CS4410

Operating Systems

Lecture 17
End-to-end view of networking
(First step to understanding network stacks)

Rachit Agarwal

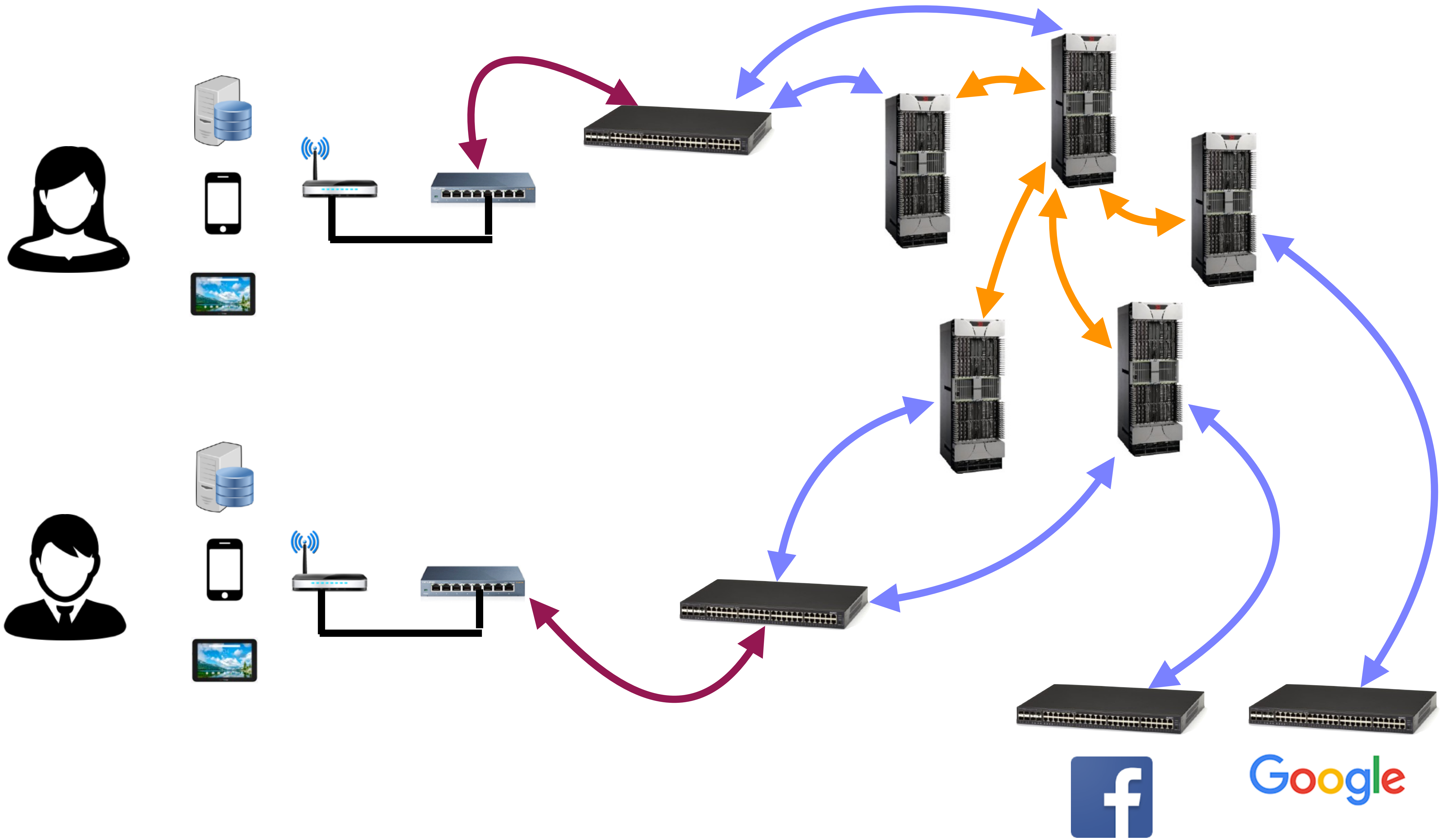
What is a computer network?

A set of network elements connected together, that implement a set of protocols for the purpose of sharing resources at the end hosts

- **Three important components**:
    - **Core infrastructure:**
        - A set of network elements connected together
    - **Protocols:**
        - Needed to use the network
    - **Purpose:**
        - Sharing resources at the end hosts (computing devices)

What is a computer network?

A set of network elements connected together, that implement a set of protocols for the purpose of sharing resources at the end hosts



What is a computer network?

A set of network elements connected together, that implement a set of protocols for the purpose of sharing resources at the end hosts

- **Three important components**:
  - **Core infrastructure:**
    - A set of network elements connected together
  - **Protocols:**
    - Needed to use the network
  - **Purpose:**
    - Sharing resources at the end hosts (computing devices)

What do computer networks do?

A computer network delivers data between the end points

- **One and only one task**: Delivering the data
- **Read that sentence again. Remember it forever.**

- This delivery is done by:
    - Chopping the data into **packets**
    - Sending individual packets across the network
    - Reconstructing the data at the end points

- **That is all!**

Data delivery as a fundamental goal

- **Support the logical equivalence of Interprocess Communication (IPC)**
  - Mechanism for "processes on the same host" to exchange messages

- **Computer networks allow "processes on two different hosts" to exchange messages**

- **Clean separation of concerns**
  - Computer networks deliver data
  - Applications running on end hosts decide what to do with the data

- **Keeps networks simple, general and application-agnostic**

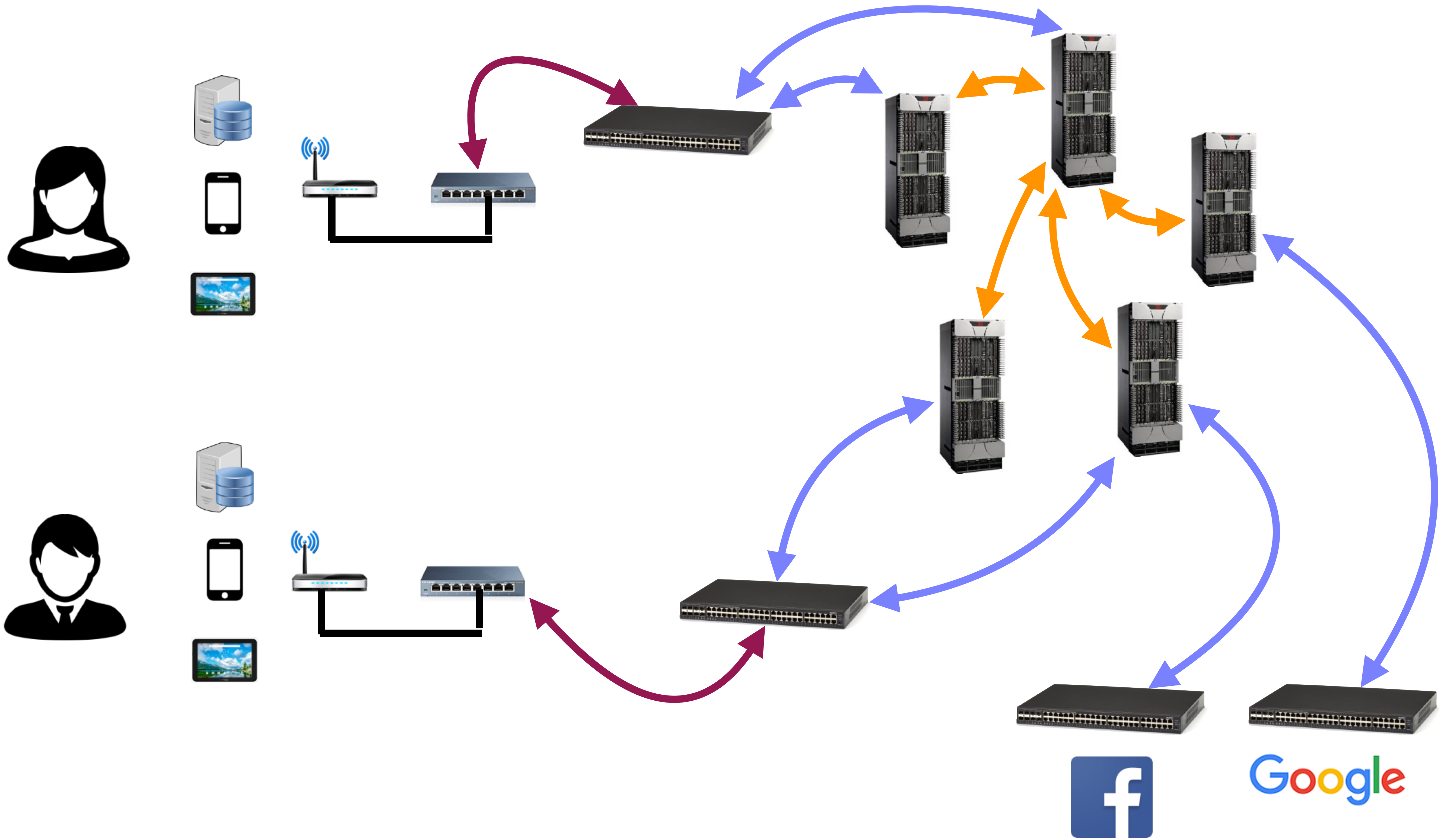
What do computer networks look like?

Three Basic pieces in the core infrastructure

- **End hosts**: they send/receive packets
- **Switches/Routers:** they forward packets
- **Links:** connect end hosts to switches, and switches to each other

What do computer networks look like?

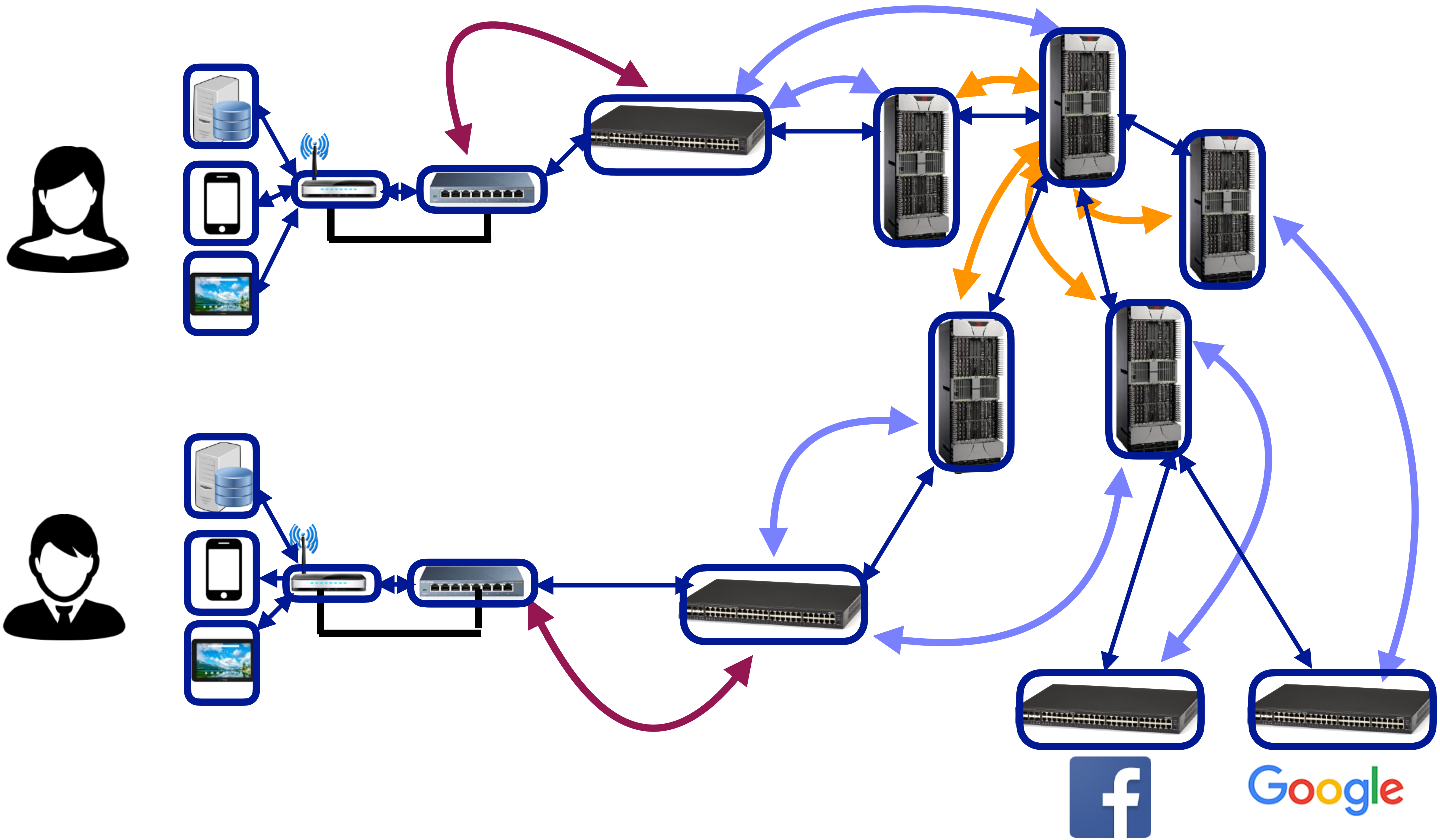
End hosts, switches/routers, links



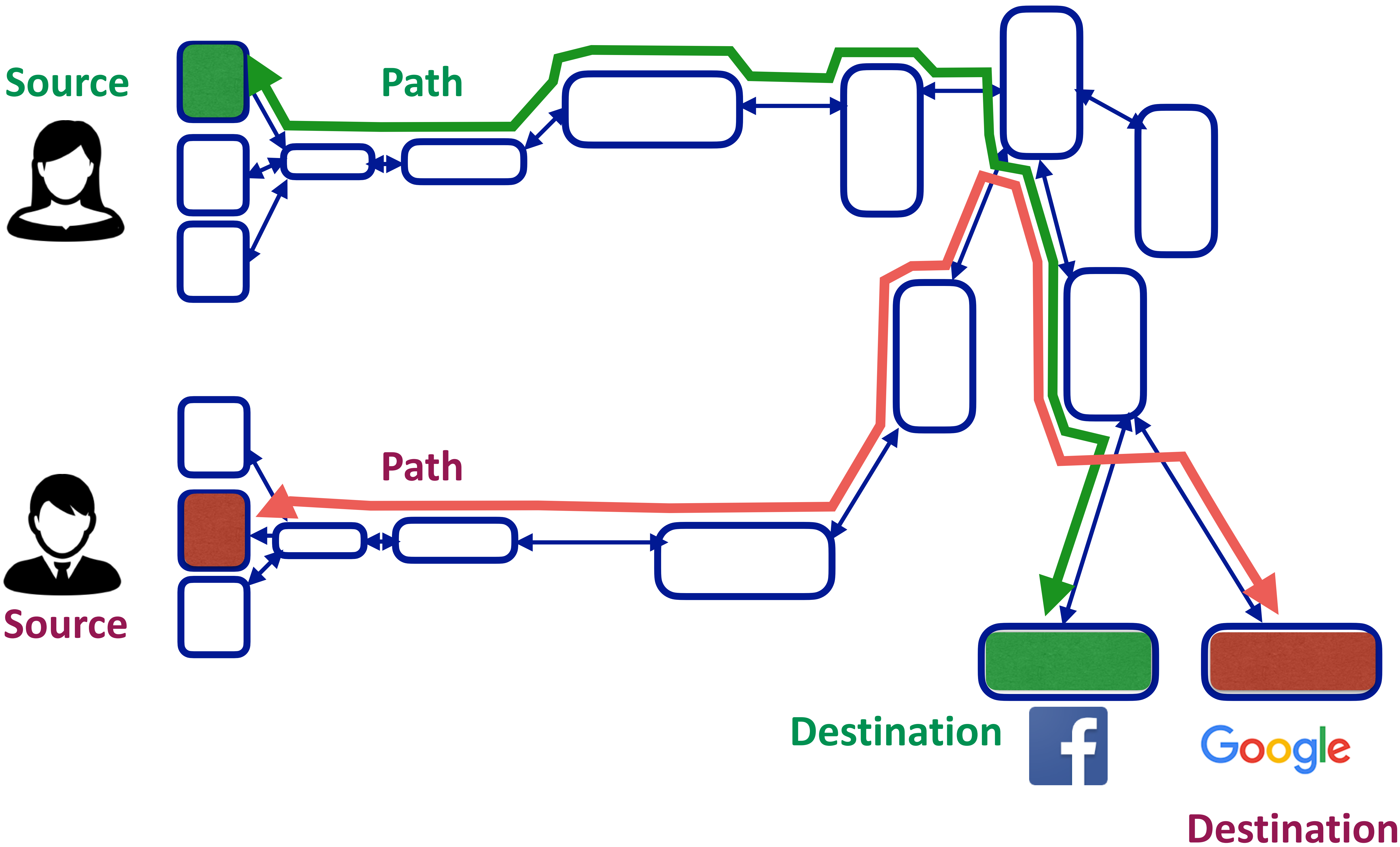
Lets make the picture simpler

What is a computer network?

A set of network elements connected together, that implement a set of protocols for the purpose of sharing resources at the end hosts



A computer network can be abstractly represented as a graph



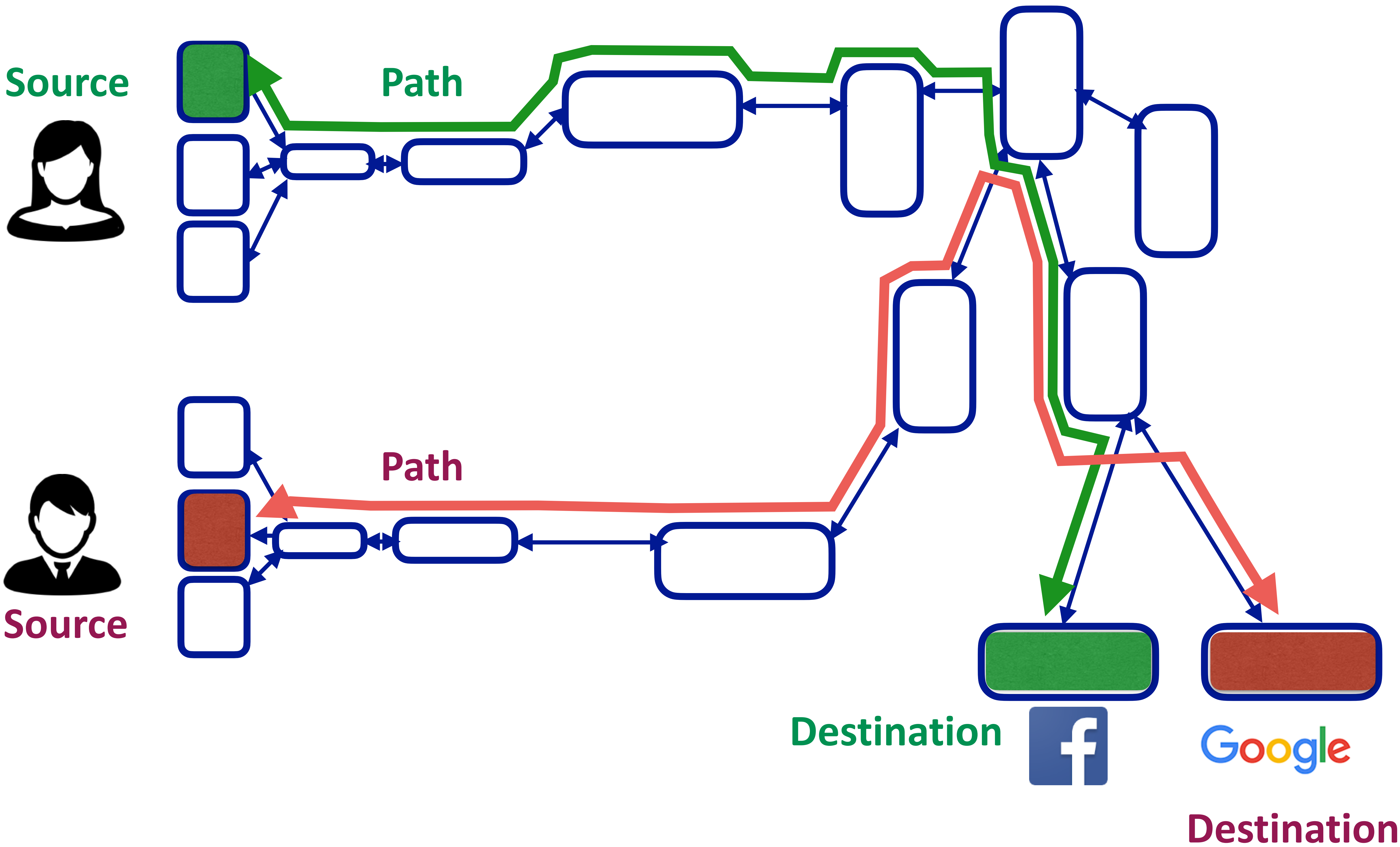
Many mechanisms underneath!

- **Locating the destination**: Naming, addressing
- **Finding a path to the destination**: Routing
- **Sending data to the destination:** Forwarding
- **Failures, reliability, etc.:** Distributed routing and congestion control

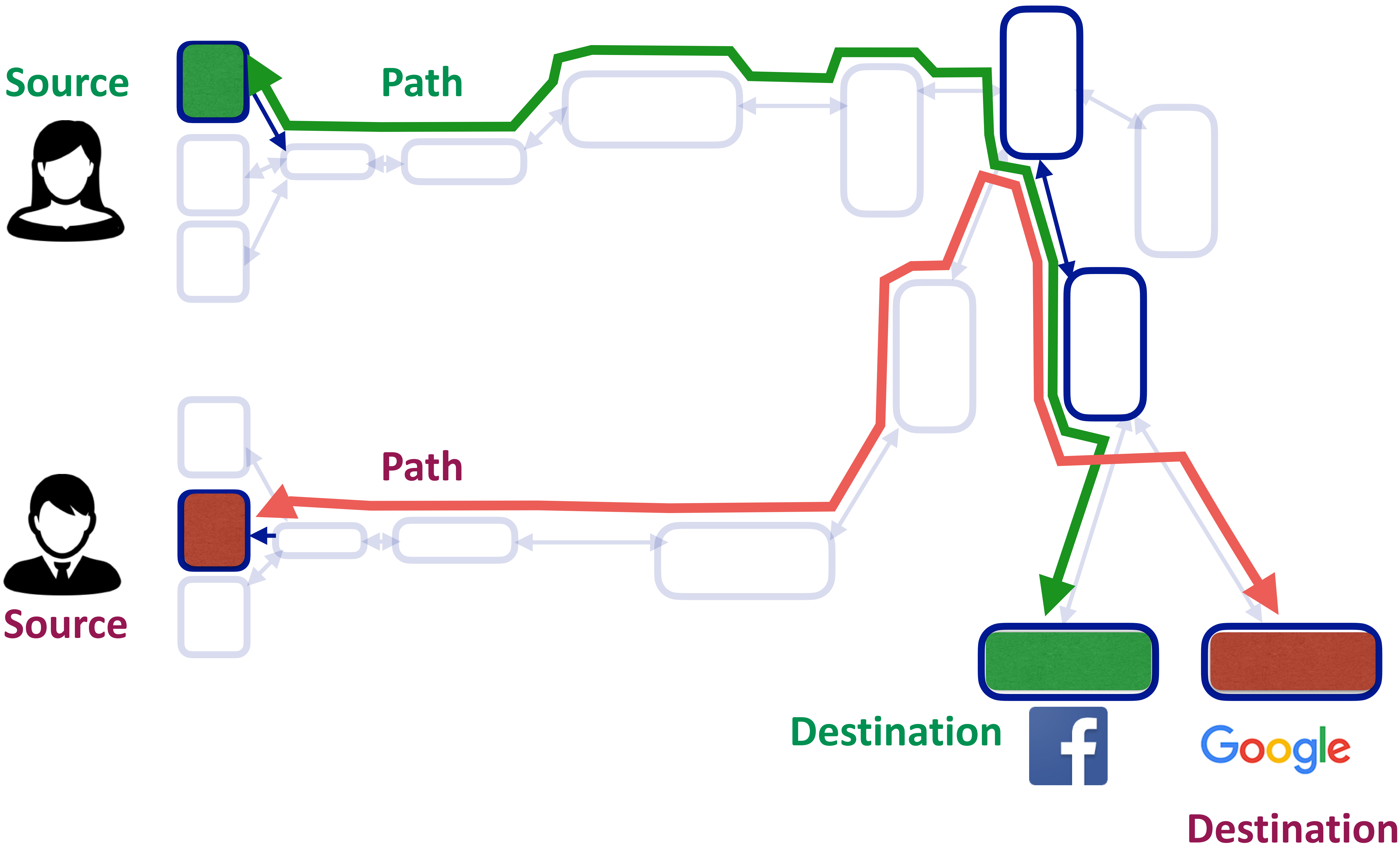
Will take the entire course to learn these!

CS4450 :-)

A computer network can be abstractly represented as a graph



Sharing the network (graph)



Today’s lecture: sharing computer networks

1. What does network sharing mean?

2. What are the performance metrics?

3. What are the various mechanisms for sharing networks?

4. **Why “packets” and “flows”?**

What does network sharing mean?

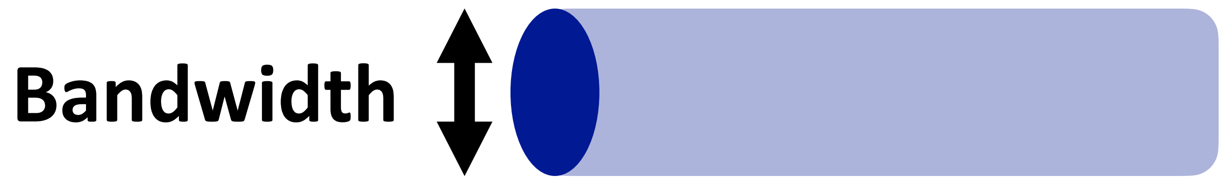
The problem of sharing networks

- Must support many “users” at the same time
- Each user wants to use the network (send and receive data)
- Limited resources
- **Fundamental question:**
  - **How does network decide which resource to allocate to which user at any given point of time?**

What are the performance metrics?

Performance metrics in computer networks!

- **Bandwidth: <u>Number of bits</u> sent per second (bits per second, or bps)**
  - Depends on
    - Hardware
    - **Throughput:** Network traffic conditions
    - ….



- **Delay: Time for <u>all bits</u> to go from source to destination (seconds)**
  - Depends on
    - Hardware
    - Distance
    - **Latency:** Traffic from other sources
    - ….
- **Many other performance metrics (reliability, etc.)**
  - We will come back to other metrics later …

What are the various mechanisms for sharing networks?

Two approaches to sharing networks

- Reservations
- On demand

Two approaches to sharing networks

- **First: Reservations**
  - Reserve bandwidth needed in advance
  - Set up circuits and send data over that circuit

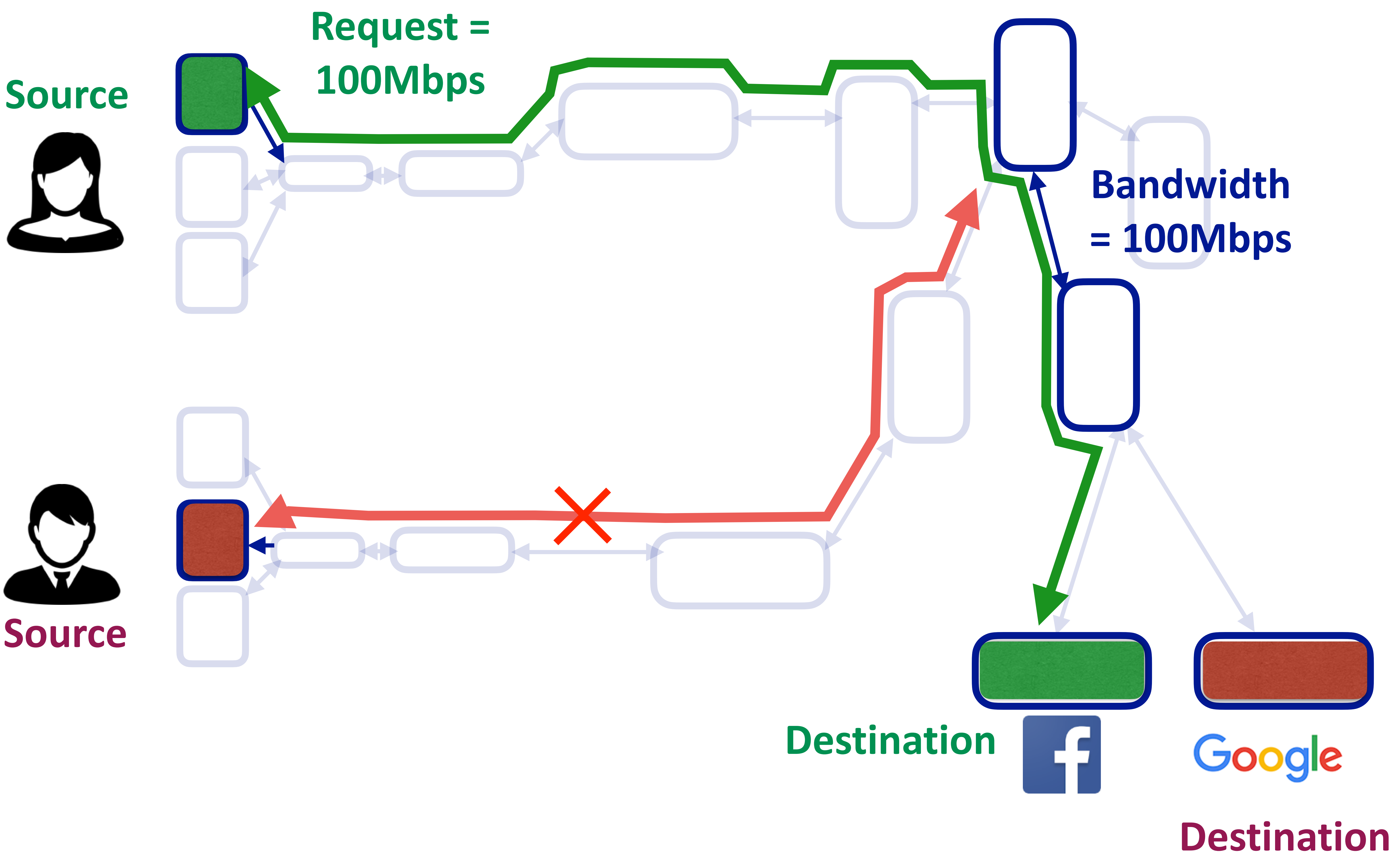
- How much bandwidth to reserve?
  - Applications may generate data at rate varying over time
  - 100MB in first second
  - 10MB in second second …
  - Must reserve for peak bandwidth (100MB)

Circuit switching: Implementing reservations since …

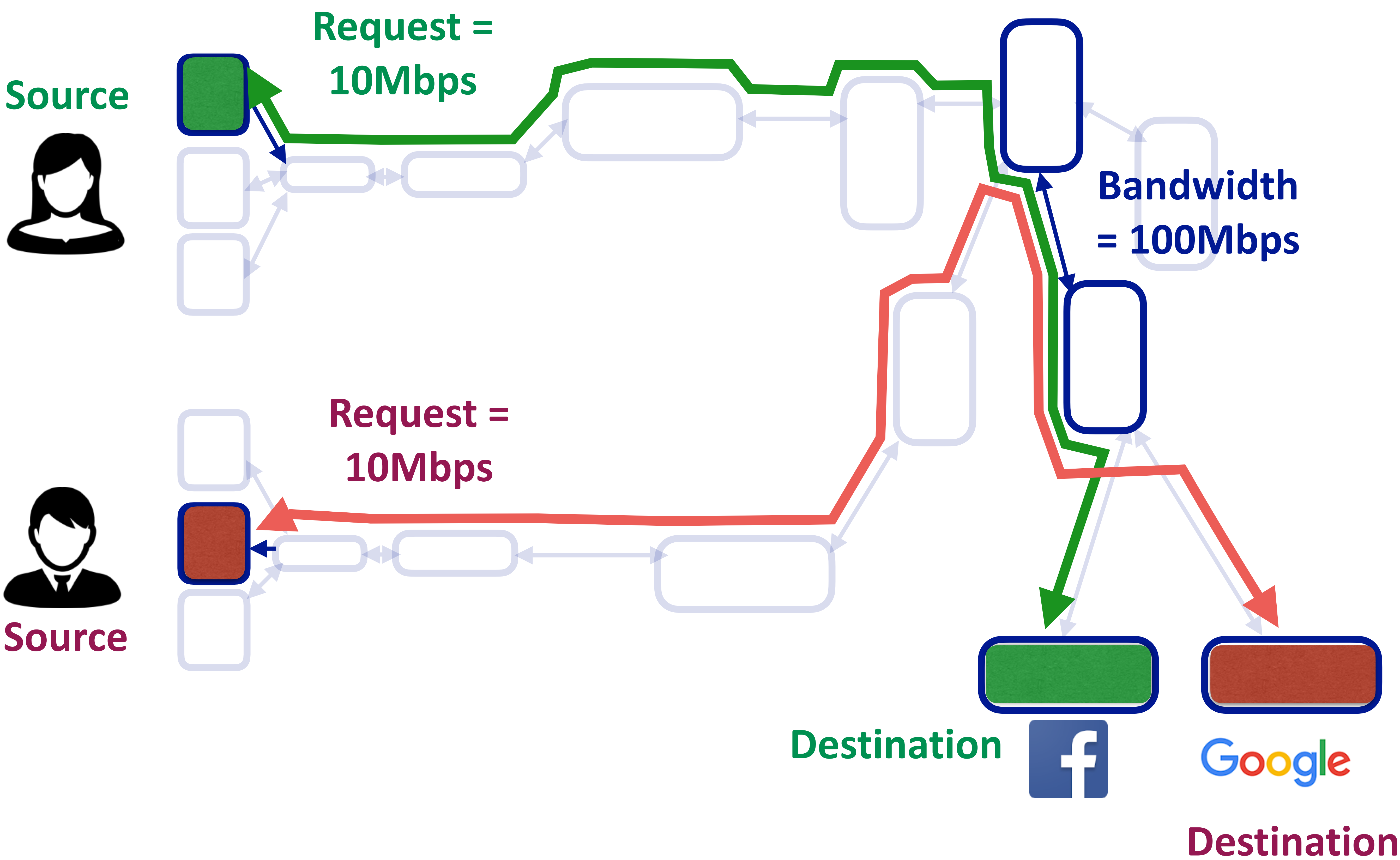
Telephone networks

- One of the many approaches to implementing reservations
- **Mechanism:**
  - Source sends a reservation request for peak demand to destination
  - Switches/routers establish a “circuit”
  - Source sends data
  - Source sends a “teardown circuit” message

Circuit switching: an example (red request fails)



Circuit switching: another example (red request succeeds)



Circuit switching and failures

- Circuit is established
- **Link fails along path (!!!!!!!)**
    - First time we have seen failures making our life complicated.
    - Remember this moment.
    - Its gonna happen, over and over again.
- Must establish new circuit

Circuit switching doesn't route around failures!!

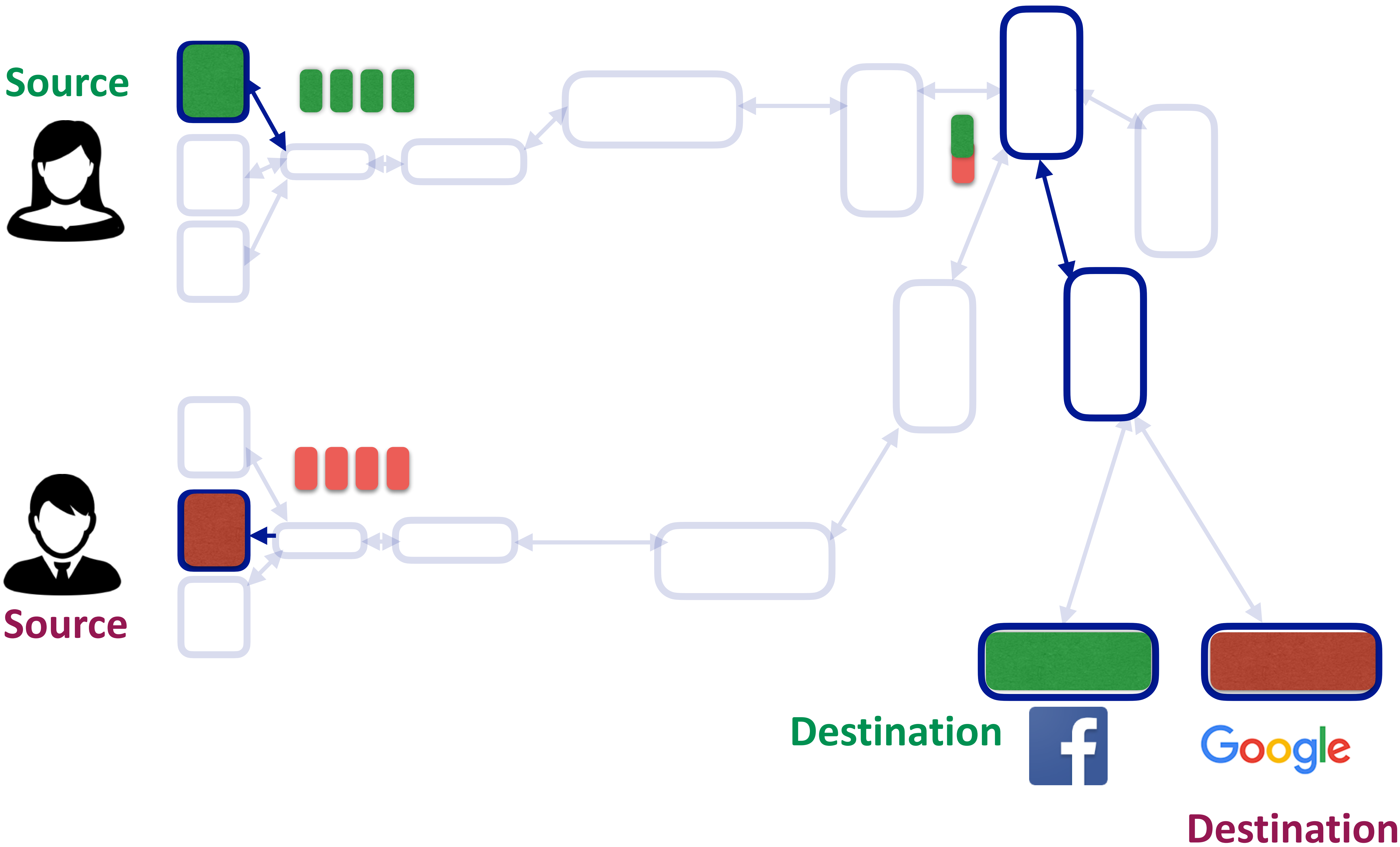
Circuit switching summary

- **Goods:**
  - Predictable performance
  - Reliable delivery (assuming no hardware failures)
  - Simple forwarding mechanism
- **Not-so-goods:**
  - **Handling hardware failures**
  - **Resource underutilization**
  - **Blocked connections**
  - **Connection set up overheads**
  - **Per-connection state in switches (scalability problem)**

Two approaches to sharing networks

- **Second: On demand (also known as “best effort”)**
  - **Designed specifically for the Internet**
  - Break data into packets
  - Send packets when you have them
  - Hope for the best …

Packet switching: an example



Packets

- **Packets carry data (are bag of bits)**:
  - Header: meaningful to network (and network stack)
    - can be multiple headers
  - Body: meaningful only to application
  - More discussion in next lecture

- **Body can be bits in a file, image, whatever**
  - can have its own application “header”

- **What information goes in the header?**

What must headers contain to enable network functionality?

- **Packets must describe where it should be sent**
  - Requires an address for the destination host
    - can be multiple headers
- **Packets must describe where its coming from**
  - why?
  - Acknowledgments, etc.
- **Thats the only way a router/switch can know what to do with the packet**

Recap: Packet switching summary

- **Goods:**
  - Easier to handle failures
  - No resource underutilization
  - No blocked connection problem
  - No per-connection state
  - No set-up cost
- **Not-so-goods:**
  - Unpredictable performance
  - High latency
  - Packet header overhead

Circuits vs packets

- Pros for circuits:
  - Better application performance (reserved bandwidth)
  - More predictable and understandable (w/o failures)

- Pros for packets:
  - Better resource utilization
  - Easier recovery from failures
  - Faster startup to first packet delivered

Statistical multiplexing

- **Statistical multiplexing:** combining demands to share resources efficiently

- Long history in computer science
  - Processes on an OS (vs every process has own core)
  - Cloud computing (vs every one has own datacenter)

- Based on the premise that:
  - **Sum of instantaneous demands << sum of peak demands**

- Therefore, it is better to share resources than to strictly partition them …

Two approaches to sharing networks

Both embody statistical multiplexing

- Reservation: sharing at <u>connection</u> level
  - Resources shared between connections currently in system
  - Reserve the peak demand for a flow

- On-demand: sharing at <u>packet</u> level
  - Resources shared between packets currently in system
  - Resources given out on packet-by-packet basis
  - No reservation of resources

End-to-end story

Four fundamental problems!

- **Naming, addressing:** Locating the destination
- **Routing:** Finding a path to the destination
- **Forwarding:** Sending data to the destination
- **Reliability:** Handling failures, packet drops, etc.

Fundamental problem #1: Naming and Addressing

- **Network Address: where host is located**
    - Requires an address for the destination host
- **Host Name: which host it is**
    - why do we need a name?
- **When you move a host to new building**
    - Address changes
    - Name ***does not*** change
- **Same thing with your own name and address!**
- **Remember the analogy: human names, addresses, post office, letters**

Names versus addresses

- **Consider when you access a web page**
  - Insert URL into browser (eg, www.cornell.edu)
  - Packets sent to web site (reliably)
  - Packet reach application on destination host
- **How do you get to the website?**
  - URL is **user-level name** (eg, www.cornell.edu)
  - Network needs address (eg, where is www.cornell.edu)?
- Must map names to addresses
  - Just like we use an address book to map human names to addresses

Mapping Names to Addresses

- On the Internet, we only name hosts (sort of)
  - URLs are based on the name of the host containing the content (that is, www.cornell.edu names a host)

- Before you can send packets to www.cornell.edu, you must resolve names into the host's address

- Done by the **Domain Name System (DNS)**

The source knows the name;

Maps that name to an address using DNS!

Questions?

Fundamental problem #2

Routing packets through network elements (eg, routers) to destination

- Given destination address (and name), how does each switch/router know where to send the packet so that the packet reaches its destination

- When a packet arrives at a router
  - a **routing table** determines which outgoing link the packet is sent on
  - Computed using **routing protocols**

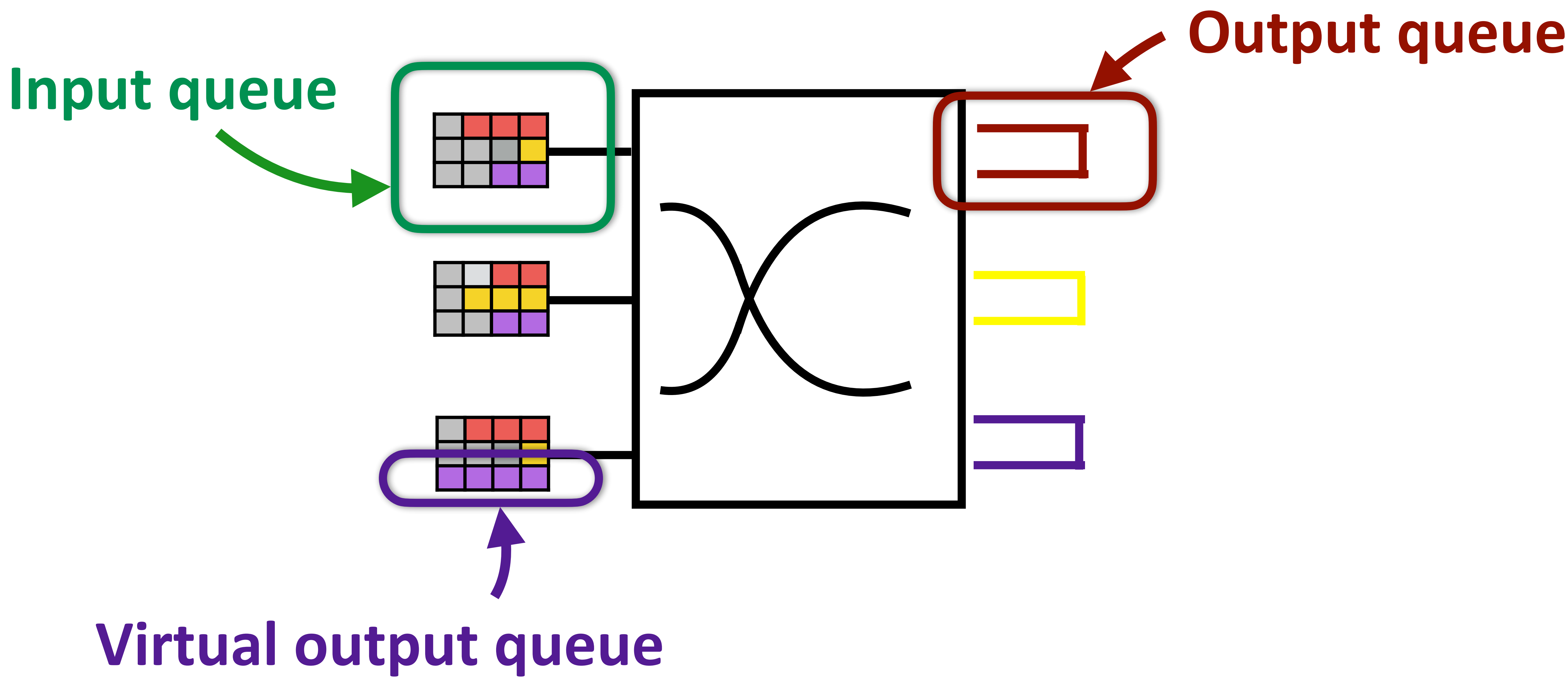
Routing protocols (conceptually)

- Distributed algorithm that runs between routers
  - Distributed means no single router has “full” view of the network
  - Exchange of messages to gather “enough” information …
- … about the network topology
- Compute paths through that topology
- Store forwarding information in each router
  - If packet is destined for X, send out using link l1
  - If packet is destined for Y, send out using link l2
  - Can packets going to different destinations sent out to same link?
- **We call this a routing table**

Questions?

Fundamental problem #3

Queueing and Forwarding of packets at switches/routers



Fundamental problem #3

Queueing and Forwarding of packets at switches/routers

- **Queueing:** When a packet arrives, store it in “input queues”
    - Each incoming queue divided into multiple virtual output queues
    - One virtual output queue per outgoing link
    - When a packet arrives:
        - Look up its destination’s address (how?)
        - Find the link on which the packet will be forwarded (how?)
        - Store the packet in corresponding virtual output queue
- **Forwarding:** When the outgoing link free
    - Pick a packet from the corresponding virtual output queue
    - forward the packet!

What must packets carry to enable forwarding?

- **Packets must describe where it should be sent**
  - Requires an address for the destination
- **Packets must describe where its coming from**
  - For handling failures, etc.
  - Requires an address for the source
- **Packets must carry data**
  - can be bits in a file, image, whatever

Switch Processing and Queueing delay

- **Processing delay**
  - Easy; each switch/router needs to decide where to put packet
  - Requires checking header, etc.

- **Queueing delay**
  - Harder; depends on “how many packets are in front of me”
  - Depends on network load
  - As load increases, queueing delay increases

- **In an extreme case, increase in network load**
  - results in packet drops

- We will return to this in much more depth later …

Questions?

Fundamental problem #4

How do you deliver packets reliable?

- Packets can be dropped along the way
  - Buffers in router can overflow
  - Routers can crash while buffering packets
  - Links can garble packets

- How do you make sure packets arrive safely on an unreliable network?
  - Or, at least, know if they are delivered?
  - Want no false positives, and high change of success

Two questions about reliability

- **Who is responsible for this? (architecture)**
  - Network?
  - Host?

- **How is it implemented? (engineering)**

- We will consider both perspectives

Questions?

Finishing our story

- We now have the address of the web site
- And, a route/path to the destination
- And, mechanisms in place to forward the packets at each switch/router
- In a reliable manner
    - So, we can send packets from source to destination
    - Are we done?

- When a packet arrives at a host, what does the host do with it?
    - To which process (application) should the packet be sent?

- If the packet header only has the destination address, how does the host know where to deliver packet?
    - There may be multiple applications on that destination

And while we are finishing our story ….

- Who puts the source address, source port, destination address, destination port in the packet header?

The final piece in the game: End-host stack

Of Sockets and Ports

- When a process wants access to the network, it opens a socket, which is associated with a port

- **Socket:** an OS mechanism that connects processes to the network stack

- **Port:** number that identifies that particular socket

- The port number is used by the OS to direct incoming packets

Implications for Packet Header

- **Packet Header must include:**
    - Destination address (used by network)
    - Destination port (used by network stack)
    - And?
    - Source address (used by network)
    - Source port (used by network stack)

- When a packet arrives at the destination host, packet is delivered to the socket associated with the destination port

- More details later

Separation of concerns

- **Network:** Deliver packets from host to host (based on address)
- **Network stack (OS):** Deliver packets to appropriate socket (based on port)
- **Applications:**
  - Send and receive packets
  - Understand content of packet bodies

The end-to-end story

- Application opens a **socket** that allows it to connect to the **network stack**
- Maps **name** of the web site to its **address** using **DNS**
- The network stack at the source embeds the address and **port** for both the source and the destination in **packet header**
- Each **router** constructs a **routing table** using a distributed algorithm
- Each router uses destination address in the packet header to look up the **outgoing link** in the routing table
    - And when the link is free, forwards the packet
- When a packet arrives the destination:
    - The network stack at the destination uses the port to forward the packet to the right application