak demands

Aggregates, Peaks, etc....

- Peak rate of flow f : $P(f)$
- Aggregate of peaks: $\sum_{\{f\}} [P(f)]$
- Peak of aggregate: $P(\sum_{\{f\}} f)$
- Typically: $\sum_{\{f\}} [P(f)] \gg P(\sum_{\{f\}} f)$
- Typically: $P(\sum_{\{f\}} f) \sim \sum_{\{f\}} A(f)$
 - Where $A(f)$ is the average rate of flow f



Statistical Multiplexing

- Statistical multiplexing merely means that you don't provision for absolute worst case
 - When everything peaks at the same time
- Instead, you share resources and hope that peak rates don't occur at same time

How would you share network resources?

Two approaches to sharing

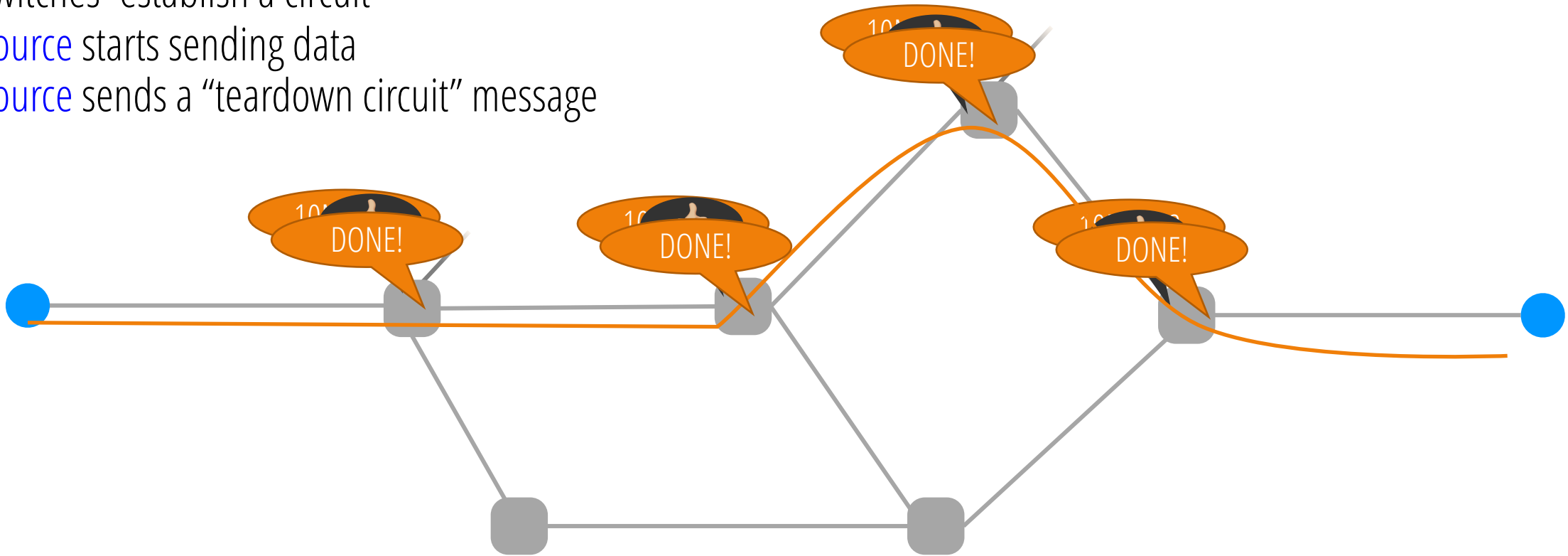
- **Reservations:** end-hosts explicitly reserve BW when needed (e.g., at the start of a flow)
 - Request/reserve resources
 - Send data
 - Release resources
- **Best-effort:** just send data packets when you have them and hope for the best ...

Implementing reservations / best-effort sharing

- Many possible approaches!
- Two canonical designs explored in research and industry
 - Reservations via **circuit switching**
 - Best-effort via **packet switching**

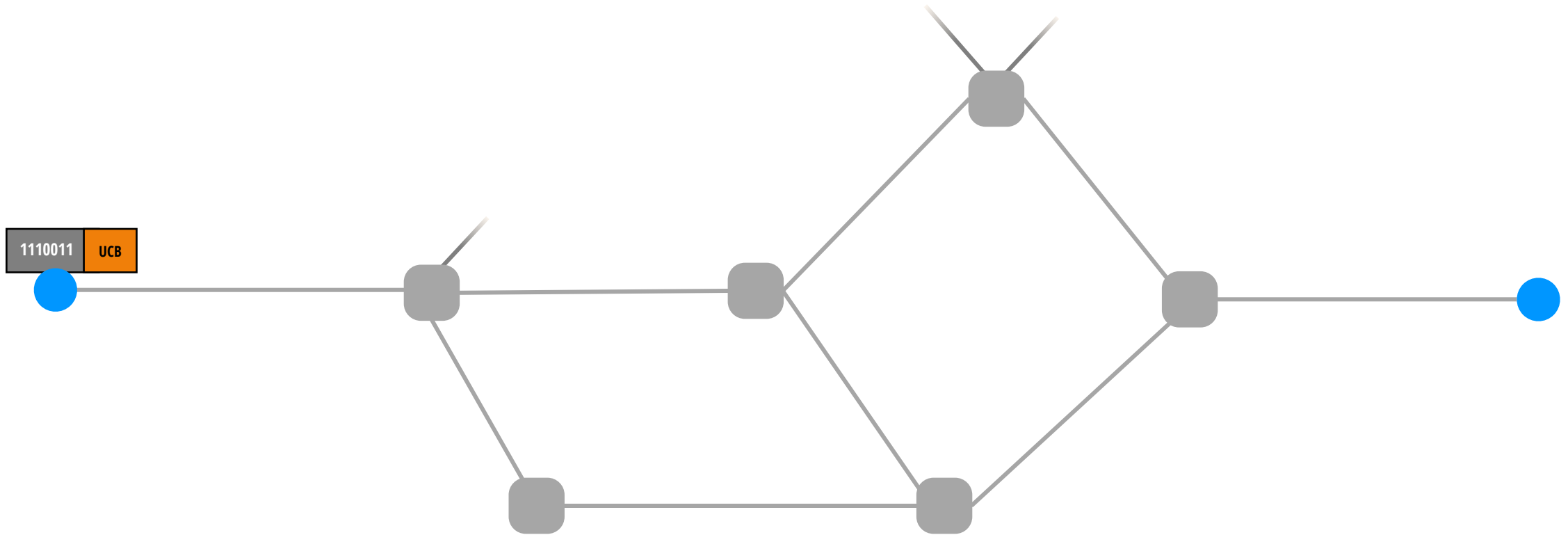
Reservations: e.g., circuit switching

- (1) **source** sends a reservation request to the **destination**
- (2) switches “establish a circuit”
- (3) **source** starts sending data
- (4) **source** sends a “teardown circuit” message



Idea: **Reserve** network capacity for all packets in a flow

Best-effort: e.g., packet switching



Allocate resources to each packet independently
(independent across switches and across packets)

Both approaches embody statistical multiplexing!

- Circuit switching: resources shared between flows currently in system
 - Reserve the peak demand for a flow
 - But don't reserve for all flows that might ever exist
- Packet switching: resources shared between packets currently in system
 - Resources given out on packet-by-packet basis
 - Never reserve resources

Circuit *vs.* Packet switching: which is better?

- What are the dimensions along which we should compare?
 - As an abstraction to applications
 - Efficiency (at scale)
 - Handling failures (at scale)
 - Complexity of implementation (at scale)

From an application viewpoint

- Circuits offer better application performance (reserved bandwidth)
- More predictable and understandable behavior (w/o failures)
- Also a very intuitive abstraction in support of business models!

Which makes more efficient use of network capacity?

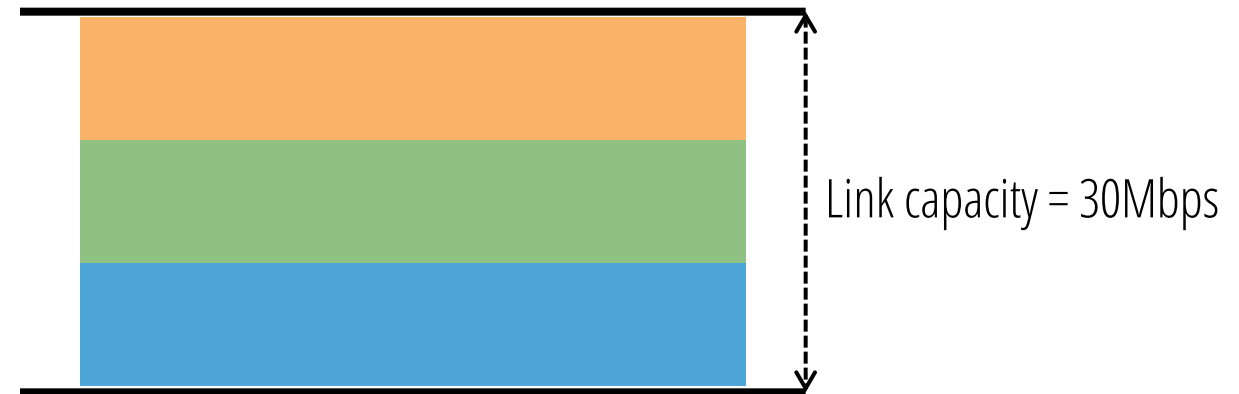
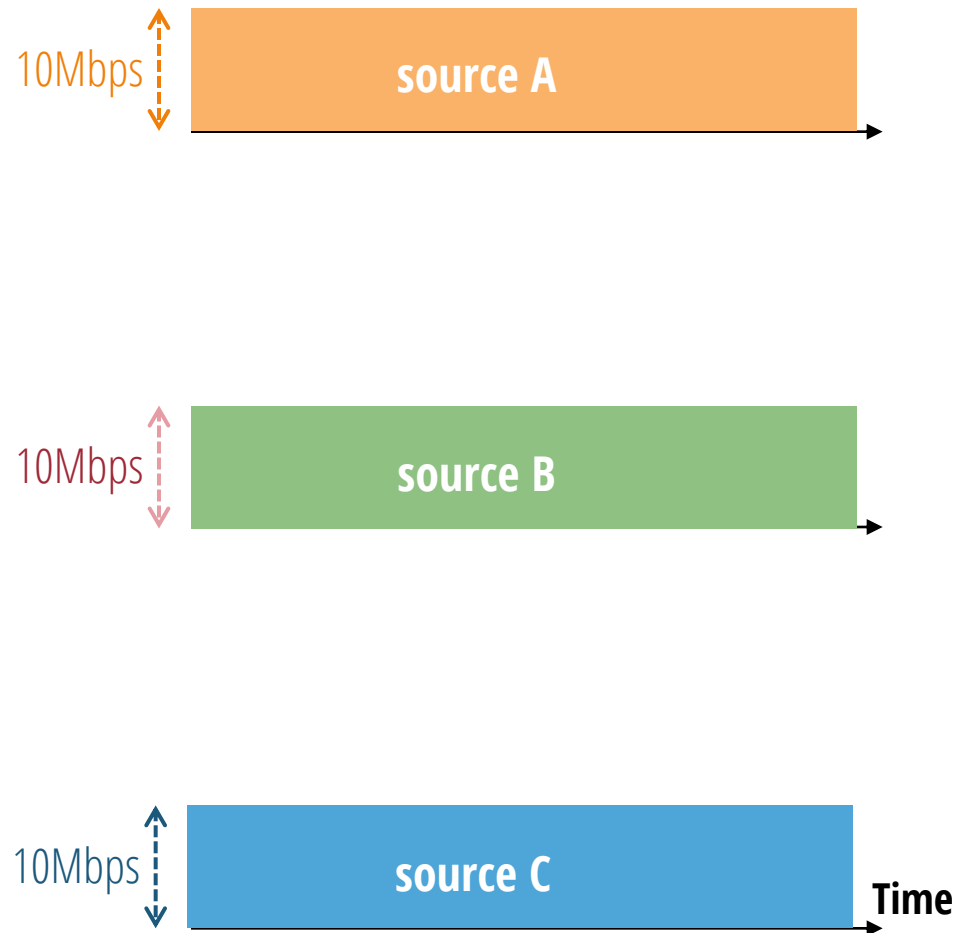
Answer: Packet switching is typically more efficient

- But how much better depends on the “burstiness” of the traffic sources

Example#1: Three constant rate sources sharing a link

- Total link bandwidth is 30Mbps
- Demands: Each source needs a constant rate of 10Mbps
- Circuit and packet switching give approximately the same result
 - Every source gets what they need
 - No wasted bandwidth
 -

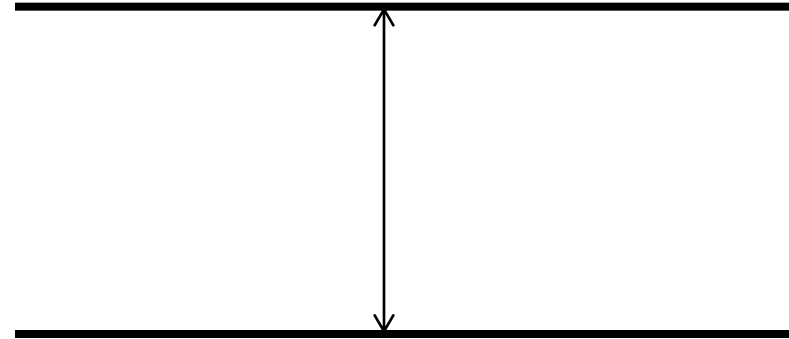
Example#1: Three constant sources



Example#2: Three “bursty” sources



Link capacity = 30Mbps

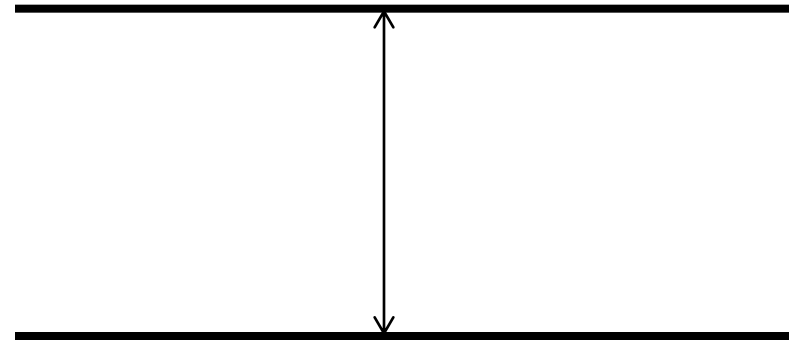


What happens with reservations?

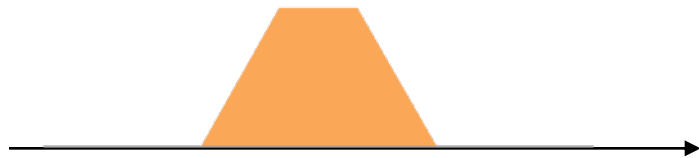
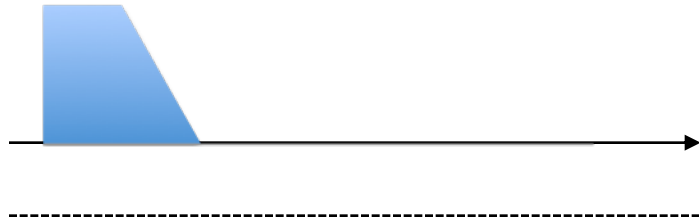


Allow two flows to reserve peak rate
Must turn away third flow!

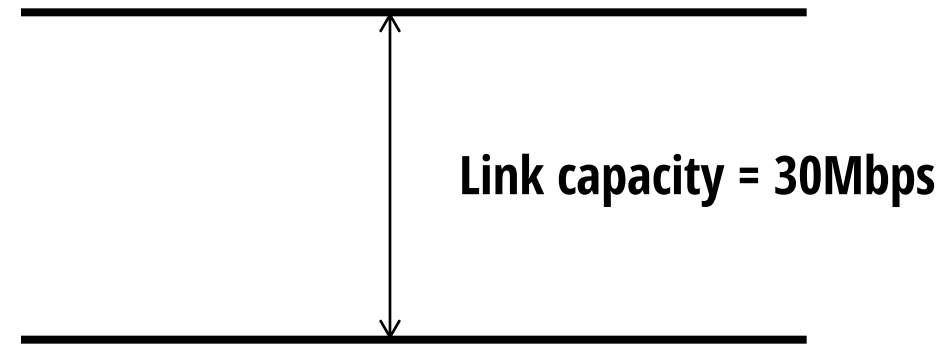
Link capacity = 30Mbps



What happens with best-effort?



All good! No overloading



Smooth vs. Bursty Applications

- Characterized by the ratio between an app's peak to average transmission rate
- Some apps have relatively small peak-to-average ratios
 - Voice might have a ratio of 3:1 or so
- Data applications tend to be rather bursty
 - E.g., ratios of 100 or greater are common when web browsing
- That's why the phone network used reservations and the Internet does not!

Which makes more efficient use of network capacity?

Answer: Packet switching is typically more efficient

- But how much better depends on the “burstiness” of the traffic sources
- This is because packet switching implements statistical multiplexing at a finer granularity than circuit switching (packets vs. flows)

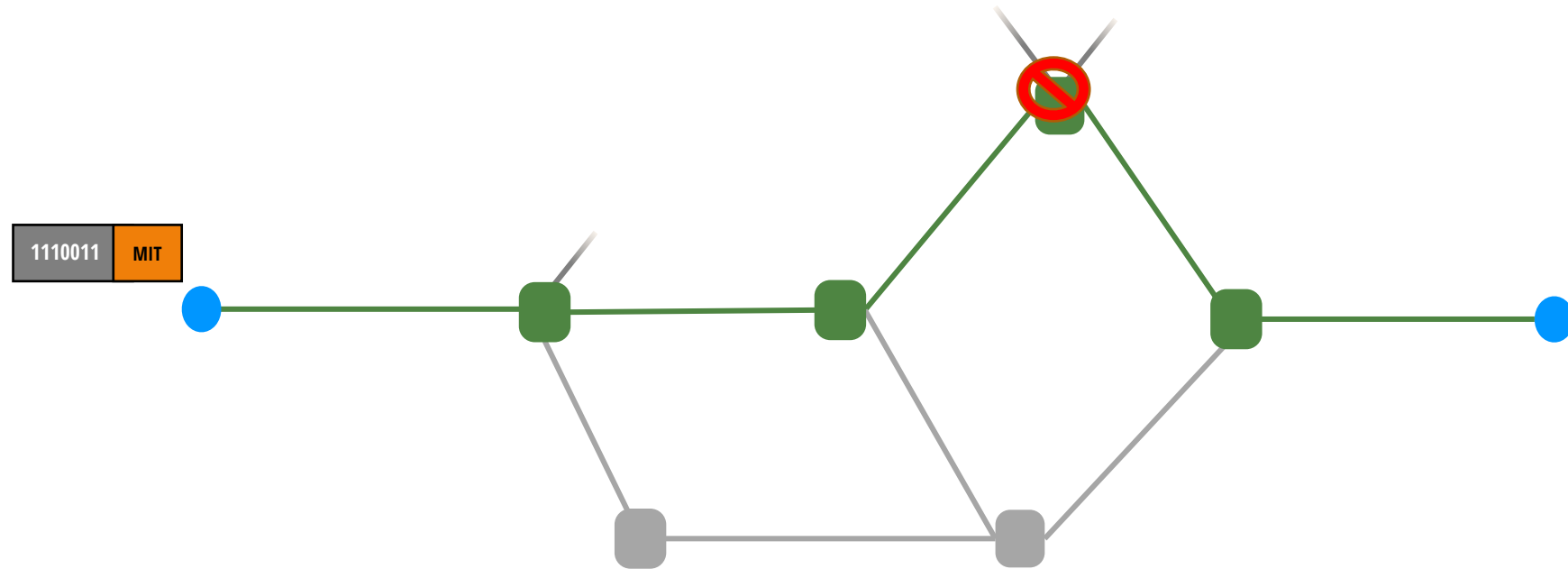
Other differences in efficiency?

- Circuit switching spends some time to setup / teardown circuits
 - Very inefficient when you don't have much data to send! (short flows)
- Packet switching typically requires larger headers

Circuit *vs.* Packet switching: which is better?

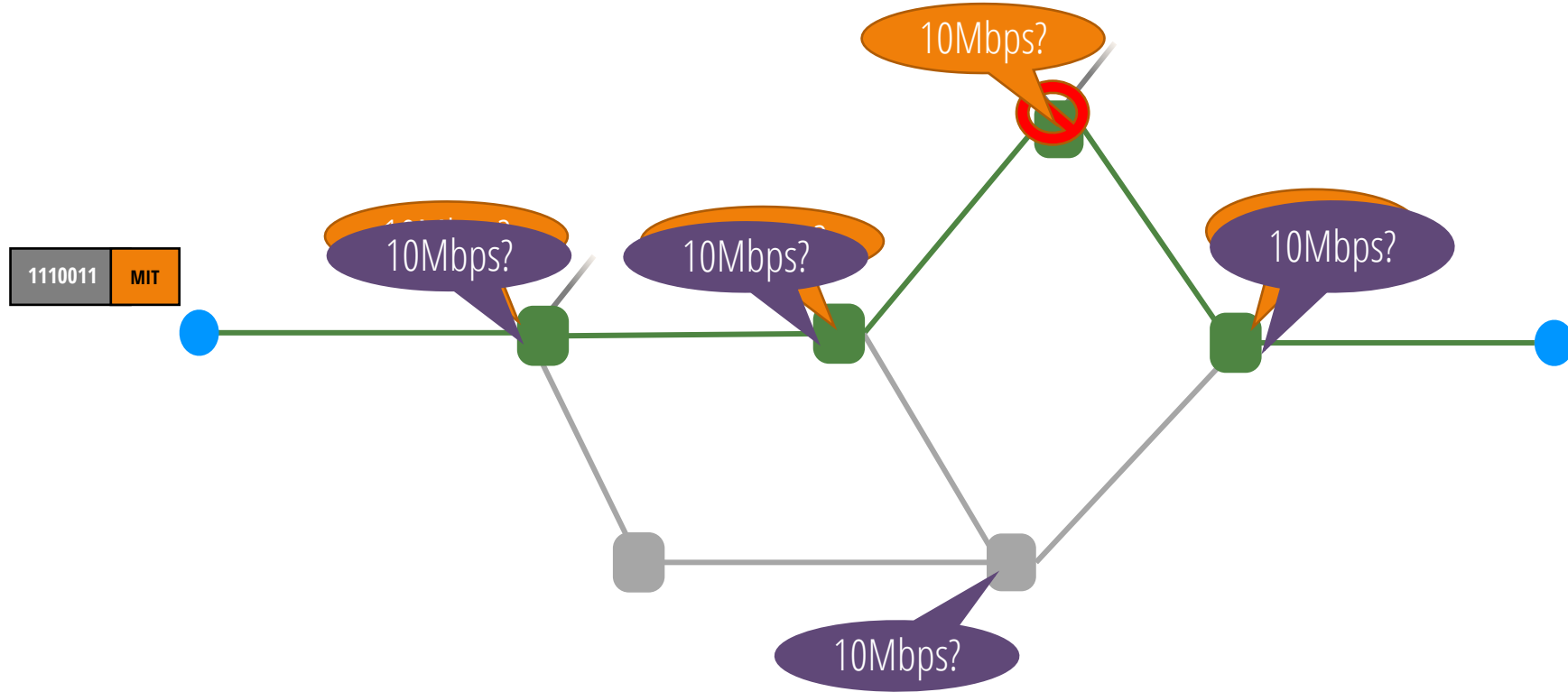
- What are the dimensions along which we should compare?
 - As an abstraction to applications (endhosts)
 - Efficiency
 - Handling failures (at scale)
 - Complexity of implementation (at scale)

What happens in the event of a failure?



With packet switching?

What happens in the event of a failure?



With circuit switching?

Recap: Failure Recovery in Packet Switching

- Link goes down, then what?
- Network must detect failure
- Network recalculates routes
 - (Job of the routing control plane)
- Endhosts and individual flows do nothing special
 - Except cope with the temporary loss of service

Recap: Failure Recovery in Circuit Switching

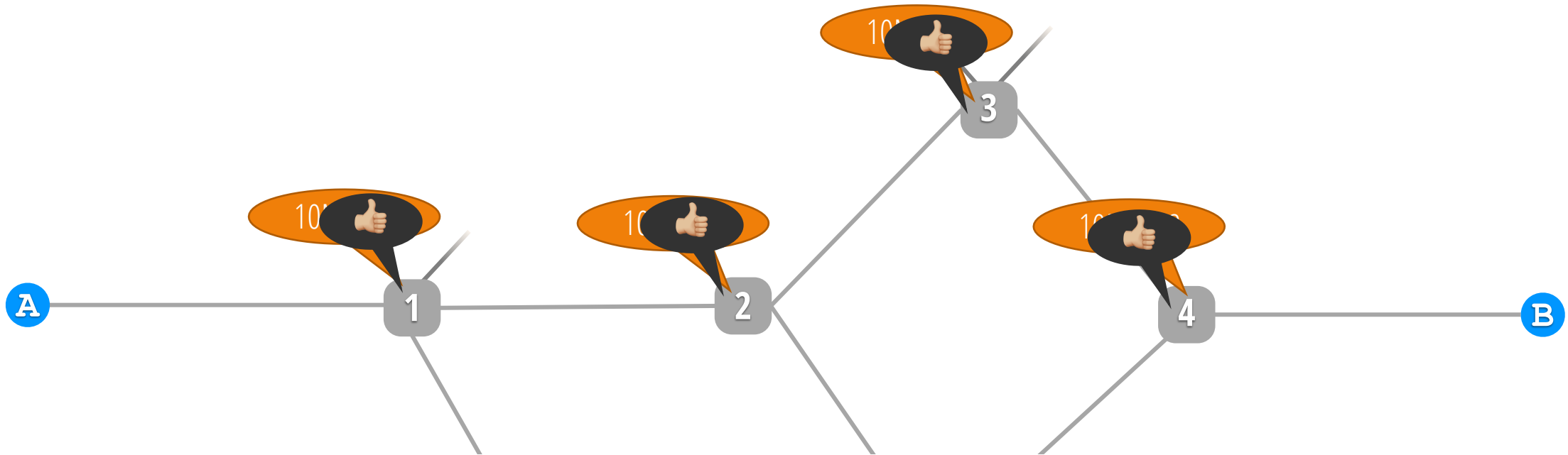
- Network must do all the things needed for packet switching
- And in addition, endhosts must
 - detect failure
 - teardown old reservations
 - send a new reservation request
- All impacted endhosts must do this, for each impacted flow!!
- If millions of flows were going through a switch, then millions of reservation requests are being simultaneously re-established!

Circuit *vs.* Packet switching: which is better?

- What are the dimensions along which we should compare?
 - As an abstraction to applications (endhosts)
 - Efficiency
 - Handling failures (at scale)
 - Complexity of implementation (at scale)

Recall...

(1) **source** sends a reservation request to the **destination**



How do switches know that the reservation went through?
What happens if the reservation request is lost mid way?
What happens if the confirmation that the reservation made it is lost?
What should the endpoint do if the reservation is declined?
What happens if the underlying route changes?
And on and on....

Recap: Circuit *vs.* Packet Switching

- Pros for circuit switching:
 - Better application performance (reserved bandwidth)
 - More predictable and understandable (w/o failures)
- Pros for packet switching:
 - Better efficiency
 - Faster startup to first packet delivered
 - Easier recovery from failure
 - Simpler implementation (avoids dynamic per-flow state management in switches)

What does the Internet use today?

- Packet switching is the default
- Limited use of RSVP (“Resource Reservation Protocol”)
 - Reservation is typically limited to a single domain
 - Not exposed to users; only invoked under operator control
- But you *can* also buy a dedicated circuit (e.g., MPLS circuits, leased lines, etc.)
 - Often used by enterprises from one branch location to another (or to/from cloud)
 - Very expensive (e.g., 10-20x higher than a normal connection)
 - Often statically set up (manually), long-lived (e.g., years), and per user (vs. per flow)
 - So, a far cry from the vision of dynamic reservations that we just discussed

Circuit *vs.* Packet Switching: A bit of history

- The early Internet (70-80s): packet switched
 - Well suited to (bursty) file transfer applications
- The next iteration (~90s): research & industry believed we'd need circuit switching
 - Envisioned that voice/live TV/ would be the Internet's true killer app
 - Spent 10+ years trying to realize this vision
- Ultimately, a failed vision. Why?
 - All the reasons we discussed...
 - ...and people rewrote apps to be adaptive (turns out we didn't really need guaranteed BW!)
 - ...and Email and the web emerged as the killer apps of that time

A lesson in how technology can transform user behavior!

Questions??