

CSC 3511 Security and Networking

Week 1, Lecture 1: Course Overview

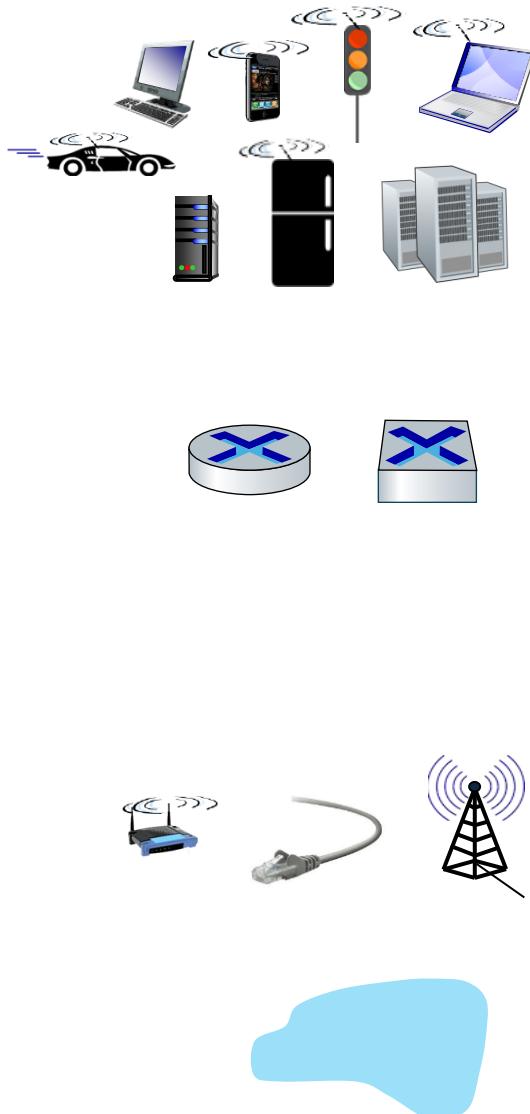
Course Personnel

- Instructor: Dr. Zhonghao Liao
 - Office: DH 436
 - Email: liao@msoe.edu
 - Office hours:
 - Monday, Tuesday: 10:00 am – 11:00 am, 2:00 pm - 3:00 pm
 - Thursday: 10:00 am - 11:00 am
 - Wednesday, Friday: Knock on door or by appointment
 - Please feel free to come over when I'm in the office
- Watch the Canvas
 - Assignments, reading materials, lecture notes

Course Materials

- Textbook:
 - Computer Networking A Top-Down Approach, 9th edition (ISBN: 9780135429334)
 - Internet Security: A Hands-on Approach 3rd Edition (ISBN: 978-17330039-6-4)
 - **Free**: An Introduction to Computer Networks by Peter L. Dordal
Link: <https://intronetworks.cs.luc.edu/current2/html/index.html>
 - **Free**: Computer Networks: A Systems Approach by Larry Peterson and Bruce Davie
Link: <https://book.systemsapproach.org/index.html>
- Lecture slides, notes, quizzes, homework, labs, etc.: Canvas
- Prerequisites: working knowledge of Python and Linux (**Don't worry if you don't have a working knowledge**)

What's the “Internet”?



Billions of connected computing *devices*:

- *hosts* = *end systems*
- running *network apps* as “*network edge*”

Packet switches:

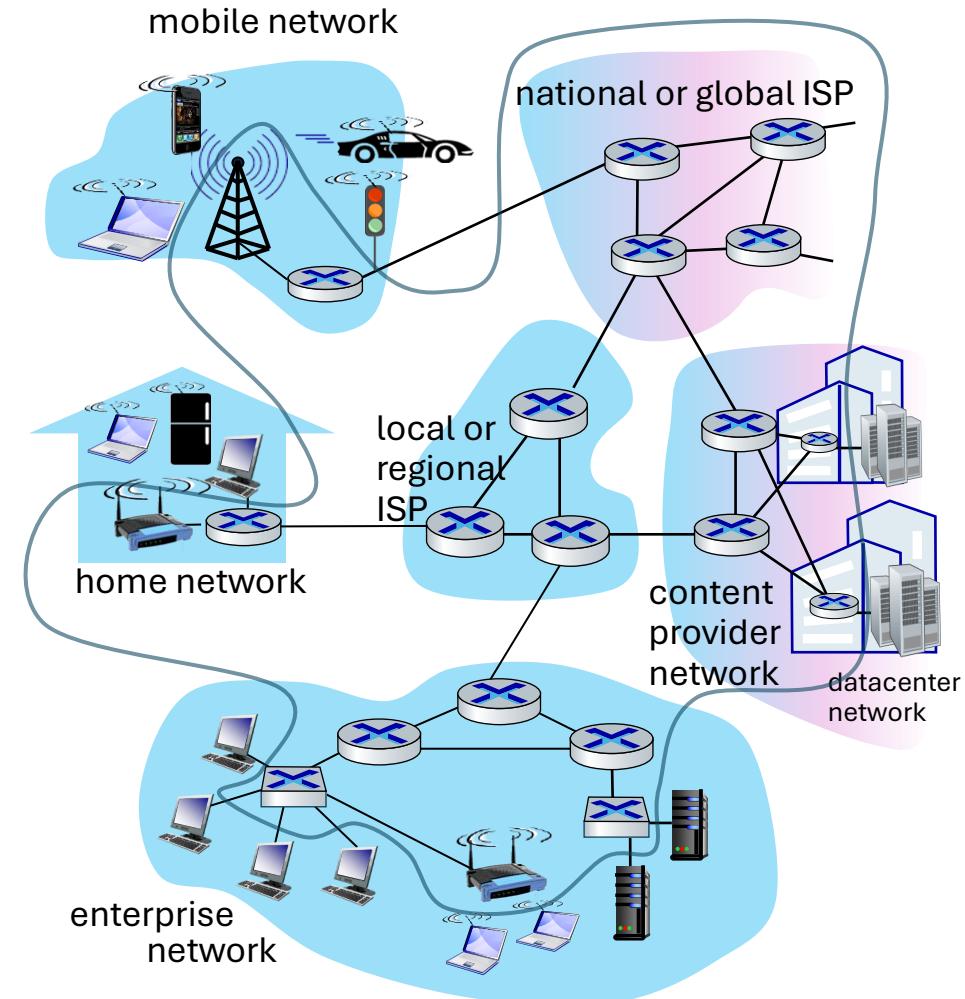
- forward packets (chunks of data) as “*network core*”
- *routers, switches*

Communication links

- fiber, copper, radio, satellite
- transmission rate: *bandwidth*

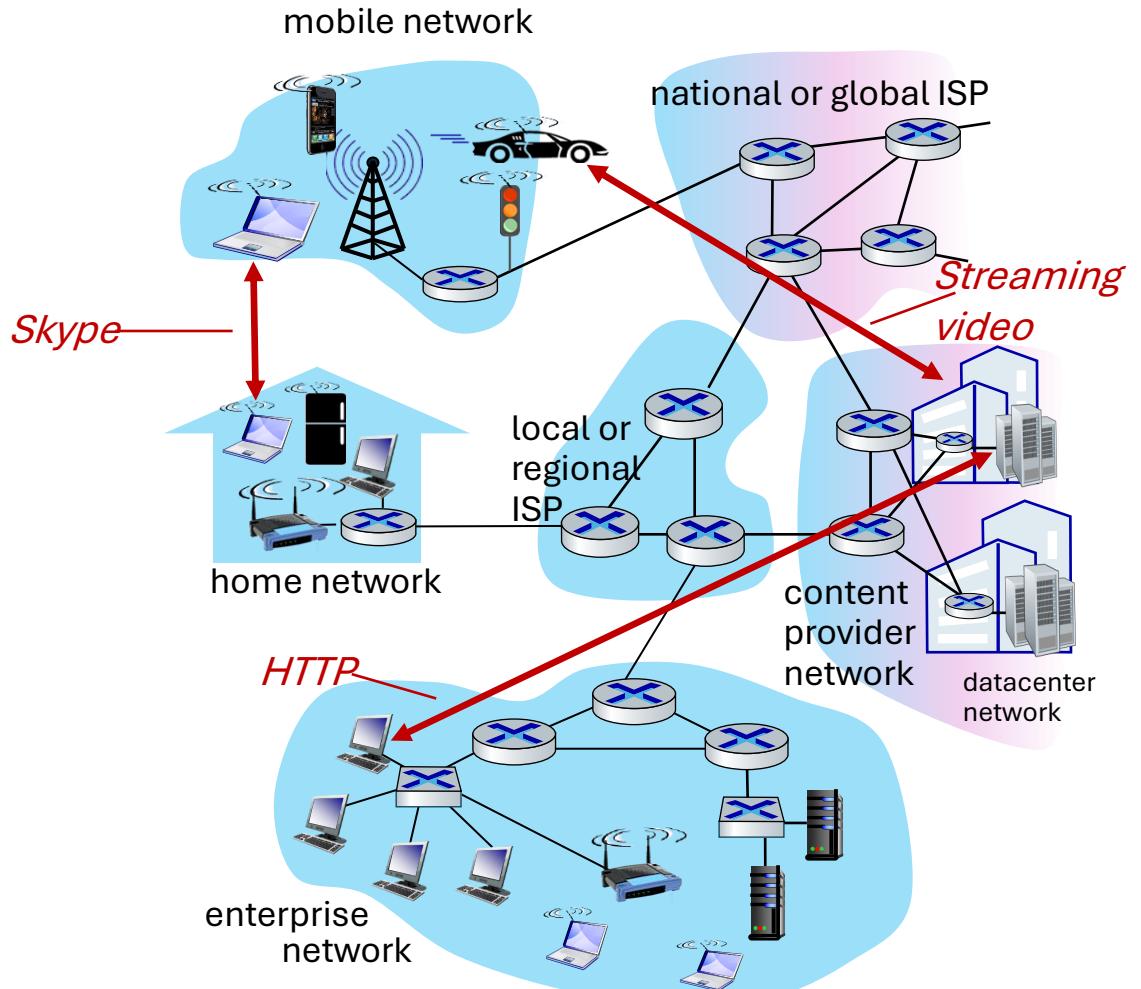
Networks

- collection of devices, routers, links: managed by an organization



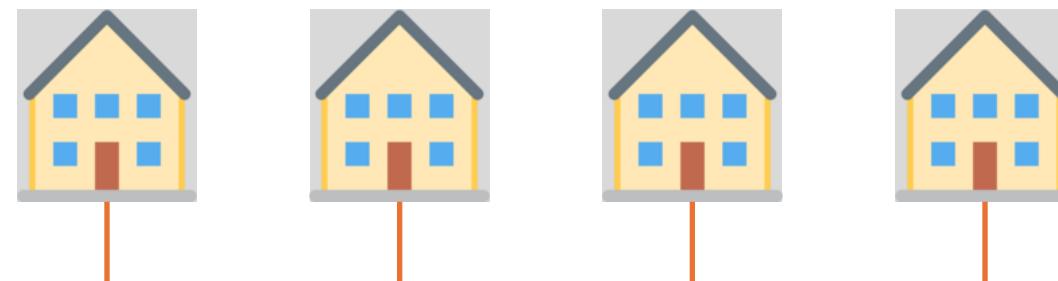
What can “Internet” do?

- *Infrastructure* that provides services to applications:
 - Web, streaming video, multimedia teleconferencing, email, games, e-commerce, social media, interconnected appliances, ...
- Provides *programming interface* to distributed applications:
 - “hooks” allowing sending/receiving apps to “connect” to, use Internet transport service
 - provides service options, analogous to postal service



Postal System Analogy – Building Block 1: Something that moves data

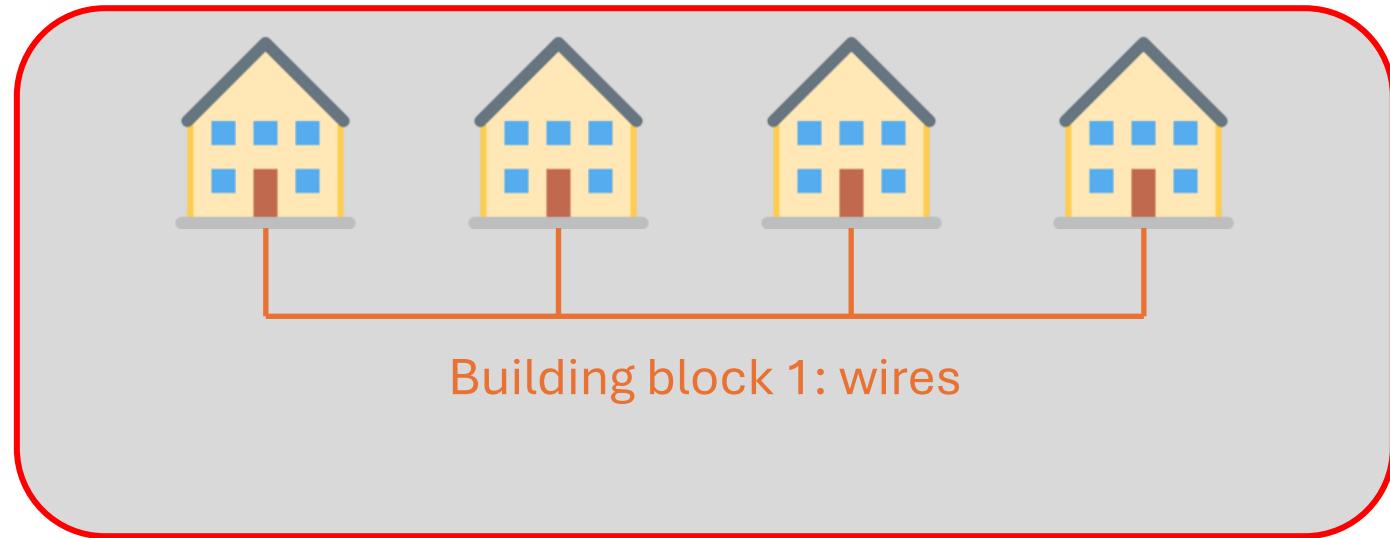
- Postal service needs a way to **deliver letters**: mailman, carrier pigeon, Pony Express, trucks, airplanes, etc.
- Internet also needs a way to move bits: wires, fiber optics, radio waves, satellites, etc.
- Risks in delivery: letters may get lost, opened by others, or delayed. Similarly, network packets can be lost, intercepted, or exposed.



Building block 1: wires

Postal System Analogy – Building Block 2: Talking to the apartment complex (Local Communication)

- Within one apartment complex, the mailman delivers letters room-to-room
- This is like a Local Area Network (LAN), connecting devices inside the same building
- People can talk to their neighbors without involving the post office.

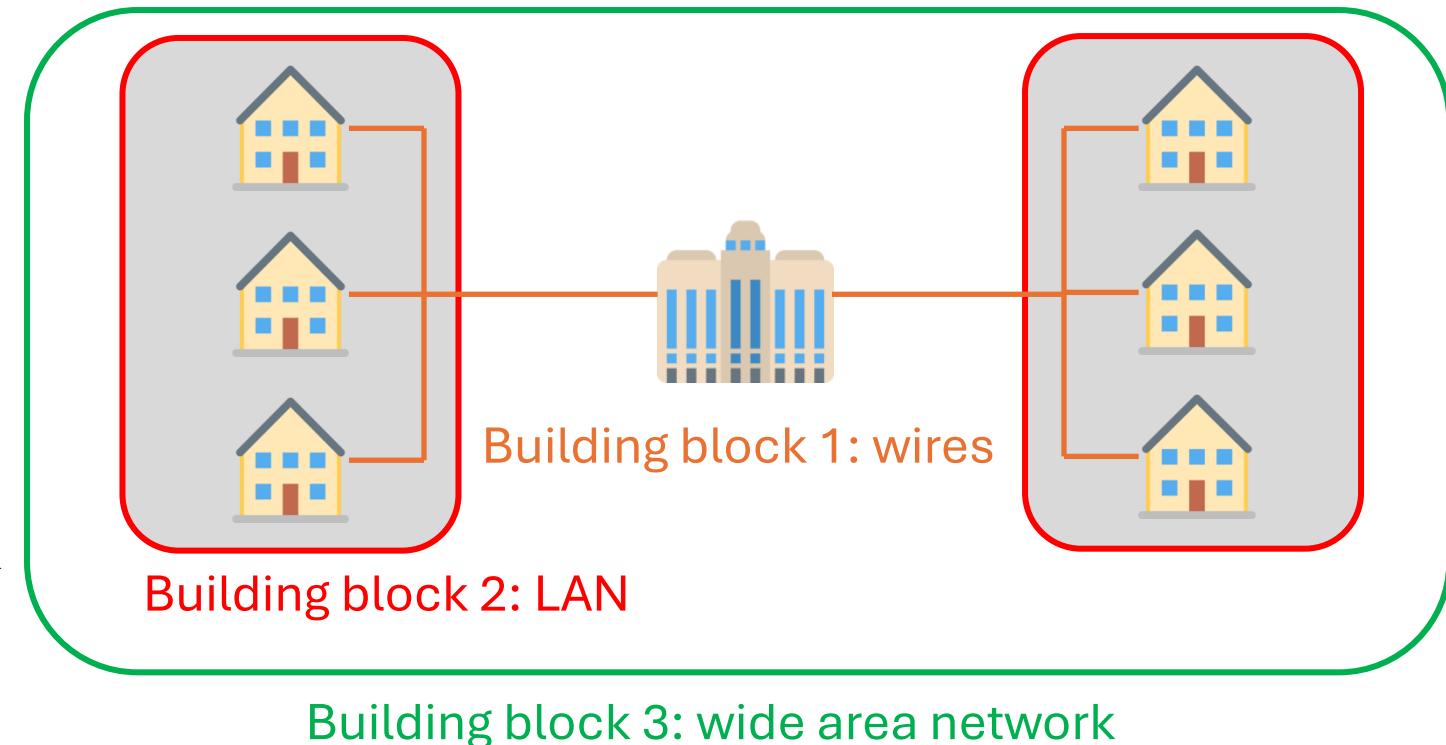


Building block 1: wires

Building block 2: LAN

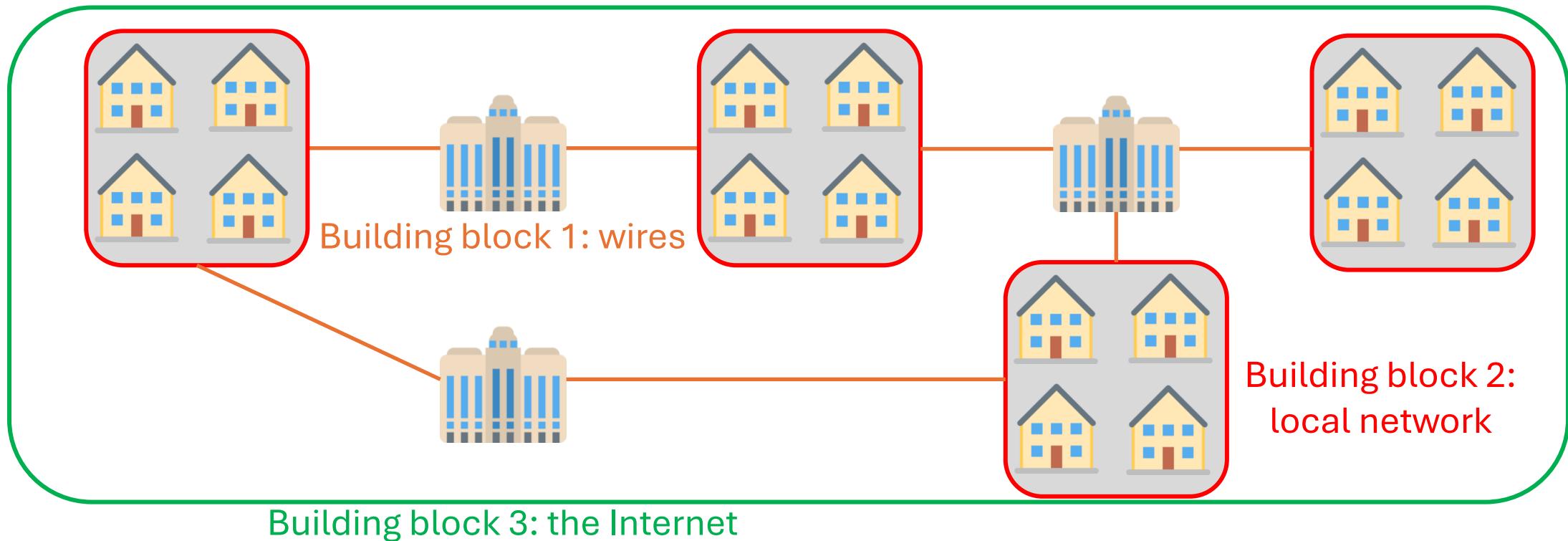
Postal System Analogy – Building Block 3: Post offices connect complexes (Wide Communication)

- A post office connects multiple apartment complexes
- Letters can travel between complexes through the post office
- This is like a Wide Area Network (WAN), connecting LANs together.



Postal System Analogy – Internet Scale: Connecting the whole world

- Multiple post offices around the world connect into a giant global system
- A letter may pass through several post offices before reaching its destination
- The Internet is the same: messages (packets) hop across multiple routers/ISPs before arriving.

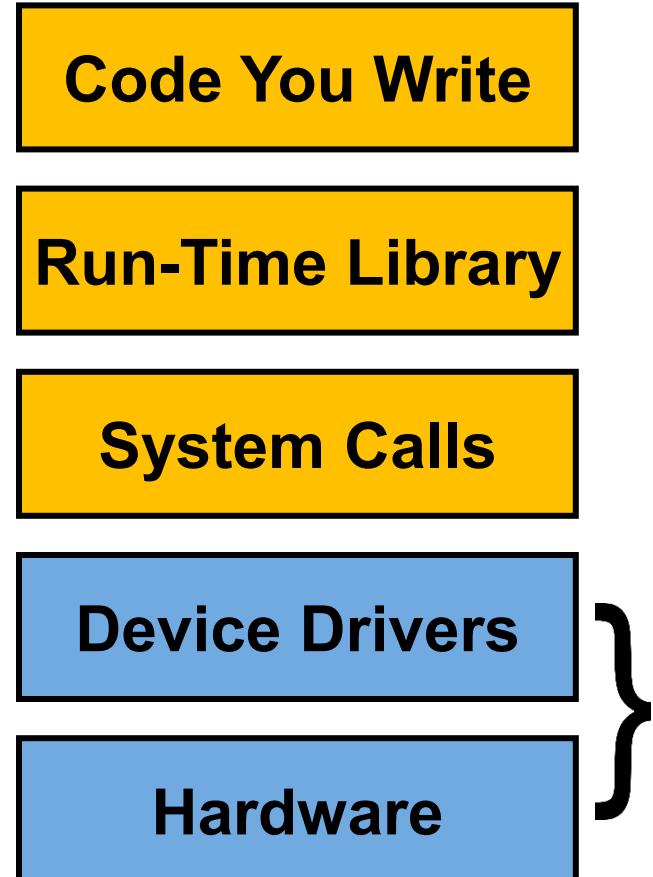


Internet: Layers of abstraction

- Postal system layers:
 - Letter (content you wrote)
 - Envelope (addressing)
 - Mail carrier (physical delivery)
 - Post office routing (deciding the next hop)
- Internet layers:
 - Application (email, web page)
 - Transport (TCP, UDP)
 - Network (IP)
 - Link (WiFi, Ethernet)
 - Physical (cables, wireless signals)
- Layer 3: Connect many local networks to form a global network
- Layer 2: Create links in a local area
- Layer 1: Move bits across space => within single computer, or between computers
- A change in one layer (e.g., switch pigeons to trucks) doesn't break the others.

Internet layering

- Internet design is partitioned into various layers. Each layer...
 - Has **protocols**
 - Relies on services provided by the layer below it
 - Provides services to the layer above it
- Analogous to the structure of an application and the “services” that each layer relies on and provides



Fully isolated
from user
programs

Internet layering: OSI vs. TCP/IP model

- Open Systems Interconnection (OSI) model:
 - A theoretical, academic model of Internet communication
 - Originally divided into 7 layers - But layers 5 and 6 aren't used in the real world, so we **ignore** them
- Same reliance upon abstraction
 - A layer can be implemented in different ways without affecting other layers
 - A layer's protocol can be substituted with another protocol without affecting other layers

Layer 3 – 7 are closer to software:
logical addressing, connections, applications

Layer 1 & 2 are closer to hardware (hardware + firmware):
signals, frames, MAC addresses

7. Application Layer

~~**6. Presentation Layer**~~

~~**5. Session Layer**~~

4. Transport Layer

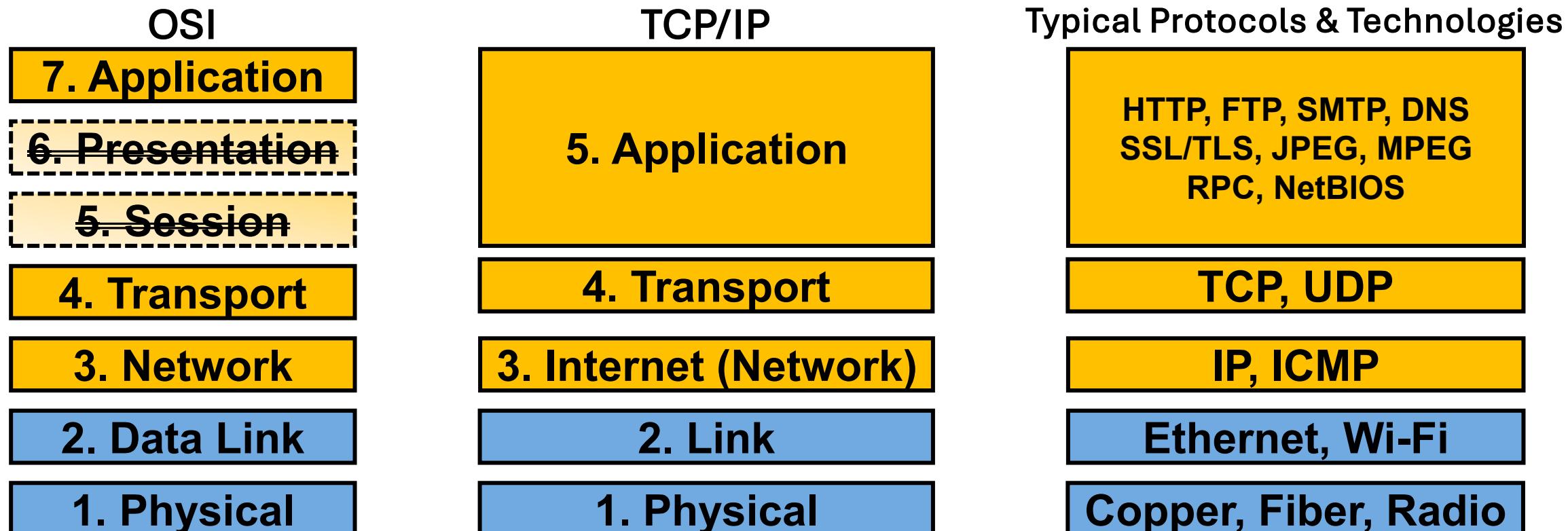
3. Network Layer

2. Data Link Layer

1. Physical Layer

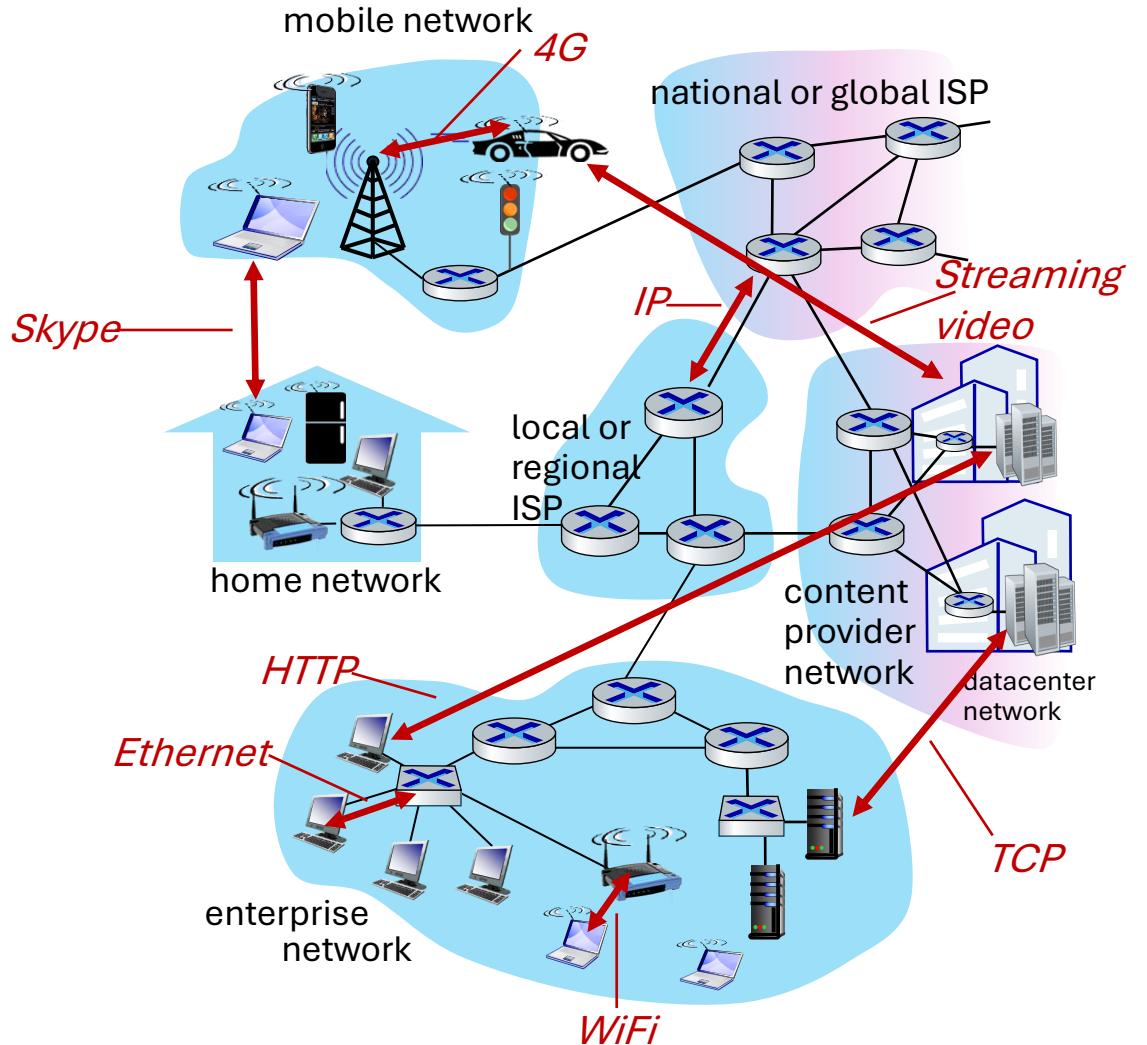
Internet layering: OSI vs. TCP/IP model

- Transmission Control Protocol/Internet Protocol (TCP/IP) model:
 - Practical model used on today's Internet → Layers 5 (Session) + 6 (Presentation) are usually bundled into Application in TCP/IP
- Different models, same idea → networking is layered abstraction, where each layer provides services to the one above
- We follow the TCP/IP model (aka the Internet protocol suite model) to organize course content



The Internet: a “nuts and bolts” view

- *Internet: “network of networks”*
 - Interconnected ISPs
- *Protocols are everywhere*
 - Control sending, receiving of messages
 - e.g., HTTP (Web), streaming video, Skype, TCP, IP, WiFi, 4/5G, Ethernet
- *Internet standards*
 - RFC: Request for Comments
 - IETF: Internet Engineering Task Force



What's a protocol?

Human protocols:

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

Rules for:

- ... specific messages sent
- ... specific actions taken when message received, or other events

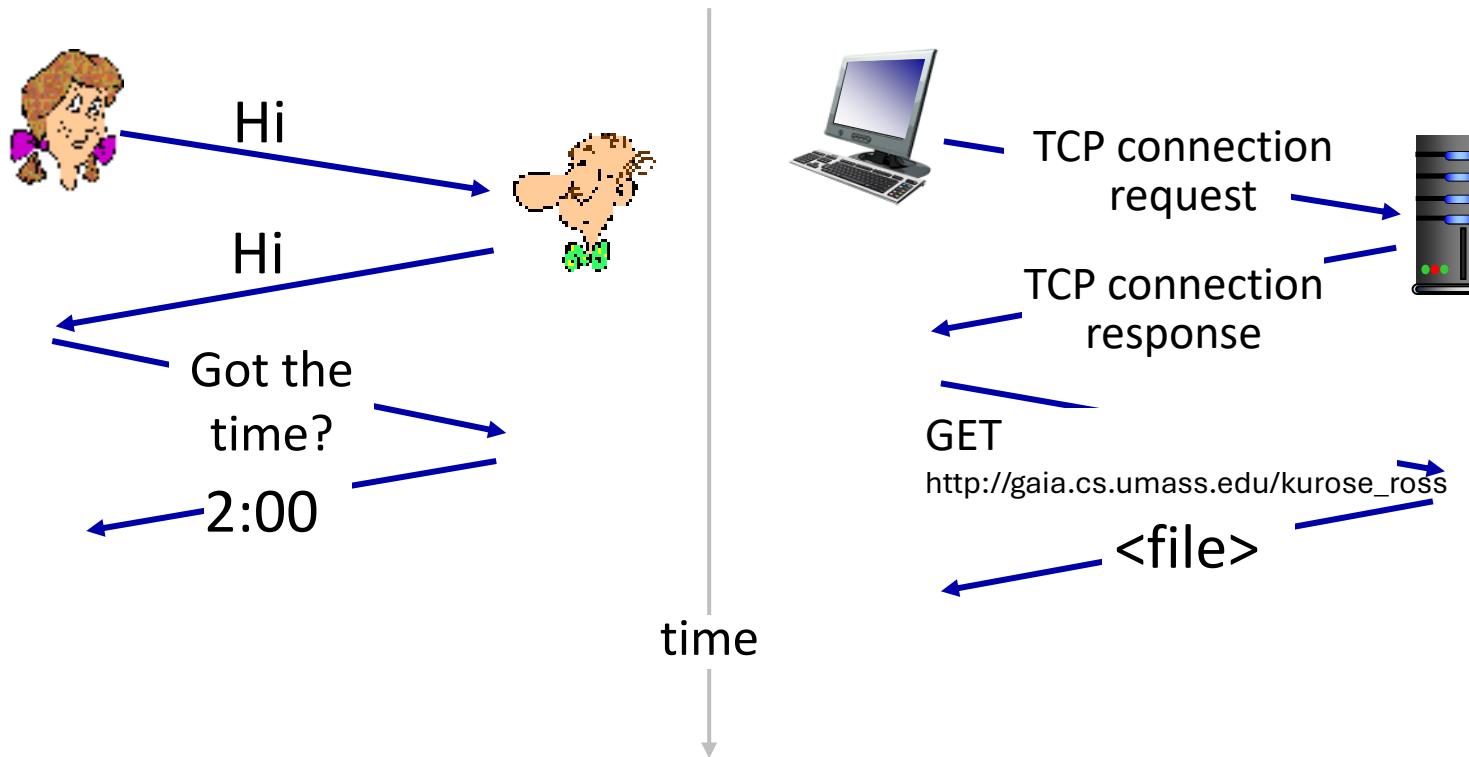
Network protocols:

- computers (devices) rather than humans
- all communication activity in Internet governed by protocols

*Protocols define the **format, order** of messages sent and received among network entities, and actions taken on message transmission, receipt*

What's a protocol – A Human Analogy

A human protocol and a computer network protocol:



Q: other human protocols?

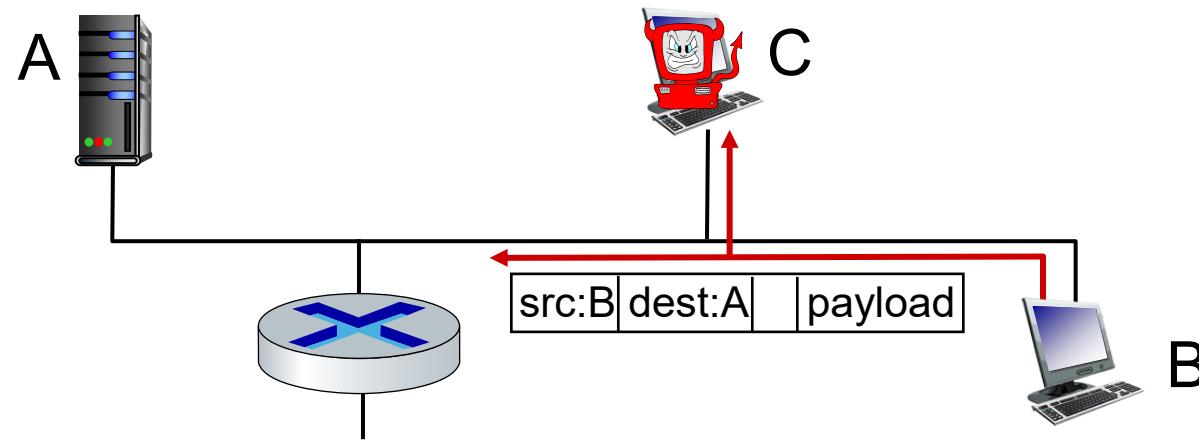
Network security

- Internet not originally designed with (much) security in mind
 - *original vision:* “a group of mutually trusting users attached to a transparent network” ☺
 - Internet protocol designers playing “catch-up”
 - security considerations in all layers!
- We now need to think about:
 - how bad guys can attack computer networks
 - how we can defend networks against attacks
 - how to design architectures that are immune to attacks

Bad guys: packet interception

packet “sniffing”:

- broadcast media (shared Ethernet, wireless)
- promiscuous network interface reads/records all packets (e.g., including passwords!) passing by

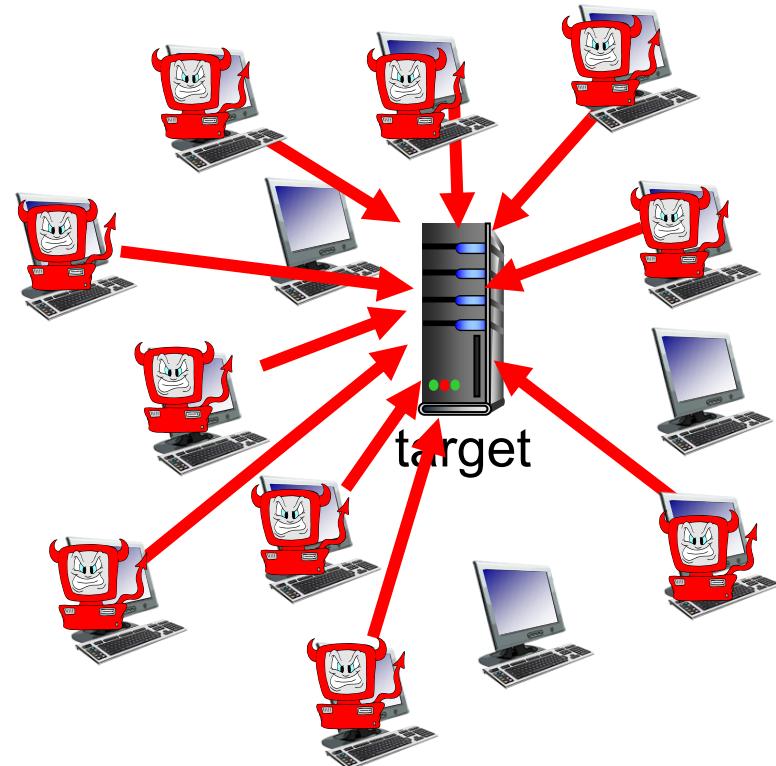


[Wireshark](#) software is a (free) packet-sniffer

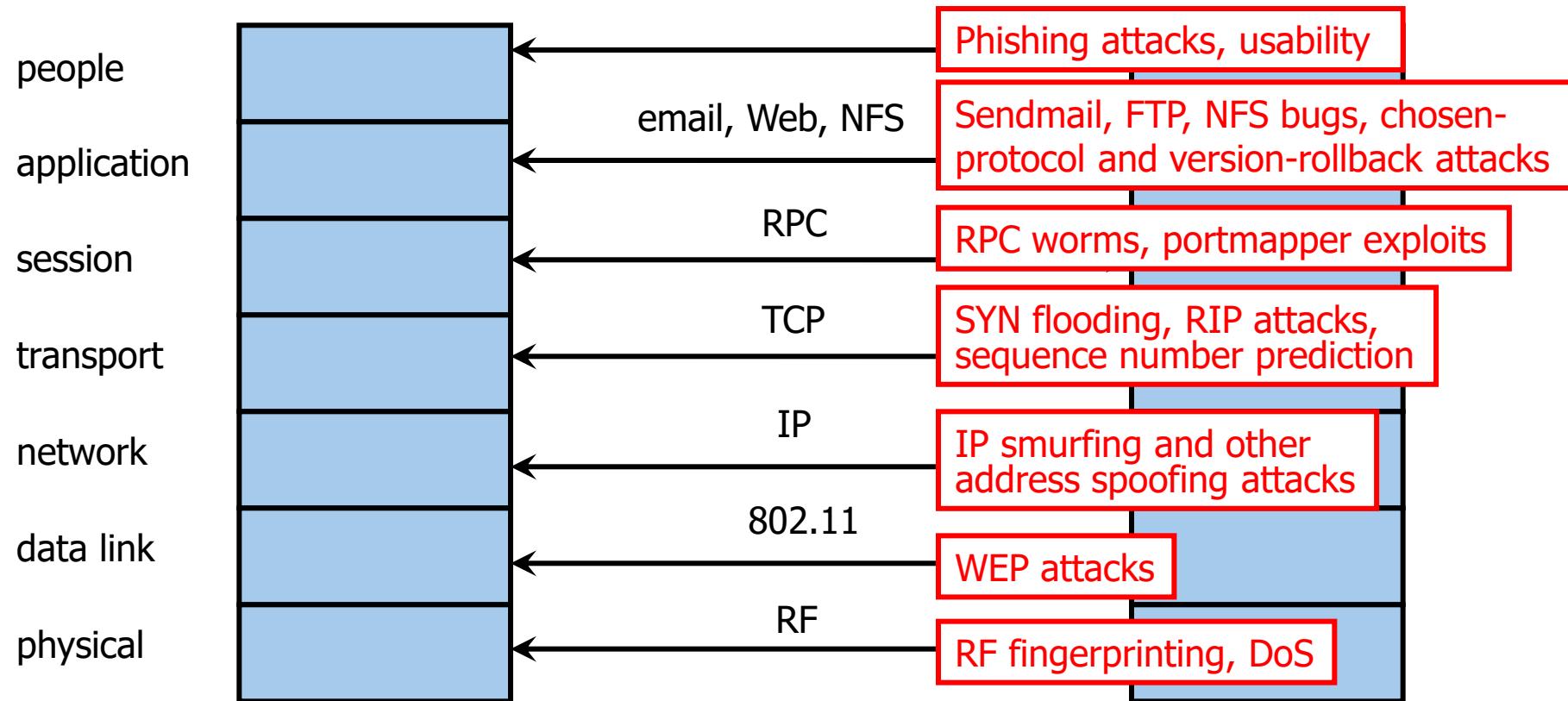
Bad guys: denial of service

Denial of Service (DoS): attackers make resources (server, bandwidth) unavailable to legitimate traffic by overwhelming resource with bogus traffic

1. select target
2. break into hosts around the network (see botnet)
3. send packets to target from compromised hosts

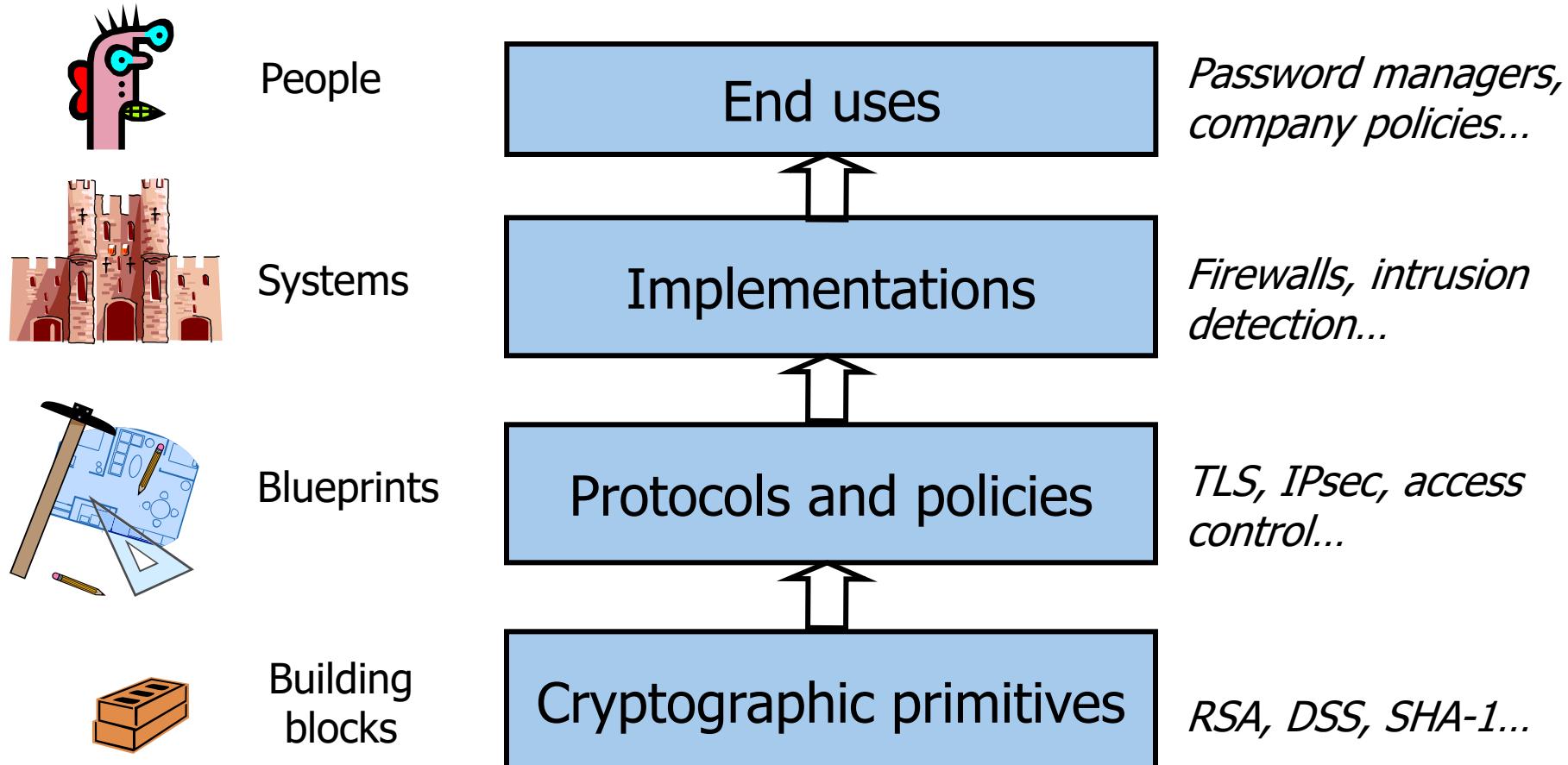


Network Threats & Attacks



Only as secure as the single weakest layer...
... or interconnection between the layers

Network Defenses



All defense mechanisms must work correctly and securely

Why do we need security and HOW TO?

- Scenarios: Trade secrets, medical records
 - Requirements: Protect vital information secret while still allowing access to those who need it (**Confidentiality**)
 - How to: Encrypted communications, secure transactions
- Scenarios: Ensure data integrity and prevent unauthorized modification
 - Scenarios: Financial transactions, software updates, legal documents
 - Requirements: Ensure data **integrity** and prevent unauthorized modification
 - How to: Digital signatures, Message authentication codes (MAC)

Why do we need security and HOW TO?

- Scenarios: Online banking, confidential email access, restricted databases
 - Requirements: Provide **authentication** and access control for resources
 - How to: Multi-factor authentication (MFA), public key infrastructure (PKI), biometric verification
- Scenarios: Cloud services, e-commerce platforms, critical infrastructure systems
 - Requirements: Guarantee **availability** of resources
 - How to: Redundant systems, load balancing

CSC 3511 Security and Networking

Week 10, Lecture 1: One-Time Pads, Stream Cipher, and Block Cipher

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	<ul style="list-style-type: none">• One-time pads• Stream ciphers• Block ciphers	<ul style="list-style-type: none">• RSA encryption
Integrity, Authentication	<ul style="list-style-type: none">• MACs (e.g., HMAC)	<ul style="list-style-type: none">• Digital signatures (e.g., RSA signatures)

Roadmap

- *One-Time Pads*
- *Stream Cipher*
- *Block Cipher*

Review: XOR

The XOR operator takes two bits and outputs one bit:

$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

Useful properties of XOR:

$x \oplus 0 = x$
$x \oplus x = 0$
$x \oplus y = y \oplus x$
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$
$(x \oplus y) \oplus x = y$

One-Time Pads: Key Generation

The key K is a *randomly-chosen* bitstring;
The key length is equal to the message length.

Alice

K	0	1	1	0	0	1	0	1	0	1	1	1
-----	---	---	---	---	---	---	---	---	---	---	---	---

Recall: We are in the symmetric-key setting, so
we **assume** Alice and Bob both know this key

One-Time Pads: Encryption

The plaintext M is the bitstring that Alice wants to encrypt

Alice												
K	0	1	1	0	0	1	0	1	0	1	1	1
	\oplus											
M	1	0	0	1	1	0	0	1	0	1	0	0
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
C	1	1	1	1	1	1	0	0	0	0	1	1

Encrypt: XOR each bit of K with the matching bit in M

The ciphertext C is the encrypted bitstring that Alice sends to Bob over the insecure channel.

One-Time Pads: Decryption

Bob receives the ciphertext C . Bob knows the key K . How does Bob recover M ?

Bob												
K	0	1	1	0	0	1	0	1	0	1	1	1
	\oplus											
C	1	1	1	1	1	1	0	0	0	0	1	1
	\downarrow											
M	1	0	0	1	1	0	0	1	0	1	0	0

Decrypt: XOR each bit of K with the matching bit in C

For one-time pads, we generate a **new key** for every message

One-Time Pads: Summary

KeyGen():

- **Randomly generate** an n-bit key, where n is the length of your message
- Recall: we **assume** that Alice and Bob can securely share this key
- For one-time pads, we generate a **new key** for every message

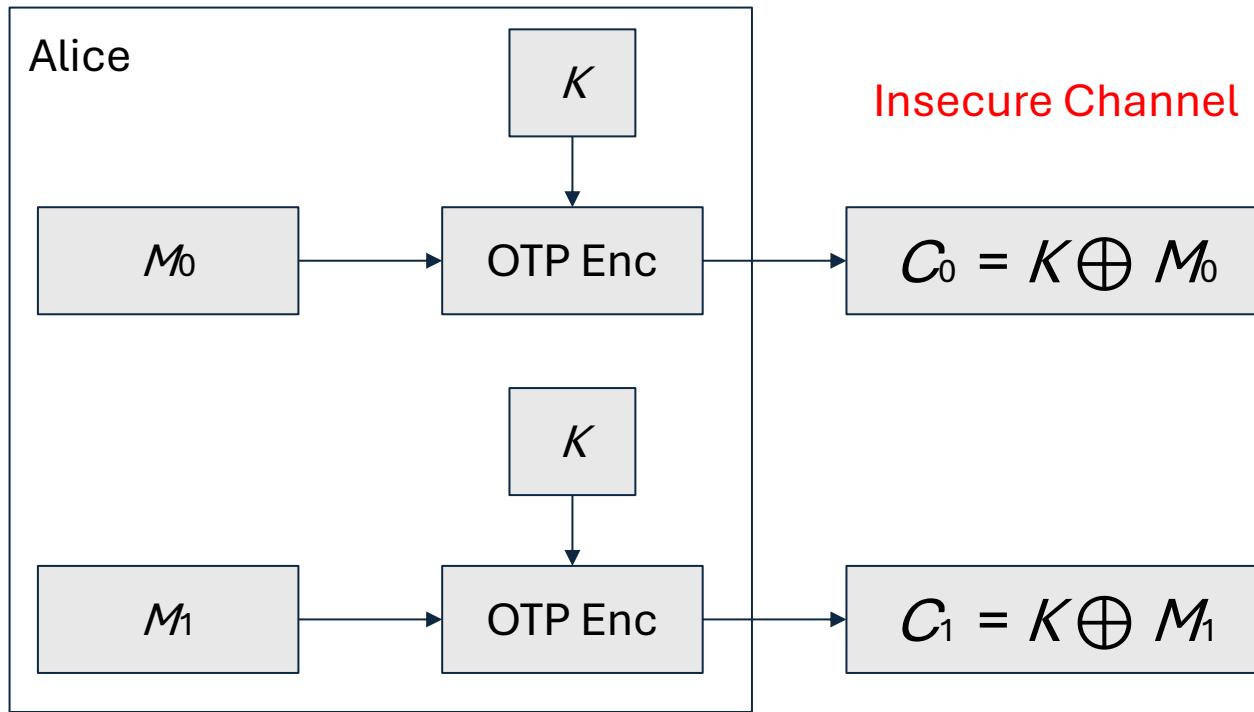
$\text{Enc}(K, M) = K \oplus M$:

- Bitwise XOR M and K to produce C
- In other words: XOR the i -th bit of the plaintext with the i -th bit of the key $\rightarrow C_i = K_i \oplus M_i$

$\text{Dec}(K, C) = K \oplus C$:

- Bitwise XOR C and K to produce M $\rightarrow M_i = K_i \oplus C_i$

Two-Time Pads?



Eve sees two ciphertexts over the insecure channel

$$C_0 \oplus C_1$$

What if Alice uses the same key K to encrypt **two different messages**?

What if Alice uses the same key K to encrypt **the same messages**?

Impracticality of One-Time Pads

Problem 1: Key generation

- Keys must be **randomly generated** for every message, and **never reused**
- Randomness is expensive

Problem 2: Key distribution

- To communicate an n-bit message, we need to securely communicate an n-bit key first
- But if we have a way to securely communicate an n-bit key, we could have communicated the message directly!

Communicate keys in advance:

- How to establish a shared secret key over an insecure channel? → [Diffie-Hellman key exchange](#) (next time)

Roadmap

- *One-Time Pads*
- ***Stream Cipher***
- *Block Cipher*

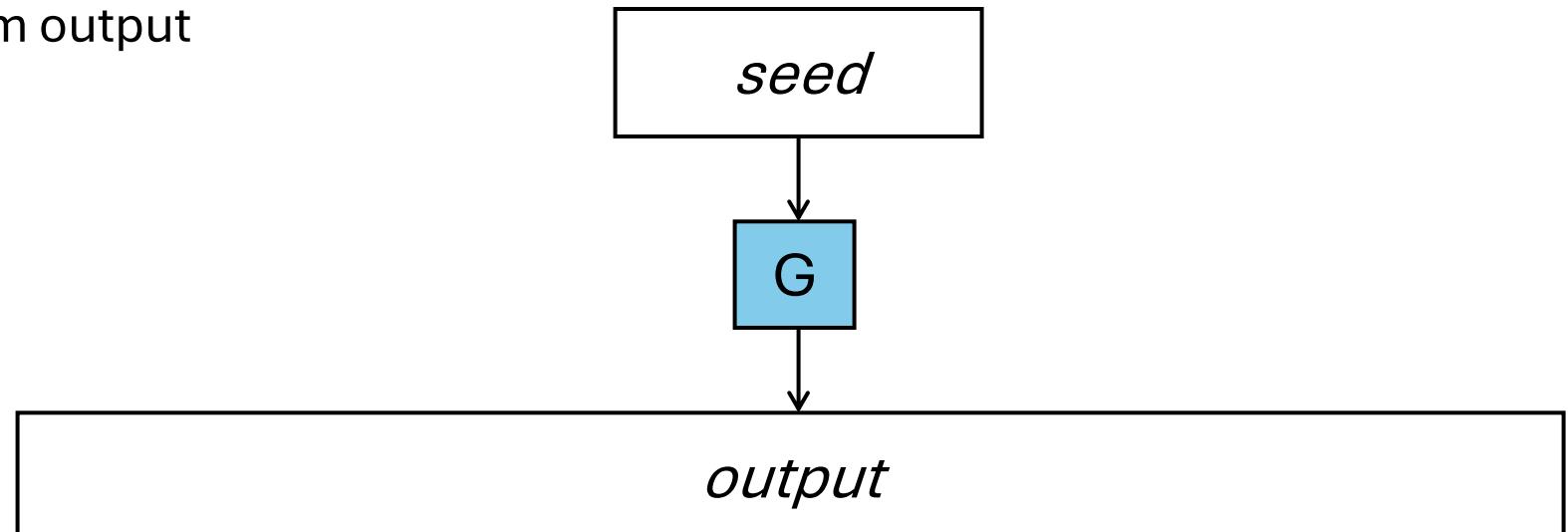
Pseudorandomness and PRGs

What does “pseudorandom” mean?

- Informal: cannot be distinguished from uniform (i.e., random)
- Informal:
 - Fix some distribution D on n -bit strings: $x \leftarrow D$ means “sample x according to D ”
 - D is pseudorandom if it passes **all** efficient statistical tests

Pseudorandom generators (PRGs)

- A PRG is an efficient, **deterministic** algorithm that expands a short, *uniform (random)* seed into a longer, pseudorandom output



Pseudorandom Generators

Functionality

- Deterministic algorithm $G \rightarrow$ Same input results in the same output
- Takes as input a *short random seed s*
- Outputs a *long* string $G(s)$

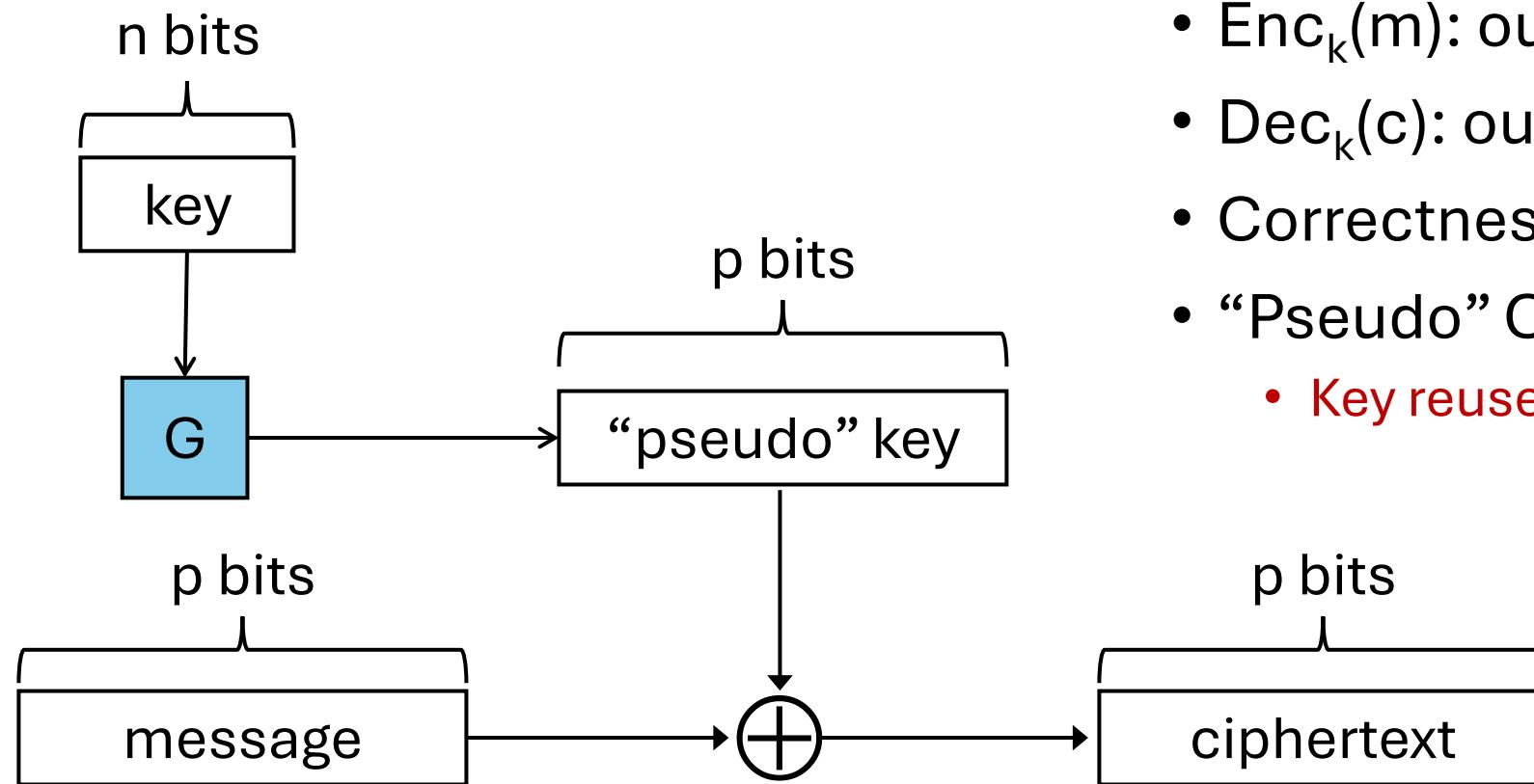
Security

- PRG output must be unpredictable
 - no “eff” adversary can predict bit $G(s)_{i+1}$
- No efficient algorithm can “distinguish” $G(s)$ from a truly random string r
 - i.e., passes all “statistical tests”

Intuition:

- Stretch a small amount of true randomness to a larger amount of pseudorandomness
- Why is this useful? \rightarrow A practical OTP instantiation?

PRGs & Stream Ciphers

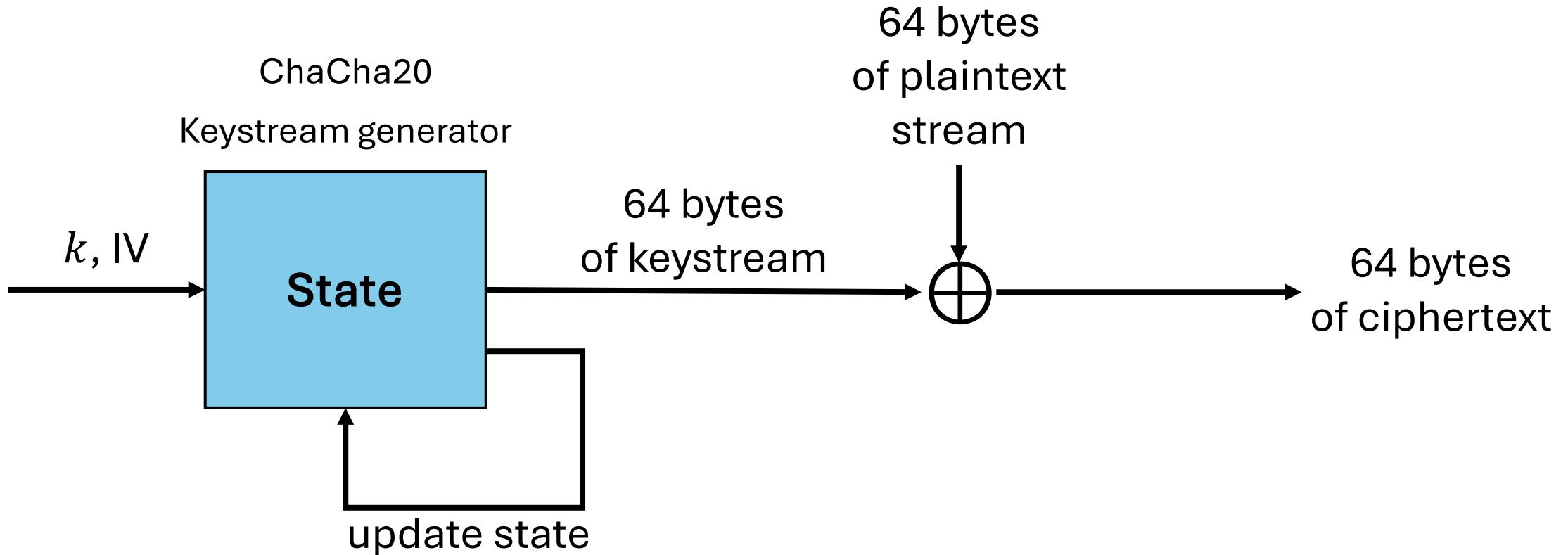


- $\text{Gen}(1^n)$: output uniform n -bit key k
 - Security parameter $n \rightarrow$ message space $\{0,1\}^{p(n)}$
- $\text{Enc}_k(m)$: output $G(k) \oplus m$
- $\text{Dec}_k(c)$: output $G(k) \oplus c$
- Correctness follows as in the OTP...
- “Pseudo” One-Time Pad
 - Key reuse problem?

ChaCha20

- Designed by [Dan Bernstein](#) in 2008
- Conceptually, it's very simple → uses arithmetic operations: integer addition modulo , XOR, and left rotations
- Extremely fast (>600 MB/sec) in software and does not require any special hardware
- No security weakness have been found
- ChaCha20 is widely used in practice, including in TLS

ChaCha20: Workflow



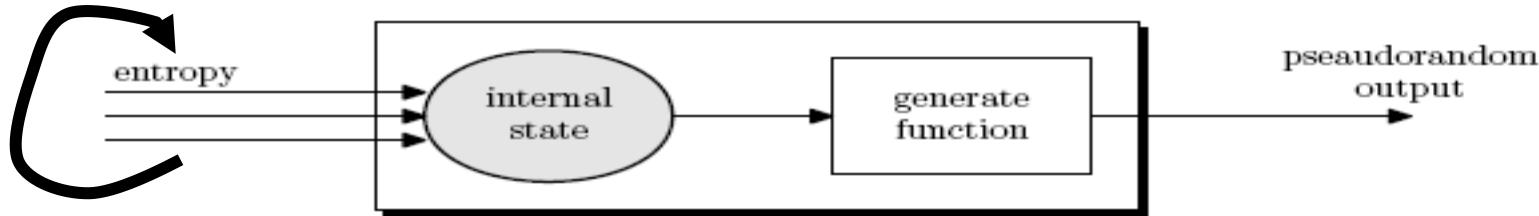
Initialization vector (IV): a public, non-repeating value for a given key k ; more details later

The pair (k, IV) is never used more than once → Can **re-use** k with a new IV

Why?

- key reuse would otherwise leak information about the plaintext: ***same plaintext* → *same ciphertext***

Generating Randomness (e.g. keys, IV)



Pseudorandom generators in practice: (e.g. /dev/random)

- Continuously add entropy (uncertainty) to internal state
- Entropy sources:
 - Hardware RNG: Intel [RDRAND](#) instruction (Ivy Bridge, 2012). 3Gb/sec.
 - Timing: hardware interrupts (keyboard, mouse)

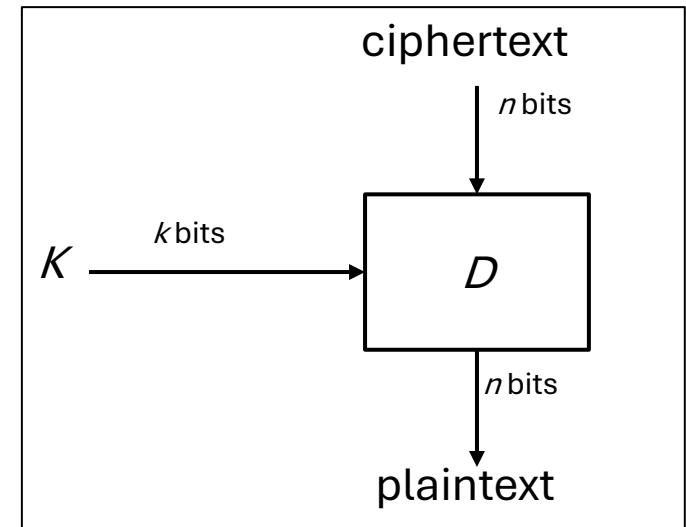
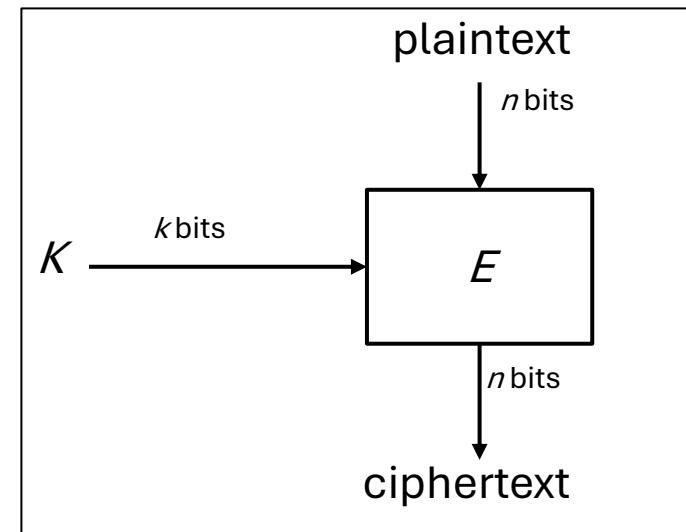
[NIST SP 800-90A](#): NIST approved generators

Roadmap

- *One-Time Pads*
- *Stream Cipher*
- ***Block Cipher***

Block Ciphers

- **Block cipher:** A cryptographic scheme consisting of encryption/decryption algorithms for a fixed-sized block of bits
- $Enc(K, M) \rightarrow C$:
 - Inputs: k -bit key K and an **n -bit** plaintext M
 - Output: An n -bit ciphertext C
 - Sometimes written as: $\{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$
- $Dec(K, C) \rightarrow M$:
 - Inputs: a k -bit key, and an n -bit ciphertext C
 - Output: An n -bit plaintext
 - Sometimes written as: $\{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$
 - The **inverse** of the encryption function

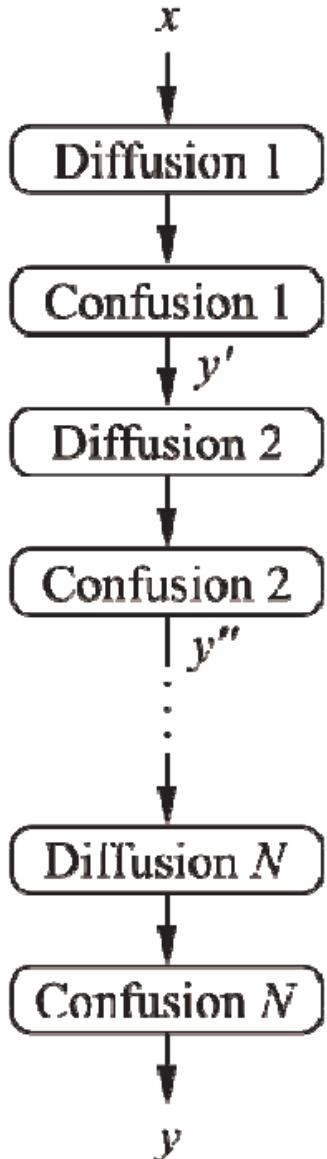


Block Cipher Primitives: Confusion & Diffusion

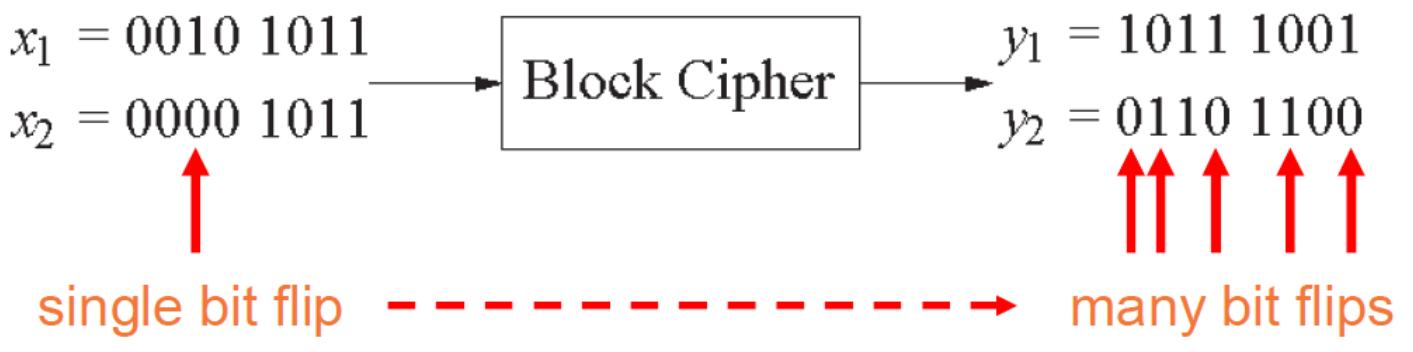
Claude Shannon: There are two primitive operations with which strong encryption algorithms can be built:

- **Confusion:** An *encryption operation* where the **relationship between key and ciphertext is obscured (or, hidden)**
 - Today, a common element for achieving confusion is substitution, which is found in both AES and DES. E.g., ***S-box***
- **Diffusion:** An *encryption operation* where the **influence** of one plaintext symbol is spread over many ciphertext symbols with the goal of **hiding** statistical properties of the plaintext.
 - A simple diffusion element is the **bit permutation**, which is frequently used within DES.
 - Both operations by themselves cannot provide security. The idea is to concatenate confusion and diffusion elements to build so called product ciphers

Product Ciphers

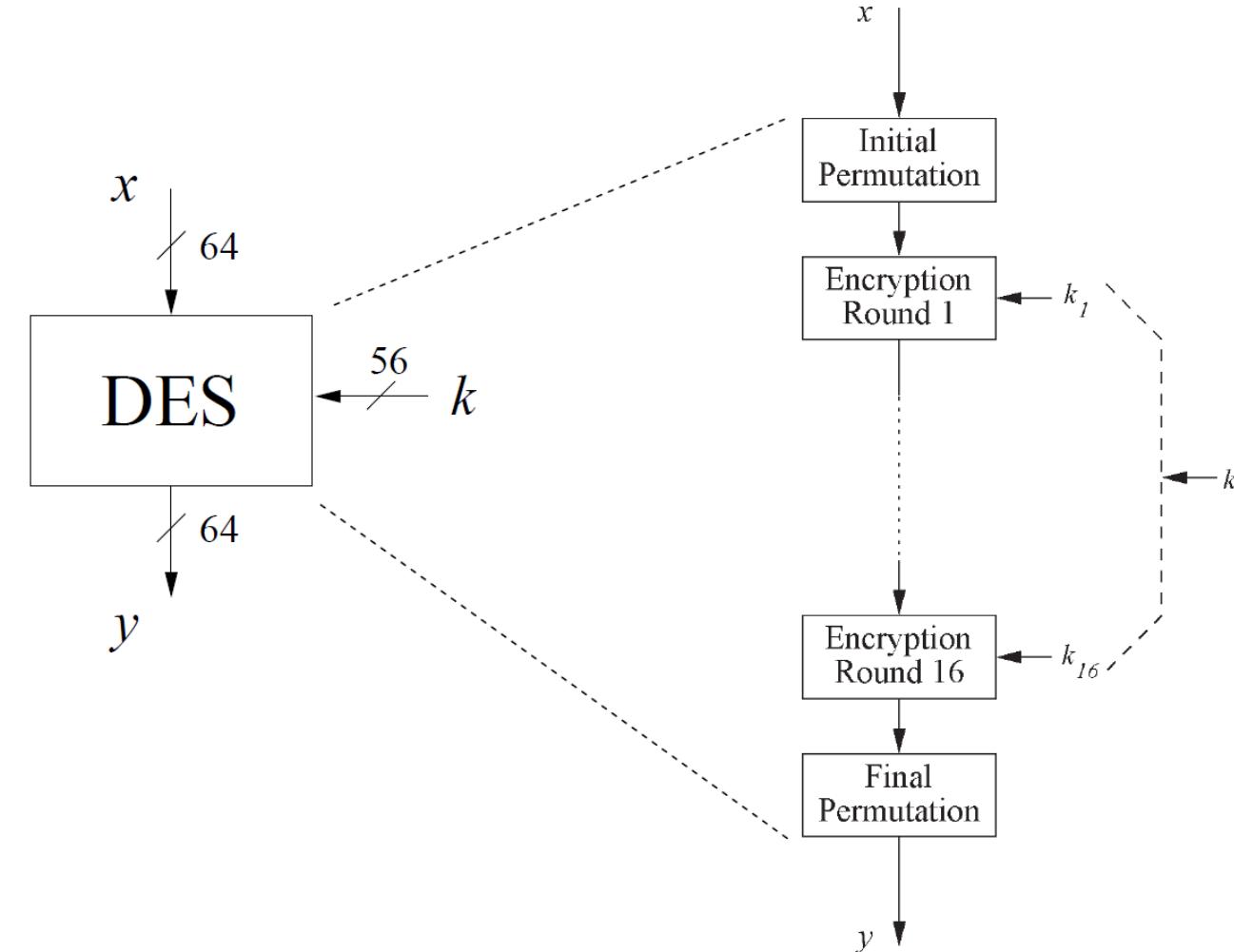


- Most of today's block ciphers are product ciphers as they consist of rounds which are applied repeatedly to the data
- Can reach excellent diffusion: changing of **one bit of plaintext** results on average in the **change of half the output bits**
- Example:



Overview of the DES (Data Encryption Standard)

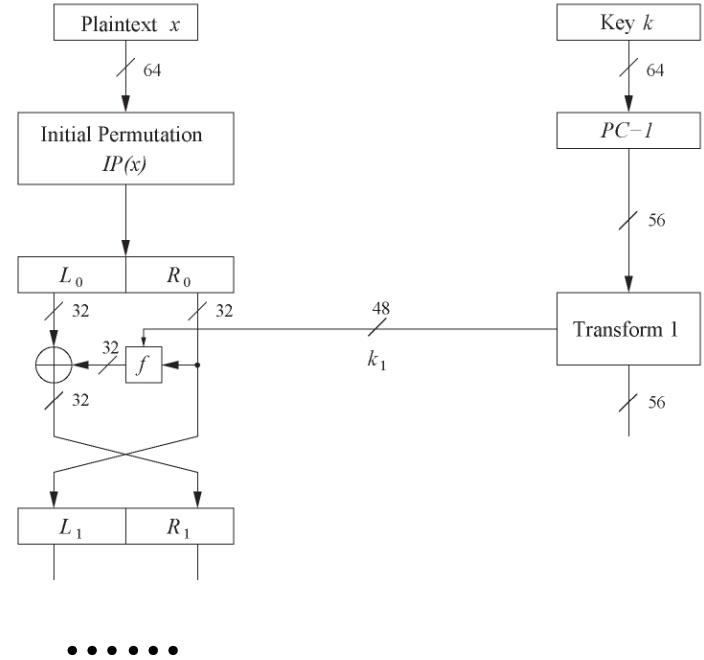
- Developed by IBM and adopted by NBS (now NIST) in 1977
- Encrypt blocks of size 64 bits
- Use a key of size 56 bits
- Use the same key for Enc/Dec
- Use 16 rounds which all perform the identical operation
- Different subkeys in each round derived from encryption key
- **DES is insecure; do not use it!**



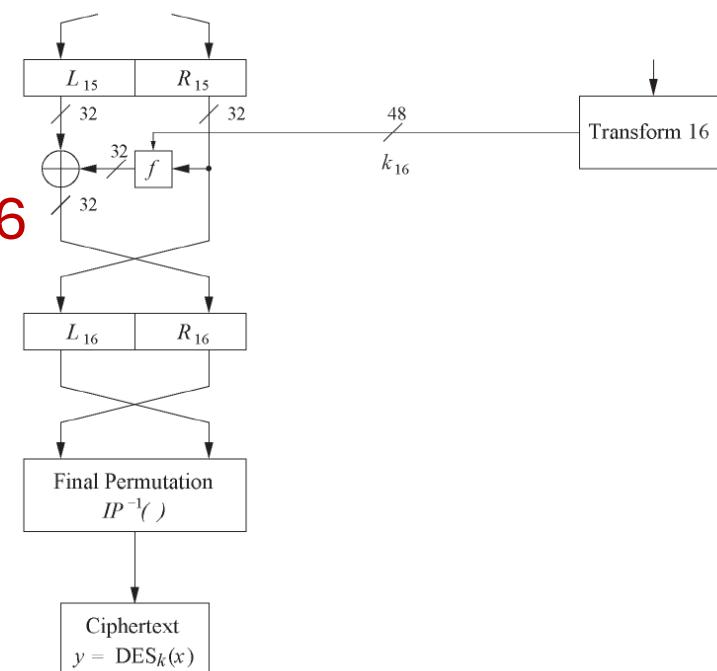
DES internals: Feistel network

- DES structure is a *Feistel network*
- Bitwise initial permutation, then 16 rounds:
 - Plaintext is split into 32-bit halves L_i and R_i
 - R_i is fed into function f , the output of which is then XORed with L_i
 - Left and right half are swapped
- Rounds can be expressed as:
 - $L_i = R_{i-1}$
 - $R_i = L_{i-1} \oplus f(k_i, R_{i-1})$
- L_{16} and R_{16} swapped again at the end of the cipher, i.e., after round 16 followed by a final permutation (IP^{-1})

Round 1

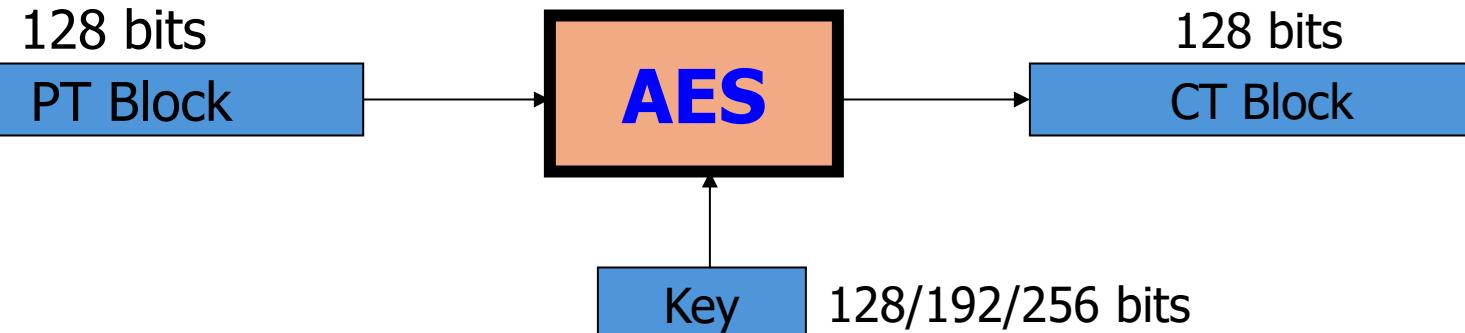


Round 16



Advanced Encryption Standard (AES)

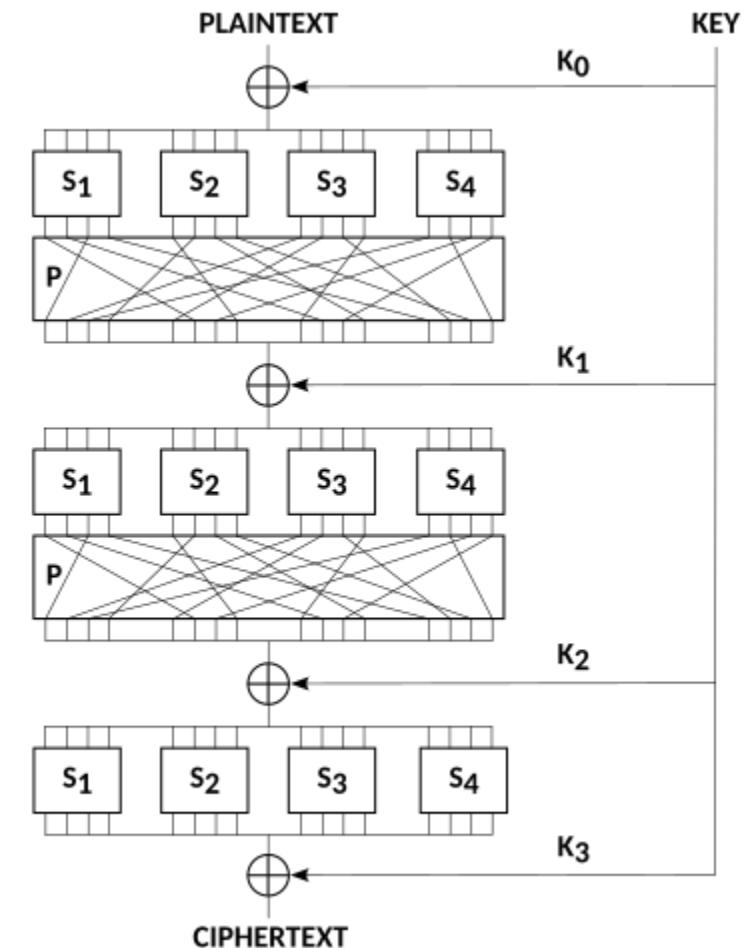
- The need for a new block cipher announced by National Institute of Standards and Technology (NIST) in January 1997
- 15 candidate schemes accepted in August 1998 → Rijndael was chosen as the AES in October 2000
- AES was formally approved as a US federal standard in November 2001 ([FIPS 197](#))
- Key lengths: 128, 192, and 256 bits
- Block length: 128 bits
- 2025: No attacks have been found on AES that are (significantly) faster than exhaustive key search
- AES is an example of a **substitution-permutation network**



	Key length (bits)	Number of rounds
AES-128	128	10
AES-192	192	12
AES-256	256	14

Substitution-permutation Networks (SPN)

- A SPN is an iterated block cipher where a round consists of a substitution operation followed by a permutation operation
- Components of a SPN cipher:
 - n : the block length (in bits)
 - l : the key length (in bits)
 - h : the number of rounds
 - A fixed invertible function $S: \{0, 1\}^b \rightarrow \{0, 1\}^b$ called a **substitution**, where b is a divisor of n .
 - A fixed **permutation** P on $\{1, 2, \dots, n\}$
 - A **key scheduling algorithm** that determines **subkeys** $k_1, k_2, \dots, k_h, k_{h+1}$ from a key k .

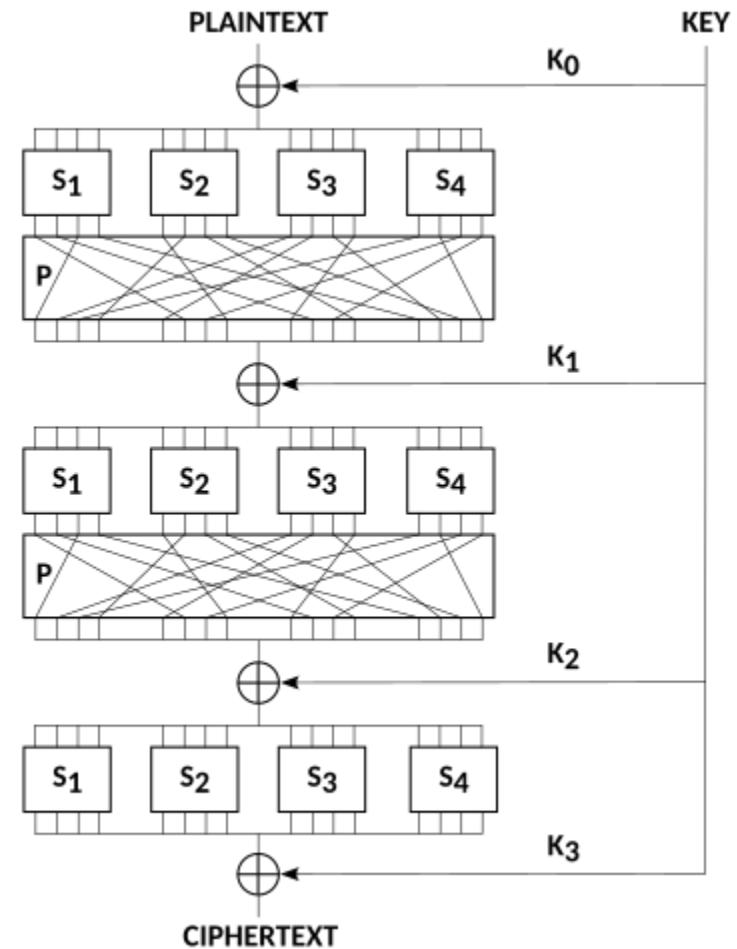


Substitution-permutation Networks (SPN)

- Here is a description of encryption

```
A ← plaintext  
for  $i = 1, 2, \dots, h$  do  
     $A \leftarrow A \oplus k_i$  (xor)  
     $A \leftarrow S(A)$  (substitution)  
     $A \leftarrow P(A)$  (permutation)  
 $A \leftarrow A \oplus k_{h+1}$   
ciphertext  $\leftarrow A$ 
```

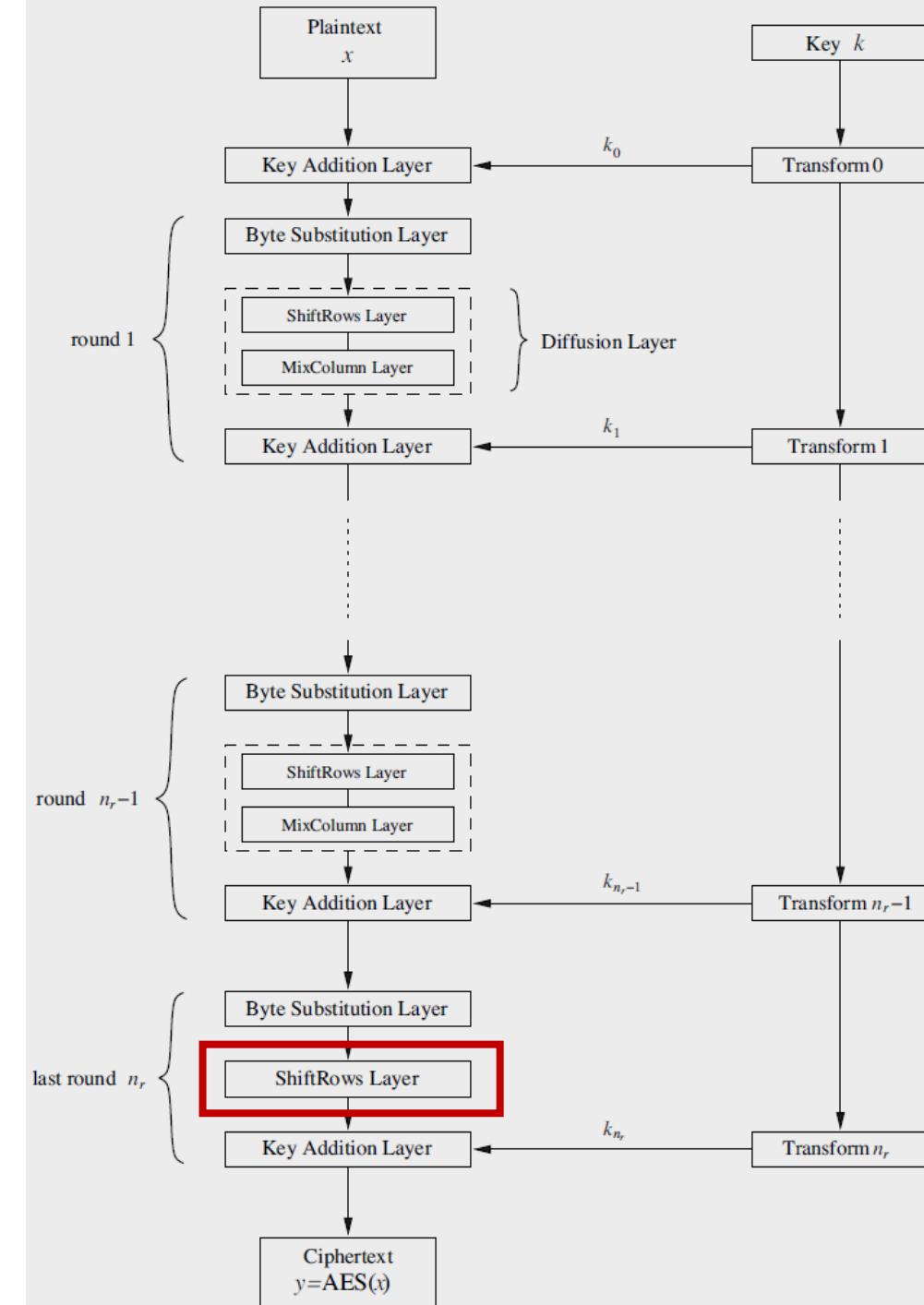
- Decryption is the reverse of encryption
- AES is an SPN, where the permutation operation is comprised of two invertible linear transformations (ShiftRows & MixColumns)



AES: structural information

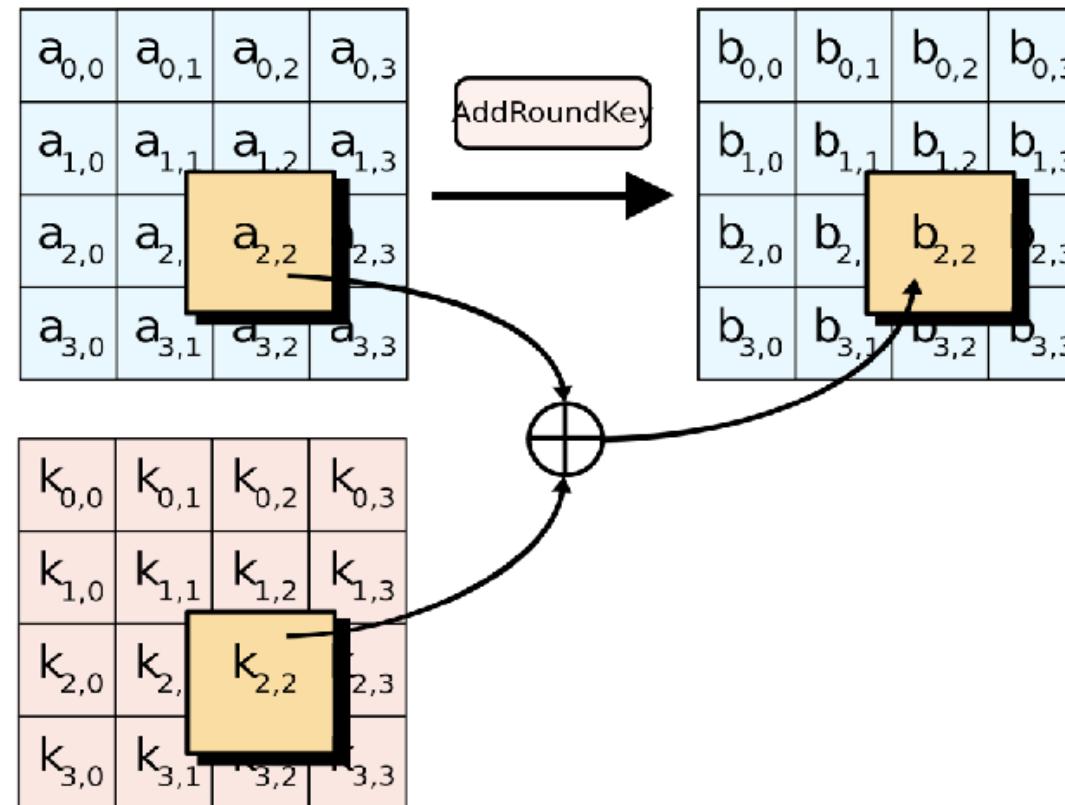
- All operations are byte oriented: Efficient in software and hardware
- From the key k derive $h + 1$ subkeys $k_1, k_2, \dots, k_h, k_{h+1}$

```
State ← plaintext
State ← State ⊕  $k_0$ 
for  $i = 1, 2, \dots, h - 1$  do
    State ← SubBytes(State)
    State ← ShiftRows(State)
    State ← MixColumns(State)
    State ← State ⊕  $k_i$ 
State ← SubBytes(State)
State ← ShiftRows(State)
State ← State ⊕  $k_h$ 
ciphertext ← State
```



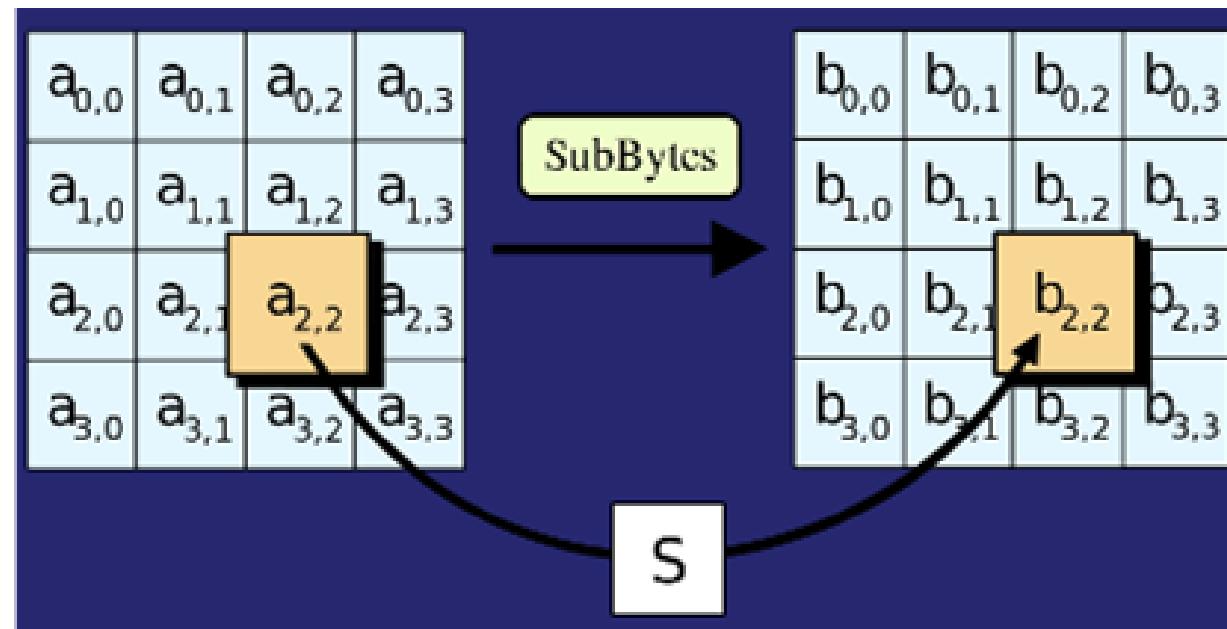
AES Algorithm: AddRoundKey()

- XOR the 16-byte block with the 16-byte round key



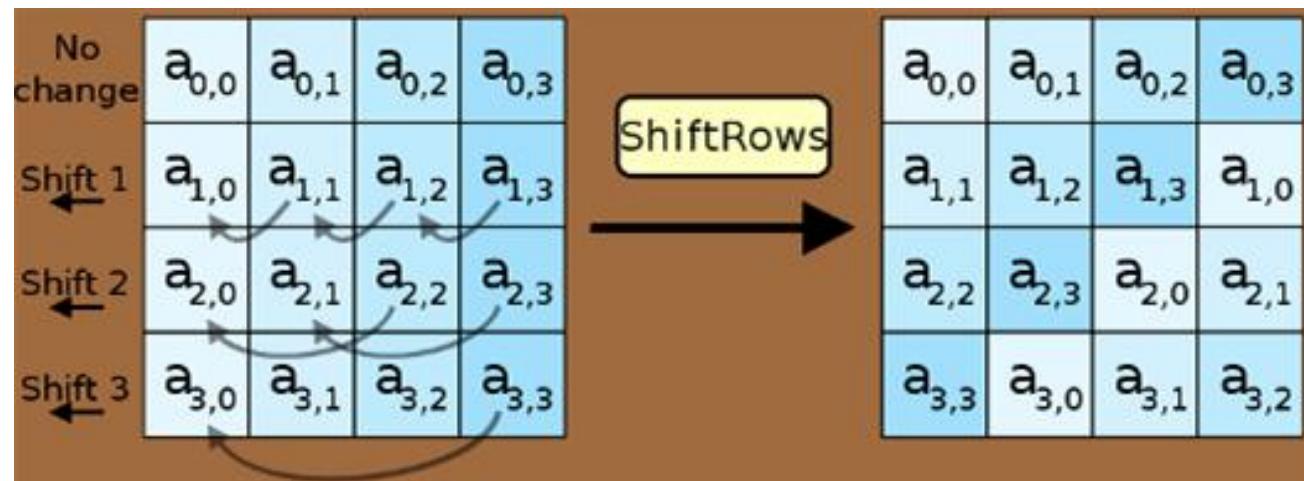
AES Algorithm: SubBytes()

- Replace each byte in the block with another byte using an 8-bit substitution box



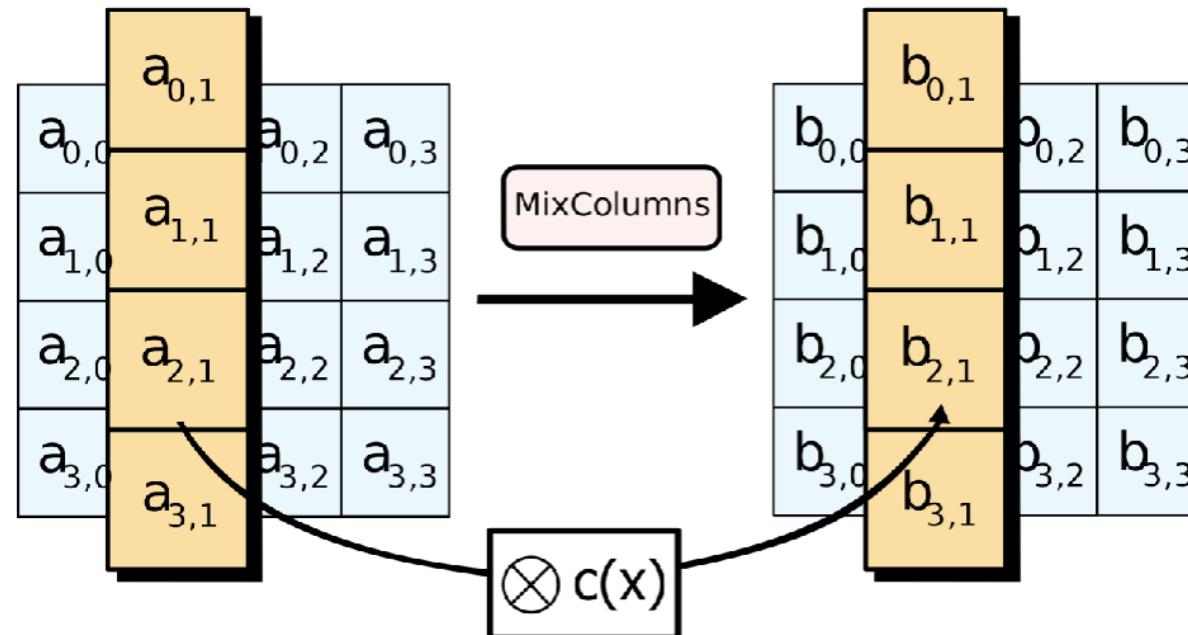
AES Algorithm: ShiftRows()

- Cyclically shifts the bytes in each row by a certain offset
- The number of places each byte is shifted differs for each row



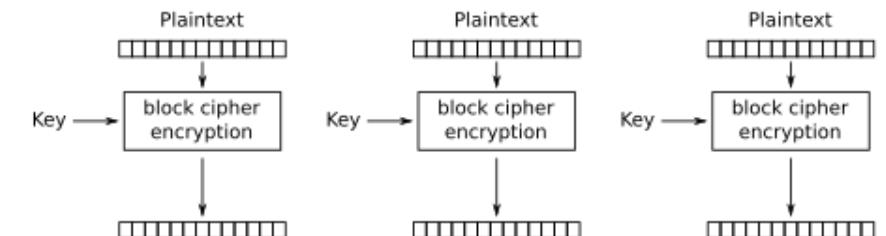
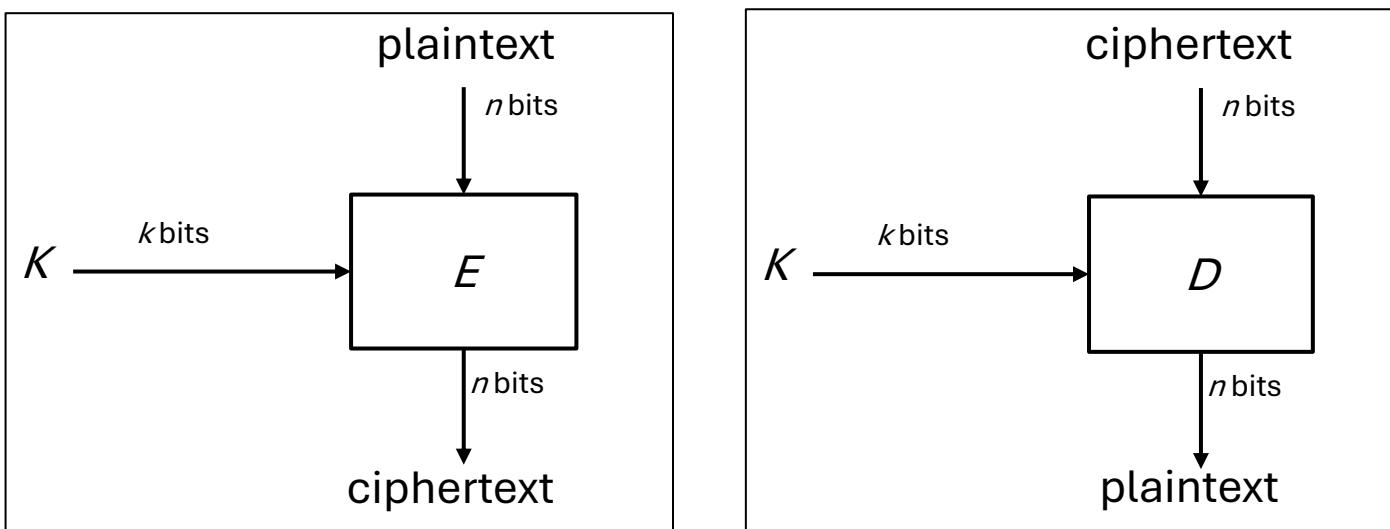
AES Algorithm: MixColumns()

- Treats the 16-byte block as a 4×4 matrix and multiply it by another matrix

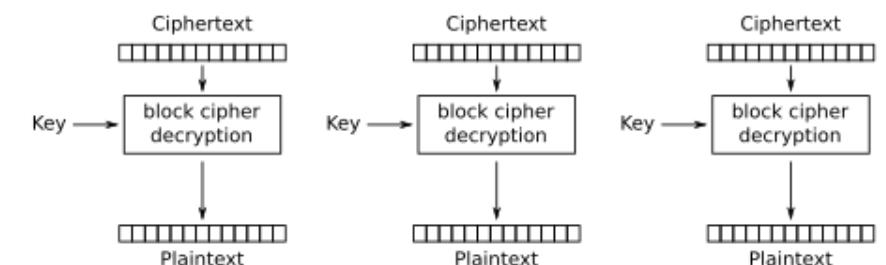


Encryption with Block Ciphers: Modes of Operation

- Message is divided into blocks
- Each block is encrypted *separately*
- Lack of diffusion



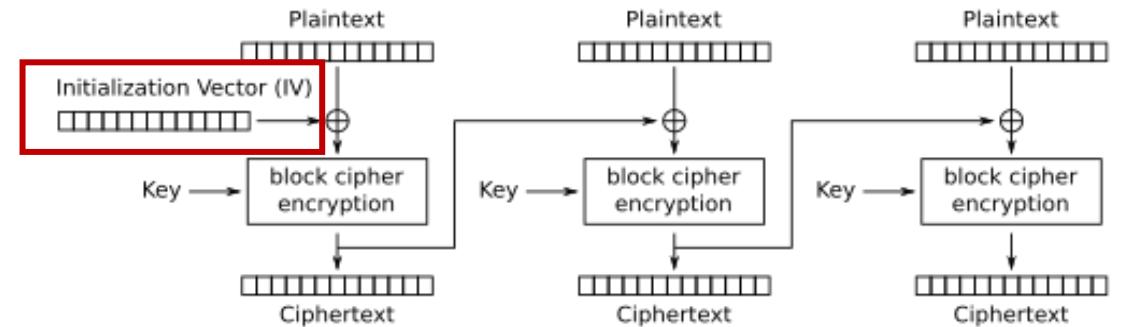
Electronic Codebook (ECB) mode encryption



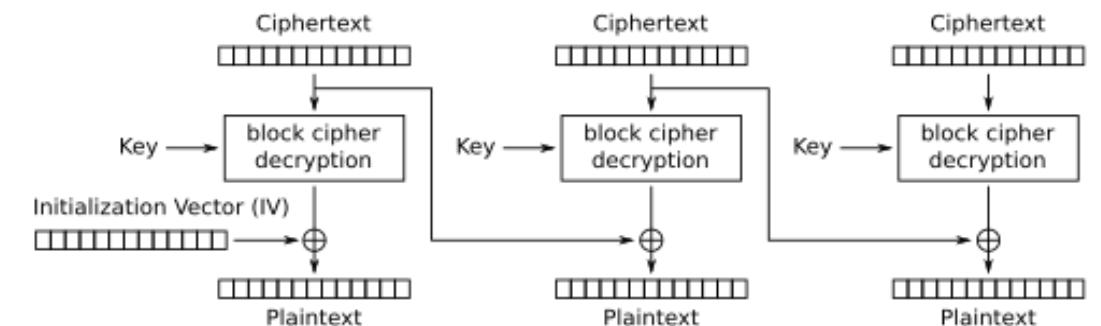
Electronic Codebook (ECB) mode decryption

Cipher Block Chaining Mode (CBC)

- There are two main ideas behind the CBC mode:
 - The encryption of all blocks are “chained together”
 - Ciphertext y_i depends not only on block x_i but on *all previous plaintext blocks* as well
- The encryption is randomized by using an **initialization vector (IV)**
 - \oplus means XOR
 - **Purpose:** Add randomness to encryption
 - Prevents identical plaintexts from producing same ciphertext
 - **Randomly** generated
 - Not secret



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

$$\text{Encryption (first block): } y_1 = e_k(x_1 \oplus \text{IV})$$

$$\text{Encryption (general block): } y_i = e_k(x_i \oplus y_{i-1}), \quad i \geq 2$$

$$\text{Decryption (first block): } x_1 = e_k^{-1}(y_1) \oplus \text{IV}$$

$$\text{Decryption (general block): } x_i = e_k^{-1}(y_i) \oplus y_{i-1}, \quad i \geq 2$$

CSC 3511 Security and Networking

Week 10, Lecture 2: Public Key Cryptography, RSA

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	<ul style="list-style-type: none">• One-time pads• Stream ciphers• Block ciphers	<ul style="list-style-type: none">• RSA encryption
Integrity, Authentication	<ul style="list-style-type: none">• MACs (e.g., HMAC)	<ul style="list-style-type: none">• Digital signatures (e.g., RSA signatures)

Roadmap

- *Public-Key Cryptography*
- *RSA Encryption*
- *Hybrid Encryption*

Public Key Cryptography

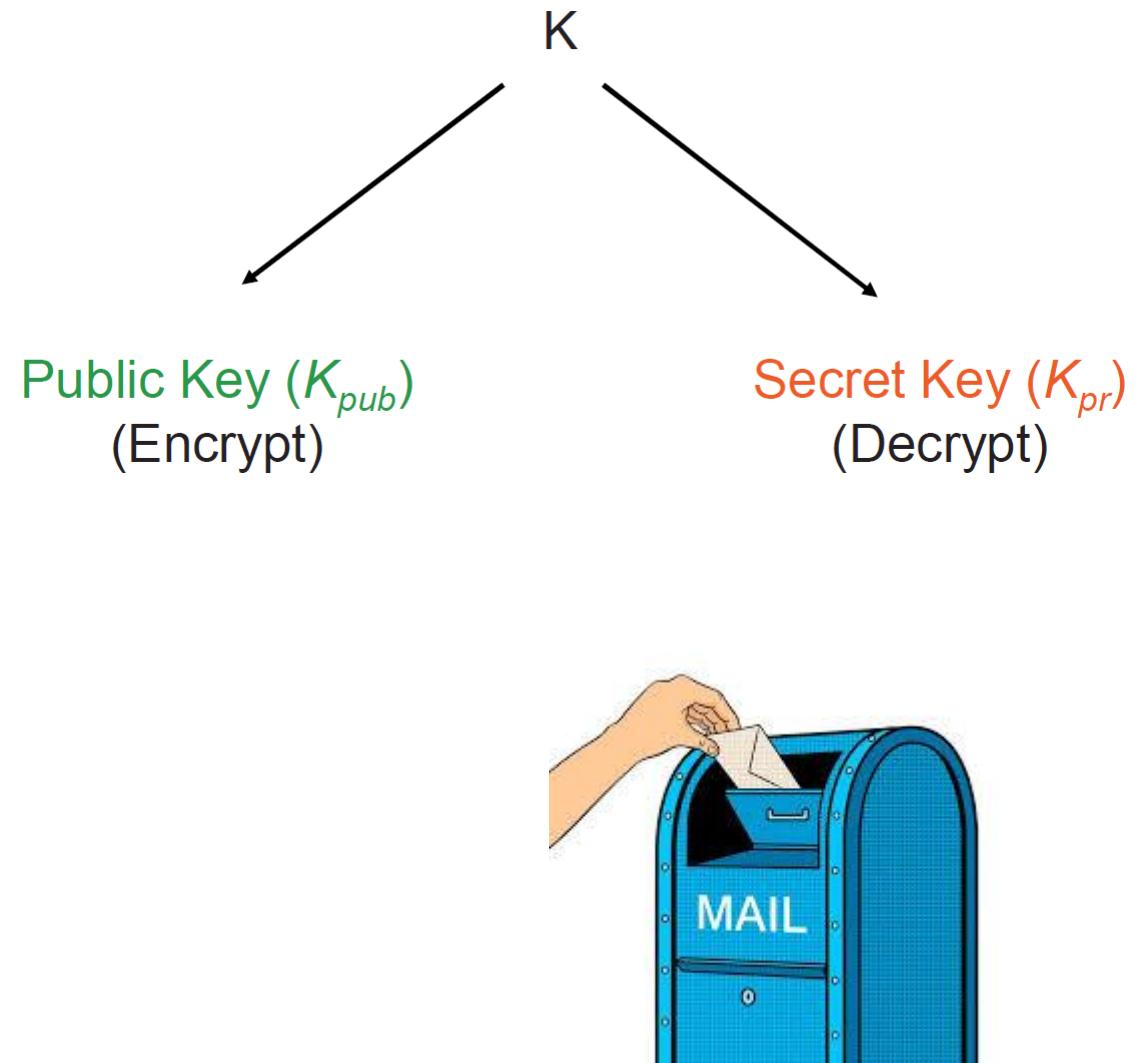
- In public-key schemes, each person has two keys
 - *KeyGen()*: output a pair of public key and private key; every public key corresponds to one private key
 - *Public key*: available to anyone
 - *Private key*: known only to individual
- Use cases
 - **Confidentiality:**
 - Encrypt using public key, decrypt using private key
 - *Key management problem: Share symmetric key K*
 - **Integrity/Authentication:**
 - Encrypt using private key, decrypt using public one
 - Digital Signature

Public-Key Encryption: Definition

- Three parts:
 - $\text{KeyGen}() \rightarrow PK, SK$: Generate a public/private keypair, where PK is the public key, and SK is the private (secret) key
 - $\text{Enc}(PK, M) \rightarrow C$: Encrypt a plaintext M using public key PK to produce ciphertext C
 - $\text{Dec}(SK, C) \rightarrow M$: Decrypt a ciphertext C using secret key SK
- Properties
 - **Correctness**: Decrypting a ciphertext should result in the message that was originally encrypted $\rightarrow \text{Dec}(SK, \text{Enc}(PK, M)) = M$ for all $PK, SK \leftarrow \text{KeyGen}()$ and M
 - **Efficiency**: Encryption/decryption should be fast
 - **Security**:
 - Computationally infeasible to derive the private key from the public key
 - Computationally infeasible to determine the private key from a chosen plaintext attack

Idea Behind Public-Key Encryption

- “Split up” the key: during the key generation phase, a key pair: public key K_{pub} and private key K_{pr} is computed
- Analogy for public-key encryption: a mailbox
 - Everyone can drop a letter
 - Only the owner has the correct key to open the mailbox



Public Key Encryption: Constructions

- Asymmetric schemes are based on a “one-way function” $f()$:
 - Computing $y = f(x)$ is computationally easy
 - Computing $x = f^{-1}(y)$ is computationally infeasible
- Constructions generally rely on mathematically hard problems:
 - **Factoring integers (RSA, ...)**: Given a composite integer n , find its **prime factors** is hard (Multiply two primes is easy)
 - **Discrete Logarithm** (Diffie-Hellman, ElGamal, DSA, ...): Given a , y and m , find x such that $a^x = y \text{ mod } m$ (Exponentiation a^x : easy)
 - **Elliptic Curves (EC)** (ECDH, ECDSA): Generalization of discrete logarithm

Note: The problems are considered mathematically hard, but **no proof exists** (so far)

Roadmap

- *Public-Key Cryptography*
- ***RSA Encryption***
- *Hybrid Encryption*

RSA Encryption: Key Generation

KeyGen():

- Randomly pick **two large primes, p and q** (> 2048 bits in total)
 - Done by picking random numbers and then using a test to see if the number is (probably) prime
- Compute $N = pq$
 - N is usually between 2048 bits and 4096 bits long
- Choose public exponent e
 - Requirement: e is **relatively prime** to $\phi(N) = (p - 1)(q - 1)$
 - Requirement: $2 < e < (p - 1)(q - 1)$
 - A common choice for e is 65537, a prime number that is efficient for encryption and decryption
- Compute private exponent $d = e^{-1} \bmod (p - 1)(q - 1)$
 - d is the modular inverse of $e \bmod (p - 1)(q - 1)$: $ed \equiv 1 \bmod (p - 1)(q - 1)$
 - Algorithm: Extended Euclid's algorithm
- **Public key:** N and e
- **Private key:** d

RSA Encryption: Encryption and Decryption

$\text{Enc}(e, N, M)$:

- Output $M^e \bmod N$

$\text{Dec}(d, C)$:

- Output $C^d = (M^e)^d \bmod N$

RSA encryption: Correctness

1. Theorem: $M^{ed} \equiv M \pmod{N}$
2. Euler's theorem: for all positive coprime-with- N a , $a^{\phi(N)} \equiv 1 \pmod{N}$
 - a. Coprime: greatest common divisor (GCD) of a and N is 1
 - b. $\phi(N)$ is the totient function of N
 - c. If N is a product of two distinct prime numbers p and q , then: $\phi(N) = (p - 1)(q - 1)$
 - d. If N is prime, $\phi(N) = N - 1$ (Fermat's little theorem)

Notice: $ed \equiv 1 \pmod{(p-1)(q-1)}$ so $ed \equiv 1 \pmod{\phi(N)}$

This means that $ed = k\phi(n) + 1$ for some integer k

Theorem in 1. can be written as $M^{k\phi(N)+1} \equiv M \pmod{N}$

- $M^{k\phi(N)}M \equiv M \pmod{N} \rightarrow$ (Assume M is coprime to N) The first half can be rewritten as:
 $(M^{\phi(N)})^k \equiv 1^k \equiv 1 \pmod{N}$
- $1M \equiv M \pmod{N}$ by Euler's theorem $M \equiv M \pmod{N}$

RSA Encryption: Security

- **RSA problem:** Given large $N = pq$ and $C = M^e \text{ mod } N$, it is hard to find M
 - Given the public key (N, e) and the ciphertext C , it is computationally infeasible to reverse-engineer the message M without knowing the private key d
 - No harder than the factoring problem (if you can factor N , you can recover p and q , then use $\phi(N)$ and e to derive d)
- Current best solution is to factor N , but unknown whether there is an easier way
 - Factoring problem is assumed to be hard (if you don't have a massive quantum computer, that is)

RSA Encryption: Issues

- Is RSA encryption described above secure?
 - No (*Textbook RSA*). It's deterministic: No randomness was used at any point!
 - The same plaintext always produces the same ciphertext under a given public key
- Sending the same message encrypted with different public keys also leaks information
 - Small m and e leaks information → e is usually small (~16 bits) and often constant (3, 17, 65537)
- Therefore, we need a probabilistic padding scheme (e.g., OAEP padding scheme)

Roadmap

- *Public-Key Cryptography*
- *RSA Encryption*
- ***Hybrid Encryption***

Hybrid Encryption

- Disadvantages of public-key encryption: Much slower than symmetric-key encryption (~1000-10,000x Difference)
 - AES-128 (CBC mode): ~150 MB/s (without hardware acceleration)
 - ChaCha20-Poly1305: 400-720 MB/s (software implementation)
 - RSA-2048 can encrypt a maximum of ~250 bytes per operation; ~10,000-23,000 public key operations (encryption/verification) per second → 0.25~0.6 MB/s
- Advantages of public-key cryptography:
 - No requirement for a secured channel
- ***Hybrid encryption***: Randomly generate a key K , encrypt message under key K using symmetric encryption, and encrypt K using public-key encryption
 - Now we can encrypt large amounts of data quickly using symmetric encryption, and we still have the benefits of public-key encryption
- Almost all cryptographic systems use the idea of hybrid encryption (more or less)

Establishing a Shared Secret

Alice

$(pk_A, sk_A) \leftarrow KeyGen()$

Bob

“Alice”, pk_A

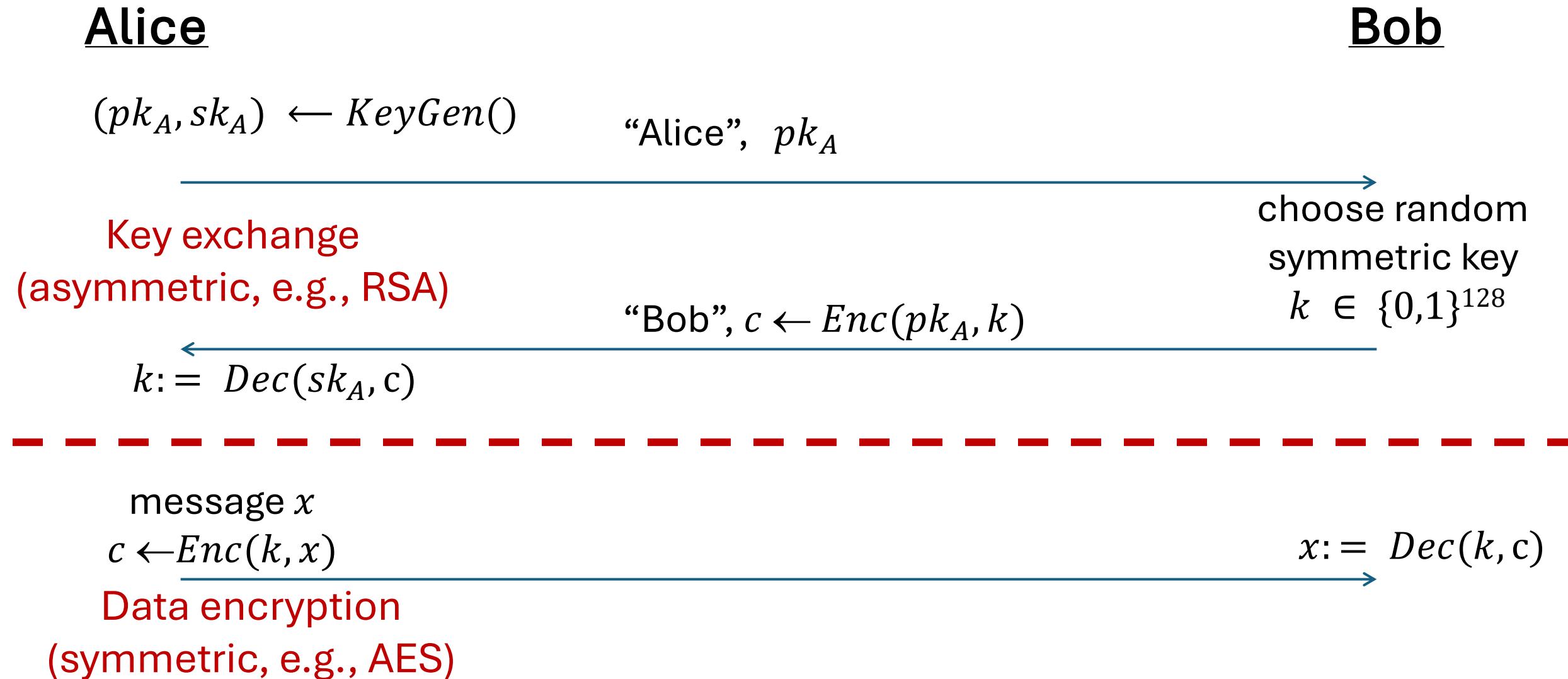
choose random
 $x \in \{0,1\}^{128}$

“Bob”, $c \leftarrow Enc(pk_A, x)$

$x := Dec(sk_A, c)$

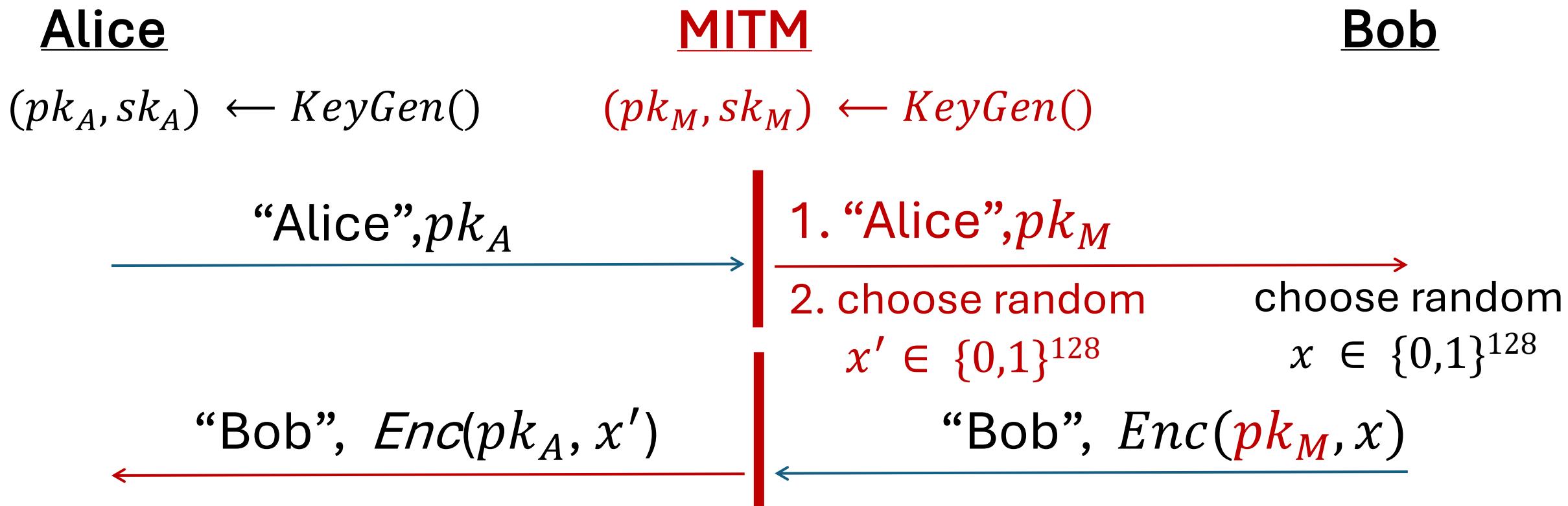
Strawman Key Exchange and Data Transfer Protocol

Example: Hybrid protocol with AES as the symmetric cipher



Insecure Against MITM

The above protocol is insecure against **MITM** attacks



CSC 3511 Security and Networking

Week 11, Lecture 1: Digital Signatures, Certificates, and PKI

Roadmap

- *Digital Signatures and RSA Signature*
- *Certificates and Chain of Trust*
- *Public Key Infrastructure*

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	<ul style="list-style-type: none">● One-time pads● Block ciphers with chaining modes (e.g. AES-CBC)● Stream ciphers	<ul style="list-style-type: none">● RSA encryption
Integrity, Authentication	<ul style="list-style-type: none">● MACs (e.g. HMAC)	<ul style="list-style-type: none">● Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)
- Key management (certificates)

Last Week: Strawman Key Exchange and Data Transfer Protocol

Alice

$(pk_A, sk_A) \leftarrow KeyGen()$

“Alice”, pk_A



Key exchange

(asymmetric, e.g., RSA)

choose random
symmetric key
 $k \in \{0,1\}^{128}$

“Bob”, $c \leftarrow Enc(pk_A, k)$

$k := Dec(sk_A, c)$



message x

$c \leftarrow Enc(k, x)$

Data encryption

(symmetric, e.g., AES)

$x := Dec(k, c)$

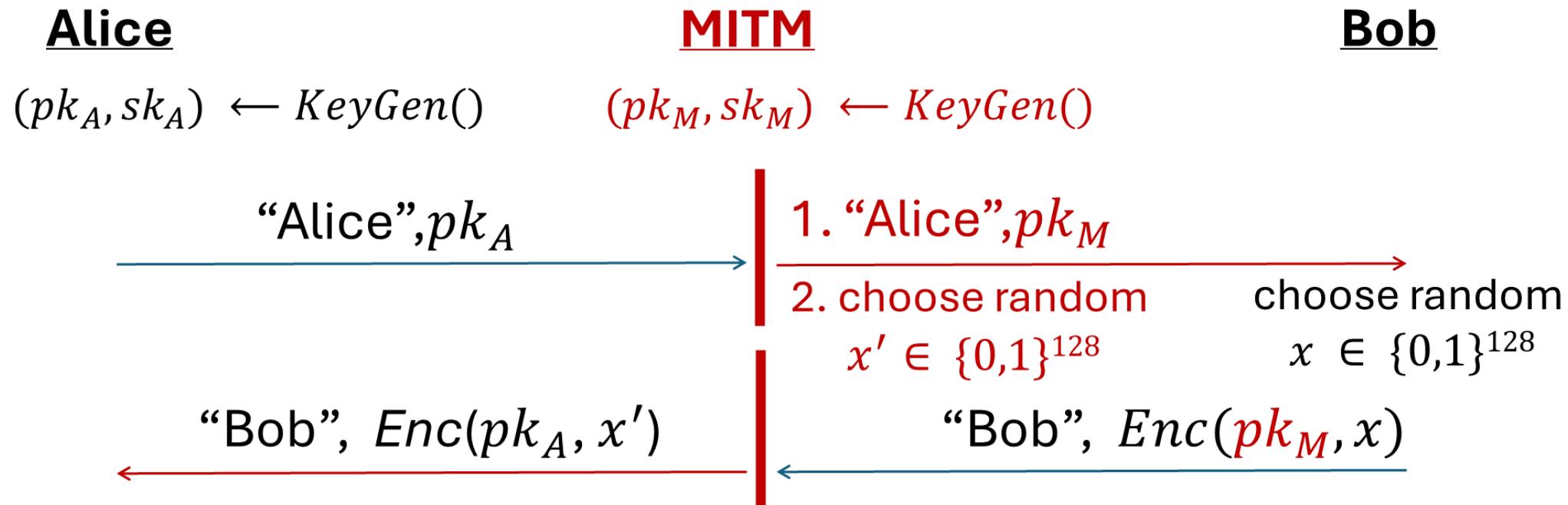


Last Week: MITM Targeting the Strawman Key Exchange and Data Transfer Protocol

The above protocol is insecure against **MITM** attacks

Root cause: Bob has no way to tell the message is from Alice or Mallory

Solution: Find a way to verify the integrity & authenticity of the received information



Digital Signatures

- Digital signatures are the *asymmetric way* of providing **integrity** and **authenticity** to data
- Assume that Alice and Bob can communicate public keys without Mallory changing them
 - We will see how to fix this limitation later using certificates
- Only the owner of the private key can **sign messages** with the private key
- Everybody can **verify the signature** with the public key



Digital Signatures: Definition

- Three parts:
 - $\text{KeyGen}() \rightarrow PK, SK$: Generate a public/private keypair, where PK is the verify (public) key, and SK is the signing (secret) key
 - $\text{Sign}(SK, M) \rightarrow sig$: Sign the message M using the signing key SK to produce the signature sig
 - $\text{Verify}(PK, M, sig) \rightarrow \{0, 1\}$: Verify the signature sig on message M using the verify key PK and output 1 if valid and 0 if invalid
- Properties
 - **Correctness**: Verification should be successful for a signature generated over any message $\rightarrow \text{Verify}(PK, M, \text{Sign}(SK, M)) = 1$ for all $PK, SK \leftarrow \text{KeyGen}()$ and M
 - **Efficiency**: Signing/verifying should be fast
 - **Security**: EU-CPA (existentially unforgeable under chosen plaintext attack)

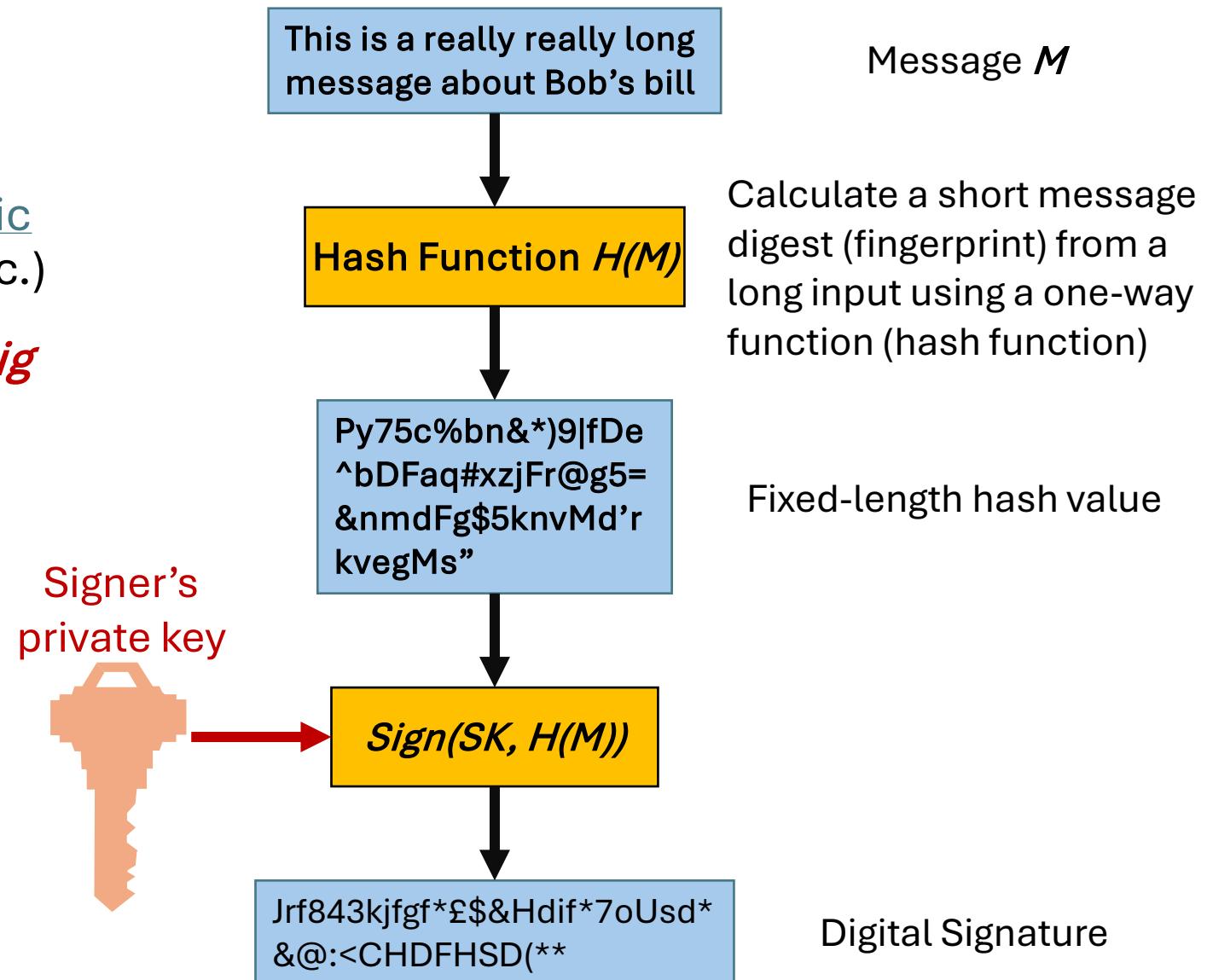
Digital Signatures in Practice: Sign

If you want to **sign** message M:

- First hash message M using cryptographic hash function $H()$ (e.g., SHA-1, SHA-2, etc.)
- Then sign (Enc) $H(M)$: $Sign(SK, H(M)) \rightarrow sig$
- Send M and sig to the receiver

Why do digital signatures use a hash?

- Asymmetric encryption/decryption is slow! → “*Encrypt*” a short message
- Allows signing arbitrarily long messages

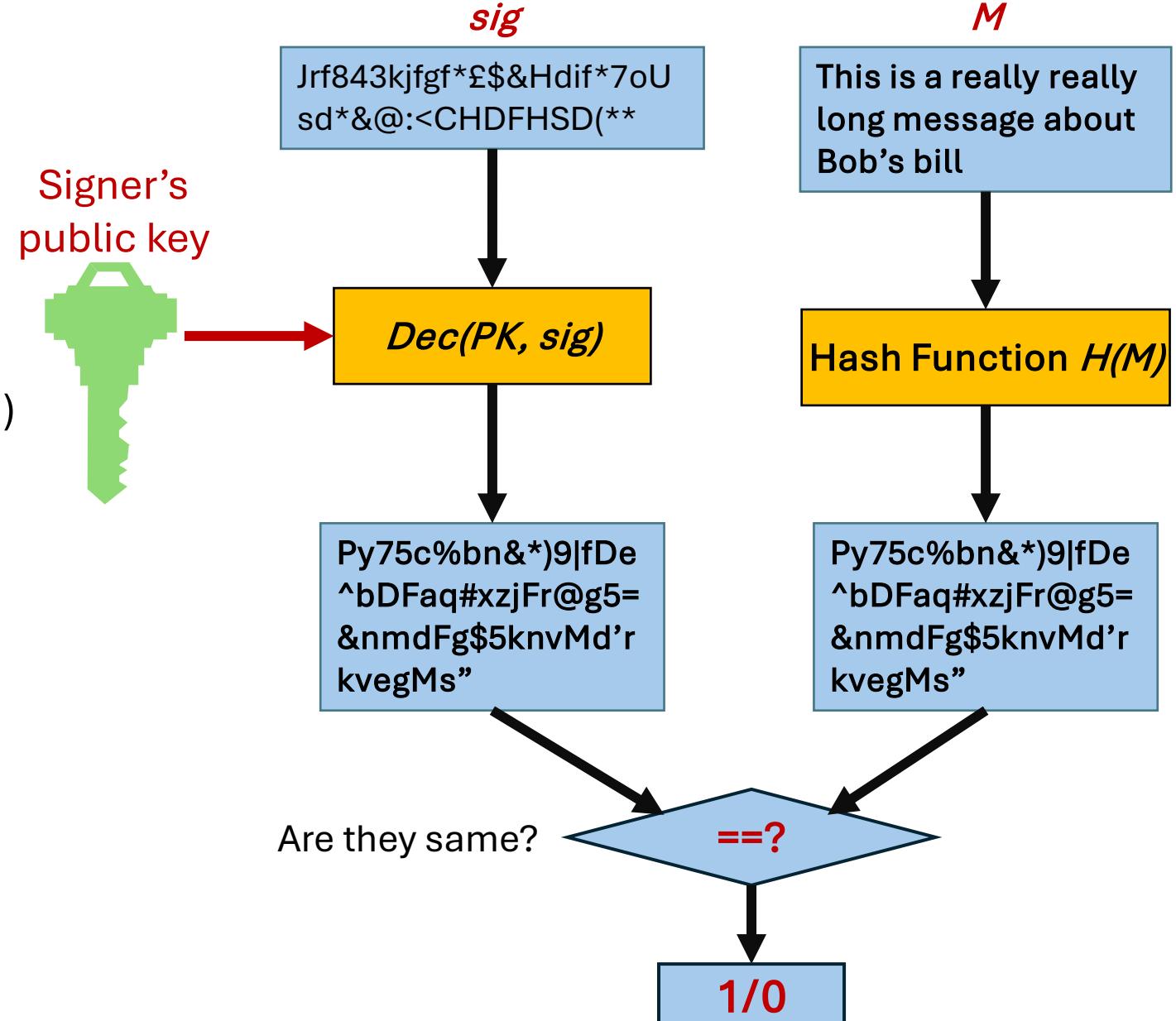


Digital Signatures in Practice: Verify

- If you want to **verify** the signature of message M:
 - Decrypt ***sig*** using signer's public key
 - Hash **M** using the same cryptographic hash function (e.g., SHA-1, SHA-2, etc.)
 - Compare results in step 1 and 2

Digital signatures provide **integrity** and **authenticity** for M

- The digital signature acts as proof that the private key holder signed $H(M)$, so you know that M is authentically endorsed by the private key holder



RSA Signatures: Definition

KeyGen():

- Same as RSA encryption:
 - **Public key:** N and e
 - **Private key:** d

Recall RSA encryption: $M^{ed} \equiv M \pmod{N}$

Sign(d, M):

- Compute: $\text{sig} = H(M)^d \pmod{N}$

Verify(e, N, M, sig):

- Verify that $H(M) \equiv \text{sig}^e \pmod{N}$
- Correctness: $\text{sig}^e \pmod{N} \equiv H(M)^{de} \pmod{N} \equiv H(M) \pmod{N}$
 - Because recall $x^{de} \pmod{N} \equiv x \pmod{N}$ for all x)

- Last week:

- $\text{Enc}(e, N, M) \rightarrow C$
- $\text{Dec}(d, C) \rightarrow M$

- There is nothing special about using e first or **using d first!**

- If we **encrypt using d** , then anyone can “decrypt” using e
- Given x and $x^d \pmod{N}$, can’t recover d because of discrete-log problem, so d is safe

Summary: Public-Key Cryptography

- Public-key cryptography: Two keys, private and public
- Public-key encryption: One key encrypts, the other decrypts
 - Security properties similar to symmetric encryption
 - *RSA Enc*: Produce a pair e and d such that $M^{ed} = M \bmod N$
- ***Hybrid encryption***: Encrypt a symmetric key, and use the symmetric key to encrypt the message
- ***Digital signatures***: Integrity and authenticity for asymmetric schemes
 - *RSA Signature*: Sign (Encrypt) the hash with the *private* key
 - *Other popular signature schemes*: DSA, ECDSA, EdDSA, BLS...

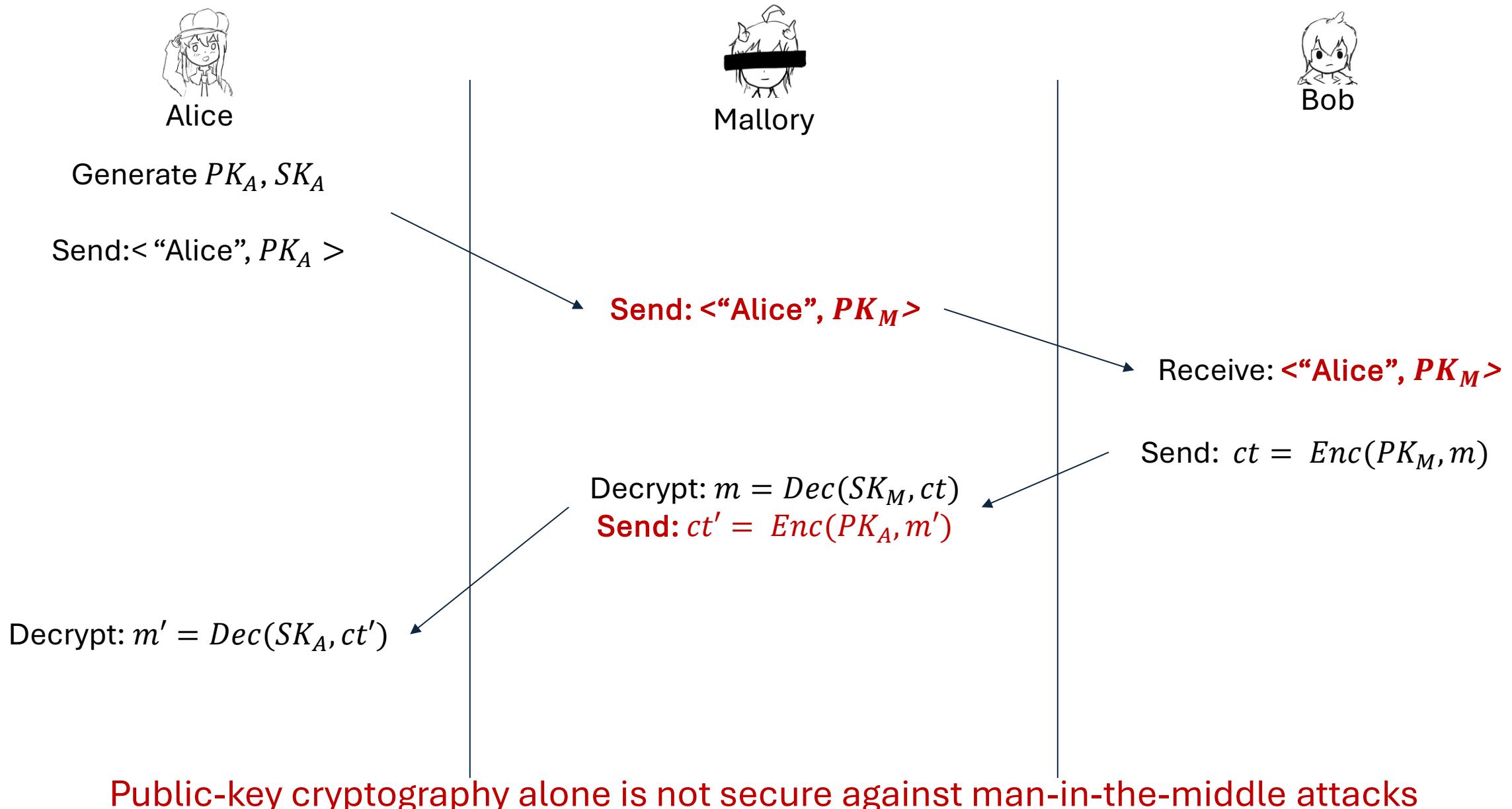
Roadmap

- *Digital Signatures and RSA Signature*
- ***Certificates and Chain of Trust***
- *Public Key Infrastructure*

Certificates

- Public-key cryptography is great! We can communicate securely without a shared secret
 - **Public-key encryption:** Everybody encrypts with the public key, but only the owner of the private key can decrypt
 - **Digital signatures:** Only the owner of the private key can sign, but everybody can verify with the public key
- How do we **know the public key** of an entity?
- How do we **distribute public keys** securely?

Problem: Distributing Public Keys



Problem: Distributing Public Keys

- *Root cause:* No mechanism to **verify** that a public key belongs to its claimed owner
- Idea: Sign Bob's public key to prevent tampering → ***Do no trust, verify!***
- Problem: **who should sign Bob's public key?**
 - If Bob signs his public key, we need his public key to verify the signature
 - But Bob's public key is what we were trying to verify in the first place!
 - *Chicken-and-egg problem:* Alice can never trust any public key she receives
- Solution: You cannot gain trust if you trust nothing. You need a **root of trust!**
 - ***Trust anchor:*** Someone that we implicitly trust (e.g., DNS root name server?)
 - From our trust anchor, we can begin to trust others

Certificates

- **Certificate:** A signed endorsement (a signature) of someone's *public key* and *identity/attributes*
 - A certificate contains at least two things: The **identity** (Alice, Bob, David, etc.) of the person, and the **key** (PK_A, PK_B, PK_D)
- Abbreviated notation
 - Encryption under Alice's public key PK : {"Message"} PK_A
 - Signing with Alice's private key SK : {"Message"} SK_A^{-1}
 - Recall: A signed message must **contain the message** along with the signature; you can't check the signature by itself!
- Scenario: Alice wants Bob's public key. Alice trusts EvanBot (PK_E, SK_E)
 - EvanBot is our *trust anchor (root of trust)*
 - If we trust PK_E , a certificate we would trust is {"Bob's public key is PK_B "} SK_E^{-1}

Certificates

Q1: How to use EvanBot?

- Suppose Alice wants Bob's key. Alice trusts EvanBot (PK_E , SK_E)
- Alice -> EvanBot: "What is Bob's public key?"
- EvanBot -> Alice:
 - *Message M:* "Bob's public key is PK_B "
 - Signature: $\{M\}_{SK_E^{-1}}$

Q2: How to build EvanBot?

- Idea 1: Make a central, trusted directory (a powerful server) from where you can fetch anybody's public key
- Idea 2: A hierarchical trust chain

Idea 1: A Central Trusted Directory

- Idea: Make a central, trusted directory (TD) from where you can fetch anybody's public key
 - The TD has a public/private keypair PK_{TD}, SK_{TD}
 - The directory publishes PK_{TD} so that everyone knows it (baked into computers, phones, OS, etc.)
 - When you request Bob's public key, the directory sends a certificate for Bob's public key
- What do we have to trust?
 - We have received TD's key correctly
 - TD won't sign a key without verifying the identity of the owner

Idea 1: A Central Trusted Directory

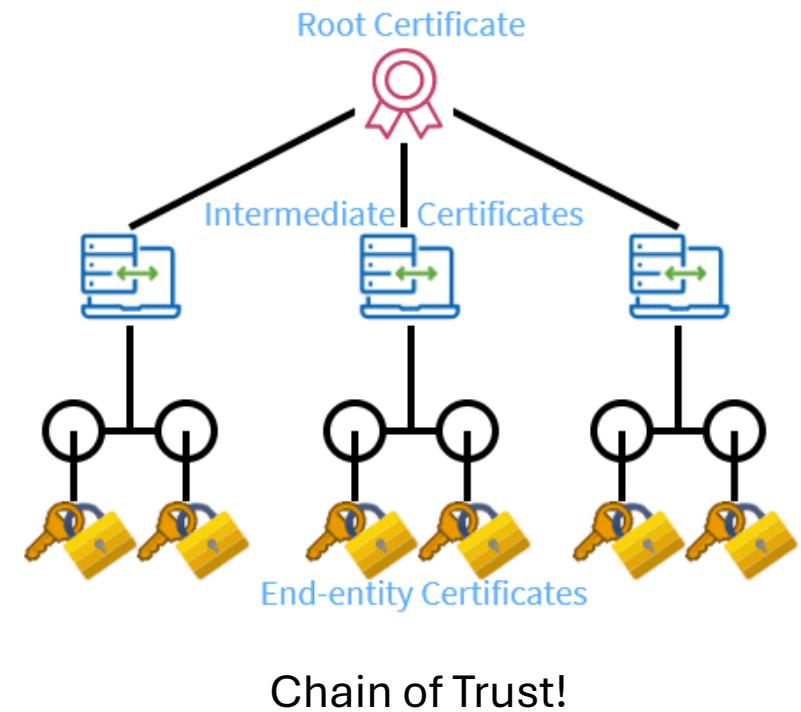
- Problem: Scalability
 - One directory won't have enough compute power to serve the entire world
- Problem: Single point of failure
 - If the directory fails, *services depending on this become unavailable*
 - If the directory is compromised, you can't trust anyone
 - If the directory is compromised, it is difficult to recover

Any ideas? → *Drawing inspiration from the DNS name server hierarchy*

Idea 2: A Hierarchical Trust Chain

Addressing scalability: Hierarchical trust

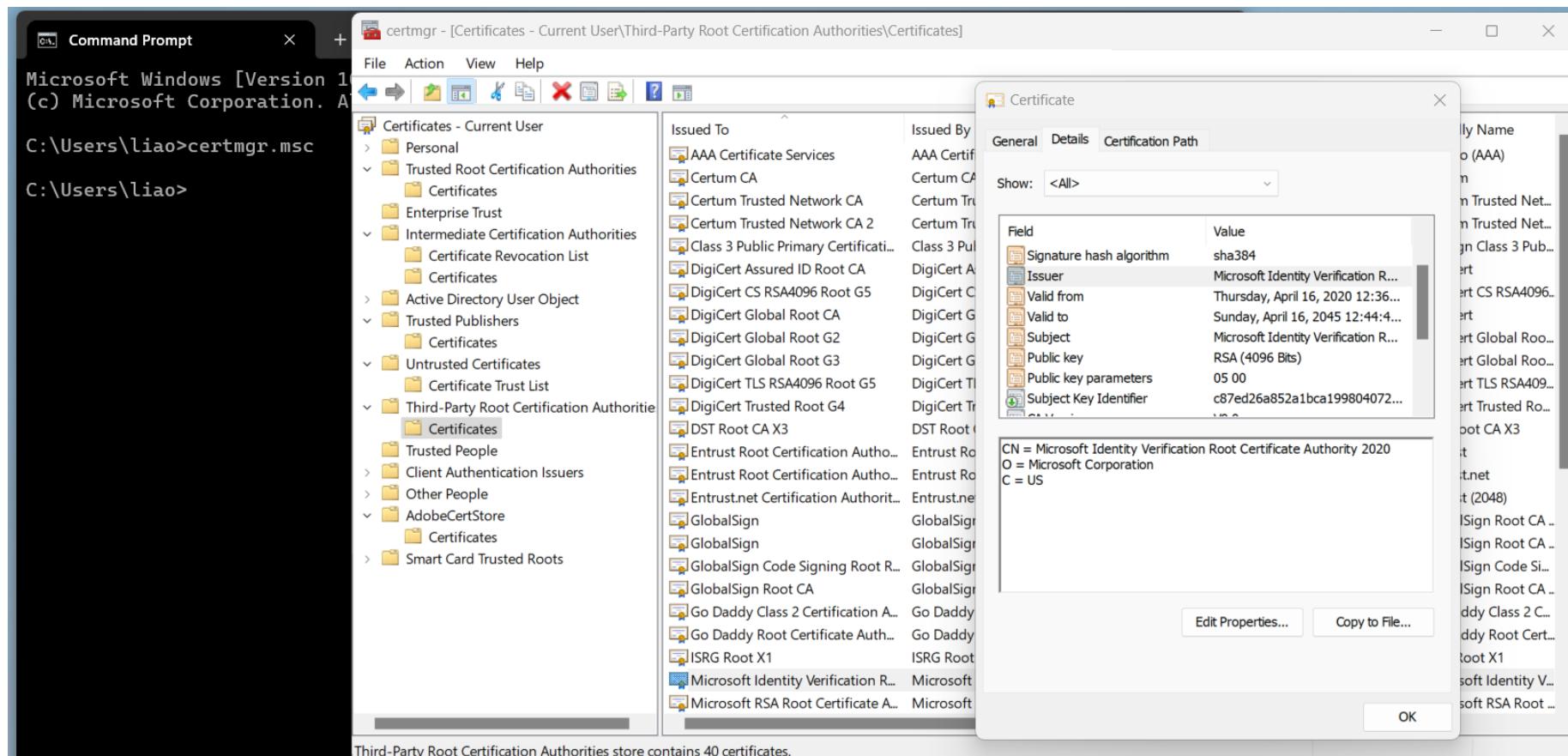
- The root of trust (trusted directory) may **delegate** trust and signing power to other authorities
 - {“John Walz’s public key is PK_{JW} , and I trust him to sign for MSOE”} SK_{TD}^{-1}
 - {“Chris Taylor’s public key is PK_{CT} , and I trust him to sign for the DSAC”} SK_{JW}^{-1}
 - {“Zhonghao Liao’s public key is PK_{ZL} (but I don’t trust him to sign for anyone else)”} SK_{CT}^{-1}
 - {“Sean Jones’s public key is PK_{SJ} (but I don’t trust him to sign for anyone else)”} SK_{CT}^{-1}
 - TD is still the root of trust (**root certificate authority**, or **root CA**)
 - JW and CT receive delegated trust (**intermediate CAs**)
 - ZL’s **identity and public key** can be trusted
 - SJ’s **identity and public key** can be trusted



Idea 2: A Hierarchical Trust Chain

Addressing scalability/Single point of failure: Multiple trust anchors

- There are ~150 root CAs who are implicitly trusted by most devices
- Public keys are **hard-coded** into operating systems and devices
- Each delegation step can restrict the **scope** of a certificate's validity
- Browse certificates on your machine:
 - Open cmd and type “certmgr.msc”



Browse Digital Certificates

Let's use [OpenSSL](#) on the SEED virtual machine to obtain Google's certificate chain

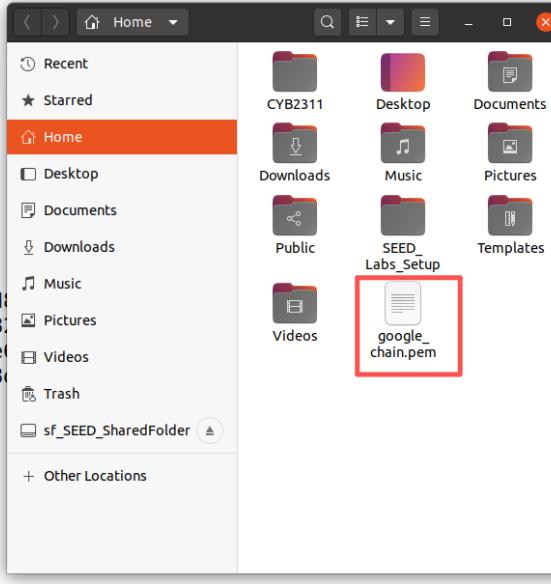
Get the entire certificate chain and save them to the file google_chain.pem

```
$ echo | openssl s_client -showcerts -connect google.com:443 2>/dev/null > google_chain.pem
```

Use the following command decode this file and print the first certificate out

```
$ openssl x509 -in google_chain.pem -text -noout
```

```
seed@VM: ~
[11/09/25]seed@VM:~$ echo | openssl s_client -showcerts -connect google.com:443 2>/dev/null > google_chain.pem
[11/09/25]seed@VM:~$ openssl x509 -in google_chain.pem -text -noout
Certificate:
Data:
Version: 3 (0x2)
Serial Number:
    d5:57:08:79:0a:56:e4:fb:12:ba:6a:0b:81:3c:a3:e0
Signature Algorithm: ecdsa-with-SHA256
Issuer: C = US, O = Google Trust Services, CN = WE2
Validity
    Not Before: Oct 13 08:37:46 2025 GMT
    Not After : Jan  5 08:37:45 2026 GMT
Subject: CN = *.google.com
Subject Public Key Info:
    Public Key Algorithm: id-ecPublicKey
        Public-Key: (256 bit)
            pub:
                04:d6:5d:b6:0a:96:d0:cb:06:9a:3e:7c:90:d1
                a3:40:bd:ef:92:45:79:db:fd:bc:b1:43:be:83
                21:c4:49:24:0f:99:26:84:af:2c:ec:9f:fd:ef
                04:ec:e9:23:8f:ea:22:98:47:a8:9b:e6:55:8d
                35:35:93:0d:20
            ASN1 OID: prime256v1
            NIST CURVE: P-256
X509v3 extensions:
    X509v3 Key Usage: critical
        Digital Signature
    X509v3 Extended Key Usage:
        TLS Web Server Authentication
```



***.google.com**
Identity: *.google.com
Verified by: WE2
Expires: 01/05/2026
WE2
Identity: WE2
Verified by: GTS Root R4
Expires: 02/20/2029
GTS Root R4
Identity: GTS Root R4
Verified by: GlobalSign Root CA
Expires: 01/28/2028

Subject Name
C(Country): US
O(Organization): Google Trust Services LLC
CN(Common Name): GTS Root R4
Issuer Name
C(Country): BE
O(Organization): GlobalSign nv-sa
OU(Organizational Unit): Root CA
CN(Common Name): GlobalSign Root CA

Certificate Validation Process

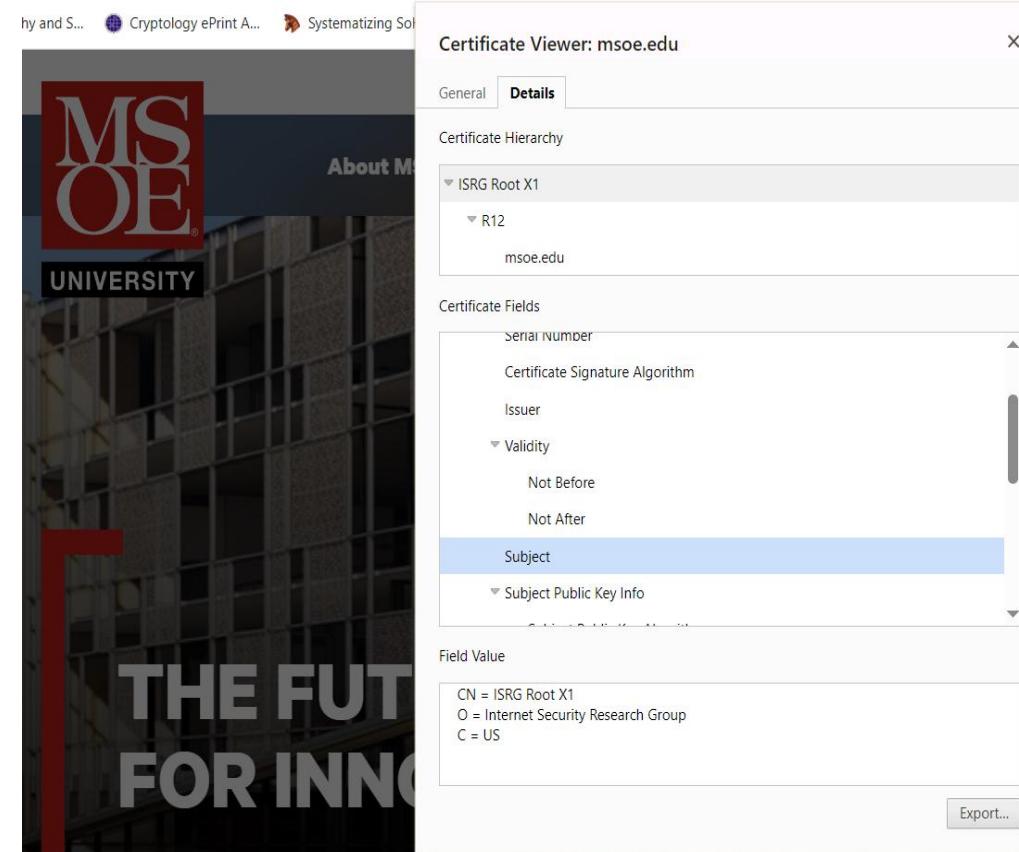
When you connects to <https://www.msoe.edu>:

Step 1: Receive Certificate Chain

- www.msoe.edu cert
 ↓ (signed by)
- R12 Let's Encrypt Server CA
 ↓ (signed by)
- ISRG Root CA

Step 2: Validate each certificate

- Check validity period: Not expired or not yet valid?
- Verify signature
- Check revocation: Is this certificate revoked?
[\(certificate revocation list \(CRL\)\)](#)
- Verify domain name: Does the certificate match the requested domain?
- Check trust chain: Can we trace back to a trusted root CA?



X509v3 Certificate Policies:
Policy: 2.23.140.1.2.1

X509v3 CRL Distribution Points:

Full Name:
URI: http://c.pki.goog/we2/xuzt3PU9F_w.crl

CT Precertificate SCTs:
Signed Certificate Timestamp:
Version : v1 (0x0)
Log ID : 16:83:2D:AB:F0:A9:25:0F:0F:F0:3A:A5
C8:23:D0:87:4B:F6:04:29:27:F8:E7:1F
Timestamp : Oct 13 09:37:48.838 2025 GMT

Roadmap

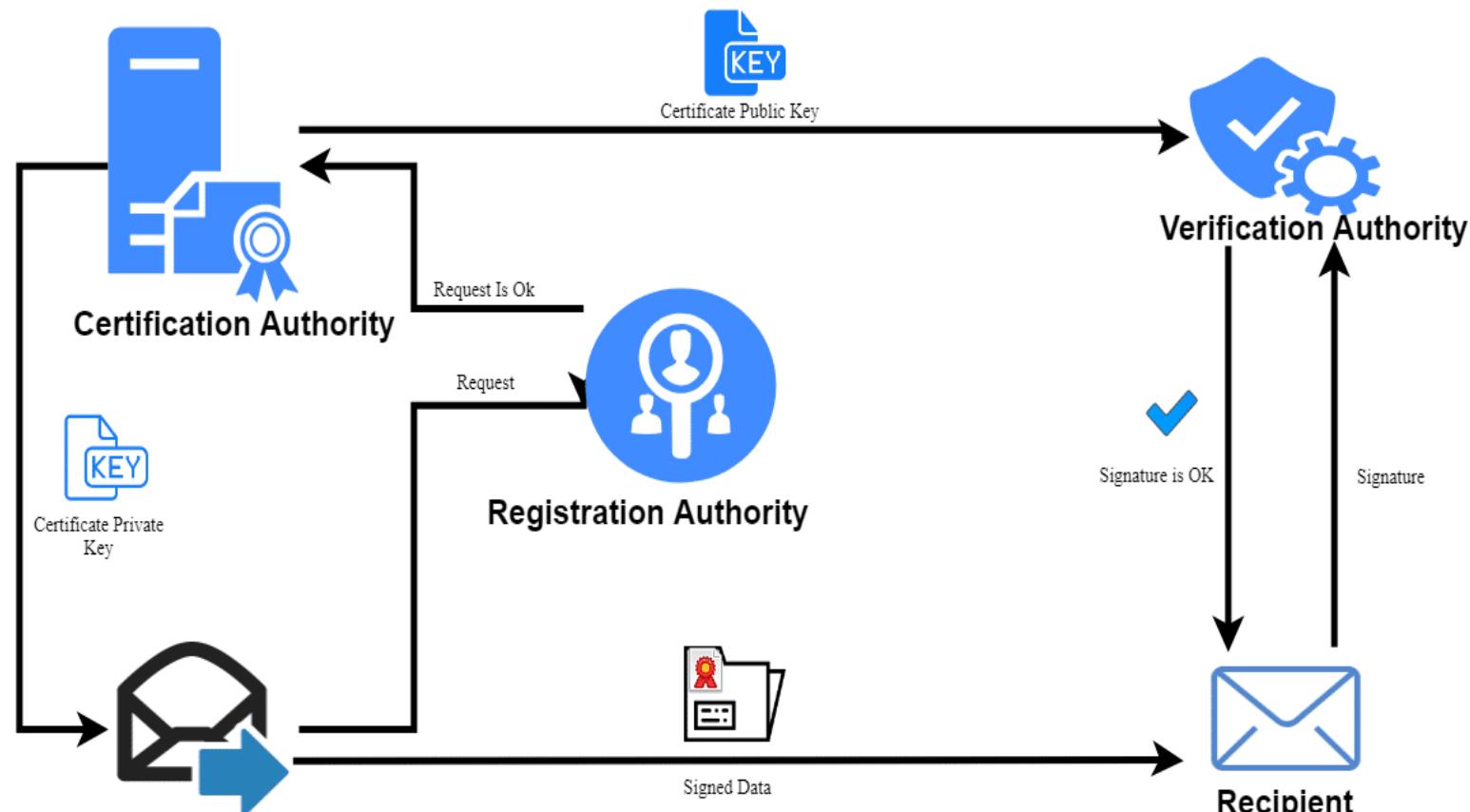
- *Digital Signatures and RSA Signature*
- *Certificates and Chain of Trust*
- ***Public Key Infrastructure***

What is PKI?

A comprehensive framework for managing digital certificates and public-key cryptography system

Core Functions:

- **Create** cryptographic keys
- **Distribute** public keys securely
- **Verify** identities
- **Revoke** compromised certificates
- **Store** certificates and keys
- **Archive** historical records

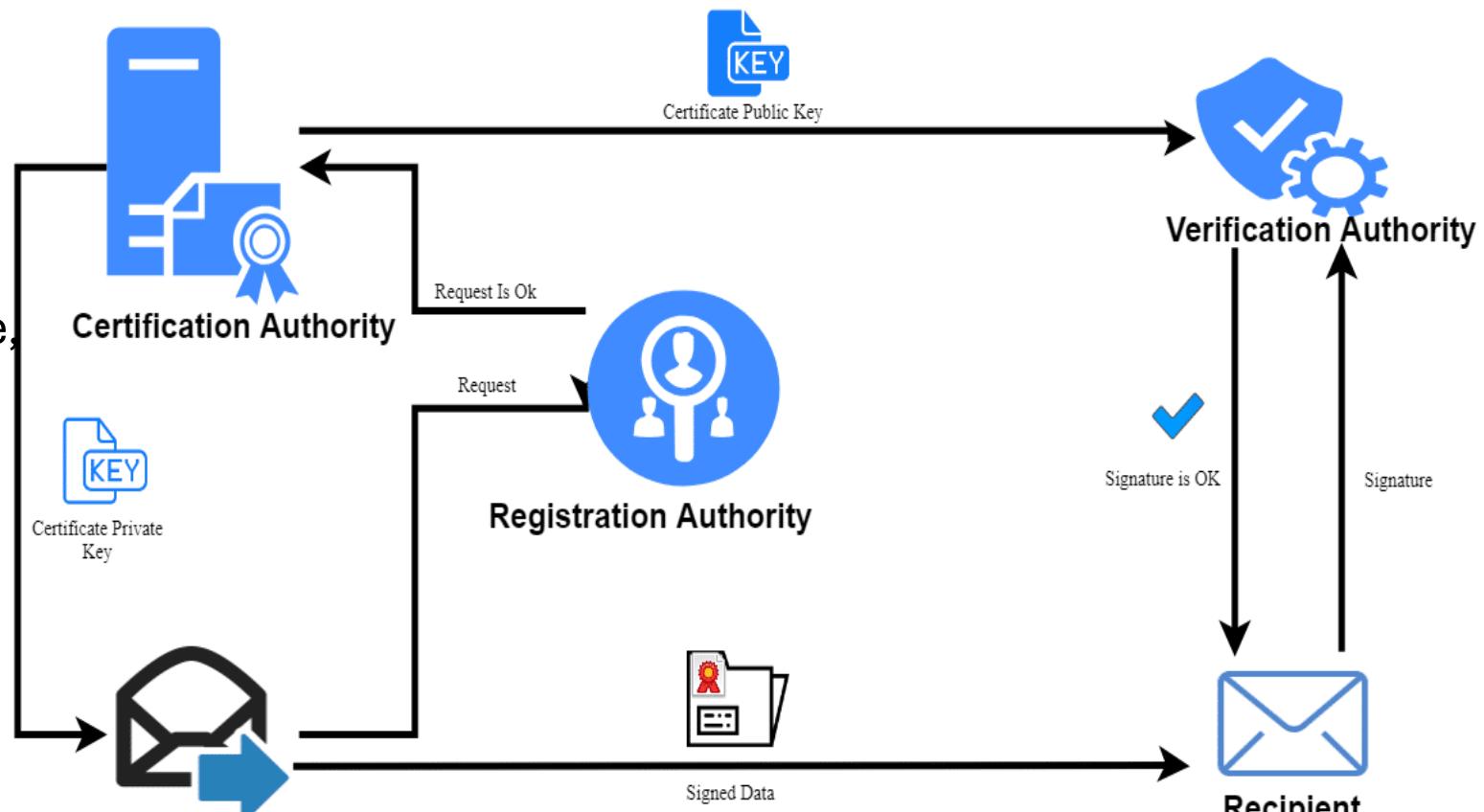


What is PKI?

A comprehensive framework for managing digital certificates and public-key cryptography system

Certificate Authority (CA):

- The trusted issuer and signer of digital certificates
- Manages certificate lifecycle: issuance, renewal, and revocation
- Types: Root CA (self-signed, top-level) and Intermediate CA (delegated authority)
- Examples: [DigiCert](#), [Let's Encrypt](#), [GlobalSign](#)

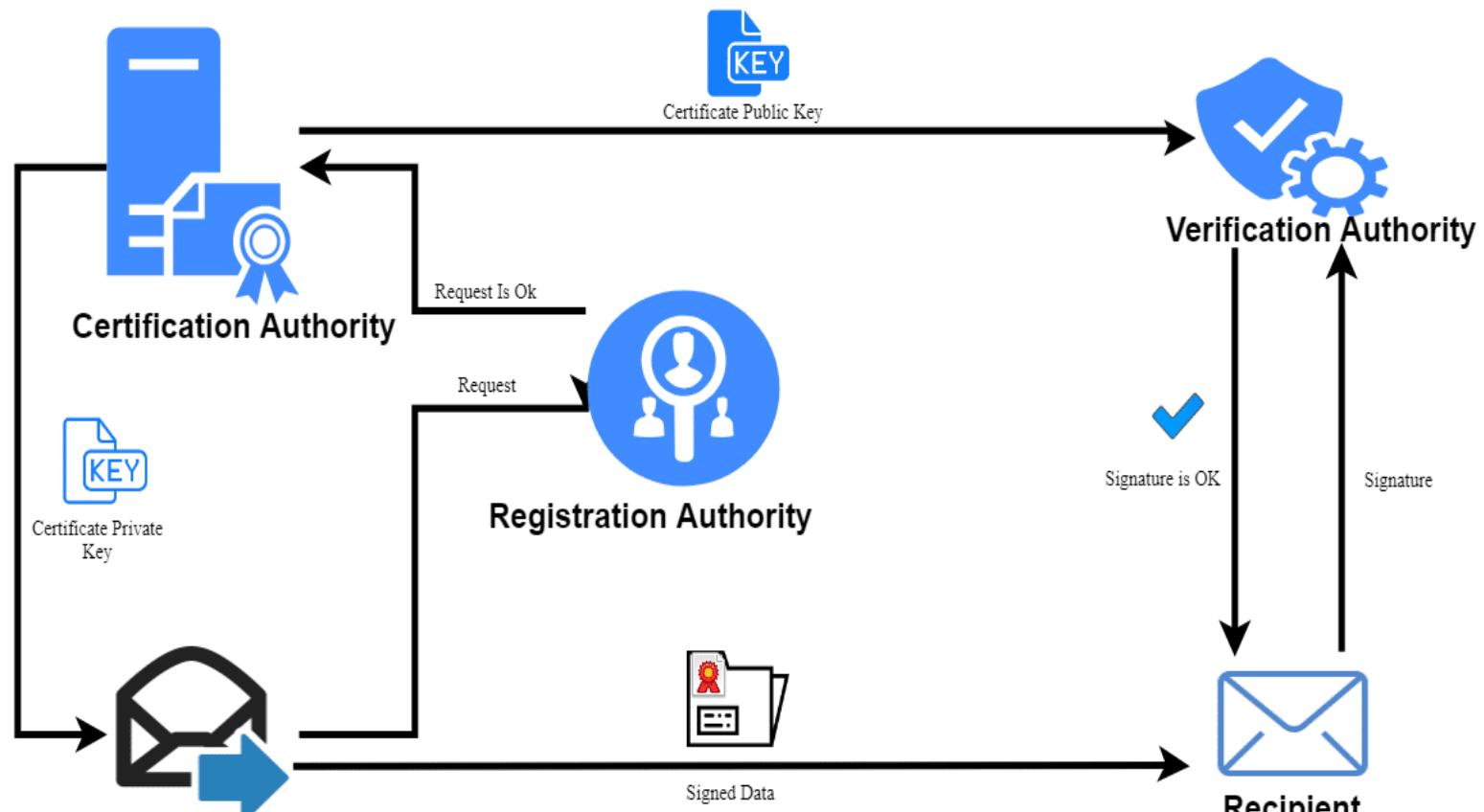


What is PKI?

A comprehensive framework for managing digital certificates and public-key cryptography system

Registration Authority (RA):

- Validates Certificate Signing Requests (CSRs) for authenticity
- Authorizes or rejects certificate requests based on identity verification
- Usually, a department or an automated system within the CA company. E.g., [Let's Encrypt ACME protocol API](#)

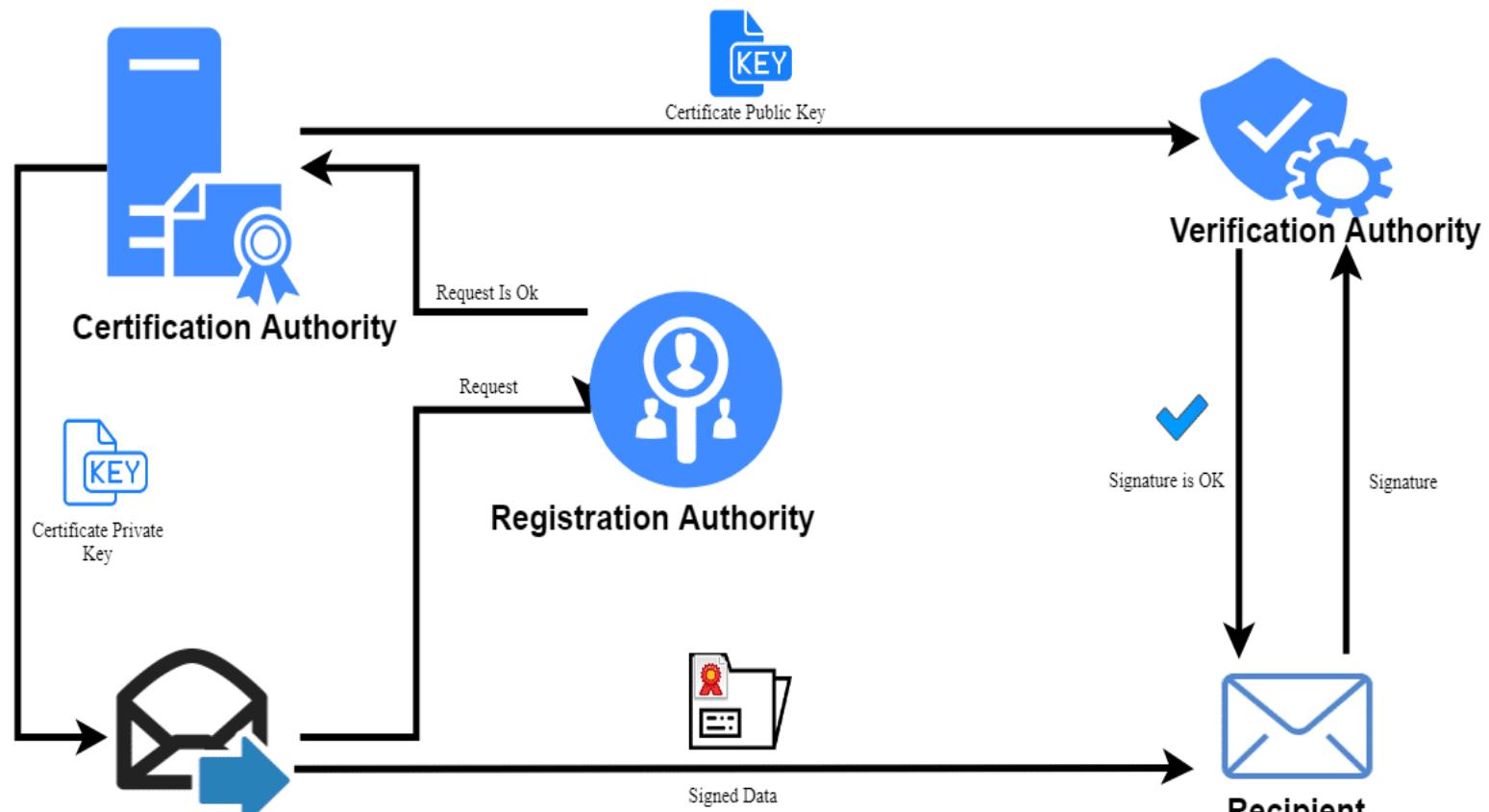


What is PKI?

A comprehensive framework for managing digital certificates and public-key cryptography system

Certificate Repository:

- Stores issued certificates and Certificate Revocation Lists (CRLs)
- Implemented as web-based databases: When you visit HTTPS sites, server sends certificates
- Try this page: <https://crt.sh/>
 - A public certificate repository anyone can search!
 - Search "msoe.edu": see every certificate ever issued!

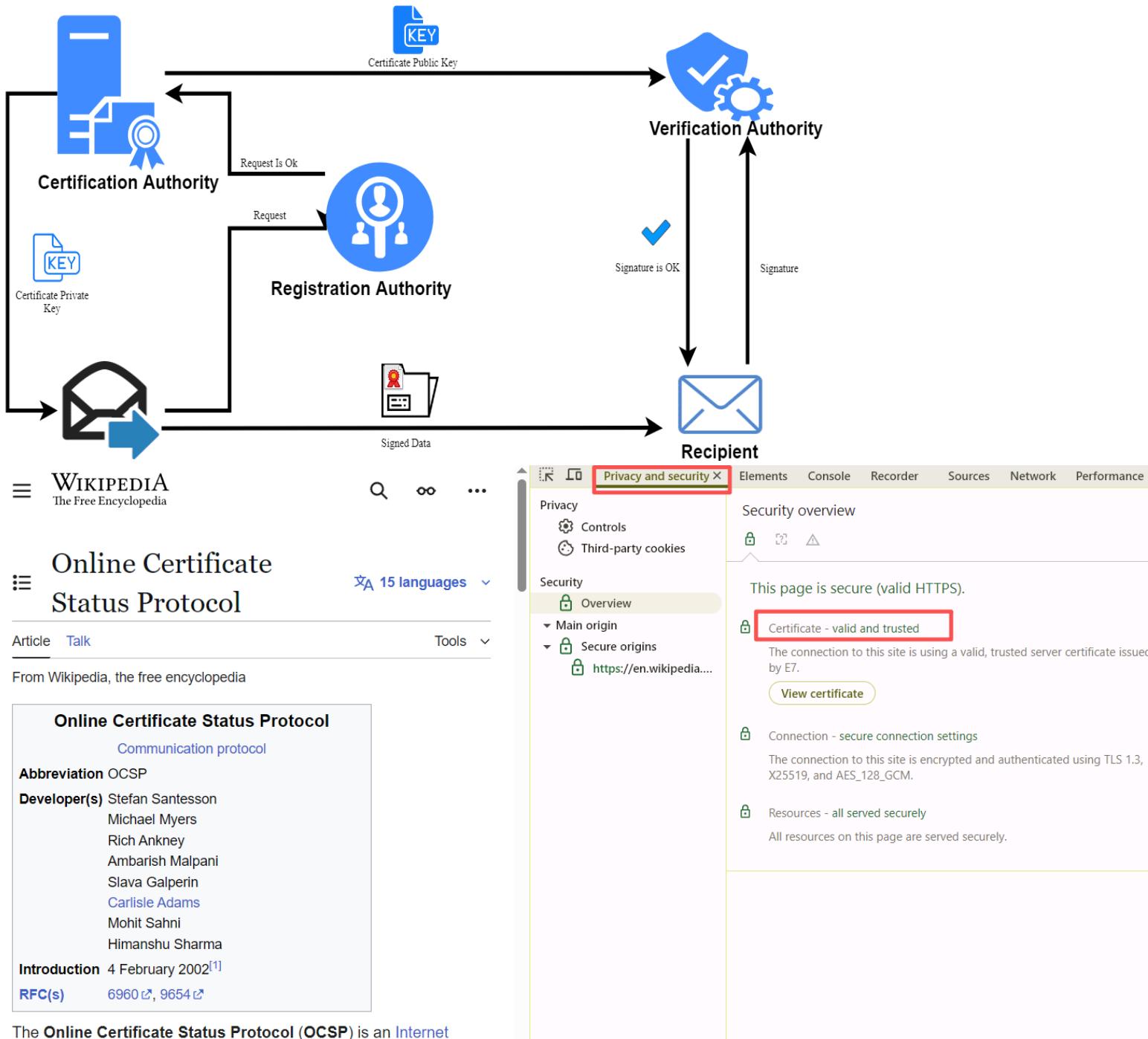


<https://doubleoctopus.com/security-wiki/digital-certificates/public-key-infrastructure/>

What is PKI?

Verification Authority (VA):

- Provides online certificate status checking via [OCSP protocol](#)
- Responds with certificate status: Valid, Revoked, or Unknown
- More efficient than downloading large CRLs
- Used by browsers and applications for real-time verification (see in DevTools)
- Note: Optional component (some PKIs use only CRLs)



CSC 3511 Security and Networking

Week 11, Lecture 2: Cryptographic Hash, MACs, and Diffie-Hellman Key Exchange

Roadmap

- ***Cryptographic Hash***
- ***Message Authentication Codes (MACs)***
- ***Diffie-Hellman Key Exchange***

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	<ul style="list-style-type: none">● One-time pads● Block ciphers with chaining modes (e.g. AES-CBC)● Stream ciphers	<ul style="list-style-type: none">● RSA encryption
Integrity, Authentication	<ul style="list-style-type: none">● MACs (e.g. HMAC)	<ul style="list-style-type: none">● Digital signatures (e.g. RSA signatures)

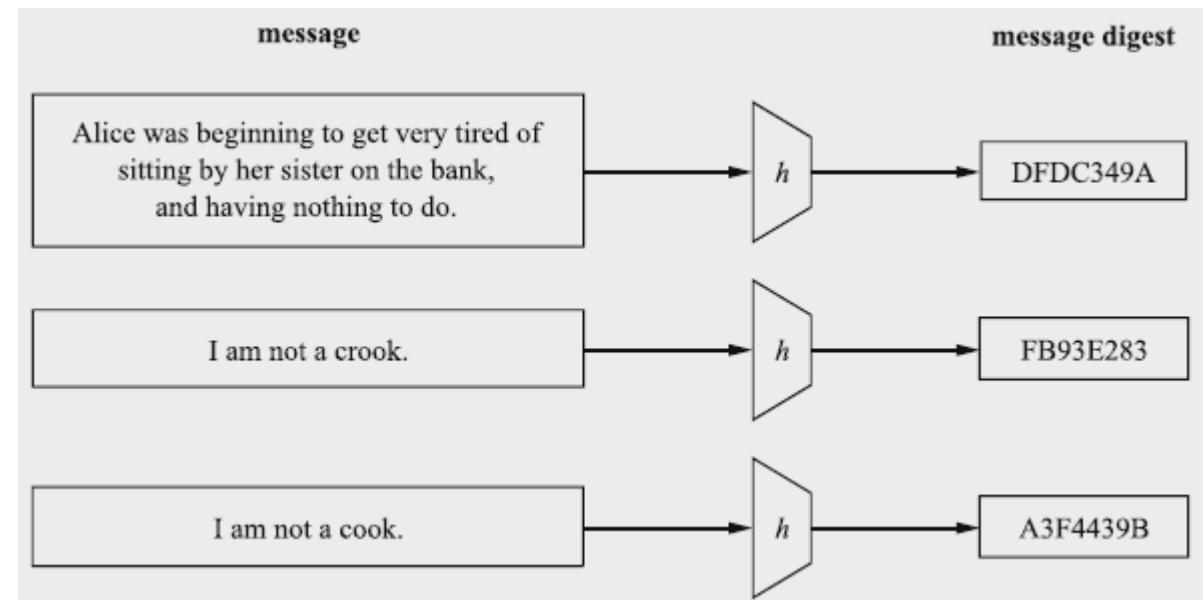
- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)
- Key management (certificates)

Cryptographic Hash Function: Definition

A hash function is a mapping H such that:

1. Map binary messages M of arbitrary lengths to outputs (digest) of a **fixed length**: $H: \{0,1\}^* \rightarrow \{0,1\}^n$ (n is small, e.g. $n = 256$)
2. Can be efficiently computed for all x

- H is called an ***n-bit hash function***
- $H(x)$ is called the **hash** or **message digest** of x
- The description of a hash function is public; there are **no secret keys**
- [SHA-1](#) ($n = 160$) and [SHA-1 hash calculator](#)
- [SHA-256](#) ($n = 256$) and [SHA-256 hash calculator](#)



Cryptographic Hash Function: Properties

- **Correctness:** Deterministic → Hashing the same input always produces the same output
- **Efficiency:** Efficient to compute
- **Security:** **Random/unpredictability**, no predictable patterns for how changing the input affects the output
 - Changing 1 bit in the input causes the output to be completely different
 - Also called “random oracle” assumption
- **Security:** **One-way-ness** (“preimage resistance”)
- **Security:** **Collision-resistance**

Hash Function: One-way-ness or Preimage Resistance

Informal description of “One-way-ness”:

- Given an output y , it is *infeasible to find any input x* such that $H(x) = y$
- e.g., $\text{SHA-256}(a) =$
 $ca978112ca1bbdcfac231b39a23dc4da786eff8147c4e72b9807785afee48bb$
- Here’s an output. Can you find an input that hashes to this output? ***NO!***
 - Note: The adversary just needs to find *any input*, not necessarily the input that was actually used to generate the hash
- Counterexample: Is $H(x) = x^2$ one-way?
 - No, because given output 100, an attacker can return a number x , s.t. $H(x) = 100$

Hash Function: Collision Resistance

Informal description of Collision:

- Two different inputs with the same hash value
- $x \neq x'$ and $H(x) = H(x')$

Can we design a hash function with no collisions?

- **No!** because there are more inputs than outputs
 - How many inputs? → Infinite
 - How large is the output space? → Fixed, e.g., SHA-256(), n = 256, 2^{256} outputs
- However, we want to make finding collisions *infeasible* for an attacker
- **Collision resistance:** It is infeasible to find any pair of inputs $x' \neq x$ such that $H(x) = H(x')$
- Intuition: Can you find *any* two inputs that collide with the same hash output for *any* output?

Hash Function: Intuition

- A hash function provides a *fixed-length “fingerprint”* over a sequence of bits
- Example: Document comparison
 - If Alice and Bob both have a 1 GB document, they can both compute a hash over the document and (securely) communicate the hashes to each other
 - If the hashes are the same, the files must be the same, since they have the same “fingerprint”
 - If the hashes are different, the files must be different
- Example: Lab 10, Task 5
 - *Do you remember Bob’s bill? How can Bob prevent Trudy from altering his bills?*
 - *Use a Hash function instead of a checksum!*

Hash Function: Examples

MD5:

- Security: Completely broken

SHA-1:

- Security: Completely broken in 2017; Was known to be weak before, but still used sometimes

SHA-2:

- Output: 256 (SHA-256), 384, or 512 bits (sometimes labeled SHA-256, SHA-384, SHA-512)
- Not currently broken, but some variants are vulnerable to a length extension attack ([demo](#))
- Current standard

SHA-3 (Keccak):

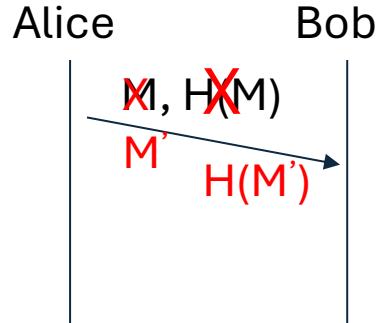
- Output: 256, 384, or 512 bits
- Current standard (not meant to replace SHA-2, just a different construction)
- Widely used in the blockchain ecosystem

Does Hash Provide Integrity?

Integrity: An adversary **cannot change** our messages without being detected

It depends on your threat model. For example:

- Alice and Bob want to communicate over an insecure channel
- Mallory might tamper with messages



Idea: Use cryptographic hashes

- Alice sends her message with a cryptographic hash over the channel
- Bob receives the message and computes a hash on the message
- Bob checks that the hash he computed matches the hash sent by Alice

Threat model: Mallory can *modify* the message ***and the hash***

- **No integrity!**
- **Why?** There is **no secret key** being used as input, so any attacker can compute a hash on any value
- Next: Use hashes to design schemes that provide integrity

Roadmap

- *Cryptographic Hash*
- ***Message Authentication Codes (MACs)***
- *Diffie-Hellman Key Exchange*

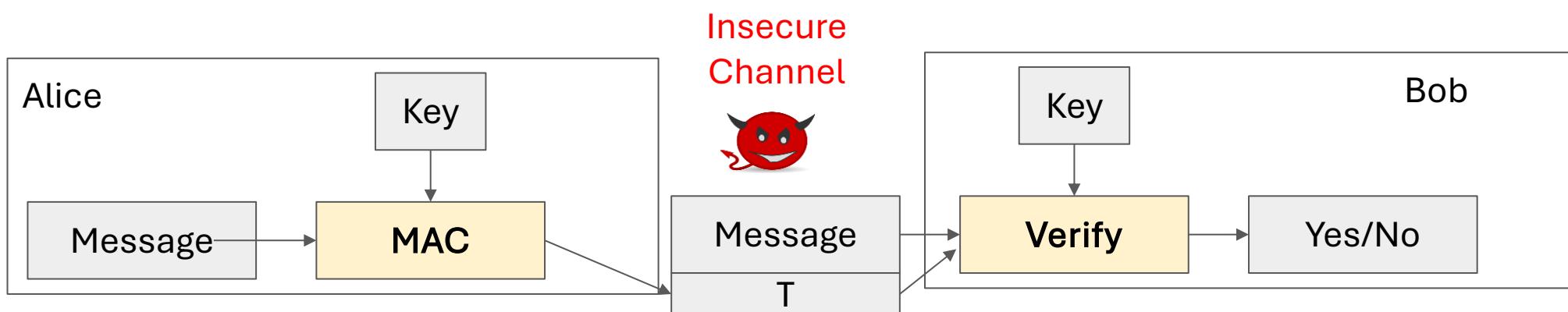
How to Provide Integrity

- Alice and Bob want to communicate over an insecure channel
- Reminder: We're in the symmetric-key setting
 - Assume that Alice and Bob share a secret key, and attackers don't know the key
- We want to attach some piece of information to *prove* that someone **with the key** sent this message
- This piece of information can only be generated by someone with the key



Message Authentication Codes (MACs): Usage

- **Integrity:** An adversary cannot *change* our messages without being detected
- Alice wants to send M to Bob, but doesn't want Mallory to tamper with it
- Alice and Bob **share a secret key**
- Alice sends **M and $T = \text{MAC}(K, M)$** to Bob
- Bob receives message, recomputes $\text{MAC}(K, M)$ and checks that it matches T
- If the MACs match, Bob is confident the message has not been tampered



MACs: Definition

- Two parts:
 - $\text{KeyGen}() \rightarrow K$: Generate a key K
 - $\text{MAC}(K, M) \rightarrow T$: Generate a tag T for the message M using key K
 - Inputs: A secret key and an *arbitrary-length message*
 - Output: A *fixed-length tag* on the message
- Properties
 - **Correctness:** Determinism
 - Note: Some more complicated MAC schemes have an additional $\text{Verify}(K, M, T)$ function that don't require determinism, but this is out of scope
 - **Efficiency:** Computing a MAC should be efficient
 - **Security:** EU-CPA (existentially unforgeable under chosen plaintext attack)

MAC Instantiation: NMAC (Nested MAC)

Can we use secure cryptographic hashes to build a secure MAC?

- Intuition: Hash output is unpredictable and looks random, so let's hash the key and the message together
- *KeyGen()*:
 - Output **two** random, n -bit keys K_1 and K_2 , where n is the length of the hash output
- $NMAC(K_1, K_2, M)$:
 - Output $H(K_1 \parallel H(K_2 \parallel M))$
- NMAC is EU-CPA secure if the two keys are different
 - Provably secure if the underlying hash function is secure
- Intuition: Using two hashes prevents a length extension attack
 - Otherwise, an attacker who sees a tag for M could generate a tag for $M \parallel M'$

MAC Instantiation: HMAC (Hash-based MAC)

- Issues with NMAC:
 - Recall: $\text{NMAC}(K_1, K_2, M) = H(K_1 \parallel H(K_2 \parallel M))$
 - We need *two different keys*, and the *length of the keys* must be the same as the length of the hash output (n bits).
- $\text{HMAC}(K, M)$:
 - Compute K' as a version of K that is the length of the hash output
 - If K is too short, pad K with 0's to make it n bits (be careful with keys that are too short, e.g., < 80 bits, and lack randomness, **not recommended**)
 - If K is too long, hash it so it's n bits
 - Recommended key length: For example, HMAC-SHA256, a key length of 64 bytes (512 bits) is recommended to achieve the full 256-bit security level
 - Output $H(K' \oplus \text{opad}) \parallel H(K' \oplus \text{ipad}) \parallel M)$

MAC Instantiation: HMAC (Hash-based MAC)

- $\text{HMAC}(K, M) = H(K \oplus opad) \parallel H(K \oplus ipad) \parallel M)$
- Use K to derive two different keys
 - $opad$ (outer pad) is the hard-coded byte **0x5c** repeated until it's the same length as K
 - $ipad$ (inner pad) is the hard-coded byte **0x36** repeated until it's the same length as K
 - As long as $opad$ and $ipad$ are different, you'll get two different keys
 - For paranoia, the designers chose two very different bit patterns, even though they theoretically need only differ in one bit
- HMAC is a hash function, so it has the properties of the underlying hash too:
 - It is collision resistant
 - Given $\text{HMAC}(K, M)$ and K , an attacker can't learn M
 - If the underlying hash is secure, HMAC doesn't reveal M , but it is still **deterministic**
- You can't verify a tag T if you don't have K
 - The attacker can't brute-force the message M without knowing K (HMAC-SHA256: $1/2^{256}$)

Do MACs provide integrity?

- Do MACs provide integrity?
 - **Yes!** An attacker cannot tamper with the message without being detected
- Do MACs provide confidentiality?
 - MACs are **deterministic** ⇒ Same M & K lead to the same tag T
 - MACs in general have no confidentiality guarantees; they can leak information about the message
- Do MACs provide authenticity?
 - *It depends on your threat model*
 - If a message has a valid MAC, you can be sure it came from *someone with the secret key*, but you can't narrow it down to one person
 - If **only two** people have the secret key, MACs provide authenticity: it has a valid MAC, and it's not from me, so it must be from the other person

Roadmap

- *Cryptographic Hash*
- *Message Authentication Codes (MACs)*
- *Diffie-Hellman Key Exchange*

The Diffie-Hellman Key Exchange Protocol (informally)

- Generate a **symmetric** cryptographic key over a **public** channel

Fix a large prime p (e.g. 2048 digits)

Alice

Fix an integer (generator) g in $\{1, \dots, p\}$

Bob

choose random a in $\{1, \dots, p-1\}$

choose random b in $\{1, \dots, p-1\}$

$$A = g^a \pmod{p}$$

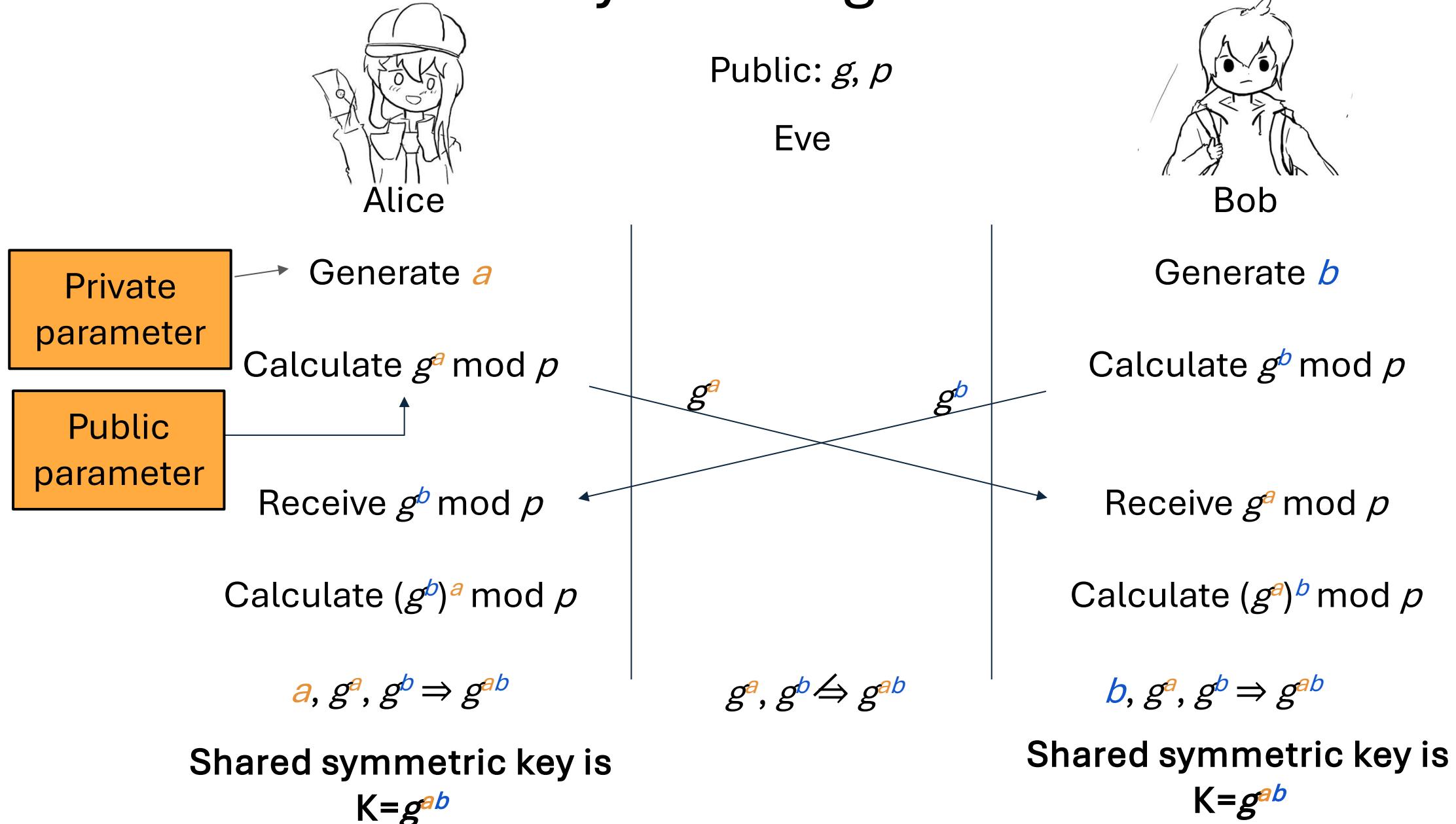
$$B = g^b \pmod{p}$$

$$B^a \pmod{p} = (g^b)^a = k_{AB} = g^{ab} \pmod{p} = (g^a)^b = A^b \pmod{p}$$

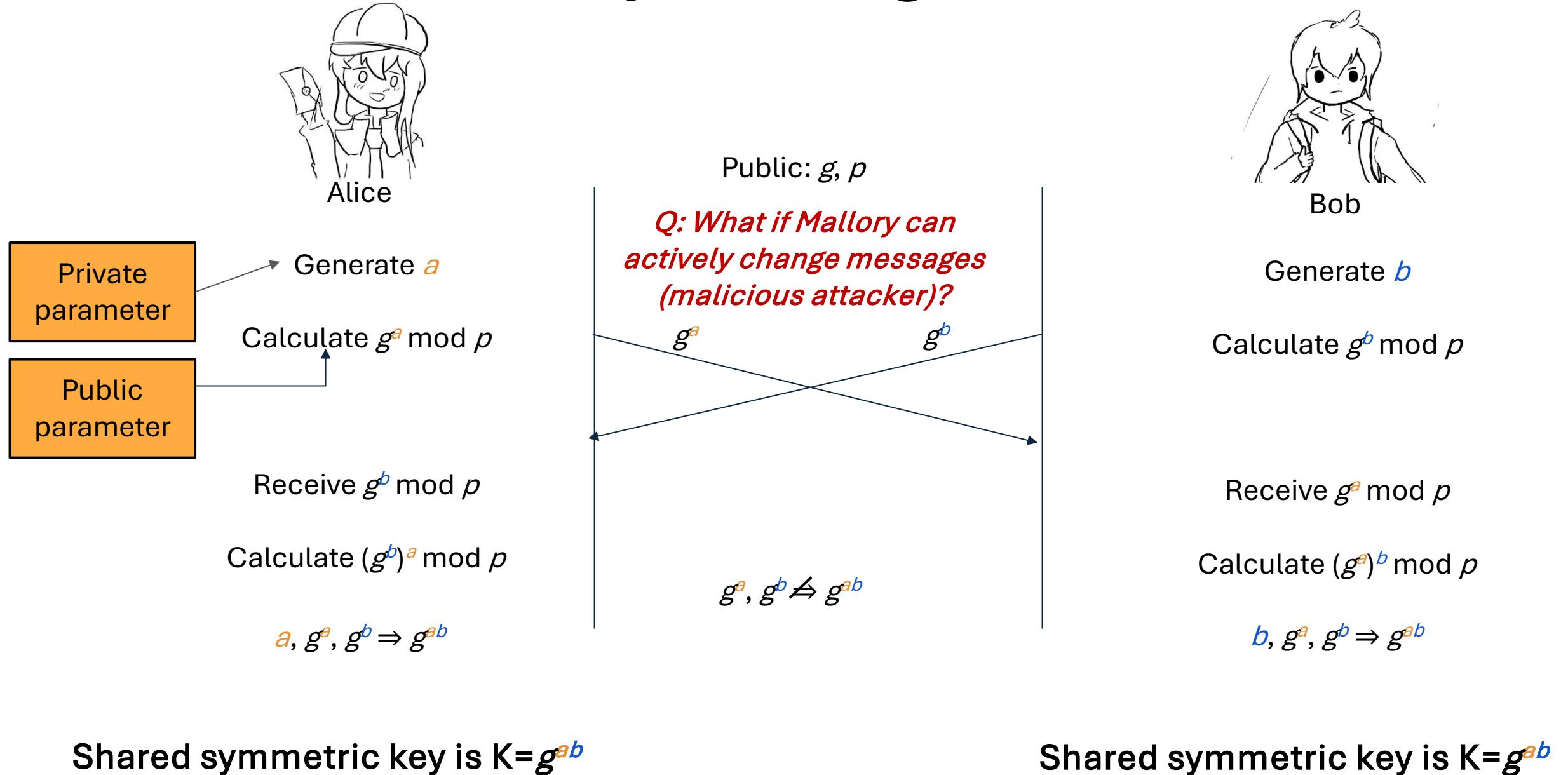
Discrete Log Problem and Diffie-Hellman Problem

- Assume everyone knows a **large prime p** (e.g. 2048 bits long) and a **generator g**
 - Don't worry about what a generator g is; just treat it as a number.
- **Discrete logarithm problem (discrete log problem):**
 - Given $g, p, g^a \text{ mod } p$ for random a , it is **computationally hard to find a**

Diffie-Hellman Key Exchange Protocol



Diffie-Hellman Key Exchange Protocol



Diffie-Hellman: Man-in-the-middle Attack



Alice



Mallory



Bob

Public: g, p

Generate a
Calculate $g^a \text{ mod } p$

Receive $g^m \text{ mod } p$

Calculate $(g^m)^a \text{ mod } p$

$$a, g^a, g^m \Rightarrow g^{am}$$

Generate m
Calculate $g^m \text{ mod } p$

Receive $g^a \text{ mod } p$

Calculate $(g^a)^m \text{ mod } p$

$$m, g^m, g^a \Rightarrow g^{am}$$

Generate b
Calculate $g^b \text{ mod } p$

Receive $g^m \text{ mod } p$

Calculate $(g^m)^b \text{ mod } p$

$$b, g^b, g^m \Rightarrow g^{bm}$$

Diffie-Hellman: Issues

Issues:

- **No Authentication:** DH doesn't verify who you're exchanging keys with
 - Vulnerable to Man-in-the-Middle (**MITM**) attacks
- **Active Protocol:** Both parties must be online simultaneously
 - Cannot encrypt a message for someone who's offline

Solutions to MITM Attacks:

- **Digital Signatures:** Sign DH public parameters ($g^a \bmod p$, $g^b \bmod p$) with certificate-based authentication
 - Used in TLS/HTTPS (next week)
- **Pre-Shared Keys (PSK):** Authenticate DH public parameters using MAC
 - PSK provides authentication, DH provides *forward secrecy*
 - Used in VPNs and IoT devices

Summary: Diffie-Hellman Key Exchange

- Algorithm:
 - Alice chooses a and sends $g^a \text{ mod } p$ to Bob
 - Bob chooses b and sends $g^b \text{ mod } p$ to Alice
 - Their shared secret is $(g^a)^b = (g^b)^a = g^{ab} \text{ mod } p$
- Diffie-Hellman provides forward secrecy: Nothing is saved or can be recorded that can ever recover the key
 - Forward secrecy (aka Perfect forward secrecy): a cryptographic feature that ensures *past* encrypted communications remain secure even if current session keys or long-term private keys are compromised
- Diffie-Hellman can be performed over other mathematical groups, such as Elliptic-curve Diffie-Hellman (ECDH)
- Issues
 - Does not provide authenticity
 - *Not* secure against MITM
 - Both parties must be online

CSC 3511 Security and Networking

Week 12, Lecture 1: Transport Layer Security

Roadmap

- ***Transport Layer Security (TLS)***
- *TLS Handshake Protocol*
- *TLS: Security Guarantees and Summary*
- *TLS: Application (HTTPS)*

Recap: TCP and UDP

Transmission Control Protocol (TCP): Reliably sending packets

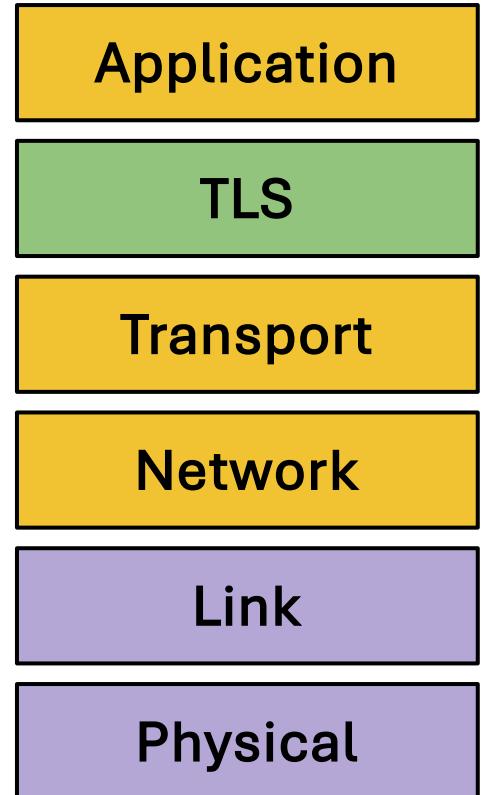
- Provides reliability, ordering, and byte stream abstraction
 - 3-way handshake
 - Packet's sequence number and ACK number
 - Flow control and congestion control
- **Attack:** TCP DoS attack by launching a SYN Flooding attack
- **Attack:** TCP reset attack by injecting TCP Reset (RST) packets
 - Blind attacks must guess the client's or server's sequence numbers
- **Attack:** TCP session hijacking attack through data injection
 - Blind attacks must guess the client's or server's sequence numbers

User Datagram Protocol (UDP): Non-reliably sending packets

- No reliability or ordering, only datagram abstraction
- Similar injection and spoofing attacks as TCP, but easier

Transport Layer Security (TLS)

- **TLS (Transport Layer Security)**: A protocol for creating a secure communication channel over the Internet
 - Replaces **SSL (Secure Sockets Layer)**, which is an older version of the protocol
- TLS is built on top of TCP
 - **Relies upon**: Byte stream abstraction between the client and the server
 - **Provides**: Byte stream abstraction between the client and the server
 - The abstraction appears the same to the end client, but TLS provides **confidentiality, integrity and (server) authenticity!**



Secure Internet Communication: Goals of TLS

Confidentiality: Ensure that attackers cannot read your traffic

- Symmetric encryption
- Protects against eavesdropping

Integrity: Ensure that attackers cannot tamper with your traffic

- HMAC
- Prevent replay attacks → The attacker records encrypted traffic and then replays it to the server
- Example: Replaying a packet that sends “Pay \$10 to Mallory”

Authenticity: Make sure you’re talking to the legitimate server

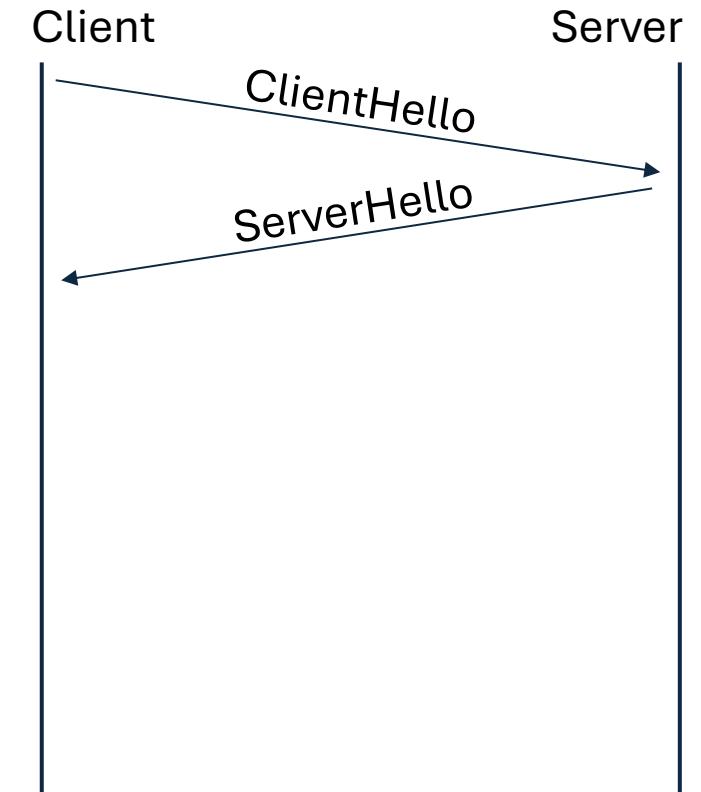
- Asymmetric encryption/Digital signature & Certificate
- Defend against an attacker impersonating the server
- Prevents man-in-the-middle (MITM) attacks

Roadmap

- *Transport Layer Security (TLS)*
- ***TLS Handshake Protocol***
- *TLS: Security Guarantees and Summary*
- *TLS: Application (HTTPS)*

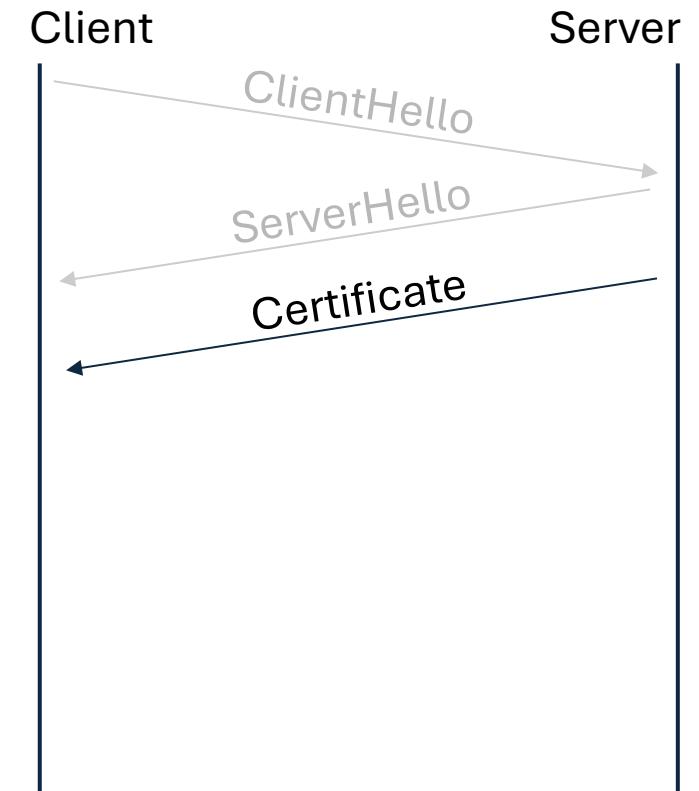
TLS Handshake Step 1: Exchange Hellos

- Assume an underlying TCP connection has already been formed; Use TLS 1.2 as an example.
- The client sends **ClientHello** with
 - A 256-bit random number R_C (“client random”)
 - A *list* of supported cryptographic algorithms (Cipher Spec)
- The server sends **ServerHello** with
 - A 256-bit random number R_S (“server random”)
 - The algorithms to use (chosen from the client’s list)
- R_C and R_S prevent replay attacks
 - R_C and R_S are **randomly** chosen for every handshake
 - This guarantees that two handshakes will never be exactly identical



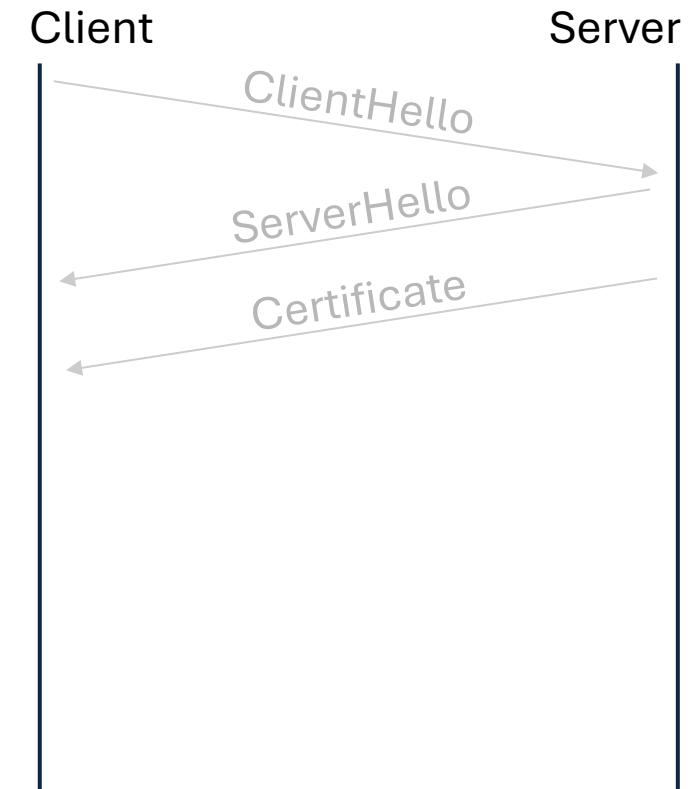
TLS Handshake Step 2: Certificate

- The server sends its certificate
 - Recall certificates: The server's **identity** and **public key (PK_{server})**, signed by a trusted certificate authority (CA)
- The client validates the certificate
 - Verify the signature in the certificate
- The client now knows the server's public key
 - The client is **not yet sure** that they are talking to the legitimate server (not an impersonator)
 - Recall: Certificates are public. Anyone can provide a certificate for anybody



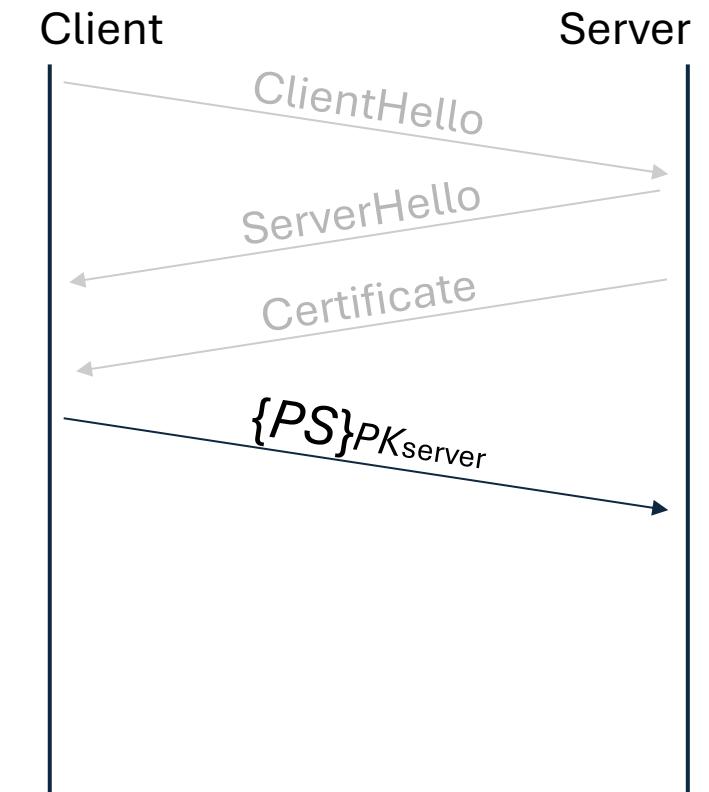
TLS Handshake Step 3: Premaster Secret

- This step has two main purposes
 1. Make sure the client is talking to the ***legitimate server*** (not an impersonator)
 - The attacker can collect legitimate certificates
 - The server must **prove** that it owns the private key corresponding to the public key in the certificate
 2. Give the client and server ***a shared secret***
 - An attacker should not be able to learn the secret
 - This will help the client and the server secure messages later
- Two approaches to sharing a premaster secret:
RSA or Diffie-Hellman (DHE)

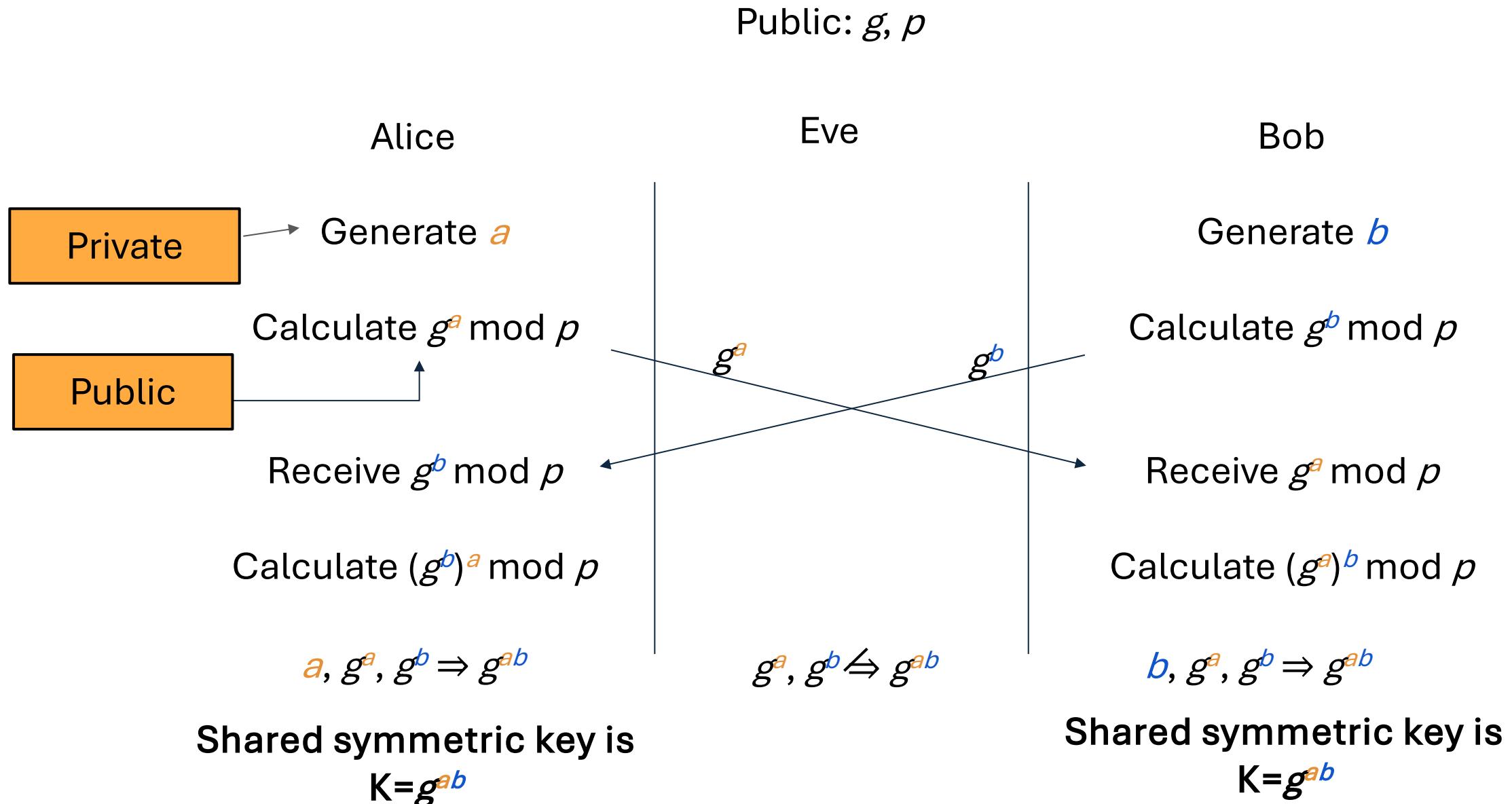


TLS Handshake Step 3: Premaster Secret (RSA)

- The client randomly generates a **premaster secret (PS)**
- The client encrypts PS with the *server's public key* and sends it to the server
 - The client knows the server's public key from the certificate
- The server decrypts the premaster secret PS
- The client and server now share a secret
 - Recall RSA encryption: Nobody except the legitimate server can decrypt the premaster secret
 - **Proves** that the server owns the private key (otherwise, it could not decrypt PS)

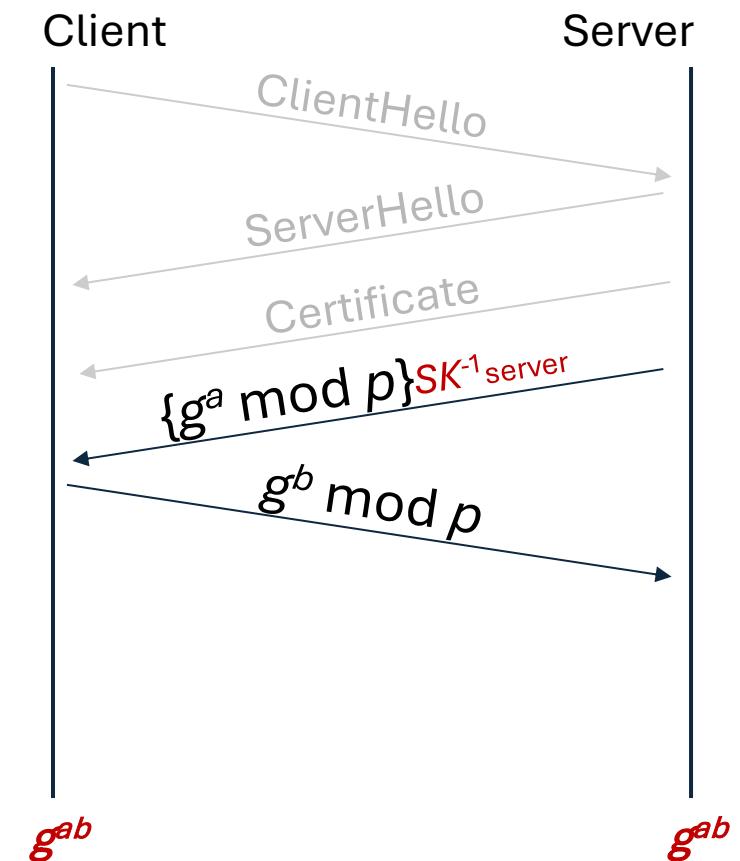


Recap: Diffie-Hellman Key Exchange



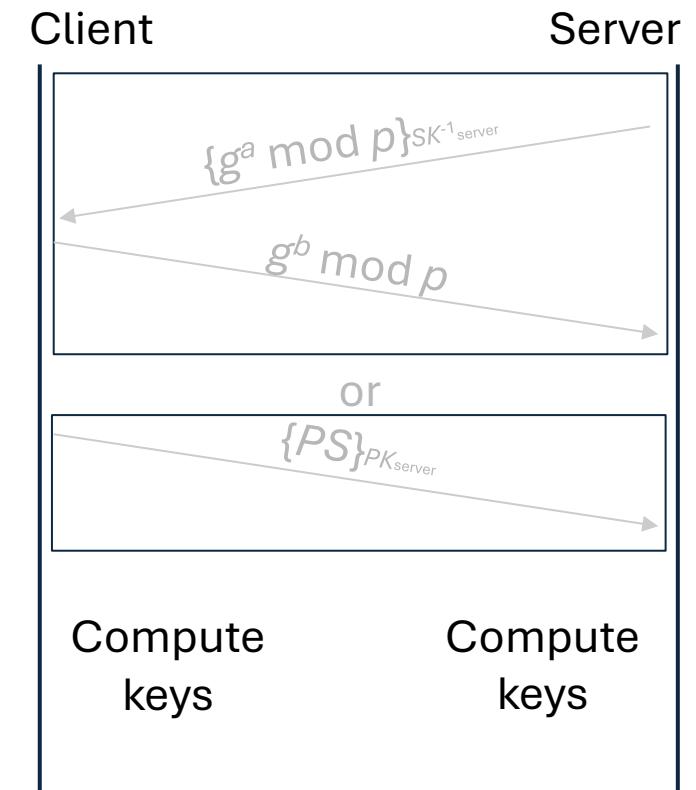
TLS Handshake Step 3: Premaster Secret (DHE)

- The server generates a secret a and computes $g^a \text{ mod } p$
- The server **signs** $g^a \text{ mod } p$ with its private key ($\textcolor{red}{SK}_{\text{server}}$) and sends the message and signature
- The client verifies the signature
 - *PKI and chain of certificates*
 - Proves that the server owns the private key
- The client generates a secret b and computes $g^b \text{ mod } p$
- The client and server now share a **premaster secret (PS)**: $g^{ab} \text{ mod } p$
 - Recall Diffie-Hellman: an attacker cannot compute $g^{ab} \text{ mod } p$



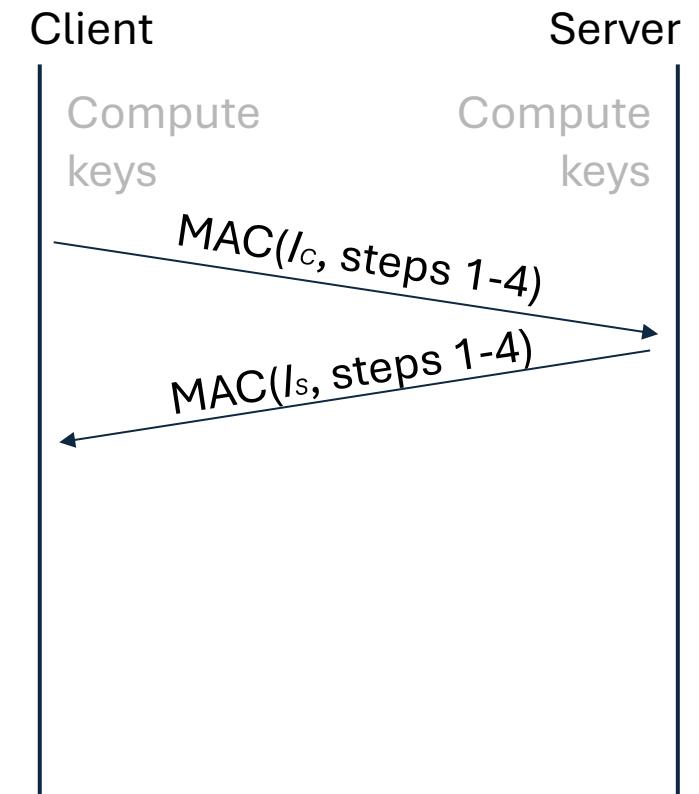
TLS Handshake Step 4: Derive Symmetric Keys

- The server and client each derive symmetric keys from random numbers R_c , R_s , and premaster secret PS
 - Usually derived by seeding a PRNG (reproducible random numbers) with the three values
 - Changing any of the values results in different symmetric keys
- Four symmetric keys are derived
 - C_c : For encrypting ***client-to-server*** messages
 - C_s : For encrypting ***server-to-client*** messages
 - I_c : For MACing ***client-to-server*** messages
 - I_s : For MACing ***server-to-client*** messages
- Note: Both client and server know all four keys



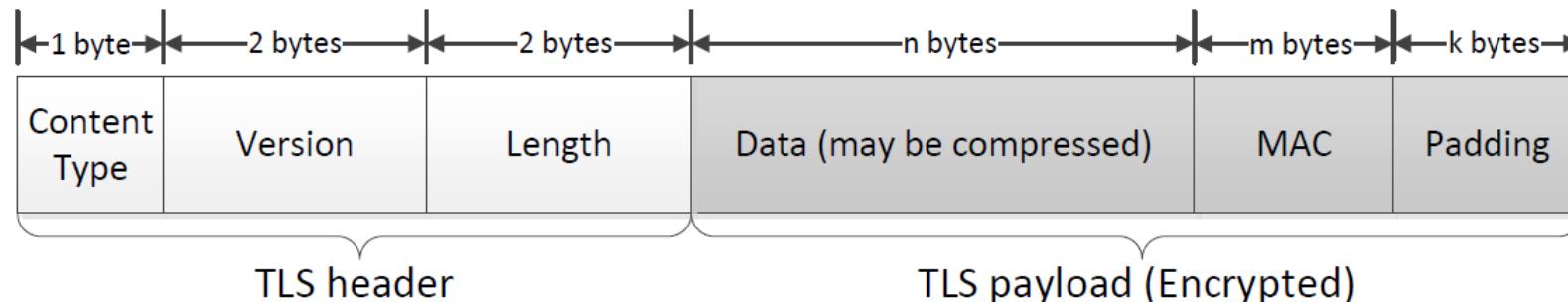
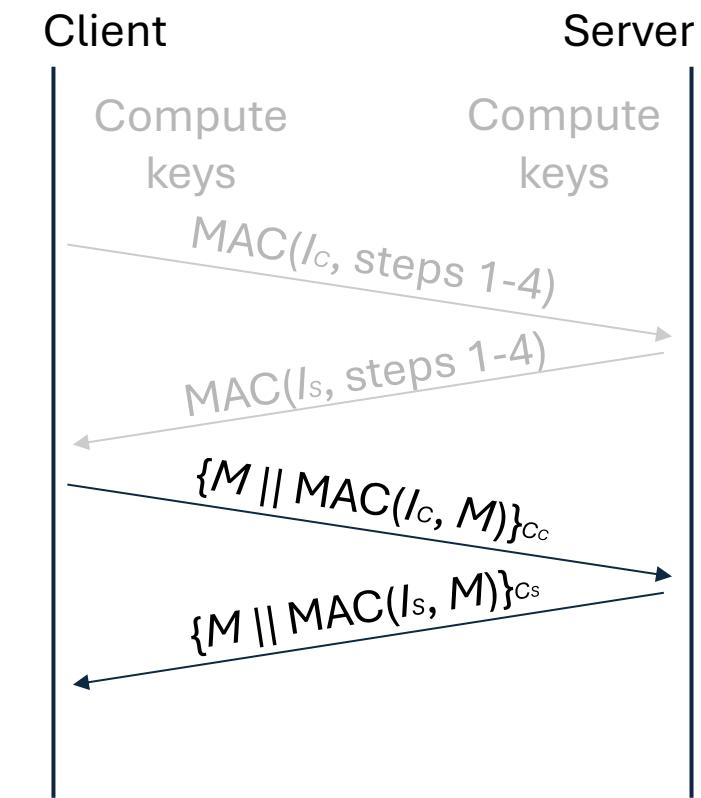
TLS Handshake Step 5: Exchange MACs

- The server and client exchange MACs on **all the messages** of the handshake so far
 - Recall MACs: Any tampering on the handshake will be detected
 - Not to be confused with MAC addresses
- Why Do We Need This Step
 - Ensures message integrity & Validates key agreement:
 - Verifying that no tampering occurred during the handshake process
 - Synchronization:
 - Confirming that both the client and server agree on the handshake messages and cryptographic parameters

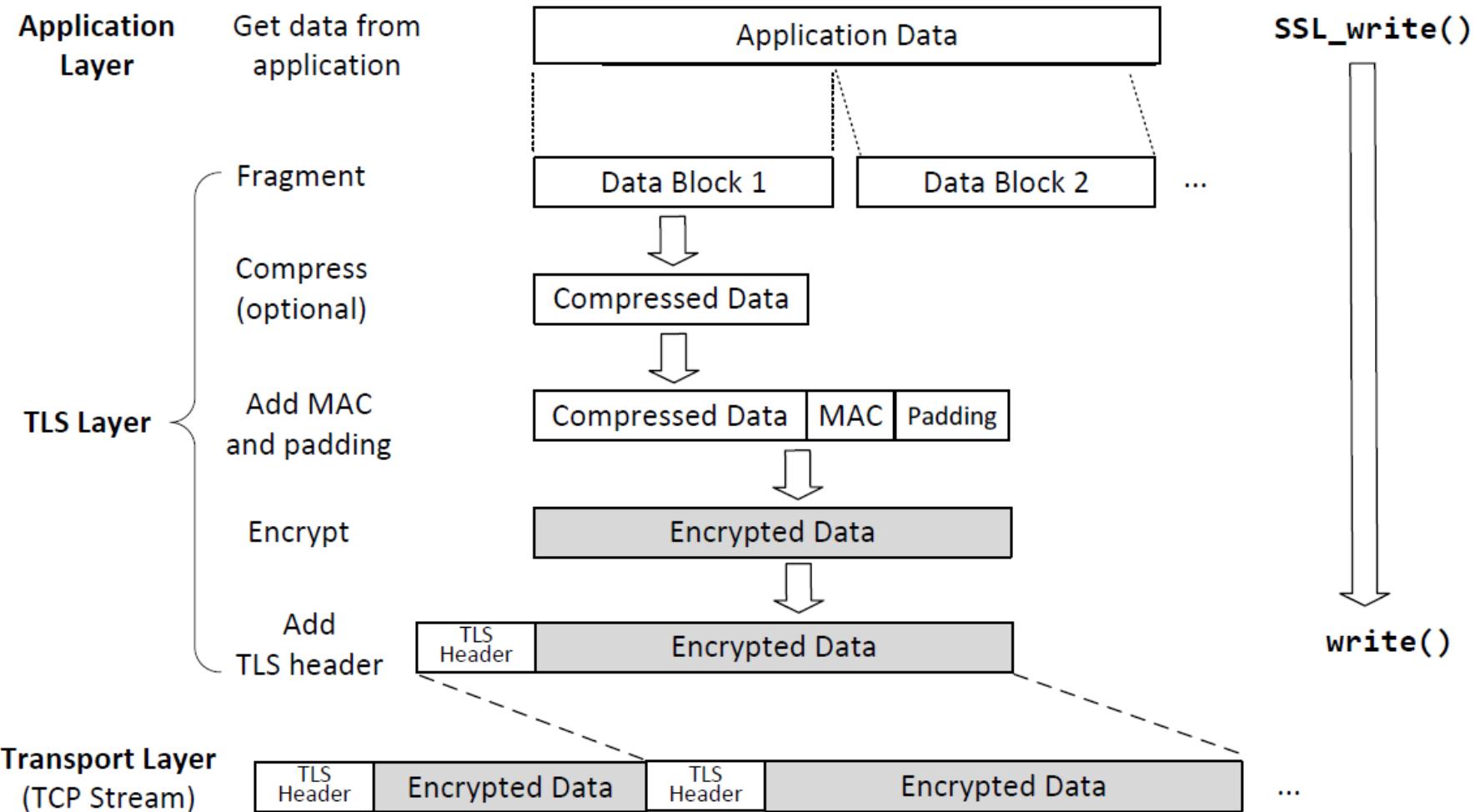


TLS Handshake Step 6: Send Messages

- Messages can now be sent securely
 - Encrypted **AND** MAC'd
 - Note: TLS (*up to version 1.2*) uses MAC-then-encrypt, as shown in the right figure
 - Modern cryptographic best practices favor **Encrypt-then-MAC**, where the message is *first encrypted*, and the MAC is calculated over the *ciphertext*. Send ciphertext and MAC
 - This is generally considered more secure but is not used in earlier versions of TLS for legacy compatibility



Sending Data with the TLS Protocol



TCP Header: Contains transport-layer metadata (e.g., ports, sequence numbers)

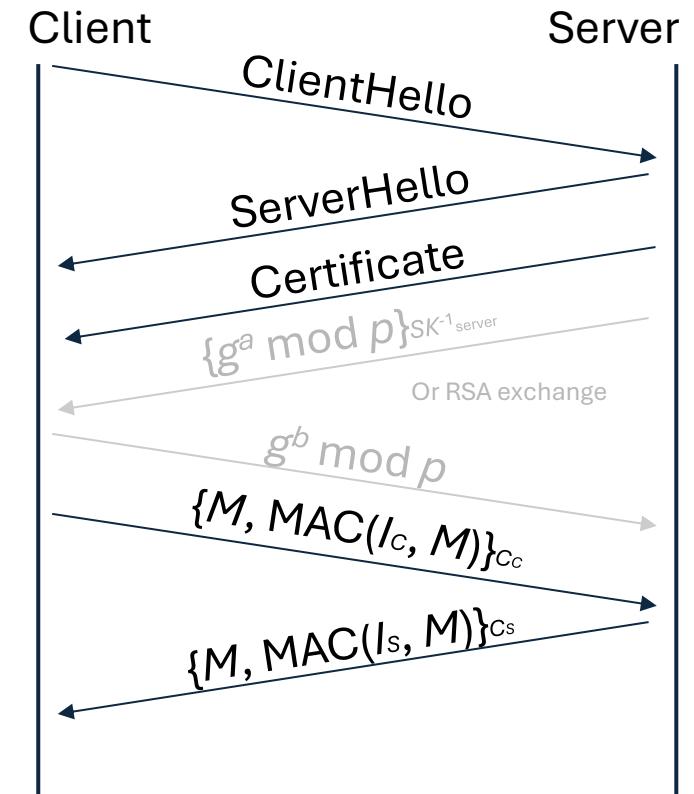
TCP Payload: Includes the **TLS Header** (unencrypted) and the **Encrypted Data**

Roadmap

- *Transport Layer Security (TLS)*
- *TLS Handshake Protocol*
- ***TLS: Security Guarantees and Summary***
- *TLS: Application (HTTPS)*

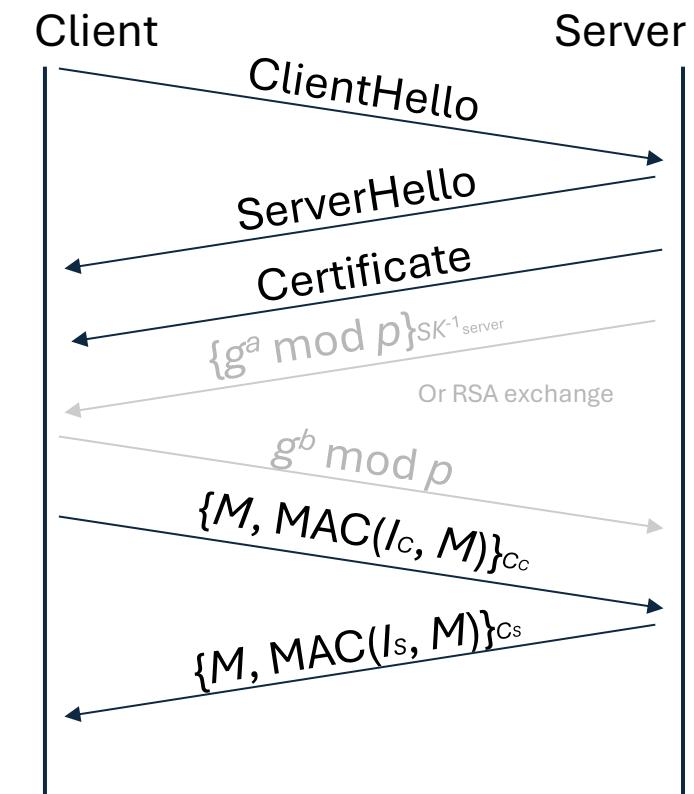
TLS: Talking to the Legitimate Server

- How can we be sure we are talking to the legitimate server (**server authentication**)?
 - The server sent its certificate, so we know the server's public key
 - The server proved that it owns the corresponding **private key**
 - RSA: The server decrypted the PS
 - DHE: The server signed its half of the exchange
- An attacker impersonating the server would not have the server's private key (assuming they have not compromised the server)



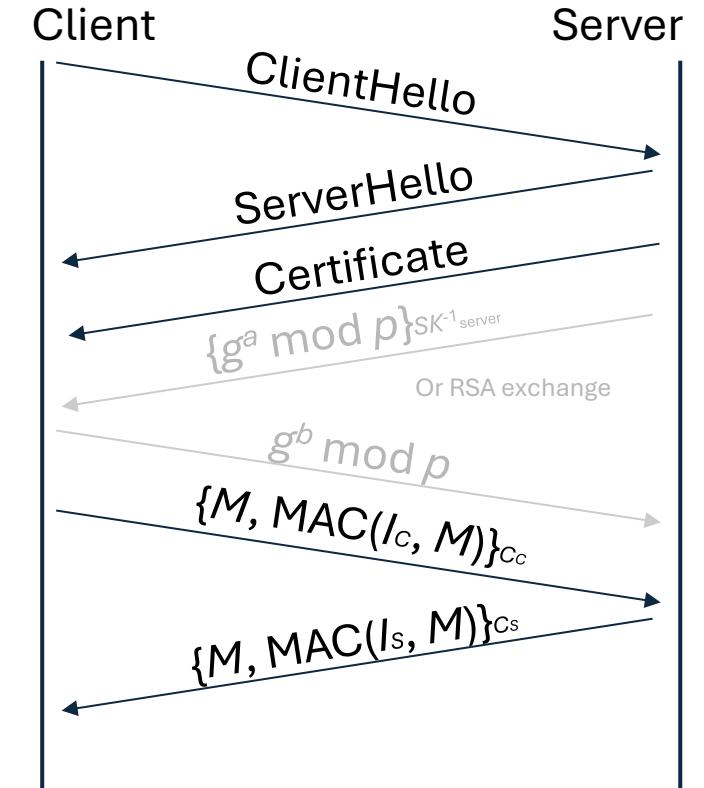
TLS: Securing Messages

- How can we be sure that network attackers can't read or tamper with our messages?
- The attacker doesn't know *PS*
 - RSA: PS was encrypted with the server's public key
 - DHE: An attacker cannot learn the Diffie-Hellman secret
- The symmetric keys are derived from PS
 - The attacker doesn't know the symmetric keys used to encrypt and MAC messages
- Encryption and MACs provide **confidentiality** and **integrity**



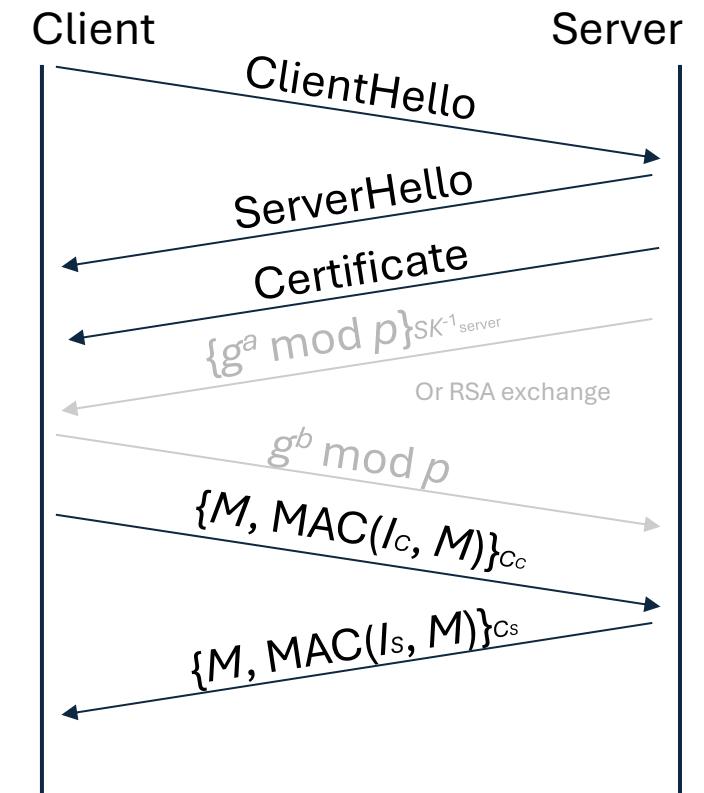
TLS: Replay Attacks

- How can we be sure that the attacker hasn't **replayed** old encrypted messages from a *past* TLS connection?
 - Replaying a packet that sends "Pay \$10 to Mallory"
- Every handshake uses a **different** R_c and R_s
- The symmetric keys are derived from R_c and R_s
 - The symmetric keys are different for every connection



TLS: Replay Attacks

- How can we be sure that the attacker hasn't replayed old messages from the *current* TLS connection?
- Add **record numbers** (in the encrypted TLS message)
 - Every message uses a **unique** record number (used in **MAC calculation**, not explicitly transmitted)
 - If the attacker replays a message, the record number will be repeated
- TLS record numbers are not TCP sequence numbers
 - Record numbers are used for **security**: ensure M is not replayed or reordered by attackers
 - Sequence numbers are used for **correctness**, in the layer below

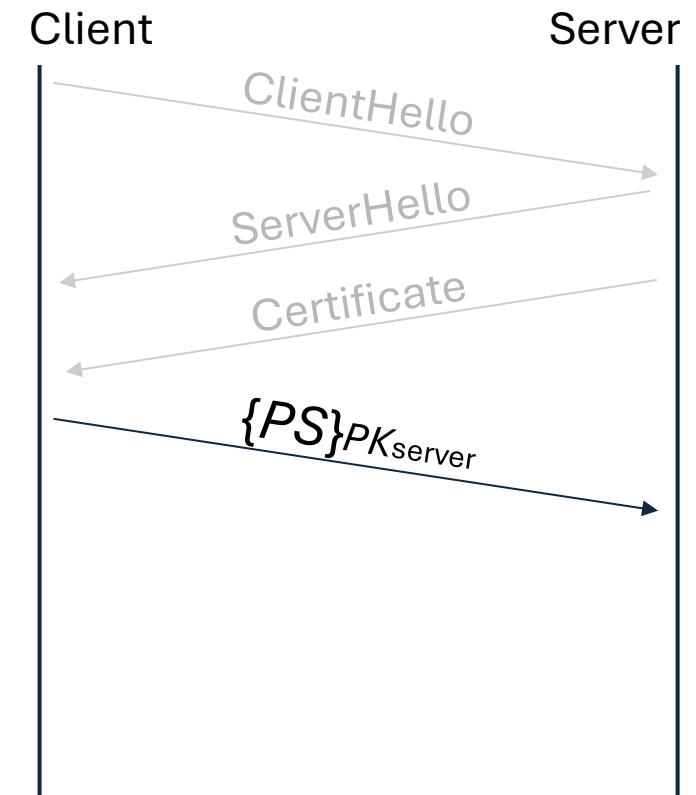


TLS: Forward Secrecy

- **Forward secrecy:** If an attacker records a connection now and compromises secret values later, they cannot compromise the recorded connection

RSA TLS: No forward secrecy is guaranteed

- The adversary can record Rc , Rs , and the encrypted PS
- If the adversary later compromises the **server's private key**, they can decrypt PS and derive the keys!



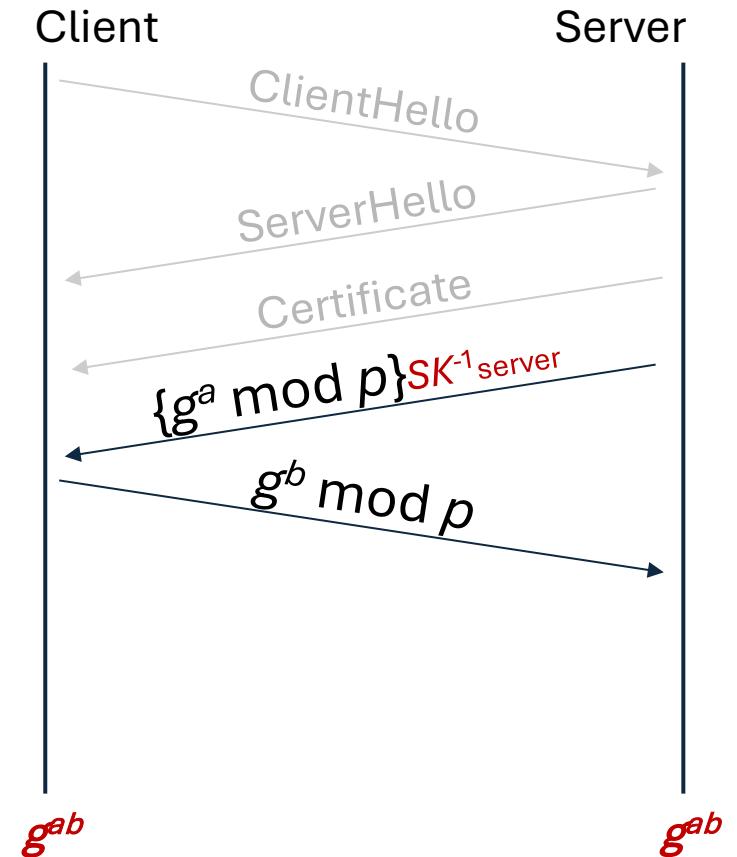
TLS Handshake Step 3: Premaster Secret (RSA)

TLS: Forward Secrecy

- **Forward secrecy:** If an attacker records a connection now and compromises secret values later, they cannot compromise the recorded connection

DHE TLS: **Guaranteed forward secrecy**

- Diffie-Hellman provides forward secrecy: g^a is deleted after the TLS session is over
- Adversary can't learn the keys, even if they later compromise the server's private key



TLS Handshake Step 3: Premaster Secret (DHE)

TLS Provides End-to-End Security

- TLS provides **end-to-end security**: Secure communication between the two endpoints, with no need to trust intermediaries
 - End-to-end security does not help if one of the endpoints is malicious (e.g. communicating with a malicious server)
 - Even if everybody between the client and the server is malicious, TLS provides a secure communication channel
 - Example: A local network attacker (on-path) tries to read our Wi-Fi session, but can't read TLS messages
 - Example: A man-in-the-middle tries to inject TCP packets, but packets will be rejected because the MAC won't be correct
- Using TLS defends against most lower-level network attacks

TLS 1.3 Changes

- TLS 1.3: The latest version of the TLS protocol (2018)
- RSA no longer supported (only DHE)
 - Guarantees forward secrecy
- Performance optimization: The client sends $g^b \text{ mod } p$ in ClientHello
 - If the server agrees to use DHE, the server sends $g^a \text{ mod } p$ (with signature) in ServerHello message
 - Potentially saves two messages later in the handshake
- Only supports AEAD mode encryption
 - AEAD (authenticated encryption with additional data): a block cipher mode that guarantees confidentiality **and integrity** at the same time
 - Eliminates attacks associated with the insecure MAC-then-encrypt pattern

TLS Does Not Provide Anonymity

- **Anonymity:** Hiding the client's and server's identities from attackers
- An attacker can figure out who is communicating with TLS
 - The certificate is sent during the TLS handshake, containing the server's name
 - The client may also indicate the name of the server in the ClientHello (called Server Name Indication, or SNI)
 - An attacker can see IP addresses and ports of the underlying IP and TCP protocols

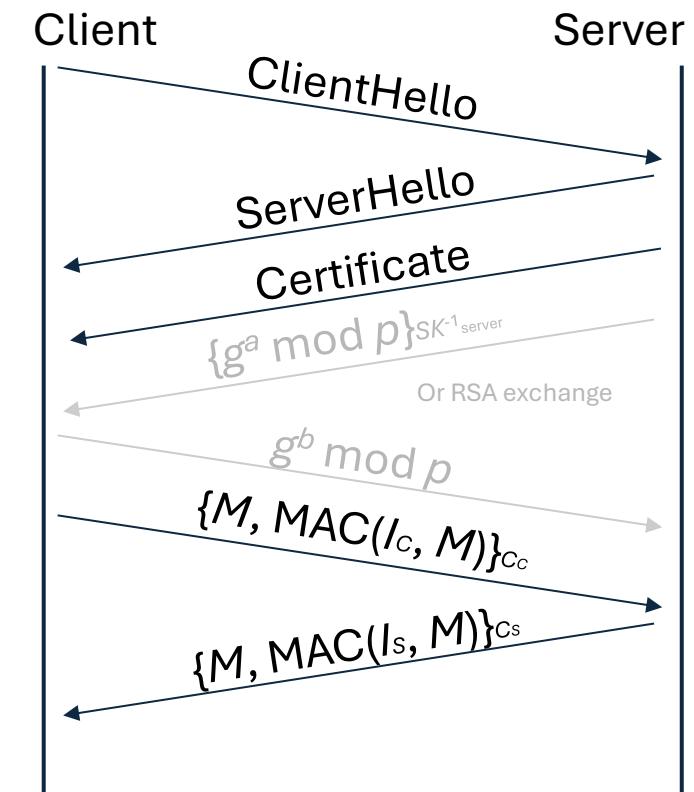
TLS Does Not Provide Availability

- **Availability:** Keeping the connection open in the face of attackers
- An attacker can stop a TLS connection
 - MITM can drop encrypted TLS packets
 - On-path attacker can still do RST injection to abort the underlying TCP connection
- Result: A TLS connection can still be censored
 - The censor can block TLS connections

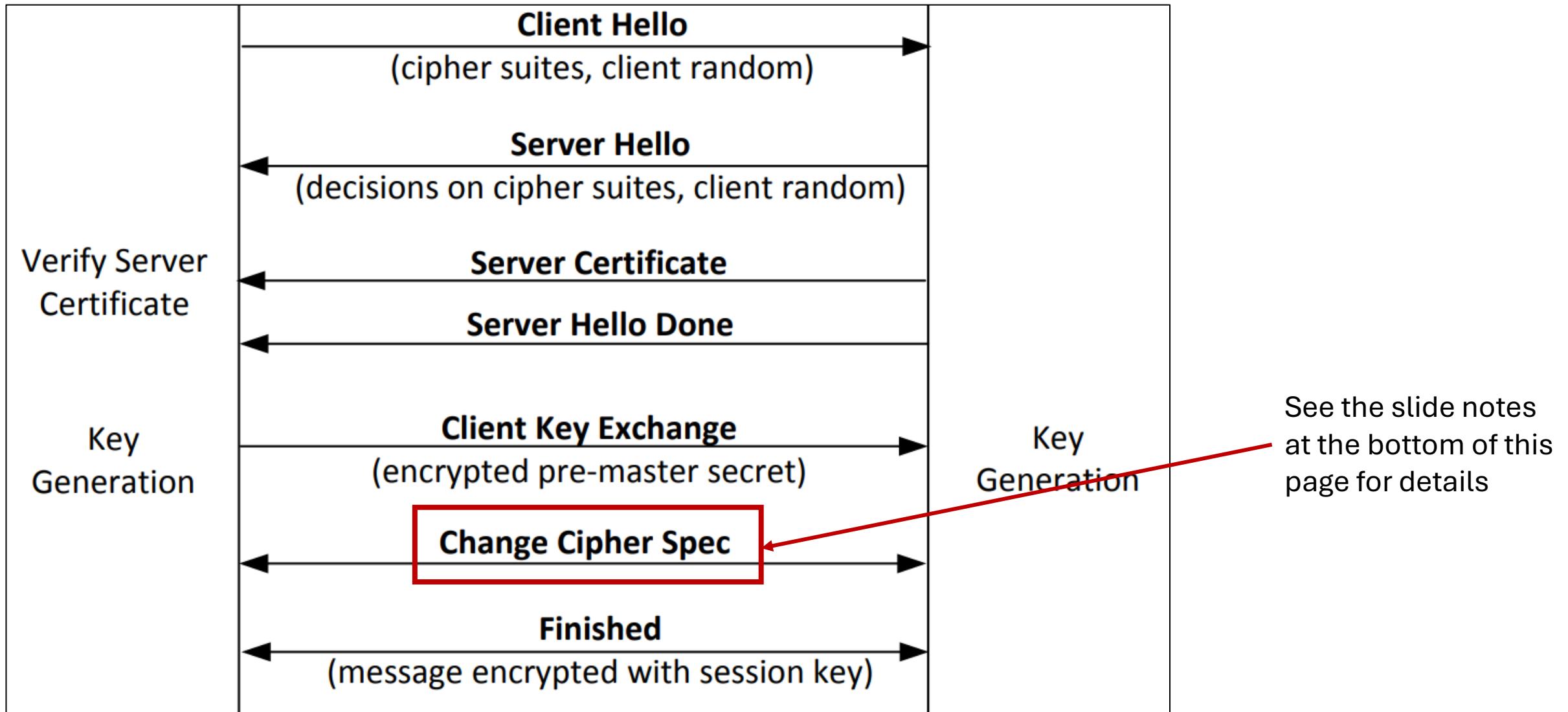
TLS: Summary

TLS Handshake

- Nonces make every handshake different (prevents replay attacks across connections)
- Certificate proves server's public key
- ~~RSA~~ or DHE proves that the server owns the private key
- ~~RSA~~ or DHE helps client and server agree on a shared secret key
- MAC exchange ensures no one tampered with the handshake
- Messages are sent with symmetric encryption and MACs
- TLS record numbers prevent replay attacks within a connection



TLS Handshake Protocol: Workflow



Roadmap

- *Transport Layer Security (TLS)*
- *TLS Handshake Protocol*
- *TLS: Security Guarantees and Summary*
- ***TLS: Application (HTTPS)***

TLS for Applications

TLS provides services to higher layers (the application layer)

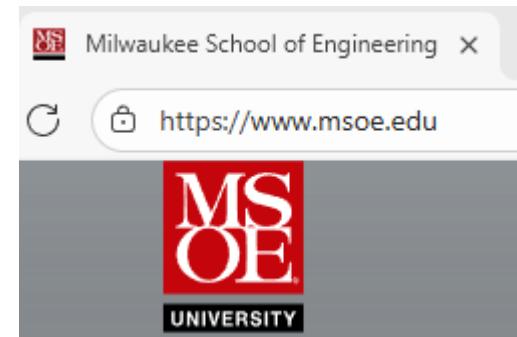
- **HTTPS:** The HTTP protocol run over TLS
 - In contrast, HTTP runs over plain TCP, with no TLS added
- Other secure application-layer protocols besides HTTPS exist
 - Pretty much anything that runs over TCP can also run over TLS, since the byte stream abstraction is maintained
 - Example: Email protocol can use the STARTTLS command to use TLS to secure communications
- TLS does not defend against application-layer vulnerabilities
 - Example: SQL injection, XSS, CSRF, and buffer overflow vulnerabilities in the application are still exploitable over TLS

TLS in Browsers

- Original design:
 - When your browser communicates with a server over TLS, your browser displays a lock icon
 - If TLS is not used, there is no lock icon
- What the lock icon means
 - Communication is **encrypted** (TLS guarantee)
 - You are talking to the **legitimate server** (TLS guarantee)
 - Any external images or scripts are also fetched over TLS



This website uses HTTP: no lock icon



This website uses HTTPS: lock icon

CSC 3511 Security and Networking

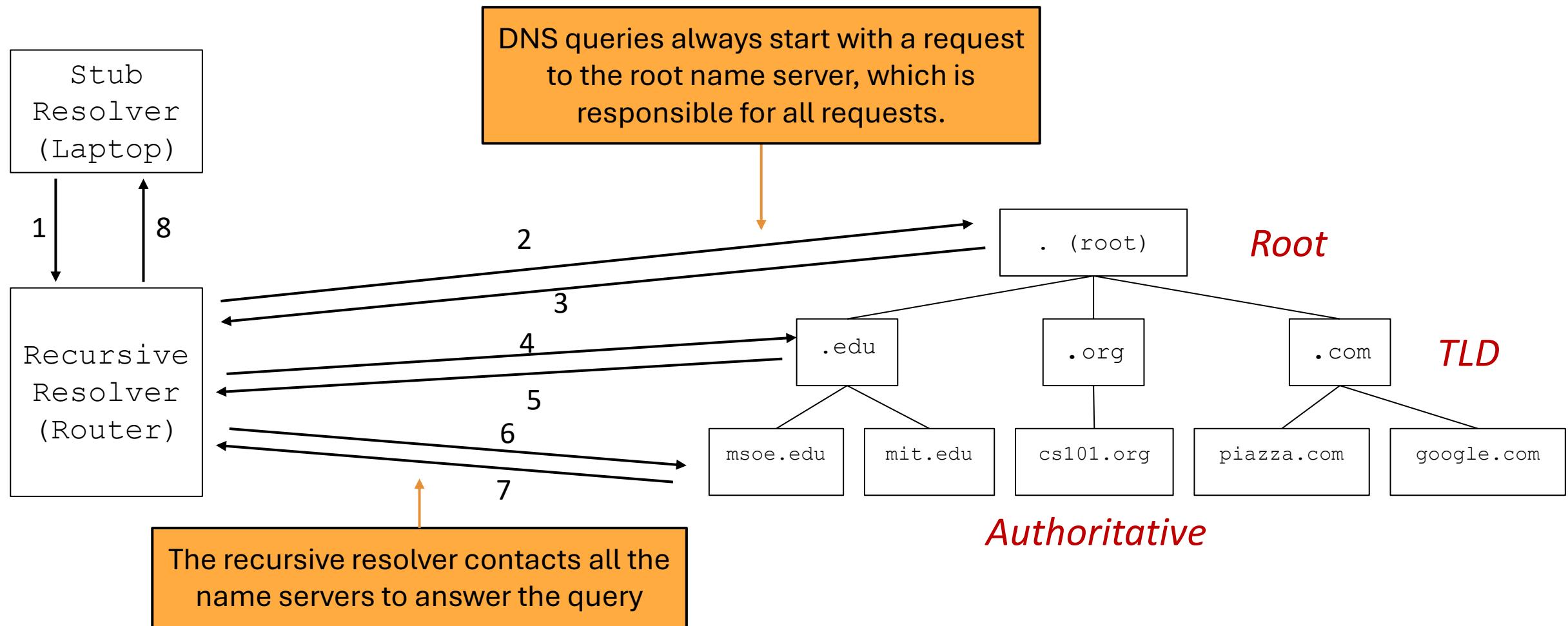
Week 12, Lecture 2: DNS Security

Roadmap

- ***Review***
 - *DNS over TLS*
 - *DNSSEC: Design Ideas*
 - *DNSSEC: Implementation*
 - *DNS Security: Summary*

Week 3 Lecture 1: DNS

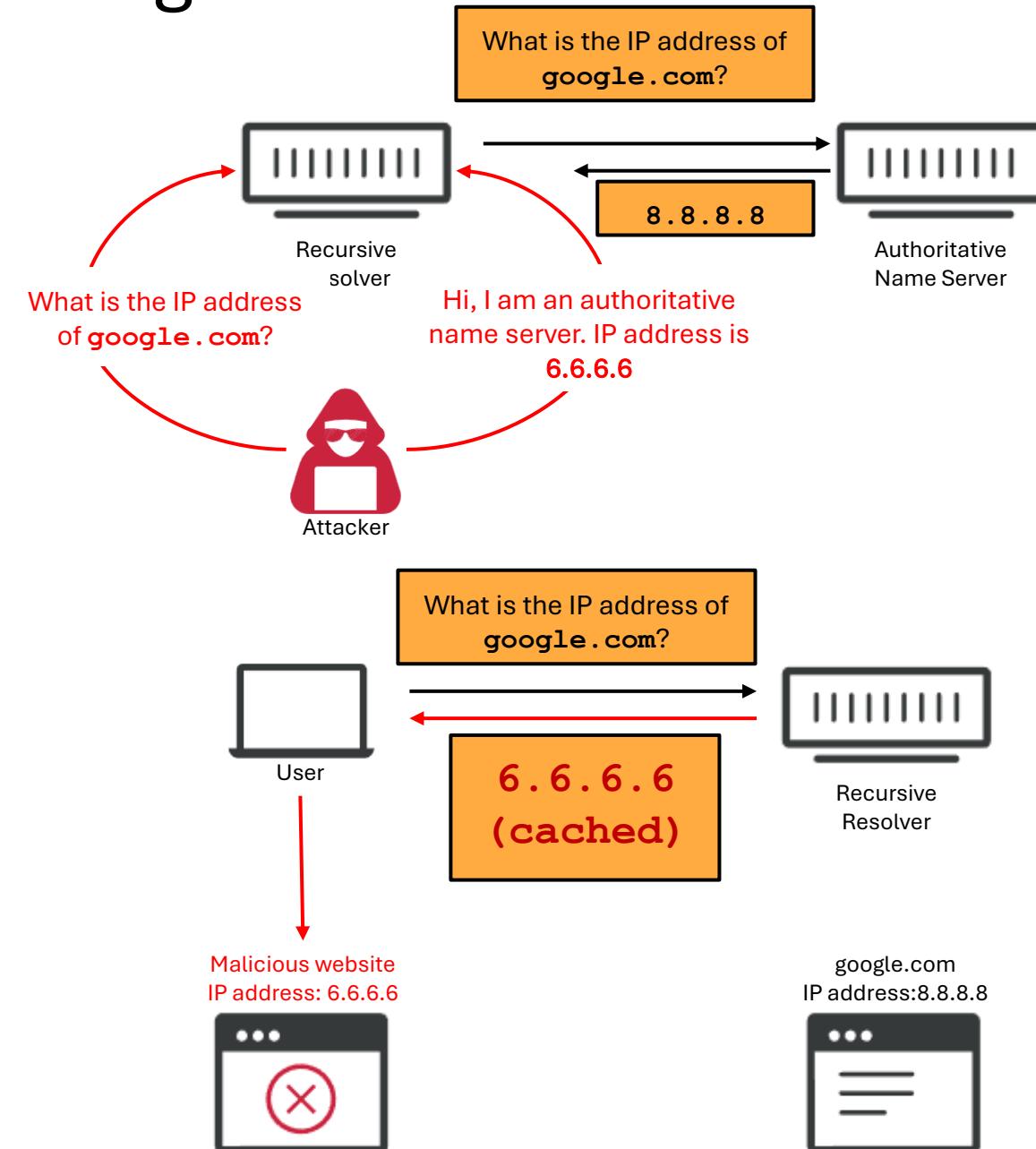
www.google.com ————— DNS —————> 74.125.25.99



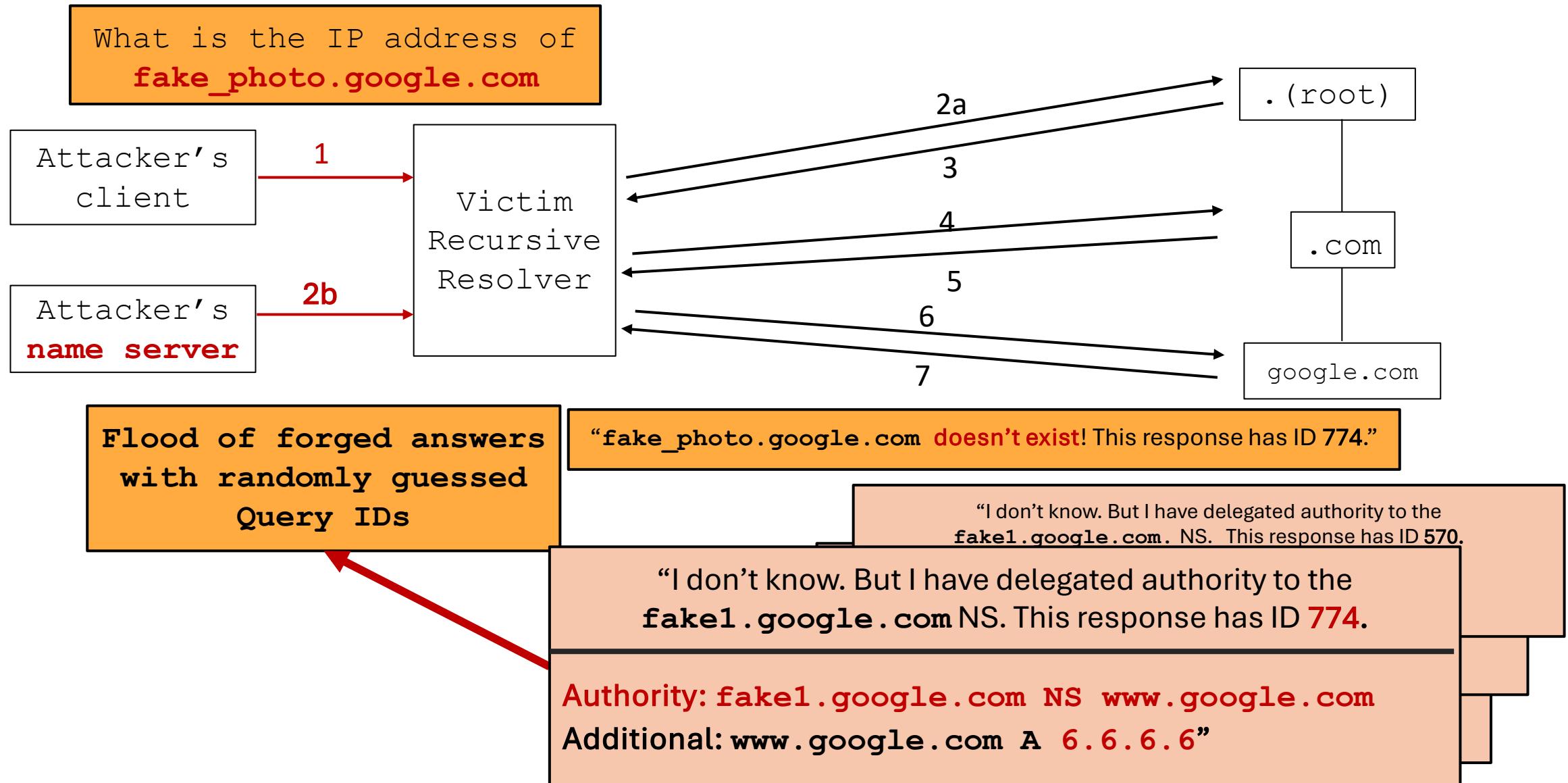
Week 9 Lecture 1: DNS Cache Poisoning Attacks

Cache poisoning attack:

- Returning a false information into the DNS cache
- The victim will cache the malicious records, “poisoning” it
- Later, DNS queries return an incorrect response, and users are directed to the wrong websites
- Example: Supply a malicious record mapping the attacker’s IP address to a legitimate domain
 - Now when the victim visits google.com, they’ll actually be sending packets to the attacker (**6.6.6.6**), who can launch the **MITM attack!**



Week 9 Lecture 1: Kaminsky Attack



Roadmap

- *Review*
- **DNS over TLS**
- *DNSSEC: Design Ideas*
- *DNSSEC: Implementation*
- *DNS Security: Summary*

Securing DNS Lookups: What do we Really Need?

We want ***integrity*** on the response

- Recall: Integrity means an attacker can't tamper with the results
- Prevent cache poisoning attacks through cryptographic verification

We want ***authenticity*** on the response

- Recall: Authenticity means verifying the response came from the legitimate DNS server
- Mitigate Kaminsky attacks and MITM attacks

We do not need confidentiality on the response

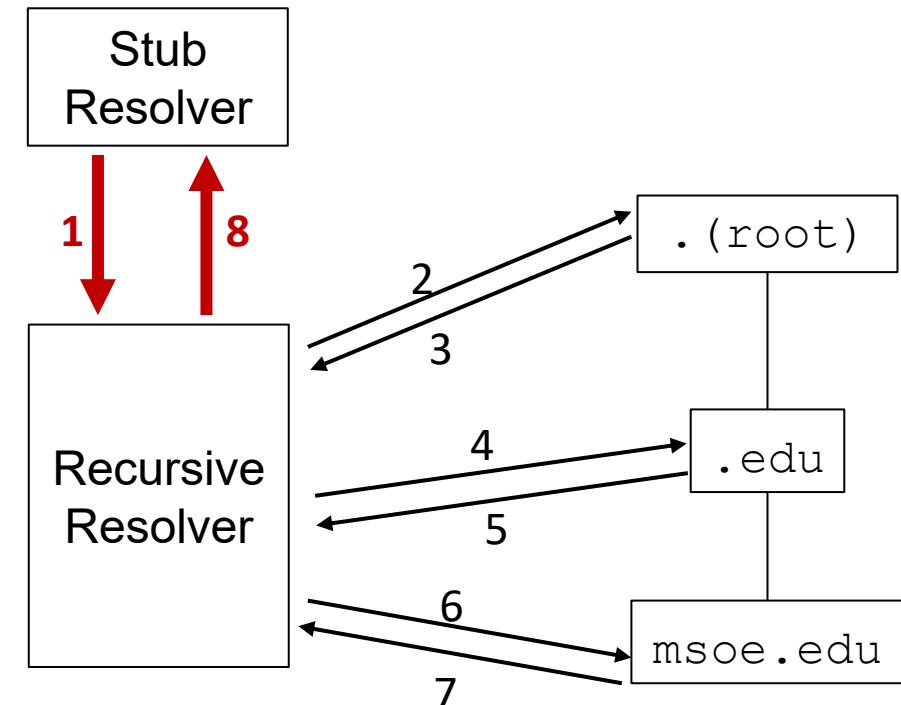
- DNS results are public: The attacker can always look up the results themselves!
- Even if the attacker couldn't see the DNS response, they can see which IP you connect to later (source and destination IPs are visible regardless, ***really?***)
 - DNS queries leak information about user activities to network observers

DNS over TLS

DNS over TLS (DoT) is one way to send DNS queries over an encrypted connection

Idea: TLS is end-to-end secure, so let's send all DNS requests and responses over TLS

- DNS traditionally uses UDP (port 53), but can use TCP; DoT uses TLS over TCP (port 853)
- Two resolvers make a TLS connection; DoT encrypts the link between resolvers.
- *Optional: Recursive resolver MAY use DoT when querying authoritative servers (not widely deployed)*



Does DoT solve our problems?

DNS over TLS: Issues

Performance: DNS needs to be lightweight and fast

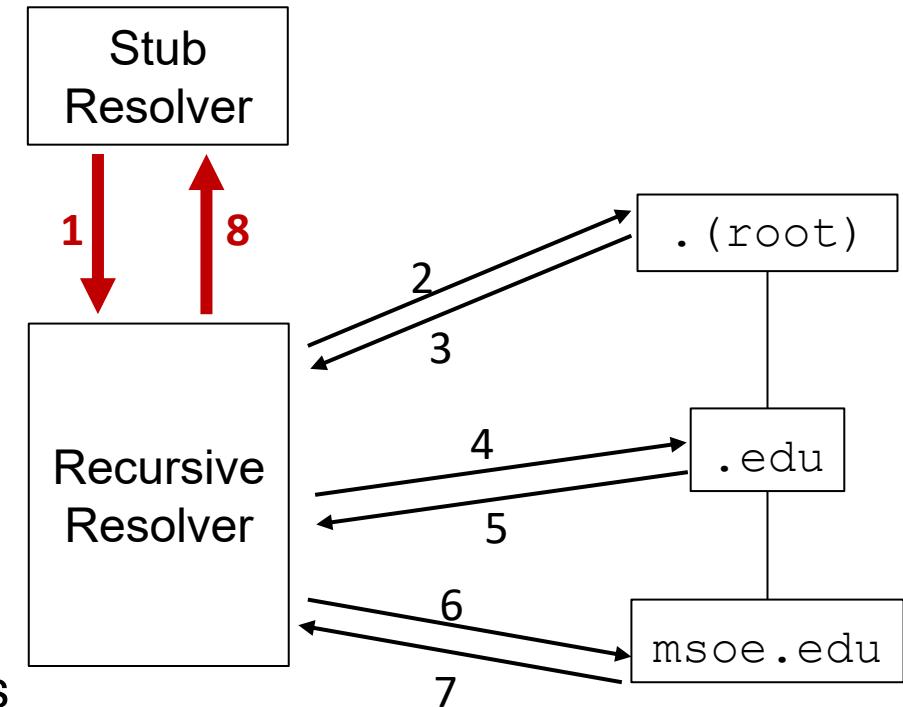
- Recall: TLS requires a long handshake

Security: DoT only *secures the link* between the stub resolver and the recursive resolver

- Malicious RR has full MITM capability
 - Can modify any responses before encryption
- User must trust their recursive resolver

Security: DoT *doesn't authenticate* authoritative responses

- Active attackers can still poison the DNS cache
- No cryptographic verification of DNS data



DNS over TLS in Practice

- Implemented by Google and Cloudflare:
 - Google Public DNS (8.8.8.8): DoT available on port 853
 - Cloudflare (1.1.1.1): DoT available on port 853
 - Want to try DoT? Install [Knot DNS tools](#) on SEED VM and follow [Cloudflare tutorial](#)
- Benefits:
 - Prevents on-path tampering of DNS traffic
 - Modern networks minimize TLS overhead (TLS 1.3); The overhead is not a big deal.
- Limitations:
 - Only defends against network attackers
 - Malicious recursive resolver can still poison responses
 - Does not **authenticate** DNS data from authoritative name servers

DNS over TLS is not enough to fully secure DNS!

DNS over TLS: Object Security and Channel Security

Channel security: Securing the communication **channel** between two end hosts

- Provides confidentiality/integrity/authenticity/anonymity for data in transit
- For example: TLS, VPN

Object security: Securing the **data** itself (in transit or in storage)

- Data is encrypted/signed/verified
- For example: Encryption, Digital Signature

Main problem of DNS over TLS:

- DoT provides channel security (stub resolver and recursive resolver)
- To fully secure DNS, we ***need object security:***
 - Cryptographically sign DNS records at the source
 - Verify signatures at the user (stub resolver), regardless of intermediaries
 - This is what **DNSSEC** provides

Roadmap

- *Review*
- *DNS over TLS*
- ***DNSSEC: Design Ideas***
- *DNSSEC: Implementation*
- *DNS Security: Summary*

DNSSEC

DNSSEC (DNS Security Extensions): An extension of the DNS protocol that ensures integrity and authenticity on the results

- Designed to cryptographically prove that returned answers are correct
- Uses a hierarchical, distributed trust system to validate records

DNSSEC is backwards-compatible

- DNSSEC is built on top of ordinary DNS
- Some, but not all name servers support DNSSEC

DNSSEC Visualization tool: [DNSViz | A DNS visualization tool](#)

DNSSEC from Scratch: Let's Build it Together

Question 1: What kind of cryptographic primitive should we use to ensure integrity on the DNS records?

- MACs (symmetric-key) or digital signatures (public-key)
- Digital signatures are the best solution: We want everyone to be able to verify integrity (not just the people with the symmetric key)

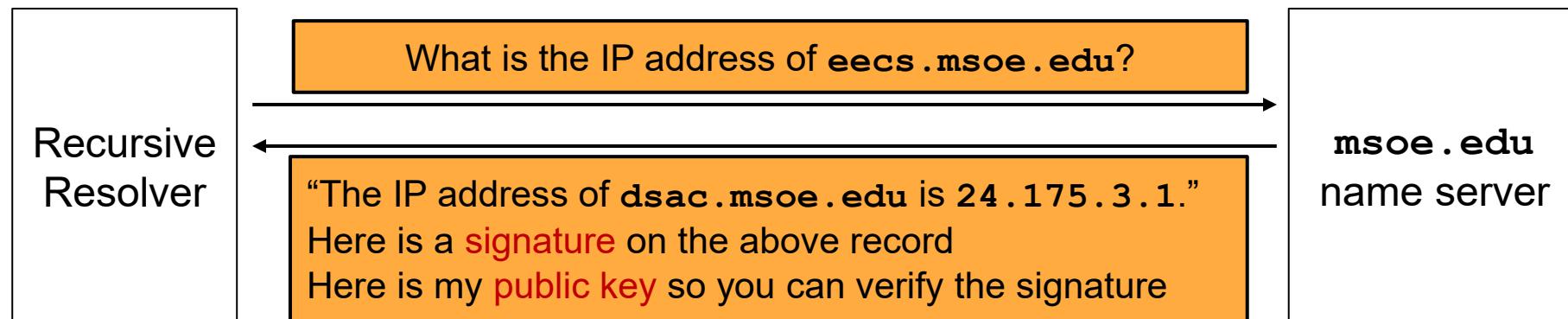
Question 2: How do we ensure the returned record is authentic?

- The name servers should sign the record with their private key
- Users verify the record with the name servers' public key

DNSSEC from Scratch: Let's Build it Together

Question 3: What does the name server need to send in order to ensure authenticity and integrity on a record?

- The DNS record
- A signature over the record, signed with the name server's private key
- The name server's public key
- How do we make sure nobody tampered with the public key?
 - ***Chain of trust!*** (but DNSSEC does not use PKI)



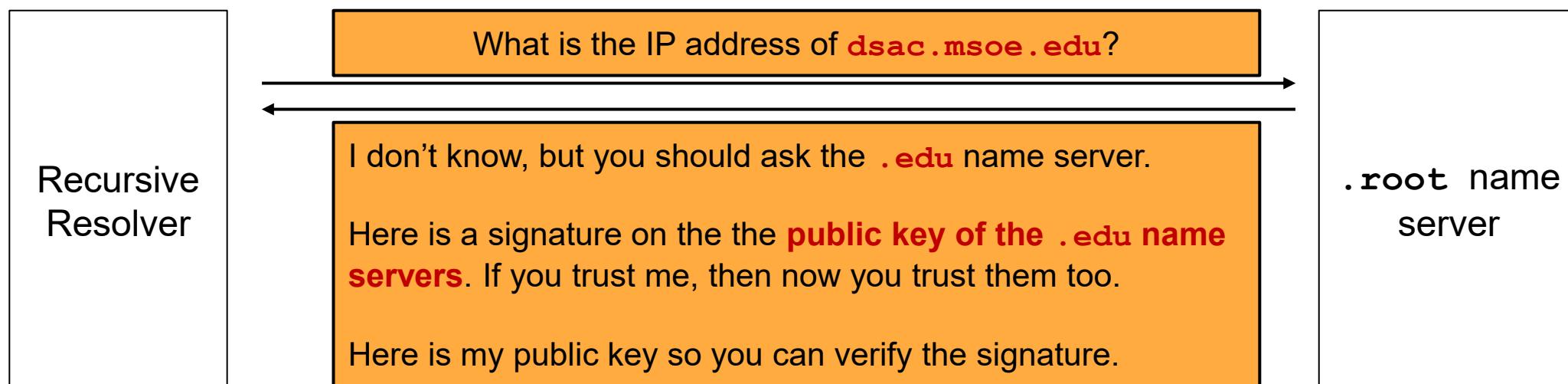
DNSSEC from Scratch: Let's Build it Together

Question 4: PKIs need a trust anchor. Who do we implicitly trust in DNSSEC?

- We implicitly trust the top of the certificate hierarchy, which is the ***root name server***
- The root name server's public key (self-signed) is **hard-coded** into resolvers

Question 5: How does a name server ***delegate*** trust to a child name server?

- Just like in a certificate chain, the parent sign the child's public key
- If you trust the parent name server, then now you trust the child name server



Roadmap

- *Review*
- *DNS over TLS*
- *DNSSEC: Design Ideas*
- ***DNSSEC: Implementation***
- *DNS Security: Summary*

Resource Record Sets (RRSETs)

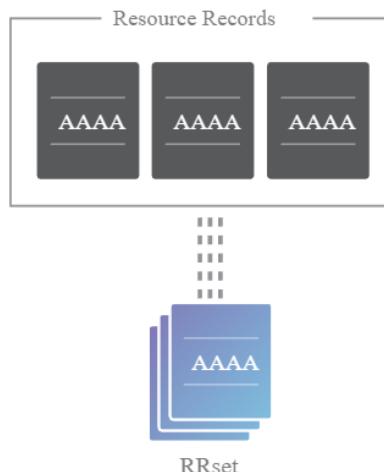
Recall: A DNS resource record has a fixed format

RR format: (name, [TTL], class, type, value)

E.g.: edu. 172800 IN NS a.edu-servers.net

In DNSSEC: A group of DNS records with the same name and type form a *Resource Record Set (RRset)*

- Example: All the type AAAA records for a given domain
- Example: *All the DNSKEY records*



mail.zone.	MX	5	server1.zone.	RRset
mail.zone.	MX	10	server2.zone.	
server1.zone.	A	10.20.30.40		RRset
	A	10.20.30.41		
	A	10.20.30.42		
server1.zone.	AAAA	2001:123:456::1		RRset
	AAAA	2001:123:456::2	[No Title]	
server2.zone.	A	11.22.33.44		RRset

RRSETs will be useful for simplifying signatures

- Instead of signing every record separately, **name server can sign an entire RRset at once**

New DNSSEC Record Types

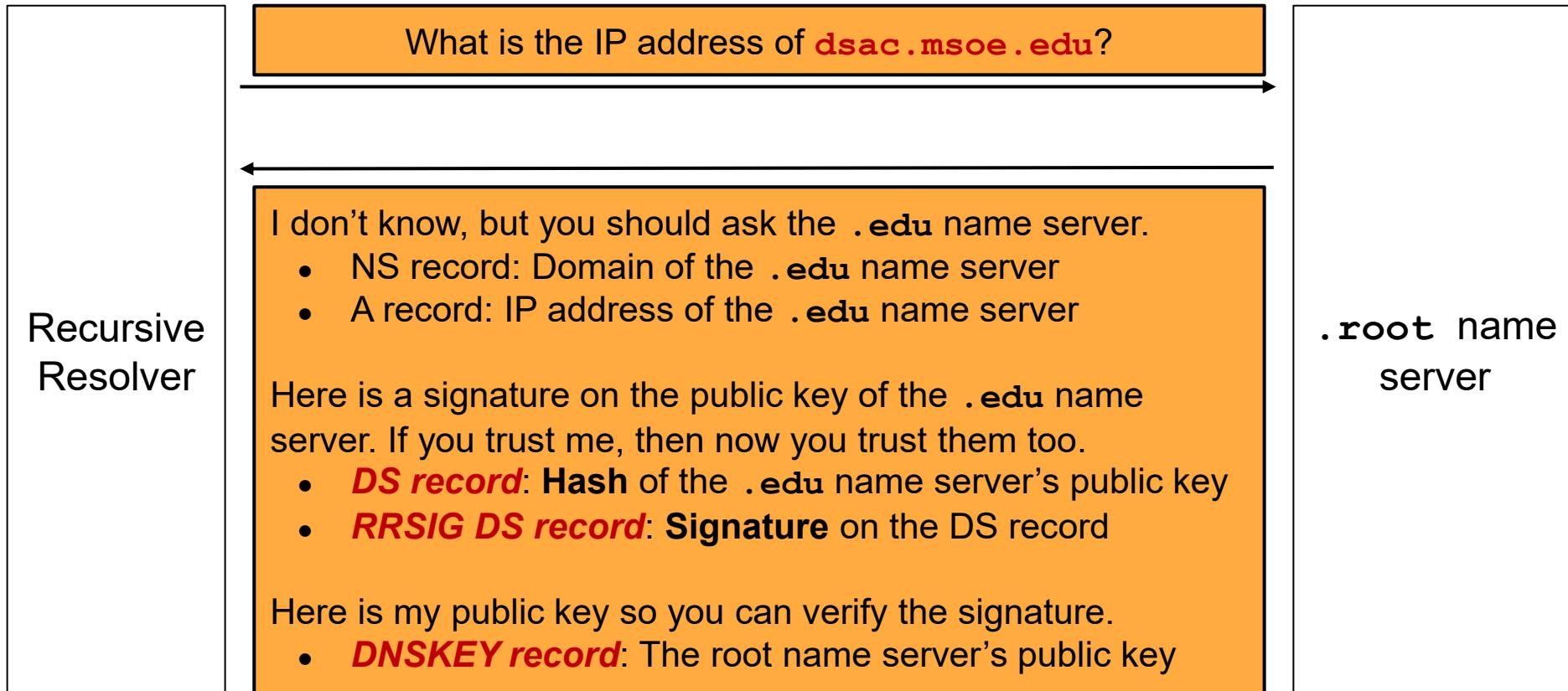
We need new record types to send cryptographic information in DNSSEC packets

- DNSKEY:
 - Contains the name server's **own public keys**
- DS (delegated signer):
 - The parent signs the DS record to ***delegate trust*** to child
 - Contains the **signer's (owner's) name**
 - Contains the **hash of child's public keys**
- RRSIG (resource record signature):
 - Contain a **Type Covered** field that specifies which type of record it covers
 - RRSIG (covering DNSKEY): Contains a signature on DNSKEY records
 - RRSIG (covering DS): Contains a signature on DS records
 - RRSIG (covering A and AAAA): Contains a signature on resource records (sign an entire RRset)

Note: DNSSEC requires a significant amount of metadata; the above description is oversimplified!

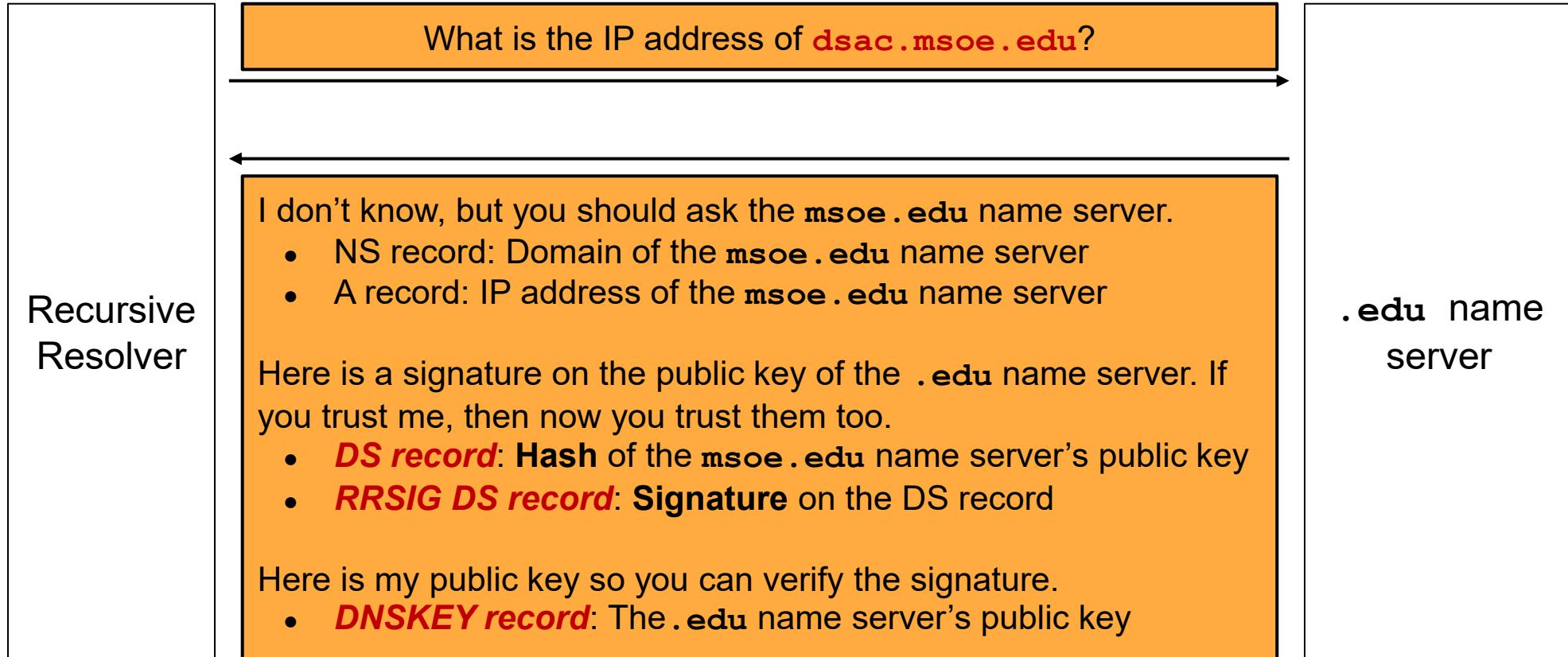
DNSSEC Lookup Steps

- Recall: We implicitly trust the top of the certificate hierarchy, which is the ***root name server***
- Recall: The root name server's public key (self-signed) is **hard-coded** into resolvers



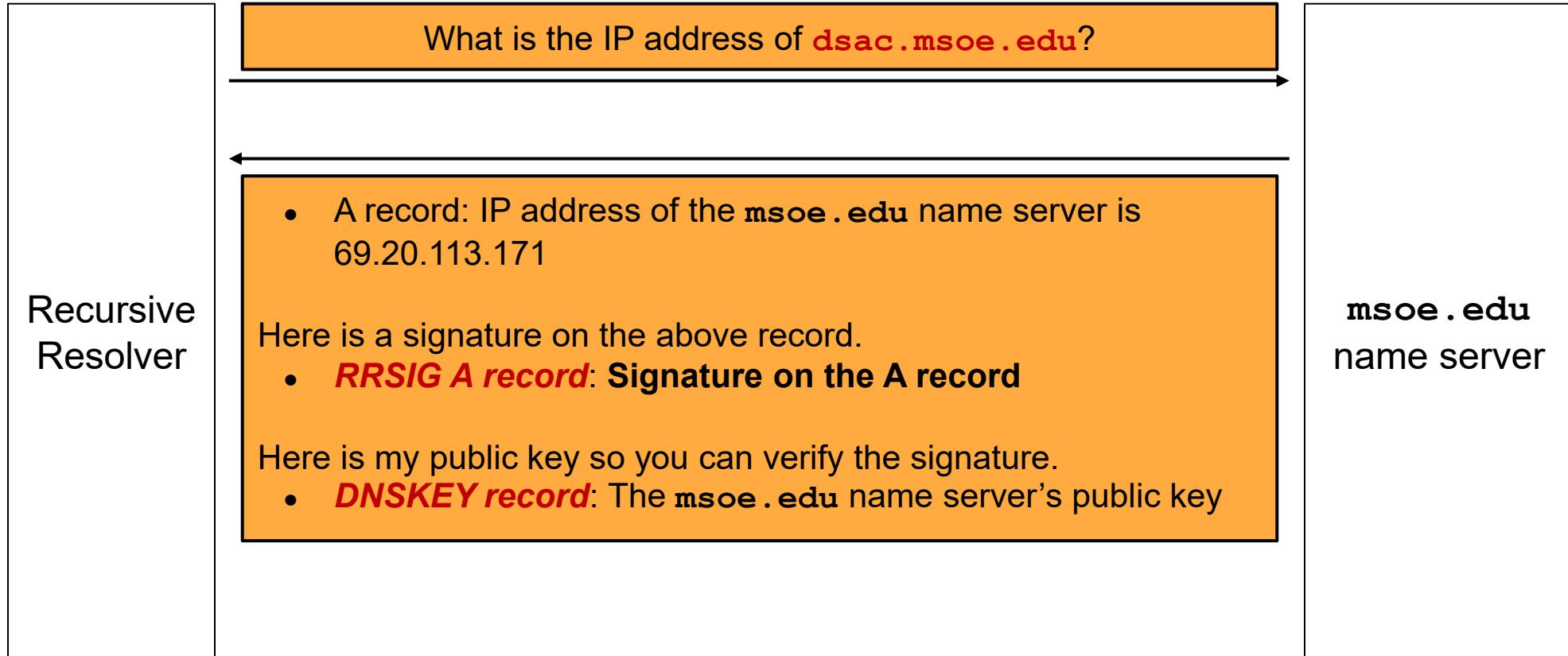
DNSSEC Lookup Steps

- Recall: We implicitly trust the top of the certificate hierarchy, which is the ***root name server***
- Recall: The root name server's public key (self-signed) is **hard-coded** into resolvers



DNSSEC Lookup Steps

- Recall: We implicitly trust the top of the certificate hierarchy, which is the ***root name server***
- Recall: The root name server's public key (self-signed) is **hard-coded** into resolvers



New DNSSEC Keys

Problem 1: What if a name server wants to change its key pair?

- A key change is necessary if, for example, an attacker steals the private key of a name server.
- Option 1: Notify its parent name server; the parent change the DS record
 - Synchronization and communication cost
 - Time consuming and easily go wrong

New DNSSEC Keys

Option 2: Each DNSSEC name server generates *two public/private key pairs*

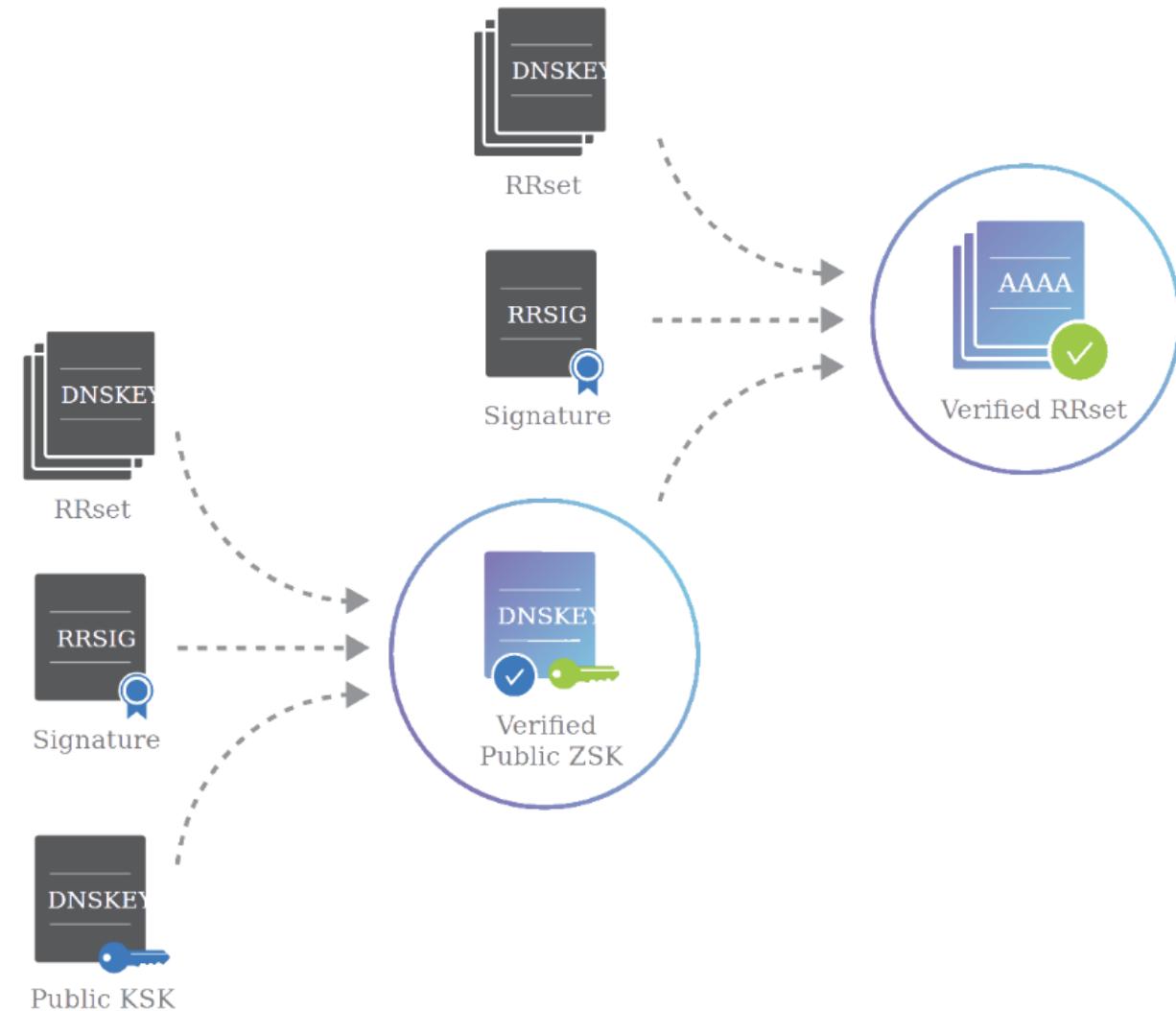
- **Key signing key (KSK):** sign the zone signing key
- **Zone signing key (ZSK):** sign everything else (e.g., RRset)

Name server sends:

1. The public KSK (endorsed by the parent)
2. The public ZSK
3. A signature on the public ZSK (signed by KSK)

The DNS resolver:

1. Uses the public KSK to verify the signature
2. Accepts the public ZSK
3. Uses ZSK to verify RRSIG and accepts RRSET



DNSSEC: Trust Delegation

- KSK is the “parent” and the ZSK is the “child”; both within one name server
- KSK endorses the ZSK by signing the public ZSK
- Each color (blue, green, orange) represents a name server
- The lighter shade represents records signed with the KSK
- The darker shade represents records signed with the ZSK

Trust anchor: root name server & its KSK

1. [DNSKEY] Public KSK (Root KSK)
2. [DNSKEY] Public ZSK
3. [RRSIG] Signature on (2), signed by KSK

Root, KSK

4. [DS] Hash of (6. child's public KSK)
5. [RRSIG] Signature on (4), signed by ZSK

Root, ZSK

6. [DNSKEY] Public KSK
7. [DNSKEY] Public ZSK
8. [RRSIG] Signature on (7), signed by KSK

.edu, KSK

9. [DS] Hash of (11. child's public KSK)
10. [RRSIG] Signature on (9), signed by ZSK

.edu, ZSK

11. [DNSKEY] Public KSK
12. [DNSKEY] Public ZSK
13. [RRSIG] Signature on (12), signed by KSK

msoe.edu, KSK

14. [A] Answer record
15. [RRSIG] Signature on (14), signed by ZSK

msoe.edu, ZSK

DS (delegated signer):

- Contains the **signer's (owner's) name**
- Contains the **hash of child's public keys**

DNSSEC: Trust Delegation

Trust anchor: root name server & its KSK

Verification process:

1. Trust anchor:

- Root KSK is hard-coded in resolvers
- Root KSK signs **DNSKEY RRset**, so we can **verify** root's **ZSK**

• Dark blue:

1. We trust the root's ZSK. The root's ZSK signs the hash of .edu's KSK, so now we can **verify** .edu's KSK.

2. **Note:** the root's ZSK also signs its own **NS RRset** at the zone

2. Steps 6-10: Similar as steps 1-5.

1. [DNSKEY] Public KSK (Root KSK)
2. [DNSKEY] Public ZSK
3. [RRSIG] Signature on (2), signed by KSK

Root, KSK

4. [DS] Hash of (6. child's public KSK)
5. [RRSIG] Signature on (4), signed by ZSK

Root, ZSK

6. [DNSKEY] Public KSK
7. [DNSKEY] Public ZSK
8. [RRSIG] Signature on (7), signed by KSK

.edu, KSK

9. [DS] Hash of (11. child's public KSK)
10. [RRSIG] Signature on (9), signed by ZSK

.edu, ZSK

11. [DNSKEY] Public KSK
12. [DNSKEY] Public ZSK
13. [RRSIG] Signature on (12), signed by KSK

msoe.edu, KSK

14. [A] Answer record

14. [A] Answer record
15. [RRSIG] Signature on (14), signed by ZSK

msoe.edu, ZSK

DNSSEC: Trust Delegation

Trust anchor: root name server & its KSK

Verification process:

3. msoe.edu name server:

- Light orange: We verify the msoe.edu's KSK; msoe.edu's KSK signs msoe.edu's ZSK; So now we trust msoe.edu's ZSK (12-13).
- Dark orange: We trust msoe.edu's ZSK. msoe.edu's ZSK ***signs the type A answer records***, so we trust the final answer.

1. [DNSKEY] Public KSK (Root KSK)
2. [DNSKEY] Public ZSK
3. [RRSIG] Signature on (2), signed by KSK

Root, KSK

4. [DS] Hash of (6. child's public KSK)
5. [RRSIG] Signature on (4), signed by ZSK

Root, ZSK

6. [DNSKEY] Public KSK
7. [DNSKEY] Public ZSK
8. [RRSIG] Signature on (7), signed by KSK

.edu, KSK

9. [DS] Hash of (11. child's public KSK)
10. [RRSIG] Signature on (9), signed by ZSK

.edu, ZSK

11. [DNSKEY] Public KSK
12. [DNSKEY] Public ZSK
13. [RRSIG] Signature on (12), signed by KSK

msoe.edu, KSK

- 14. [A] Answer record**
15. [RRSIG] Signature on (14), signed by ZSK

msoe.edu, ZSK

DNSSEC: Offline Signature Generation

Problem 2: DNS is designed to be fast and lightweight.

However, public-key cryptography is slow!

Solution: Offline signatures → The name server computes signatures (for RRSET, DNSKEY) in advance; serves the signatures when they are needed

Benefit: Efficiency

- Generating a signature each time a user requests it is slow (and can lead to DoS attacks)

Benefit: Availability and Security

- Redundancy: One secure signature generation system, and many mirrored name servers providing the same records and signatures
- If the signature generation system is separate from the name server, compromising the name server is not enough!

DNSSEC: Signing Non-Existent Domains

Problem 3: What if the user queries for a domain that doesn't exist?

Option 1: Don't authenticate nonexistent domain (NXDOMAIN) responses

- Issue: If NXDOMAIN responses don't have to be signed, the attacker can still *spoof* NXDOMAIN responses and cause denial-of-service (DoS)

Option 2: Keep the private key in the name server, ask it signs NXDOMAIN responses

- Issue: Signing in real time is slow
- Issue: Name servers stores the private key, which is an issue if they are malicious or hacked

We need a way that can *prove that a domain doesn't exist* ahead of time

New DNSSEC Record Types: NSEC

NSEC ([and NSEC3](#), [and NSEC5](#)): Prove nonexistence of a DNS record

- NSEC record: Next Secure Record
- Provide two adjacent domains **alphabetically**, so that the resolver knows that no domain **in the middle** exists
- For example, suppose a website has 3 subdomains (hostnames):
 - *b.example.com* *g.example.com* *q.example.com*
 - Query for “*c.example.com*”, name server returns a **signed NSEC record**: “No domains exist between *b.example.com* and *g.example.com*”
 - Query for “*x.example.com*”, name server returns a **signed NSEC record**: “No domains exist between *q.example.com* and *b.example.com*” (wrapping around from z to a)

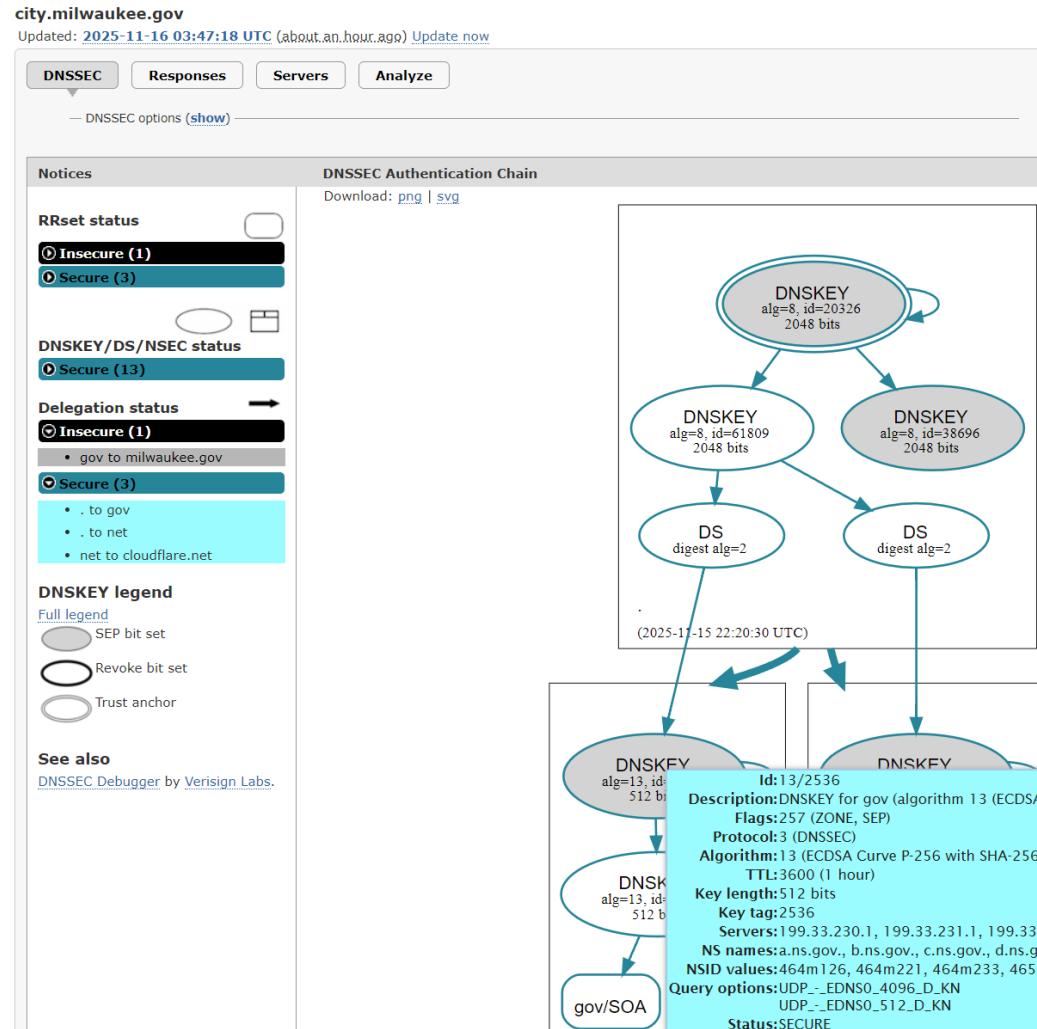
Roadmap

- *Review*
- *DNS over TLS*
- *DNSSEC: Design Ideas*
- *DNSSEC: Implementation*
- ***DNS Security: Summary***

DNSSEC in Practice

DNSSEC Visualization tool:

[DNSViz | A DNS visualization tool](#)



DNSSEC lookup walkthrough (PowerShell):

\$ dig +norecurse +dnssec [www.msoe.edu](#) "@198.41.0.4"

```
PS C:\Users\liao> dig +norecurse +dnssec www.msoe.edu "@198.41.0.4"

; <>> DiG 9.16.28 <>> +norecurse +dnssec www.msoe.edu @198.41.0.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37970
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 15, ADDITIONAL: 27

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; QUESTION SECTION:
;www.msoe.edu.           IN      A

;; AUTHORITY SECTION:
edu.          172800  IN      NS      d.edu-servers.net.
edu.          172800  IN      NS      b.edu-servers.net.
edu.          172800  IN      NS      f.edu-servers.net.
edu.          172800  IN      NS      h.edu-servers.net.
edu.          172800  IN      NS      l.edu-servers.net.
edu.          172800  IN      NS      j.edu-servers.net.
edu.          172800  IN      NS      e.edu-servers.net.
edu.          172800  IN      NS      c.edu-servers.net.
edu.          172800  IN      NS      a.edu-servers.net.
edu.          172800  IN      NS      g.edu-servers.net.
edu.          172800  IN      NS      i.edu-servers.net.
edu.          172800  IN      NS      m.edu-servers.net.
edu.          172800  IN      NS      t.edu-servers.net.
edu.          86400   IN      DS      6B5AC52B4B345357687271E85353082741F1CF3 D06A4C1D
.edu.         86400   IN      RRSIG   6B5AC52B4B345357687271E85353082741F1CF3 D06A4C1D 0251115160000 61809 . WLDpVjLW5szBaxcS0xH2d2ogpeG70KHig0axQXeoIsr0ZNv1H0G7
c37 NG6IqKi85Nr0jmAKDL3Nj0g7k5UY0g95AuXq8YmL1N9mC7vr8QLOVN07 qbpgeq1epqjaXu
b01R2q0xINO/ni0M6irjjAL1XP7vksoK94ScopFjp a6YqoXN+oVbpneGHqNyooi5Jn7/XzLuqv3
GAloTZEyHU0CvfwY1ekSr 0mtcgrcPPMJzlmir+82XrWTOBDNxU5dSrJoU9V8ImogfctgLp+Bf2
4es a5i6PkBi+LPjh+n9nD0XgSf+wS+xGXNhzRfVKG1sGr54+Um830a/Gnc1 hu3ElA==

;; ADDITIONAL SECTION:
.d.edu-servers.net.    172800  IN      A      192.31.80.30
.d.edu-servers.net.    172800  IN      AAAA     2001:500:856e::30
.b.edu-servers.net.    172800  IN      A      192.33.14.30
.b.edu-servers.net.    172800  IN      AAAA     2001:503:231d::2:30
.f.edu-servers.net.    172800  IN      A      192.35.51.30
.f.edu-servers.net.    172800  IN      AAAA     2001:503:d414::30
.h.edu-servers.net.    172800  IN      A      192.54.112.30
.h.edu-servers.net.    172800  IN      AAAA     2001:502:8cc::30
```

DNSSEC: Summary

DNSSEC: An extension of the DNS protocol that ensures integrity on the results

- Provides object security (unlike DNS over TLS, which provides channel security)
- Uses a hierarchical public key infrastructure to delegate trust from the trust anchor (root name server)
- Uses signatures to cryptographically verify records

DNSSEC Implementation:

- Each name server replies with its public key (DNSKEY type record)
- When delegating trust, each name server signs the public key of the next name server (DS and RRSIG type records)
- When providing a final answer, the name server signs the final answer (RRSIG type)
- Zones are split into key-signing keys and zone-signing keys
- NSEC signs a message saying no domains exist alphabetically between two records

DNS Security Solutions: Comparison

DoT (DNS over TLS):

- ✓ Provides confidentiality (privacy)
- ✓ Protects against on-path eavesdropping
- ✗ Requires trust in recursive resolver
- ✗ No authentication of DNS data

DNSSEC:

- ✓ Provides integrity and authenticity
- ✓ Cryptographically verifies DNS data
- ✓ Works with any name server
- ✗ No confidentiality
- ✗ Complex key management

DoT + DNSSEC: Best of both worlds!

CSC 3511 Security and Networking

Week 13, Lecture 1: Anonymity and Tor

Roadmap

- ***Anonymity***
- *Proxies and VPNs*
- *Tor*
- *Tor Onion Services*

Anonymity

Anonymity: Concealing your identity

- *Wikipedia*: “Anonymity describes situations in which the acting person's identity is unknown”
- Anonymous communication on the Internet: The identity of the source and/or destination are concealed

Anonymity is not confidentiality

- Confidentiality hides the **contents** of the communication
- Anonymity hides the **identities** of who is communicating with whom

Anonymity on the Internet is hard

- Packets contain the source IP address and destination IP address

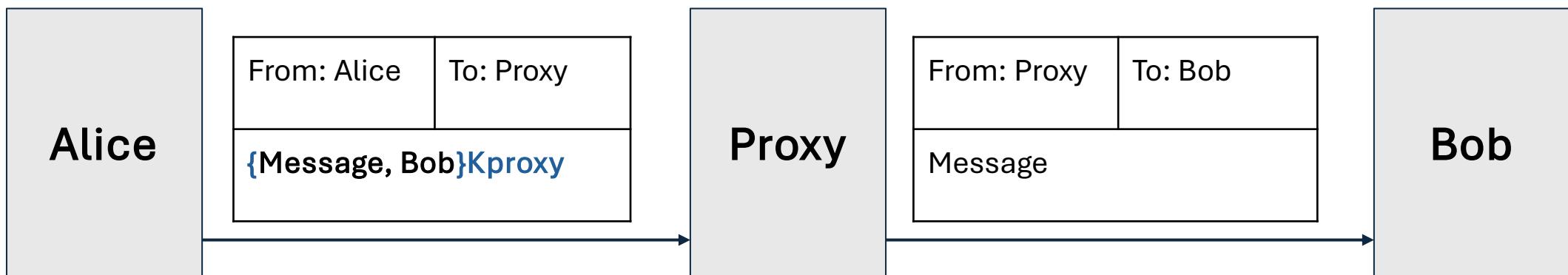
Main strategy for anonymity: **Ask someone else to send messages for you**

Roadmap

- *Anonymity*
- ***Proxies and VPNs***
- *Tor*
- *Tor Onion Services*

Proxies

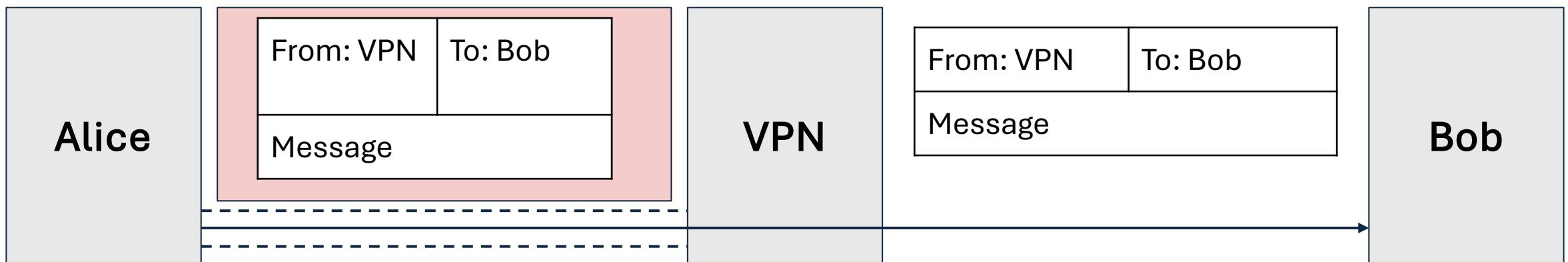
- Alice wants to send a message to Bob
 - Bob shouldn't know the message is from Alice
 - An eavesdropper (Eve) cannot deduce that Alice is talking to Bob
- **Proxy:** A third party that relays our Internet traffic, e.g., HTTP, DNS, ...
 - Alice sends the message to the proxy, and the proxy forwards the message to Bob
 - The recipient's name (and optionally the message) is encrypted (optional, not mandatory), so an eavesdropper does not see a packet with both Alice and Bob's identities in plaintext
 - Bob receives the message from the proxy, with **no indication** it came from Alice
 - Mask the original IP address



Virtual Private Networks (VPNs)

VPNs: A virtual, encrypted, point-to-point tunnel

- Creates an **encrypted** tunnel between your *device* and a *remote VPN server*, which acts as an intermediary for your internet traffic
- Creates an alternative use case:
 - Appear as though you are coming from the virtually connected network instead of your real network!
 - E.g., a remote worker using a corporate VPN to securely access company resources
 - Proxies operate at the *application layer*, while VPNs operate at the *network layer*



Proxies and VPNs: Issues

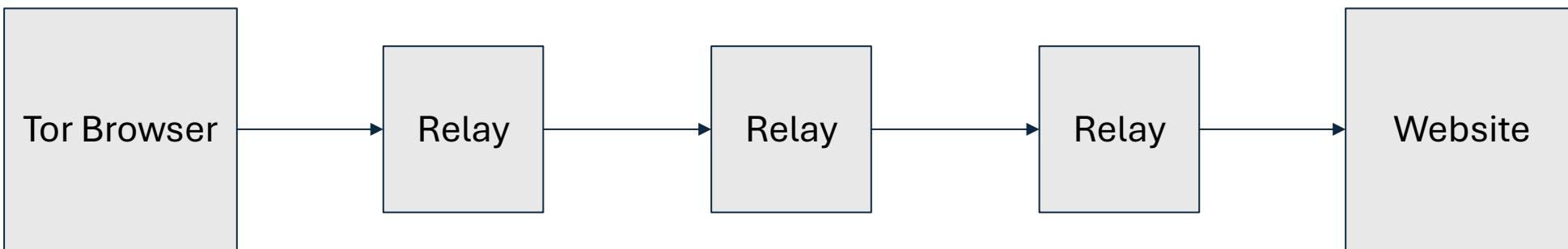
- Performance
 - Sending a packet requires additional hops across the network
- Cost
 - VPNs can cost \$80 to \$200 per year
- *Trusting the proxy*
 - The proxy can see the sender and recipient's identities
 - Attackers might convince the proxy to tell them about your identity

Roadmap

- *Anonymity*
- *Proxies and VPNs*
- ***Tor***
- *Tor Onion Services*

The Onion Router (Tor)

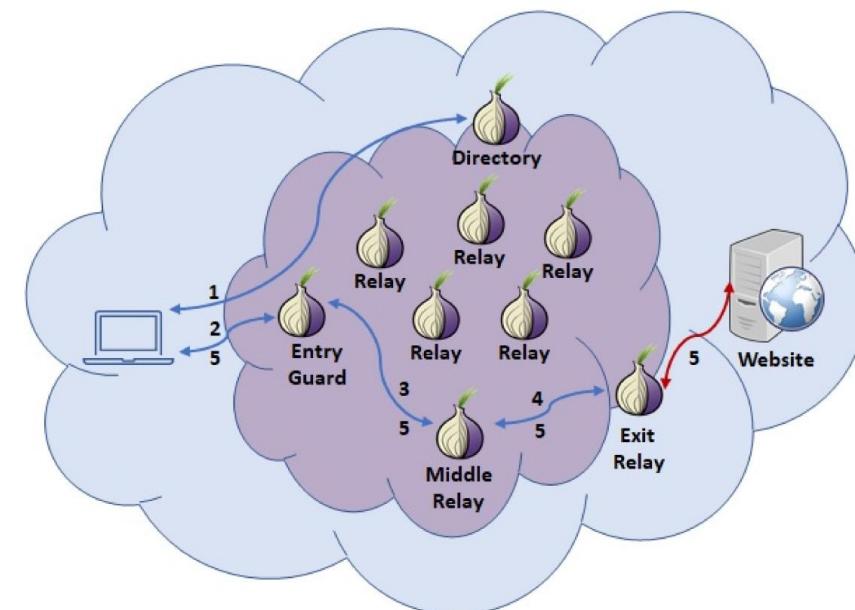
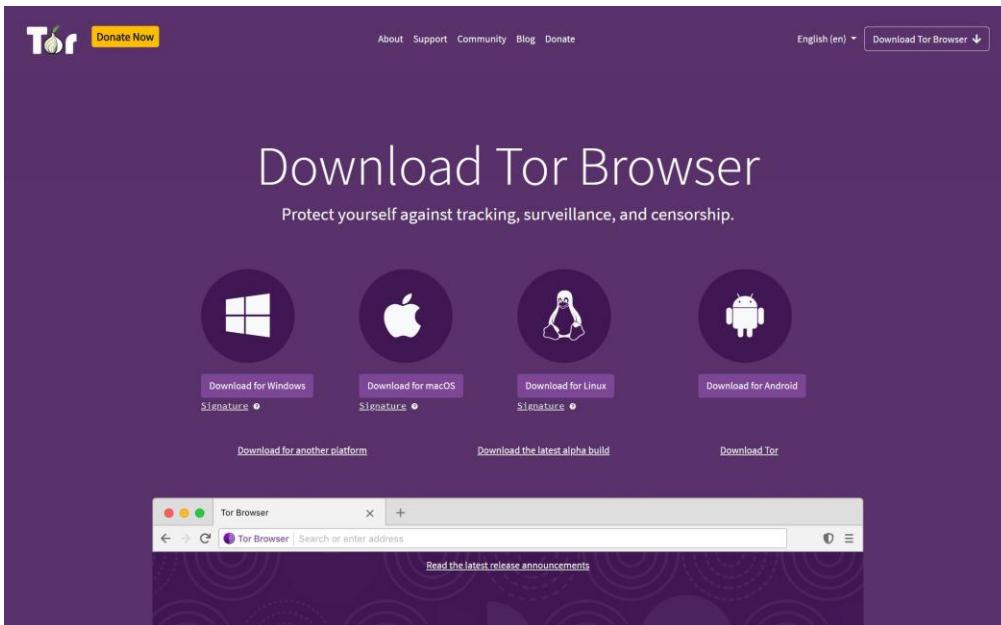
- *Idea:* Send the packet through multiple proxies instead of just one proxy
- **Tor:** A network that uses **multiple proxies** (relays) to enable anonymous communications



The Onion Router (Tor)

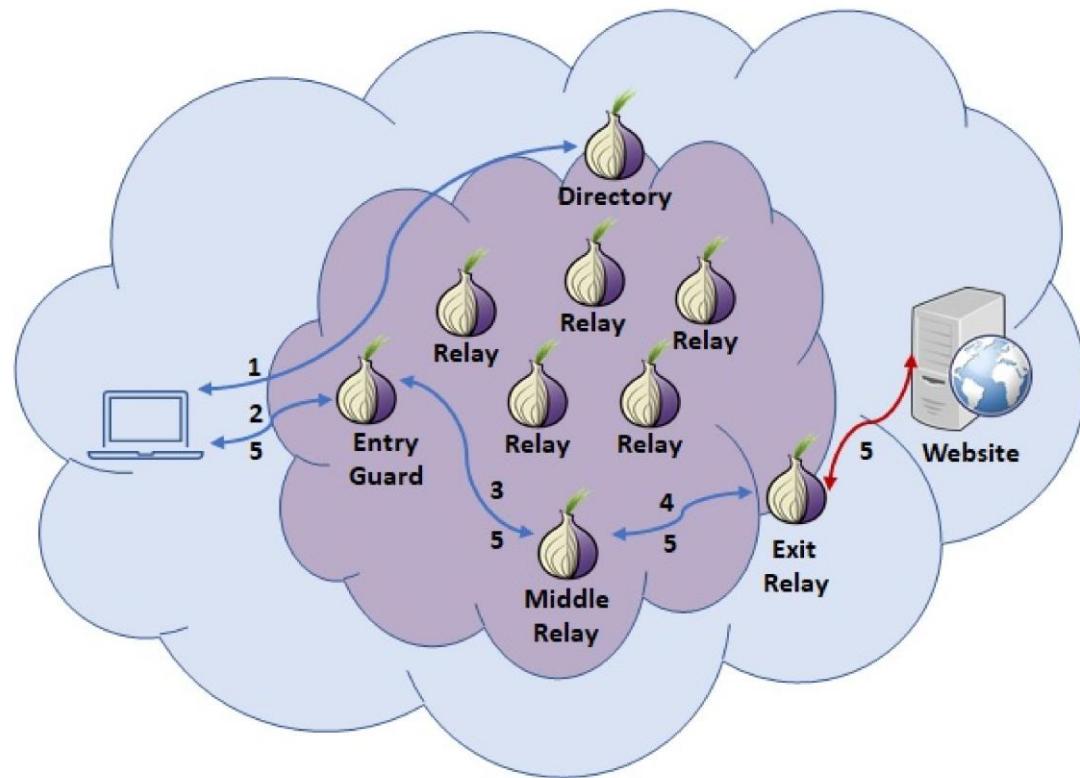
Components of Tor

- Tor Browser: A web browser configured to connect to the Tor network (based on Firefox)
- Tor network: A network of many *Tor relays (proxies)* for forwarding packets
- Directory server: A server that holds and distributes a list of all active Tor relays (and their public keys → why? MITM attacks!); also known as Directory Authorities
- Tor onion services: Servers that can only be reached through the Tor network
- Tor bridges: Tor relays that try to hide the fact that a user is connecting to the Tor network



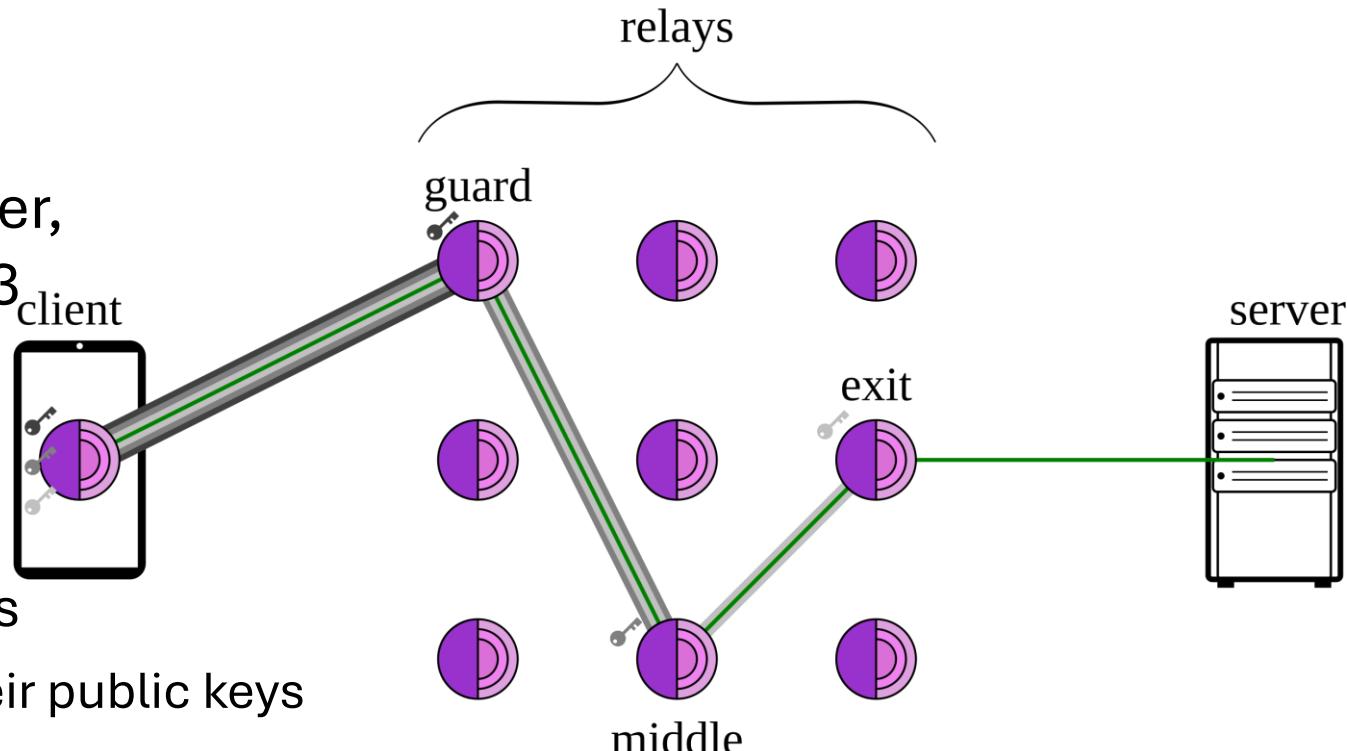
The Goals of Tor

- Security: Client anonymity and censorship resistance
 - Optional: Server anonymity with onion services
- Performance: Low latency; communication should be fast.
- Tor preserves anonymity against local adversaries
 - Example: An on-path attacker sees Alice send a message to a Tor relay, but not the final destination of the message
 - Example: The server should not know the identity of the client



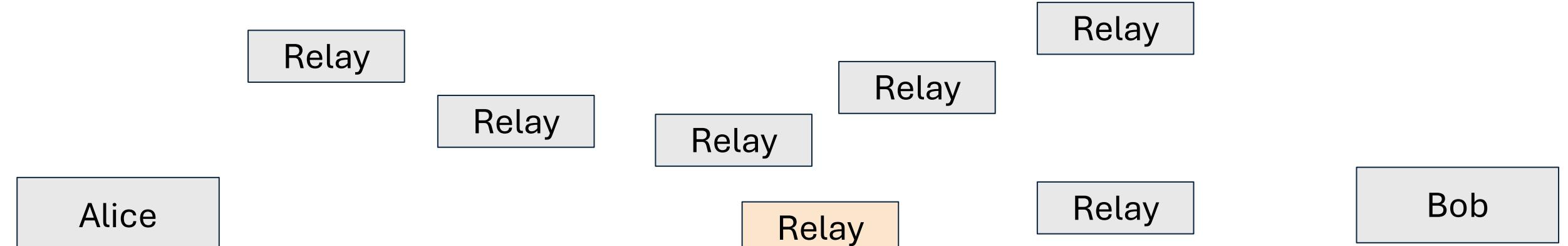
Tor Circuits

To communicate anonymously with a server, the Tor client forms a circuit consisting of 3 relays (by default)



1. Query the directory server for a list of relays
 - Directory server: Lists all Tor relays and their public keys
2. Choose 3 relays to form a Tor circuit
3. Connect to *the first relay*, forming an end-to-end TLS connection
4. Connect to the second relay *through the first relay*, forming an end-to-end TLS connection
5. Connect to the third relay *through the second relay*, forming an end-to-end TLS connection
6. Connect to the web server
 - If the web server is using HTTPS, then an end-to-end TLS connection will be formed through the third relay

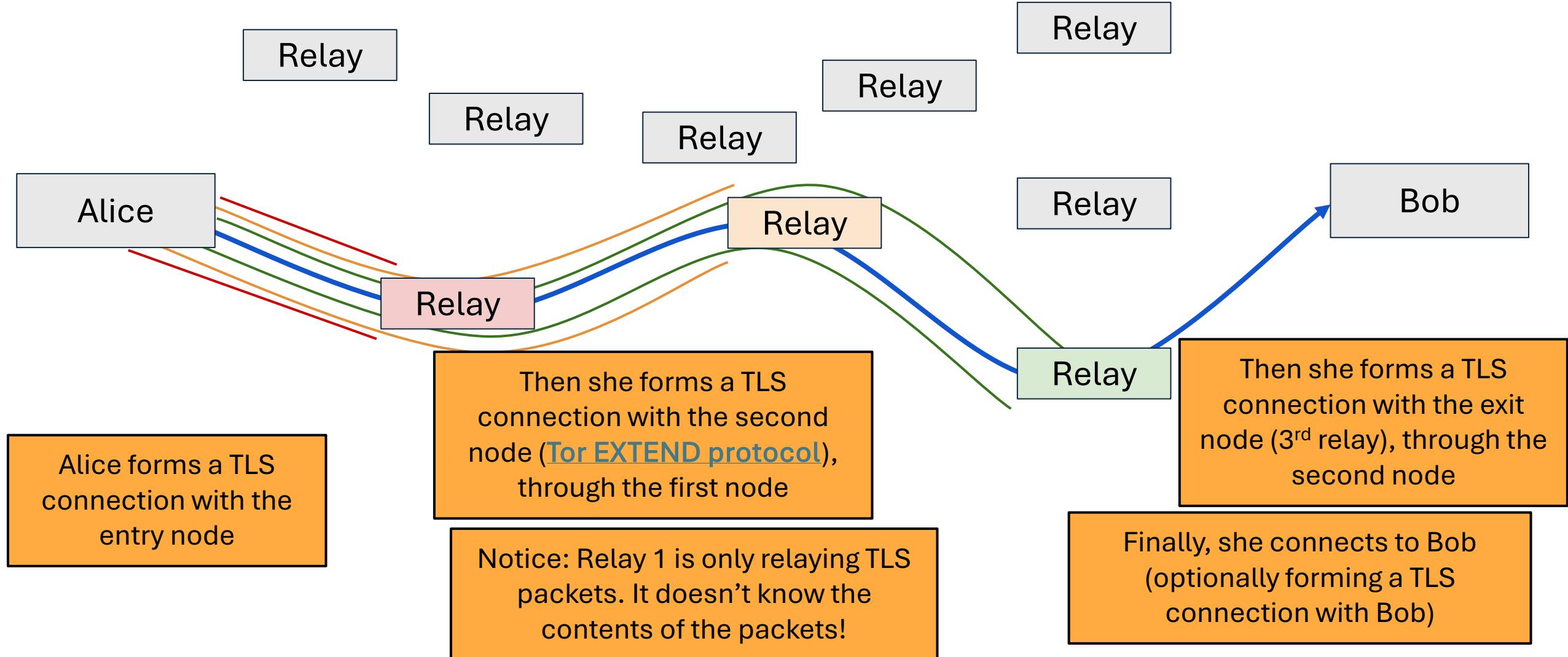
Tor Circuits



Suppose Alice wants to talk to Bob anonymously.

Alice queries the **directory server** and chooses 3 relays

Tor Circuits



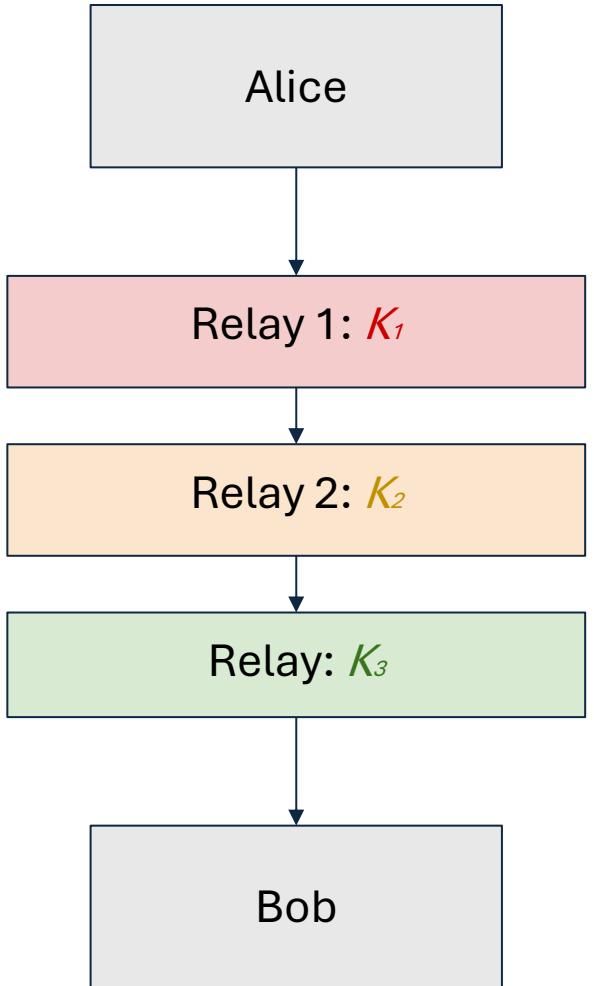
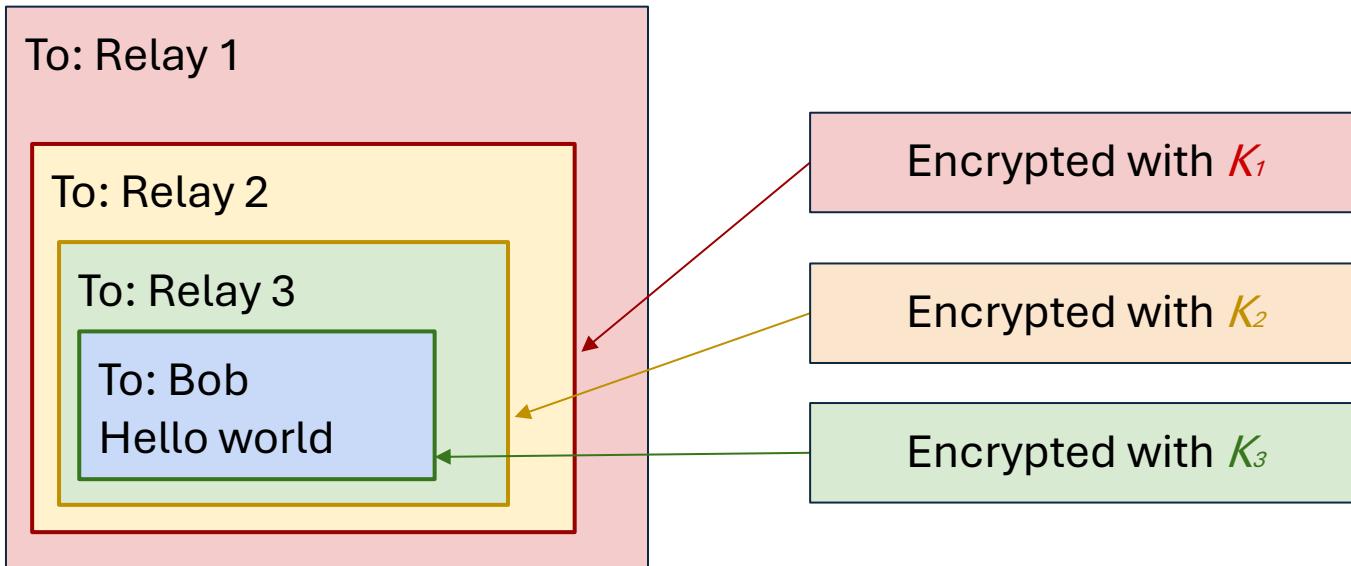
What a Relay Does

Function of the relays:

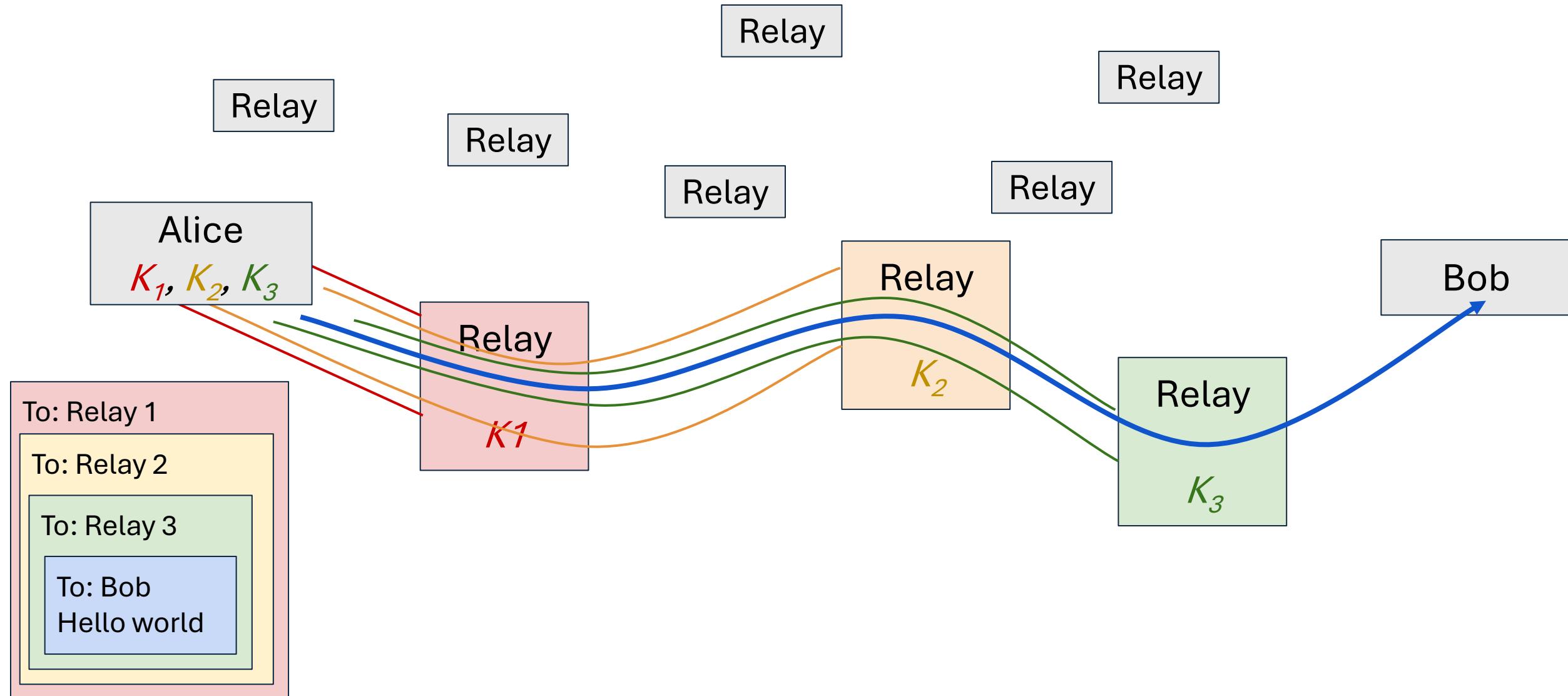
- Listen for someone to initiate a TLS connection
- When receiving a packet, decrypt using the key obtained through TLS
- If the destination of the packet is another relay, forward the packet to the next relay
- If the destination of the packet is an external server, forward the packet to that server

Tor Packet Construction

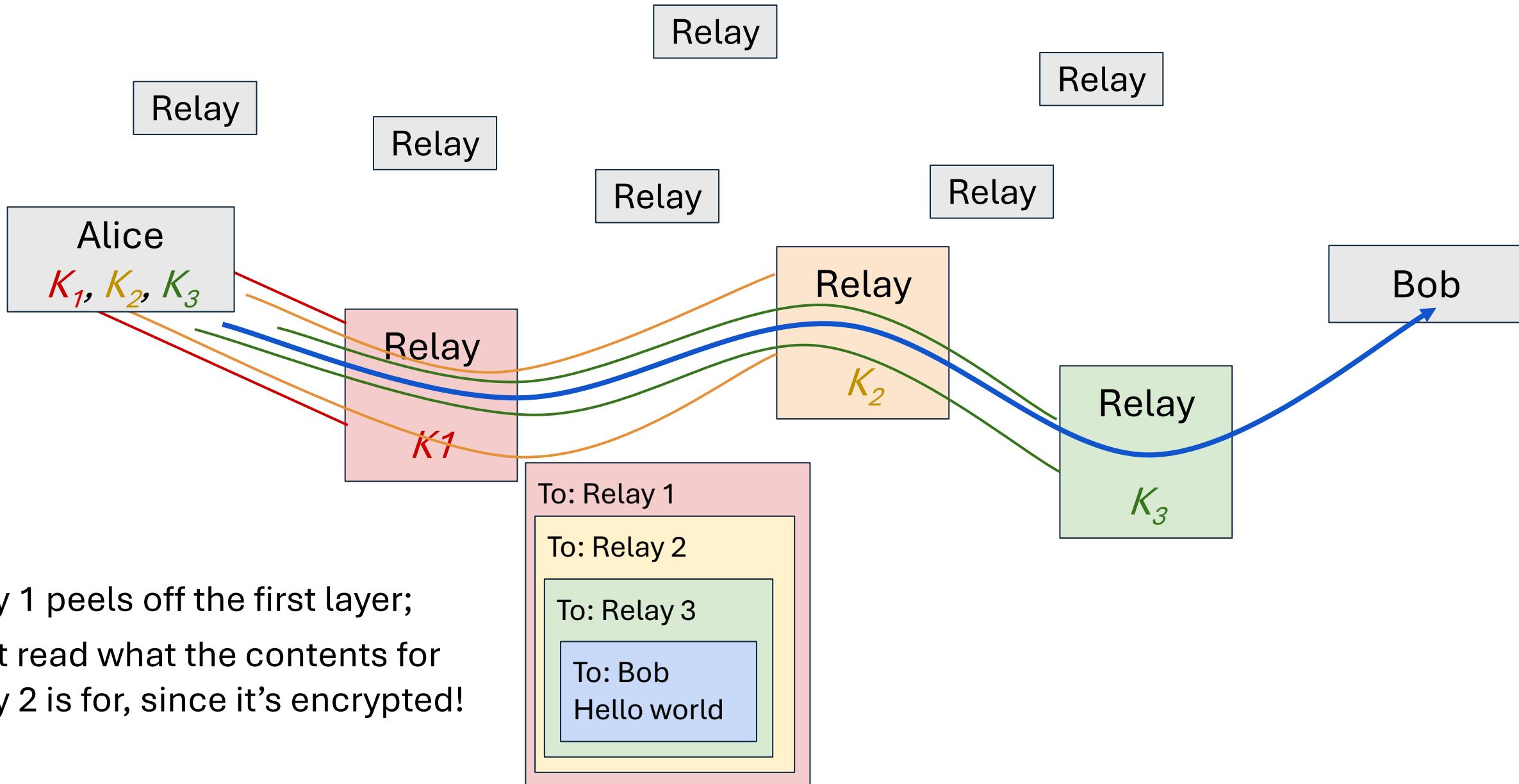
- Wrap the packets via encryption
 - For example, the packet sent to Bob is encrypted under K_3 since Relay 3 is the one to forward that information to Bob
- Ensures that no one can read or tamper with the messages, since these are all sent over TLS connections



Tor Circuits



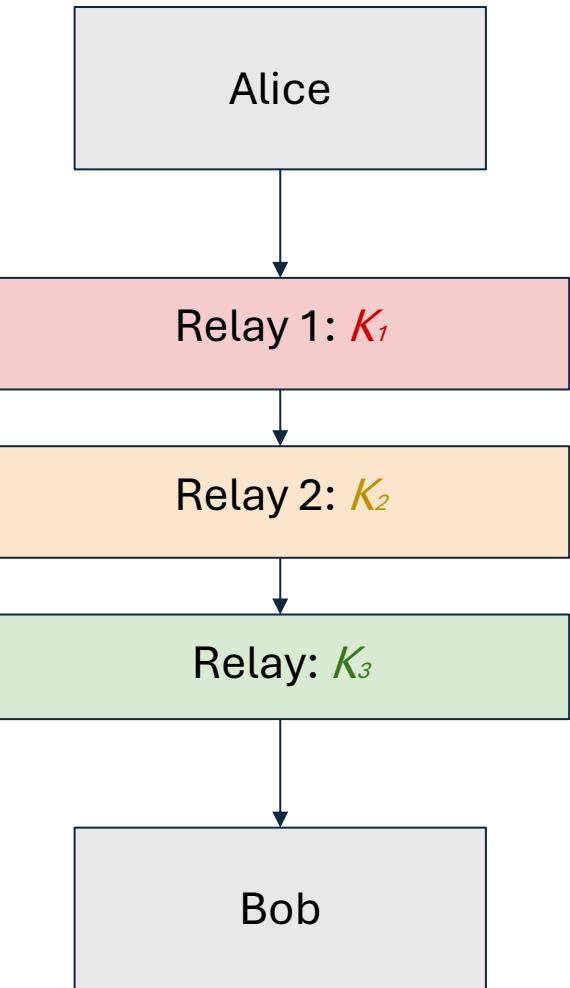
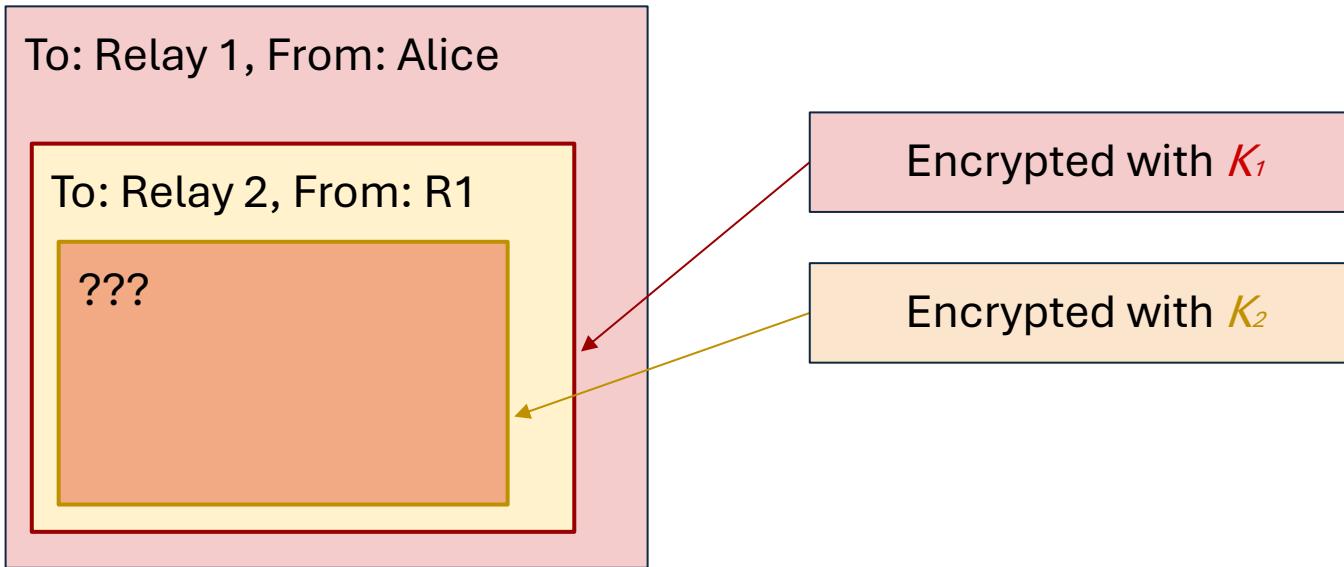
Tor Circuits



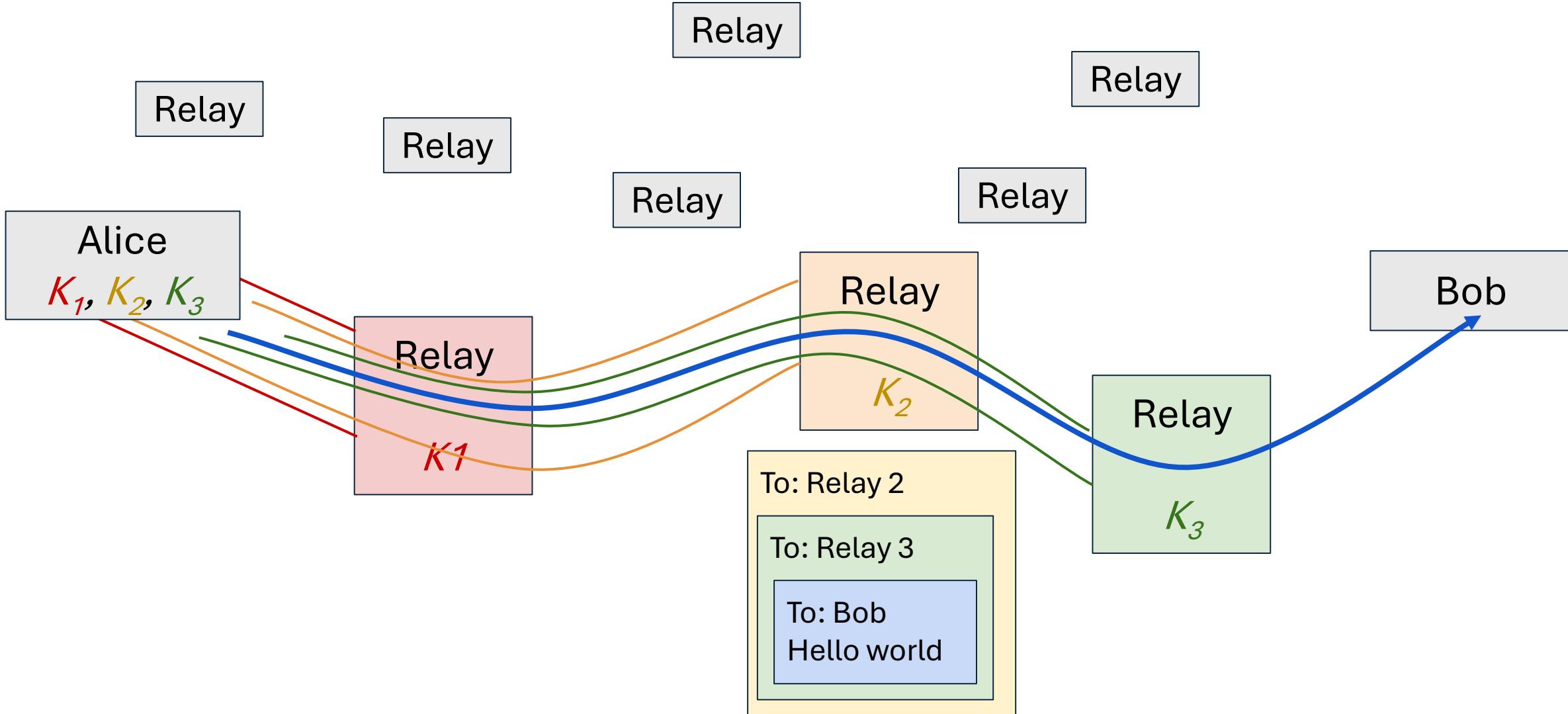
Relay 1 peels off the first layer;
Can't read what the contents for
Relay 2 is for, since it's encrypted!

Tor Packet Construction

- What does Relay 1 see?

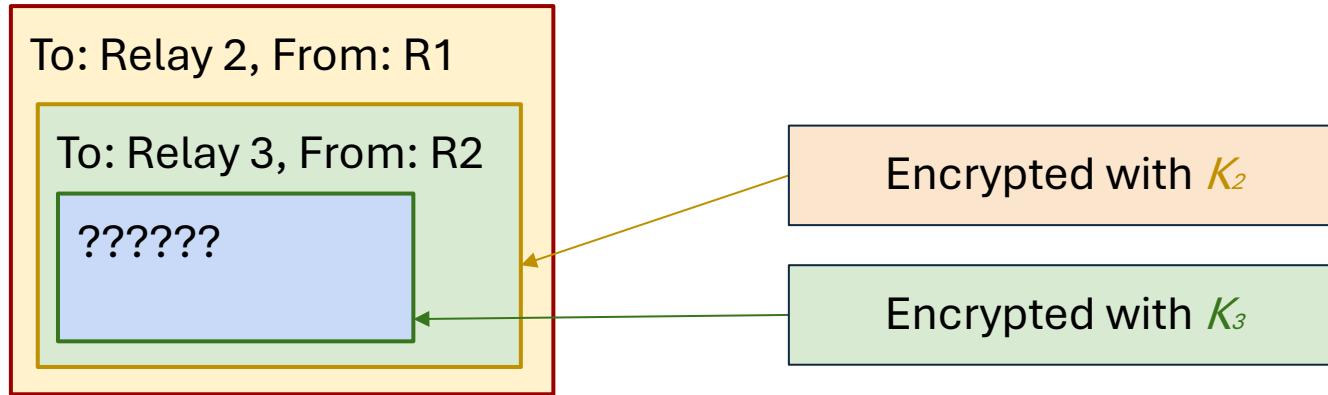


Tor Circuits

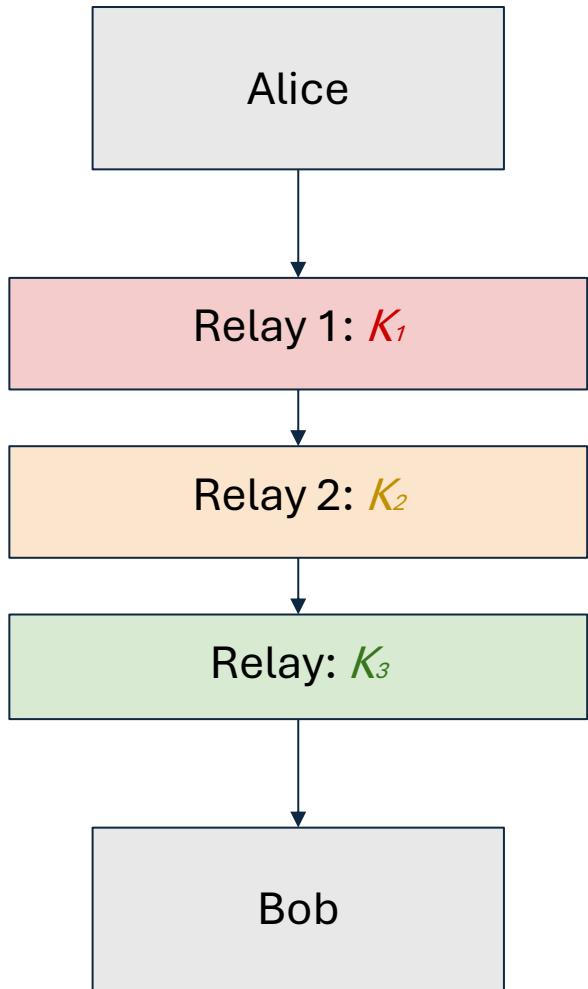


Tor Packet Construction

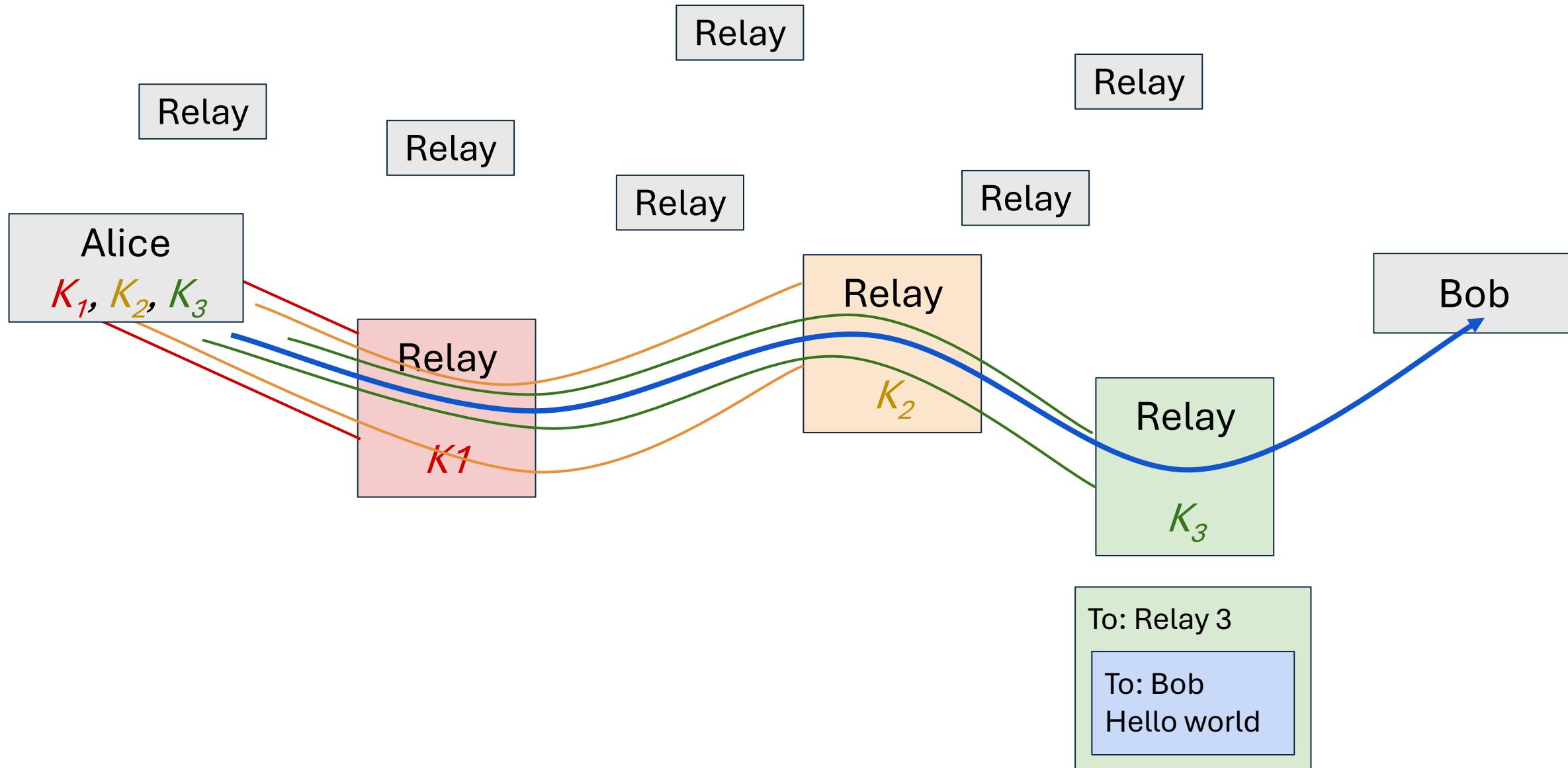
- What does Relay 2 see?



All Relay 2 knows is the message came from Relay 1 and is going to Relay 3. They know nothing about Alice and Bob!

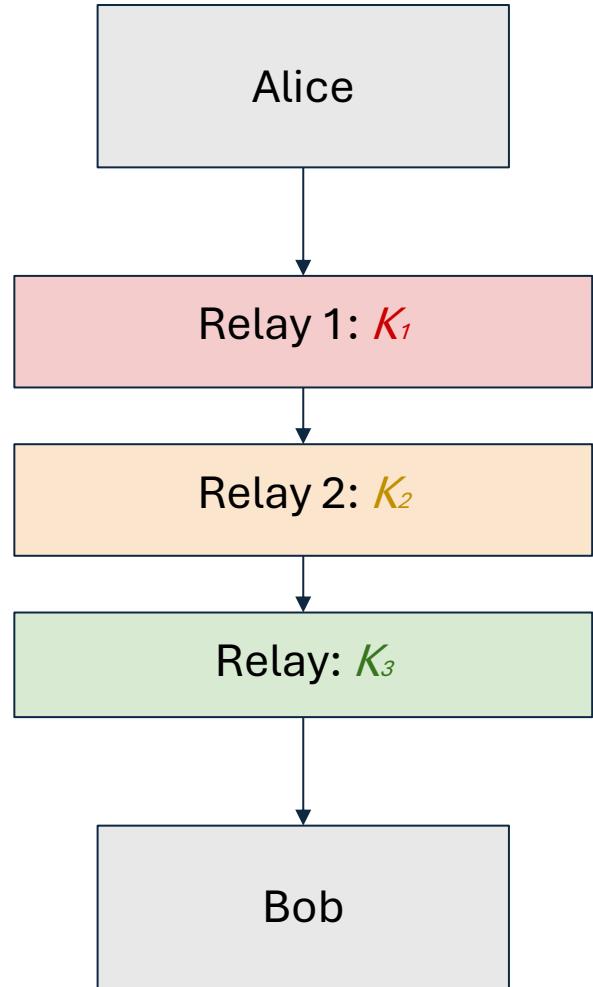
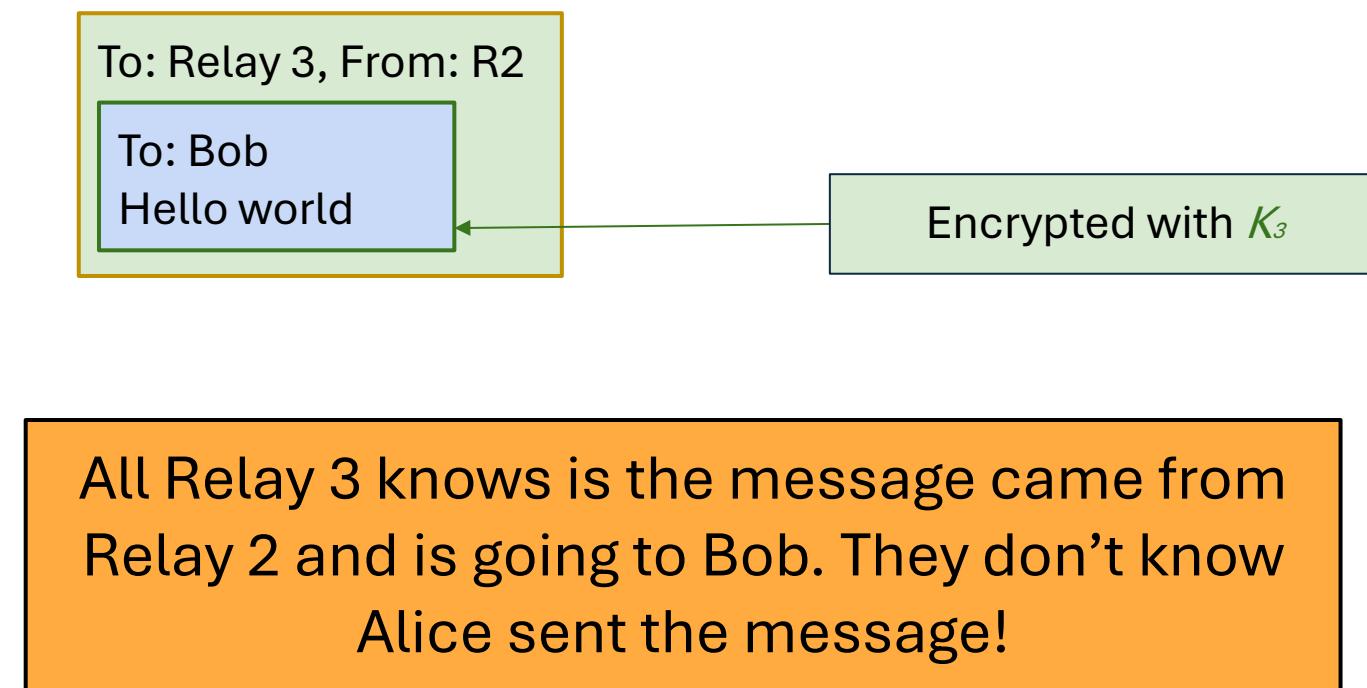


Tor Circuits

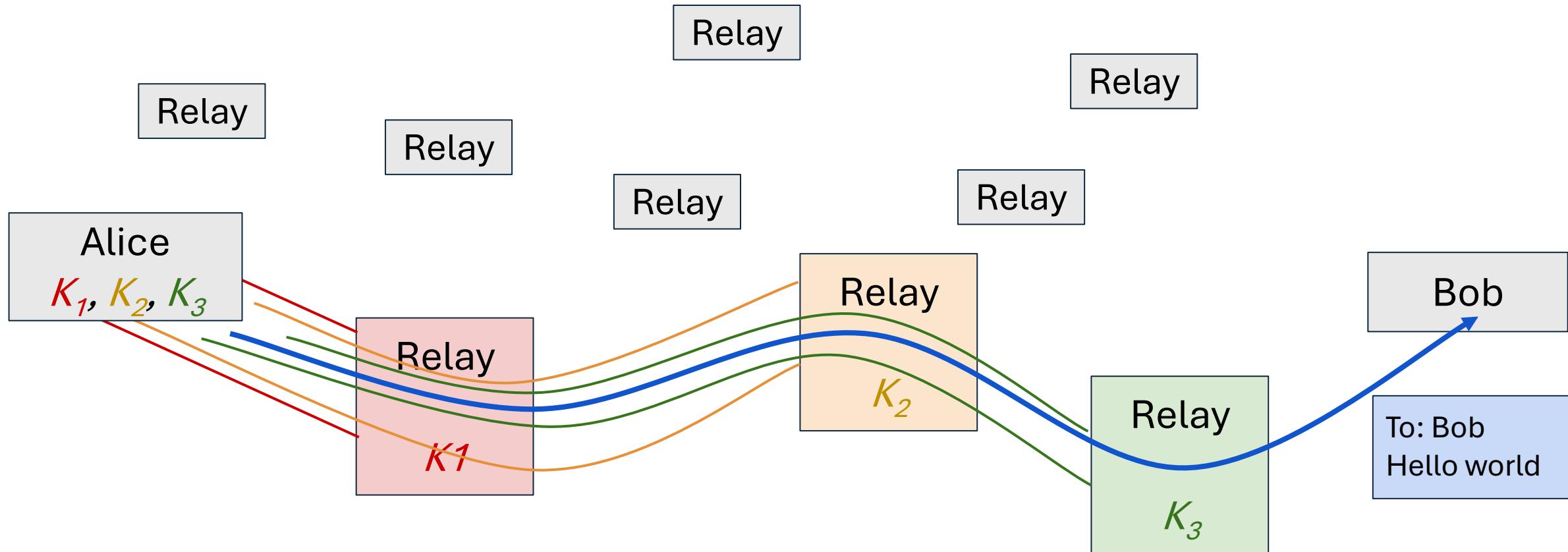


Tor Packet Construction

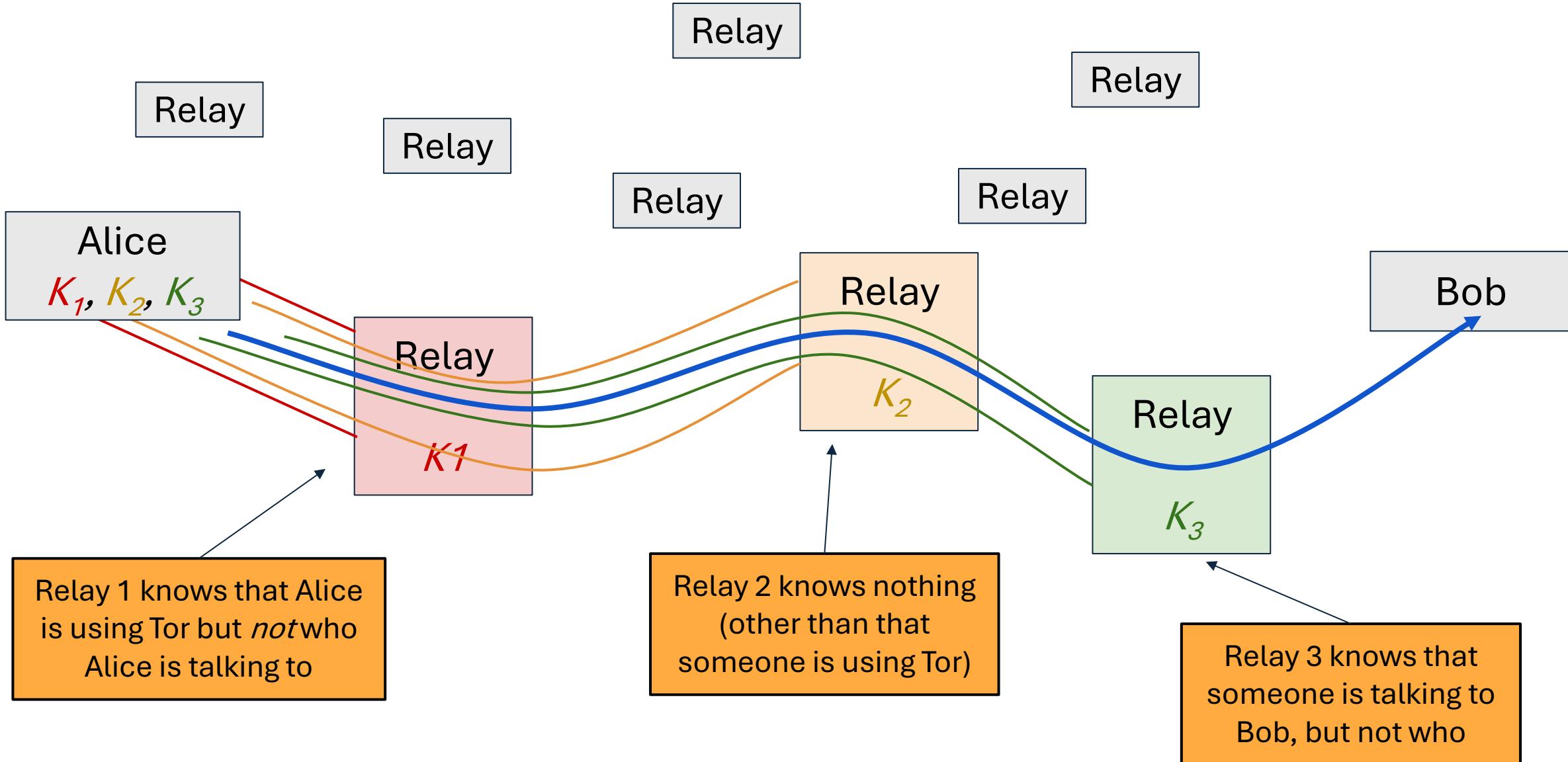
- What does Relay 3 see?



Tor Circuits



Tor Circuits



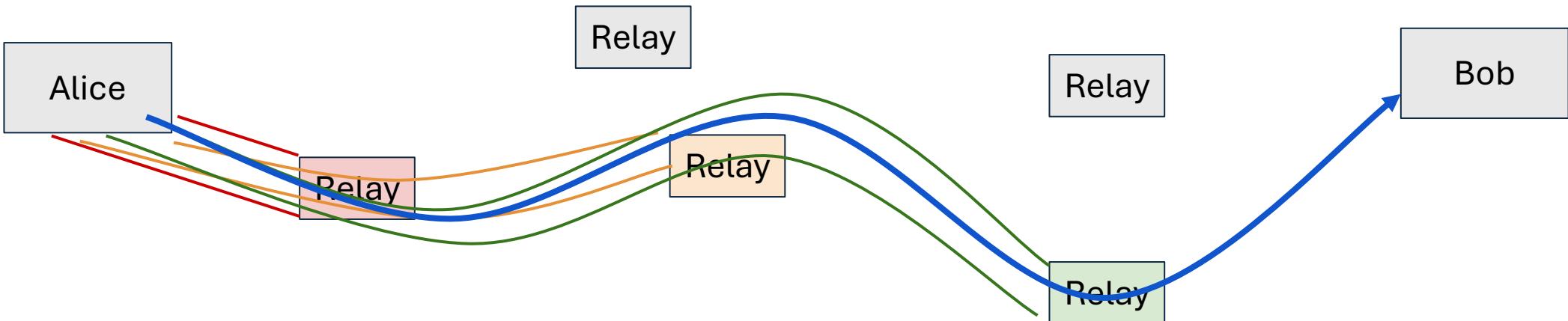
Tor Exit Nodes

Notice: The exit node (e.g., Relay 3) can see the message and the recipient

- Without collusion, the exit node doesn't know the sender

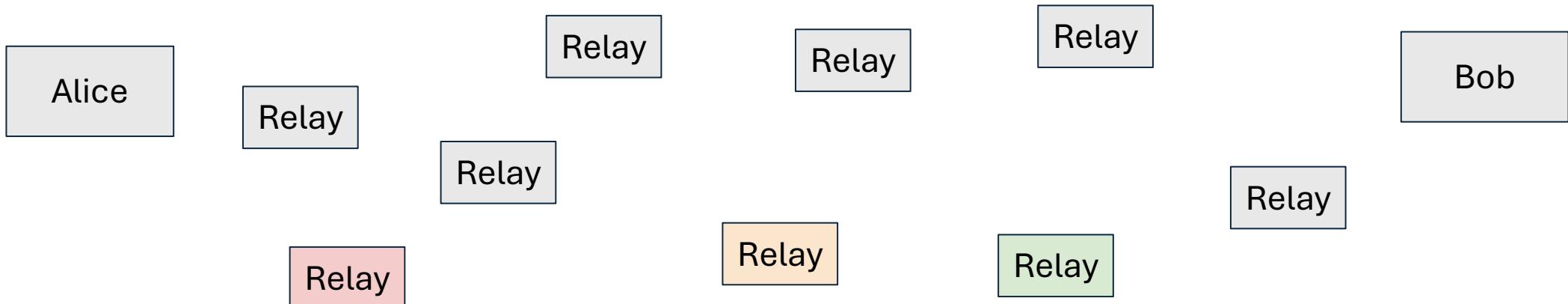
The exit node is a man-in-the-middle attacker

- If the user is not using TLS to connect to the end host (using HTTP), the exit node can see and modify the traffic
- If the user is using TLS (using HTTPS), the exit node cannot see or tamper with the traffic



Tor Weaknesses: Collusion

- Collusion: Multiple nodes working together and sharing information
 - Collusion is adversarial (dishonest) behavior
 - If all nodes in the circuit collude, anonymity is broken
 - If at least one node in the circuit is honest, anonymity is preserved
- It is easy to form some amount of colluding nodes
 - An adversary can create/control hundreds nodes in the Tor network to increase the chance that your circuit consists entirely of the attacker's nodes!
 - The more nodes we use, the more confident we are that they are not all colluding



Tor Weaknesses: Distinguishable Traffic

Tor does not hide the fact that you are using Tor!

- Packet Size Pattern:
 - HTTPS: Variable sizes reflecting the actual content length
 - Tor: Always fixed 514-byte cells
- Connection Pattern:
 - HTTPS: Direct connection to destination
 - Tor: Always connects to *Tor relay* first
- Anonymity only works in a crowd
 - Example: A Harvard student sends an anonymous threatening message using Tor. The administrators notice that **only one student** on the Harvard network is using Tor!

Tor Weaknesses: Timing Attacks

A network adversary who has a full (global) view of the network can learn that Alice and Bob are talking

- Exploit a timing attack: Observe when Alice sends a message, when Bob receives a message, and link the two together
- Global adversaries are outside of Tor's threat model and are not defended against
- Tor only defends against local adversaries with partial views of the network
- Timing attacks could be defended against by delaying the timing of packets, but this violates Tor's performance goal
- [German law enforcement](#) has been identified as having a capability to deanonymize Tor users

Summary: Anonymity and Tor

Anonymity conceals an individual's identity, but it's difficult to achieve on the web

- Proxies and VPNs relay traffic through a single machine
 - Issue: The single relay knows who you are and what you are doing!
- Tor routes your traffic through multiple relays
 - No one machine knows both who you are and what you are doing
 - Circuits are established by performing TLS handshakes with (at least) three nodes, nesting encrypted channels
- Weakness:
 - Collusion between relay nodes can deanonymize users by working together
 - Tor traffic is distinguishable from normal traffic
 - Timing attacks

Roadmap

- *Anonymity*
- *Proxies and VPNs*
- *Tor*
- ***Tor Onion Services***

Tor Onion Services

- Sometimes, the **server** wants to be anonymous, so no one knows where the server is located
- **Tor onion services:** Websites that are only accessible through the Tor network
 - Gives the server anonymity protection
 - Sometimes called the **dark web**
- Big idea: Route the server's traffic through the Tor network so that no one knows who the server is

Tor Onion Services

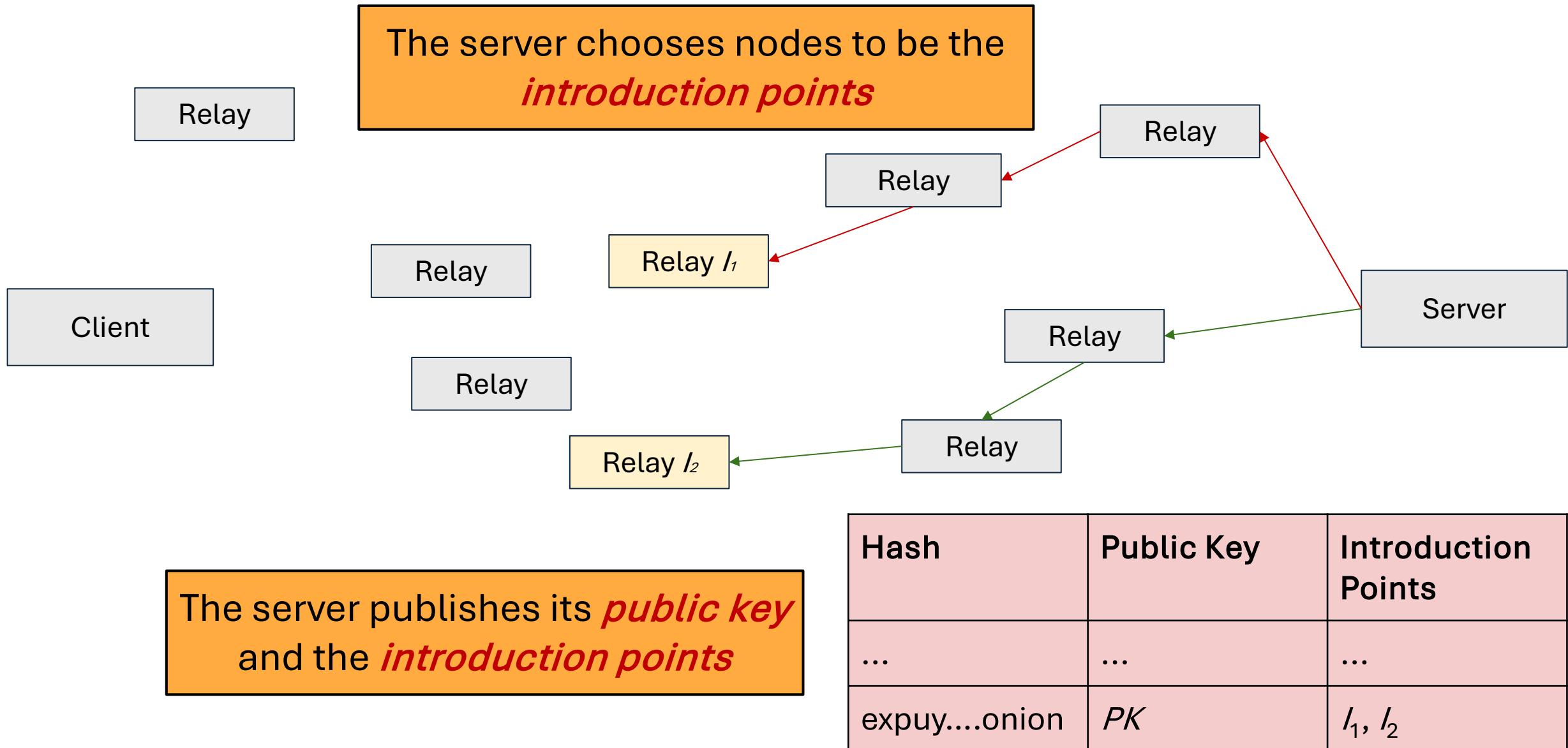
Recall: Standard domain names translate to IP addresses, which would break server anonymity

- Instead, identify servers using the **hash of the service's public key** encoded as “*abcxyz.onion*” address
 - Example: <http://pwoah7foa6au2pul.onion>
 - Example: **<https://facebookcorewwi.onion>** (Facebook brute-forced key pairs until they found one with a human-readable hash)
- Any **.onion** address can only be accessed through the **Tor browser** (or other software specifically configured to use the Tor network)
- Only the Tor network knows how to route traffic to **.onion** services through its encrypted relay system

Tor Onion Services

- Regular Domain names vs .onion:
 - Regular domains must resolve to IP addresses, which reveal the server's network location
 - .onion addresses are hashes of public keys, which don't contain any location information
- Where the anonymity comes from:
 - The .onion address being a hash of the public key means it's just a cryptographic identifier, not a network location
 - When clients connect to an onion service, they establish an encrypted connection through multiple Tor relays (detailed in the following slides)
 - The service's actual IP address is hidden behind these relay layers (hence "onion")
 - *No single relay knows both the client and the server's real location*
- Why use the hash of the public key
 - The server proves it owns the address by proving it has the private key that matches the public key hash

Tor Onion Services

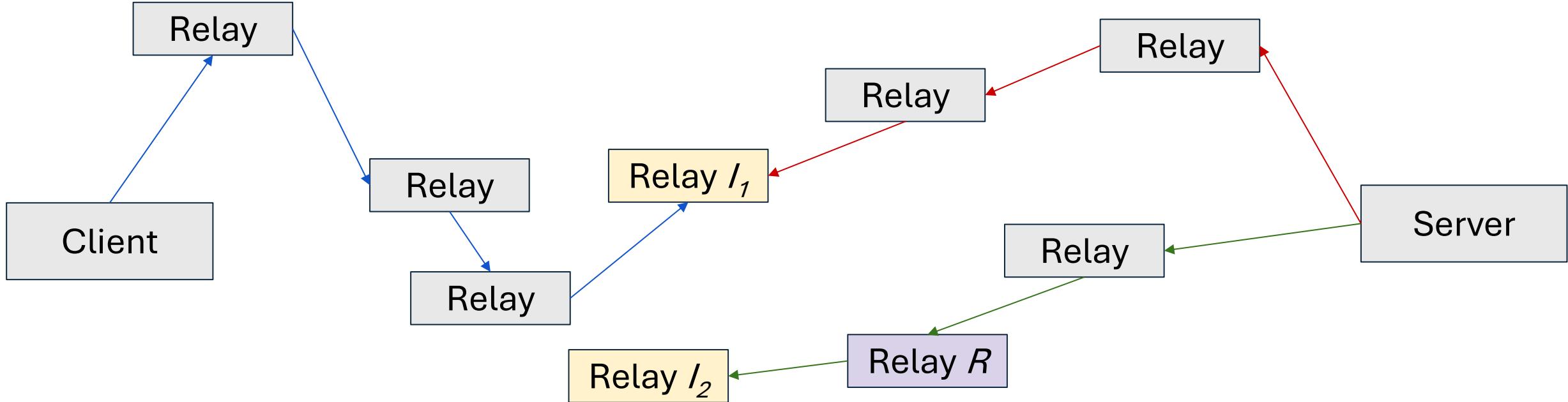


Tor Onion Services

Now, the client connects to the server

1. The client queries the directory using the hash of the public key to get the server's **full public key** and the **introduction points**
2. The client chooses an introduction point and forms a Tor circuit to it
3. The client chooses a **rendezvous point (relay)** and **secret** used to communicate to the server, encrypts them with the server's public key, and sends them to the introduction point, which relays them to the server
4. The client and server **both form Tor circuits** to the rendezvous point and perform an end-to-end TLS handshake, and the server sends the decrypted secret to the client to **authenticate** itself

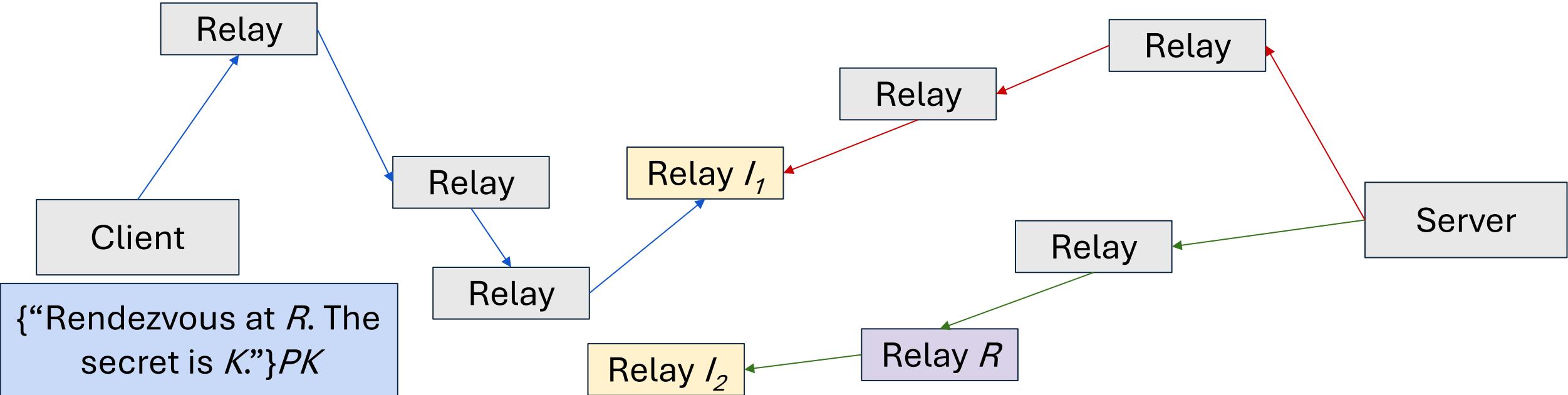
Tor Onion Services



The client queries the directory and
connects to an introduction point

Hash	Public Key	Introduction Points
...
expuy....onion	PK	I_1, I_2

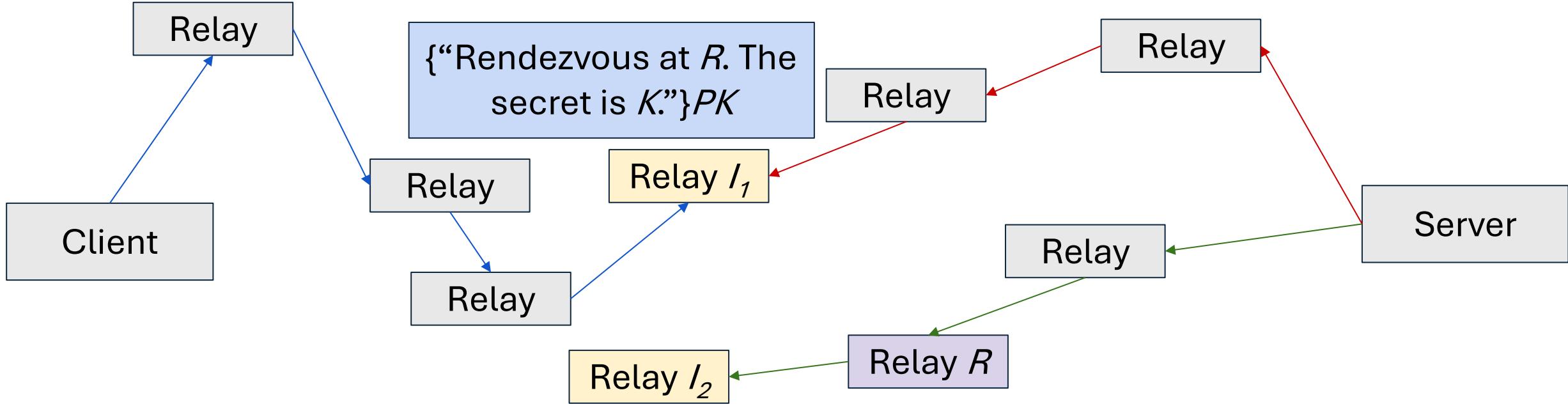
Tor Onion Services



The client chooses a ***rendezvous point and secret***, encrypts using the public key, and sends them to the server through the introduction point

Hash	Public Key	Introduction Points
...
expuy....onion	PK	I_1, I_2

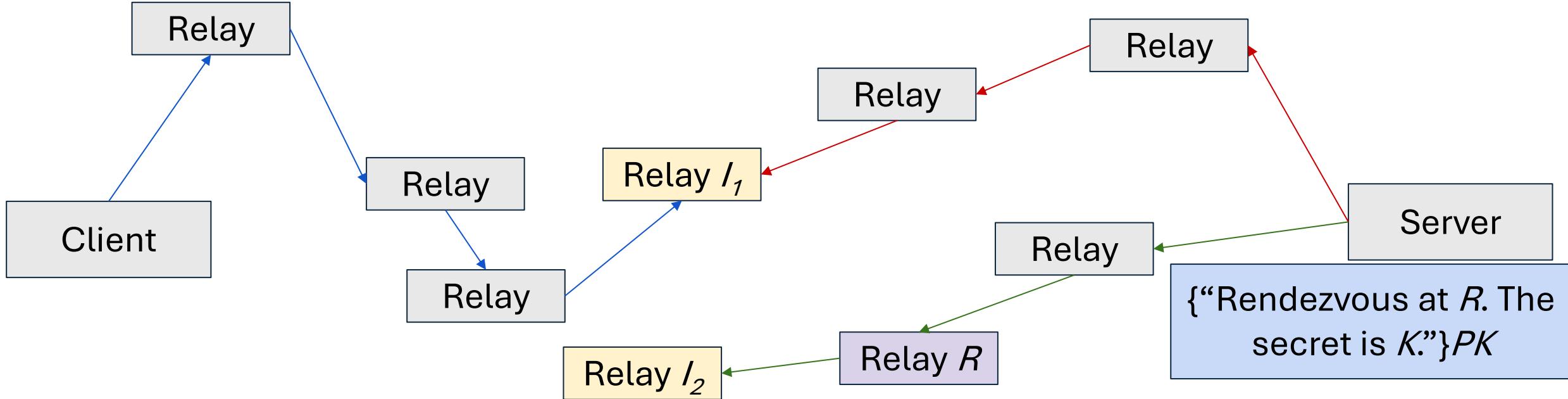
Tor Onion Services



The client chooses a *rendezvous point and secret*, encrypts using the public key, and sends them to the server through the introduction point

Hash	Public Key	Introduction Points
...
expuy....onion	PK	I_1, I_2

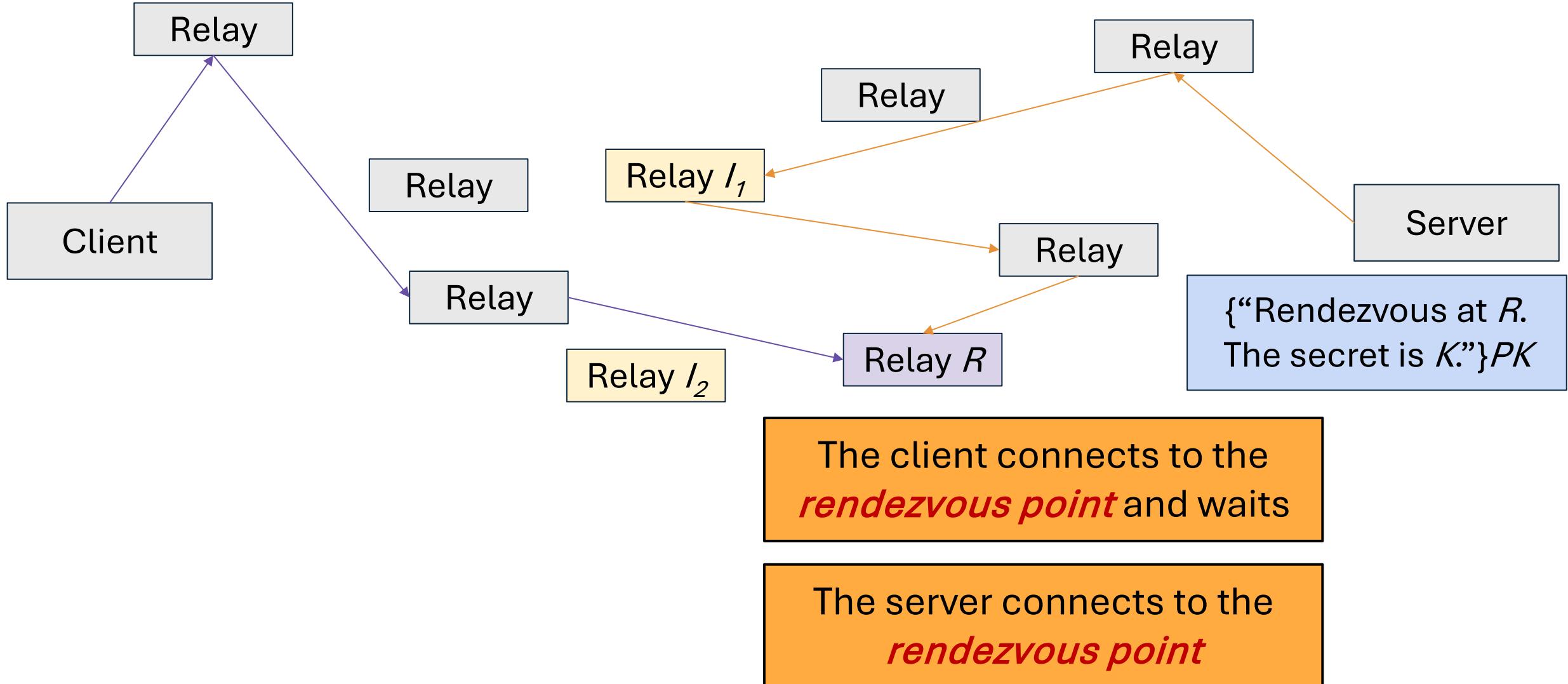
Tor Onion Services



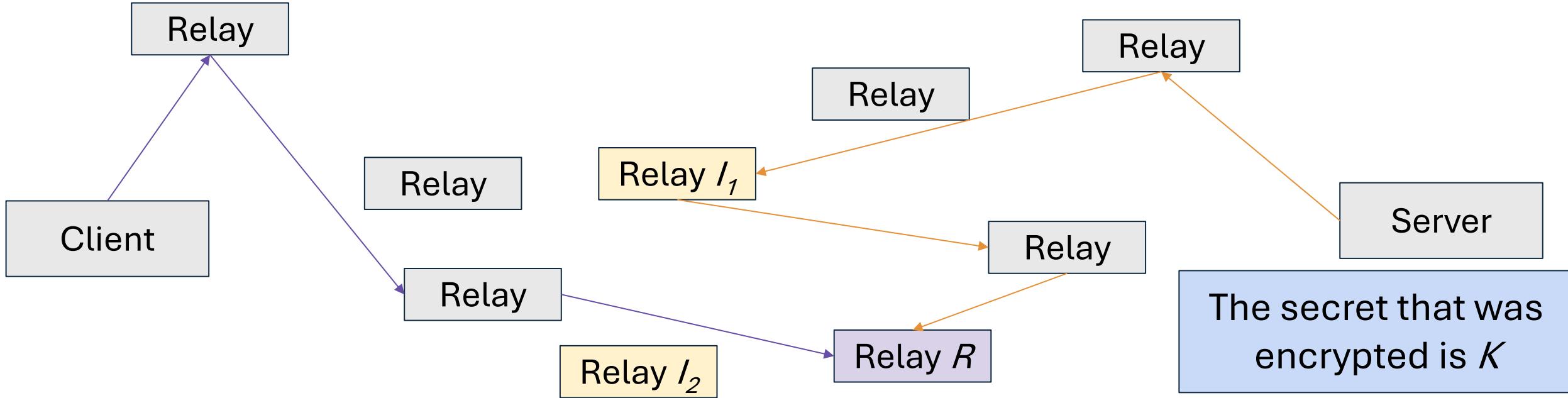
The client chooses a ***rendezvous point and secret***, encrypts using the public key, and sends them to the server through the introduction point

Hash	Public Key	Introduction Points
...
expuy....onion	PK	I_1, I_2

Tor Onion Services

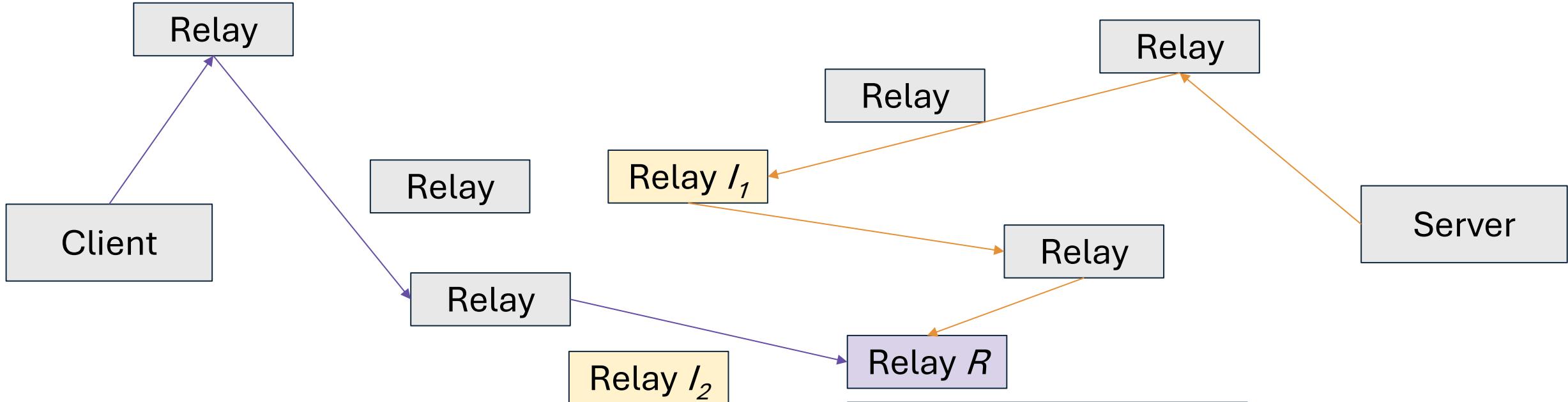


Tor Onion Services



The client and server perform a handshake, and the server sends the decrypted secret

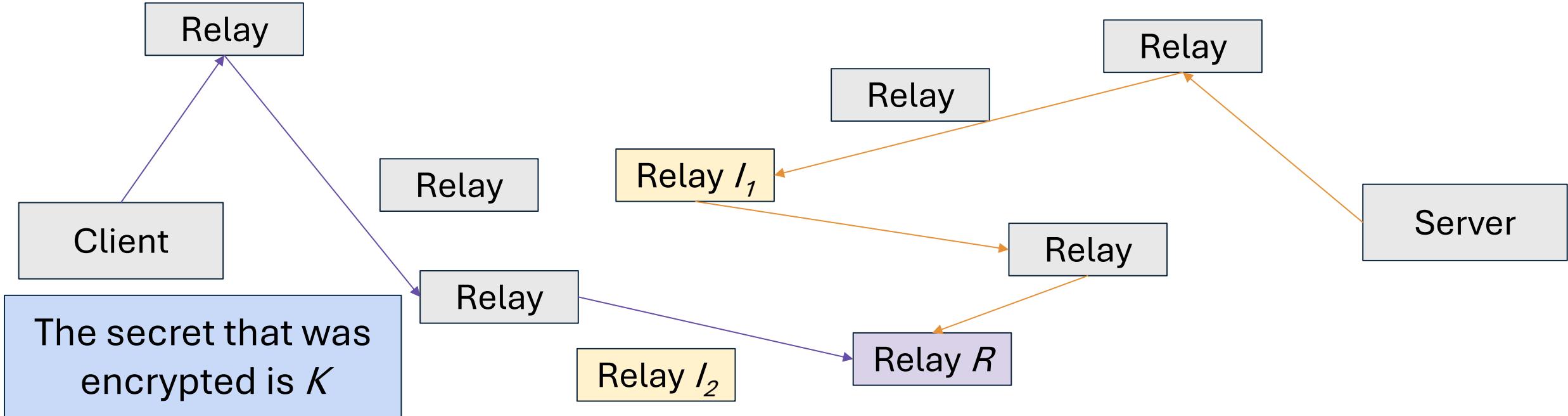
Tor Onion Services



The client and server perform a handshake, and the server sends the decrypted secret

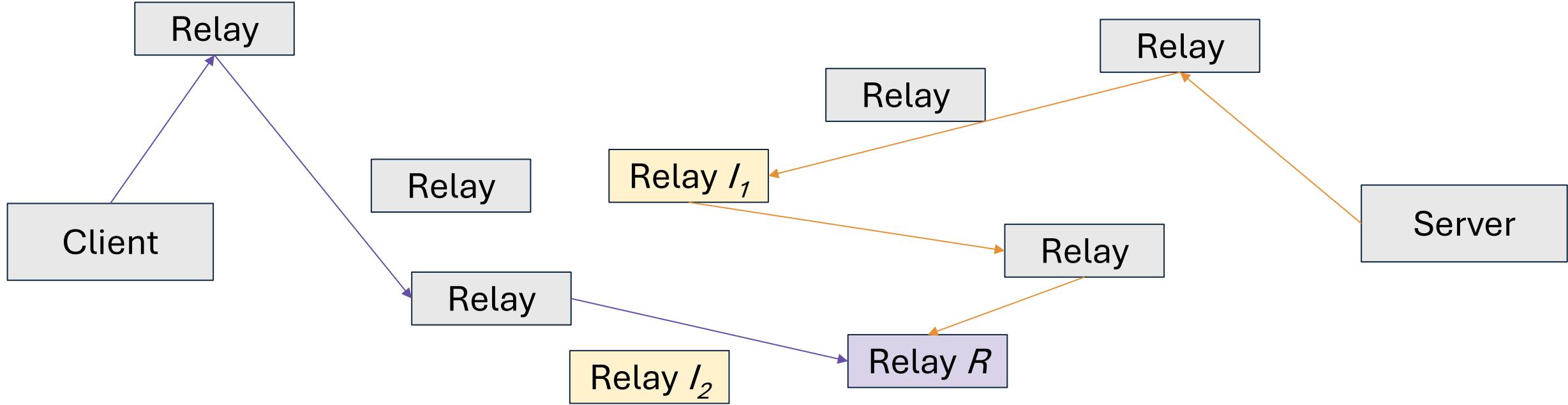
The secret that was encrypted is K

Tor Onion Services



The client and server perform a handshake, and the server sends the decrypted secret

Tor Onion Services



Notice: The client and server never directly communicate, and the introduction and rendezvous points don't know the client or the server

Hosting Illegal Services on Tor

- Tor onion services are often used for services widely considered illegal around the world
 - Legitimate hosting services like Cloudflare will refuse to host these services
 - Most countries will take legal action against these services if hosted on the regular web
- ~~Dark markets~~: Marketplaces for buying and selling illegal goods
 - Transactions processed with a censorship-resistant currency like Bitcoin
 - Services like PayPal will refuse to process illegal transactions
 - Ratings system with mandatory feedback
 - Escrow service to handle disputes between sellers and buyers
 - Can only be accessed as a Tor onion service
- ~~Cybercrime forums~~: Websites for discussing illegal activity

CSC 3511 Security and Networking

Week 14, Lecture 1: Firewalls

Roadmap

- ***Motivation and Security Policies***
- *Firewalls: Packet Filters*
- *Firewall Limitations and Evasion*

Motivation: Scalable Defenses

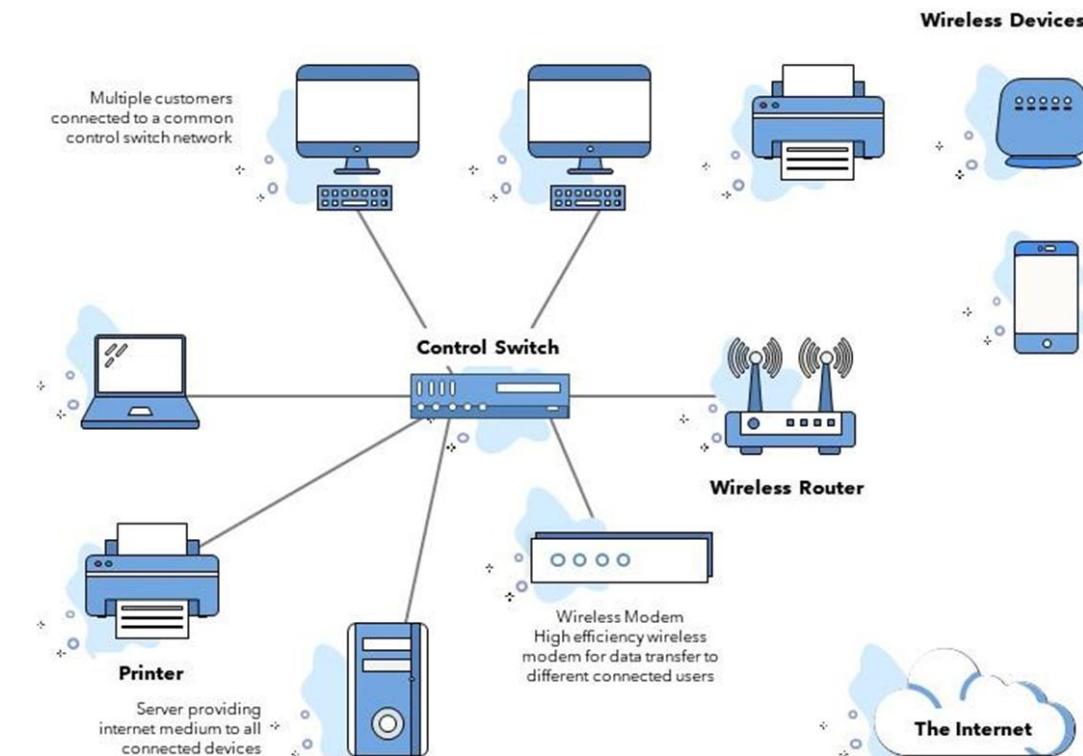
- How do you protect a set of systems against external attack?
 - Example: A company network with many servers and employee computers

Observation: More network services = more risk

- Each network connection creates more opportunities for attacks (greater attack surface)
- Various connections/protocols

Observation: More networked machines = more risk

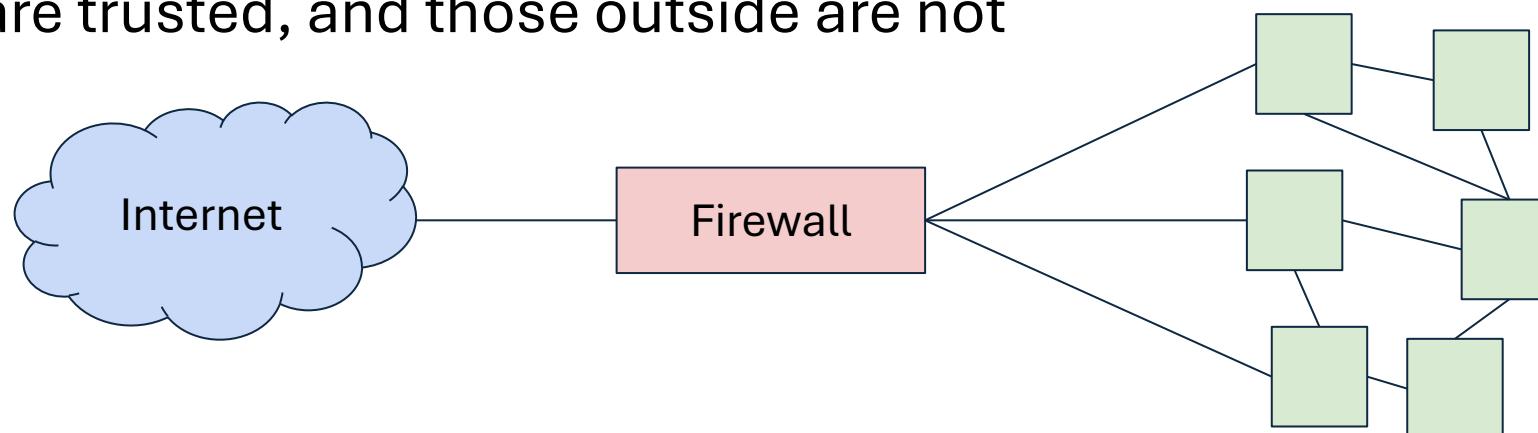
- What if you have to secure hundreds of systems with different hardware, operating systems, and users?
- What if there are some devices (e.g., guest's computer) in the network that you aren't aware of?



Instead of securing individual machines, we want to ***secure the entire network!***

Firewalls and Security Policies

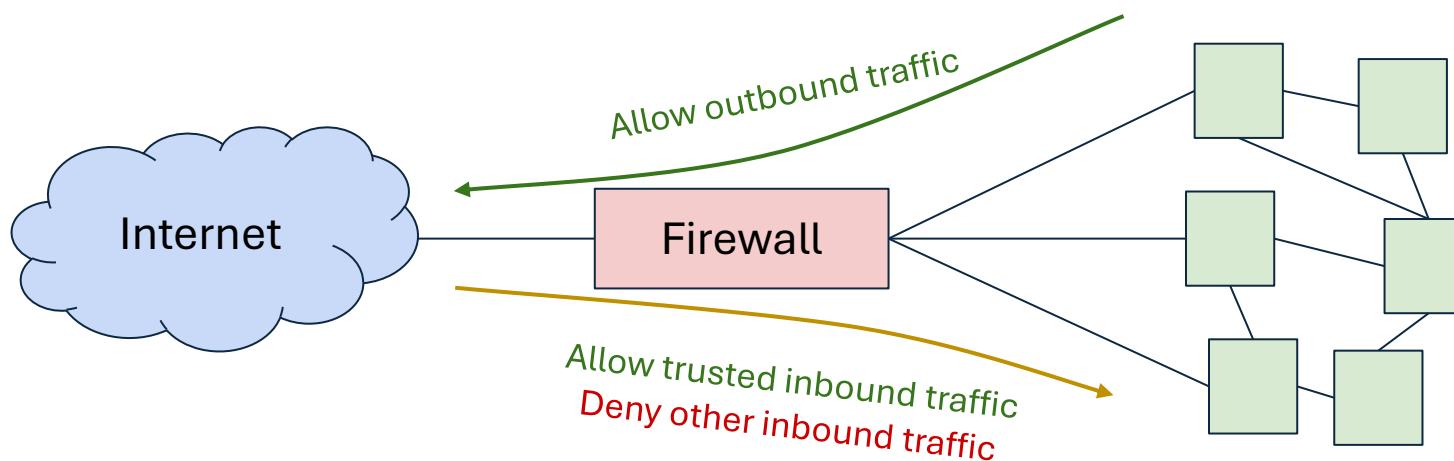
- Idea: Add a single point of access in and out of the network, with a monitor
 - Any traffic that could affect vulnerable systems must pass through the firewall
 - Ensure “complete mediation”
- Network access is controlled by a **policy**
 - A security policy is a set of rules that defines what traffic is allowed or denied
 - Defines what traffic is allowed to exit the network (**outbound policy**)
 - Defines what traffic is allowed to enter the network (**inbound policy**)
 - Policy model based on our threat model: We usually assume users “inside” the network are trusted, and those outside are not



Firewalls and Security Policies

For example, what's the policy of a standard home network?

- Outbound policy: Allow outbound traffic
 - Users inside the network can connect to any service
- Inbound policy: Only some traffic is able to enter the network
 - Allow inbound traffic in response an outbound connection (e.g., FTP)
 - Allow inbound traffic to certain, trusted services (e.g. SSH)
 - Deny all other inbound traffic (e.g., IP forwarding)



Default Security Policies?

- How should we handle traffic that isn't explicitly allowed or denied?
 - For example, an employee clicks a phishing link, and ransomware tries to connect to an AWS EC2 instance on port 8443
- ***Default-allow policy:*** Allow all traffic, but deny those on a specified **denylist**
 - For the above example: Connection succeeds → Ransomware communicates with attacker, encrypts files, exfiltrates data.
 - As problems arise, add them to the denylist
- ***Default-deny policy:*** Deny all traffic, but allow those on a specified **allowlist**
 - For the above example: Connection blocked → Attack is contained, ransomware can't receive instructions or send data out.
 - As needs arise, add them to the allowlist

Default Security Policies?

Default-allow policy vs. Default-deny policy, which default policy is better?

- Default-allow is more flexible, but flaws are vulnerabilities and can be catastrophic
- Default-deny is more conservative, but flaws are less painful
- Default-deny is generally *accepted* to be the best default policy
 - Never trust, always verify

Aspect	Default Accept (Blocklist/Denylist)	Default Deny (Allowlist)
Security	Less secure: Any traffic not explicitly blocked is allowed, which can expose vulnerabilities.	More secure: All traffic is blocked unless explicitly allowed, minimizing the attack surface.
Functionality	Maximizes functionality: All traffic works unless explicitly blocked.	Reduces functionality initially: Services and applications may not work until specific rules are created.
Ease of Configuration	Easier to configure: No need to identify every necessary traffic flow up front.	More effort required: Every service, port, or protocol needs to be explicitly configured.
Use Case	Suitable for home networks, personal devices, or less critical systems where functionality is prioritized.	Best for organizations, data centers, or sensitive systems where security is a top priority.

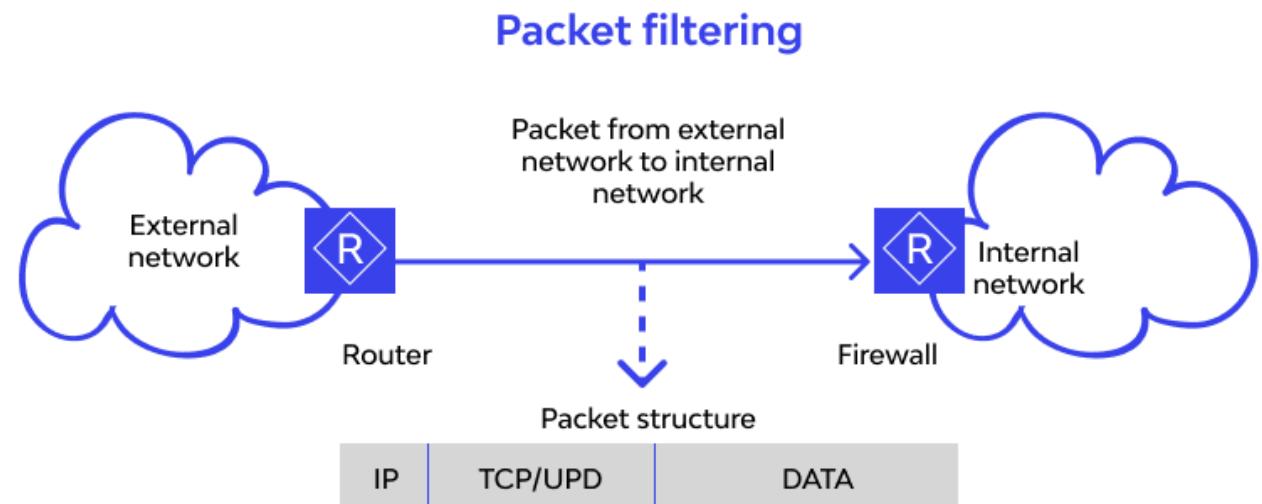
Roadmap

- *Motivation and Security Policies*
- ***Firewalls: Packet Filters***
- *Firewall Limitations and Evasion*

Packet Filters

Firewalls are often **packet filters**

- Filtering with *incoming and outgoing* interfaces
- Inspect network packets and chooses a handling method: forward or drop
- Works at network layer and transport layer;
- Filter based on packet header fields:
 - Source & destination IP addresses
 - Protocols (TCP, UDP, ICMP):
 - Source & destination port number
 - Control flags (TCP SYN, ACK, FIN, etc.)



Fast and efficient: Minimal processing overhead

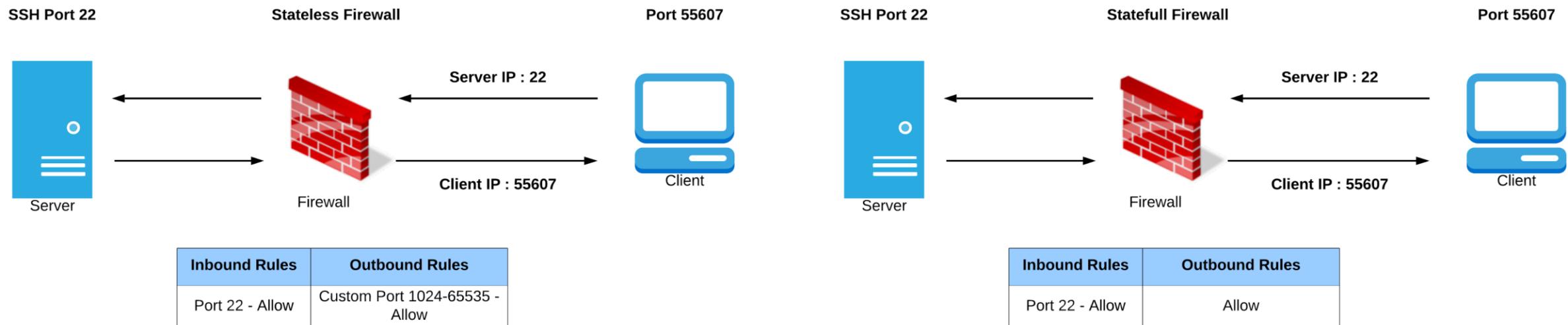
Packet Filters

Stateless packet filters

- Packet filters that have no history
- All decisions must be made using only the *information in the packet itself*
- Can have trouble implementing complex policies that require knowledge of history

Stateful packet filters

- Maintain a state table that tracks all active connections and sessions passing through it



Stateless Packet Filters

Consider implementing the typical home network policy:

- Allow outbound traffic
- Allow inbound traffic in response to an outbound connection
- Deny all other inbound traffic

Issue: How do we know what inbound traffic is in response to an outbound connection?

- TCP: Can be implemented with a hack
 - Deny inbound traffic without an ACK flag set
 - Allow inbound traffic with the ACK flag set
 - If the internal computer sees an ACK packet without having formed a connection (through the OS), it will ignore it or send a RST packet
- UDP (e.g., DNS responses): Impossible to implement
 - UDP “connections” are typically implemented at the application layer, so we can’t inspect much

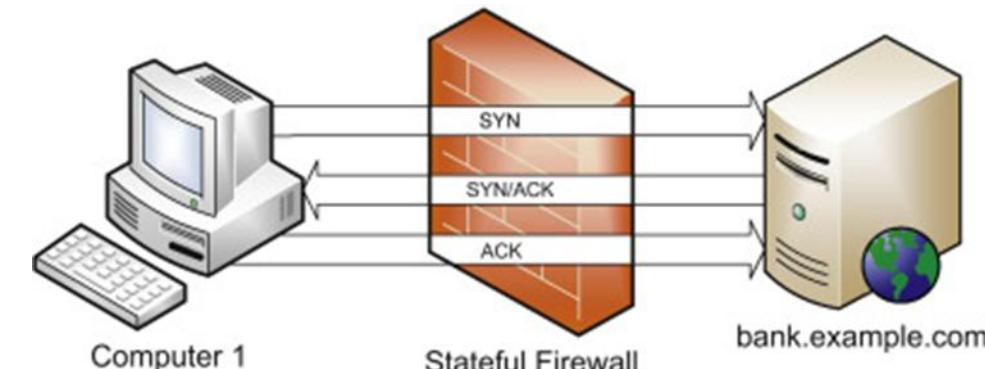
Stateless packet filters are not good enough!

Stateful Packet Filters

A better idea: Keep “state” in the implementation of the packet filter

- The filter keeps track of inbound/outbound connections and sessions passing through it, rather than individual packets
- **Phase 1: Session handling**
 - The firewall tracks and remembers active connections by maintaining a **connection state table**
 - Remembers details about active connections
 - Source/destination IP addresses and ports
 - **Connection states** (NEW, ESTABLISHED, RELATED, etc.)
 - Protocol-specific parameters (e.g., TCP SEQ and ACK #, TCP flags seen, ICMP Seq #)
 - Timestamps (created, last activity)

Source Address	Source Port	Destination Address	Destination Port	Connection State
192.168.1.100	1030	192.0.2.71	80	Initiated
192.168.1.102	1031	10.12.18.74	80	Established
192.168.1.101	1033	10.66.32.122	25	Established
192.168.1.106	1035	10.231.32.12	79	Established



Stateful Packet Filters

A better idea: Keep “state” in the implementation of the packet filter

- Phase 2: Decision making

- Makes decisions based on both **static rules** and **connection context**
 - For ESTABLISHED connections: Automatically allowed (already in state table)
 - For NEW connections: Must pass all firewall rules
 - For INVALID packets: Rejected (don't match any known connection)

Example rules:

- **allow tcp connection 4.5.5.4:*** -> **3.1.1.2:80**
 - Allow connections from **4.5.5.4** to **3.1.1.2** with destination port **80**
- **allow tcp connection *:*/int -> *:80/ext**
 - Allow outbound connections with destination port **80**
- **allow icmp connection *:*/int -> *:*/ext**
 - Allow outbound ICMP connections from any internal host to any external host
- **allow udp connection *:*/int -> *:53/ext**
 - Allow outbound UDP connections from any internal host to any external DNS servers
 - **Stateful firewall** automatically allows UDP responses back

Stateful Packet Filters

A better idea: Keep “state” in the implementation of the packet filter

- Phase 2: Decision making

- Makes decisions based on both static rules and connection context

1. Outbound ICMP Echo Request to *bank.example.com*

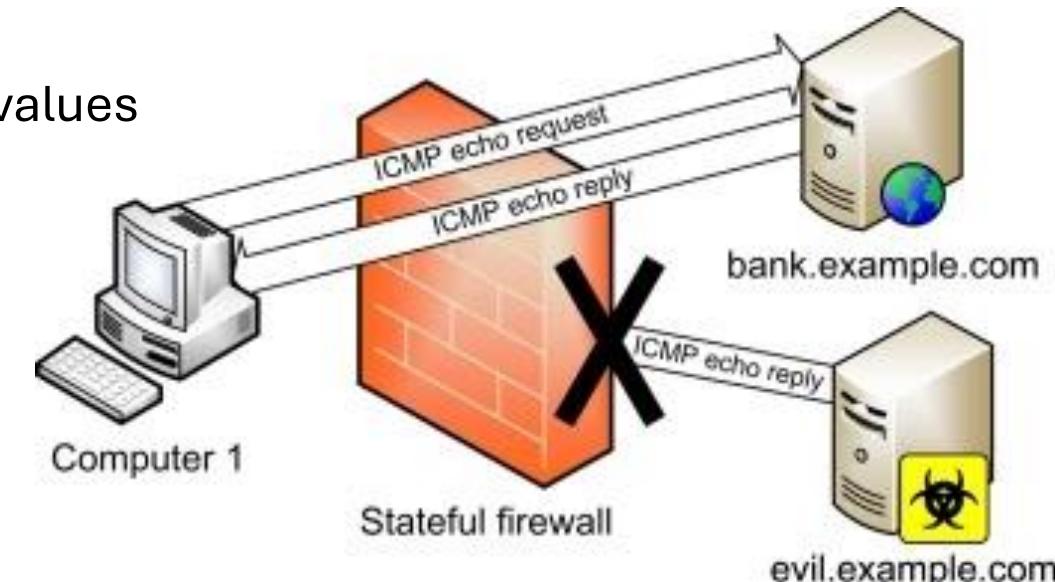
- Firewall *creates state table entry*, containing Session ID, connection type, des IP, ICMP ID & Seq, State (NEW → ESTABLISHED), and expected reply

2. Inbound ICMP Echo Reply from *bank.example.com*

- *Validates* session ID, ICMP ID and Seq match expected values

3. Unsolicited ICMP Echo Reply from *evil.example.com*

- Checks state table: No matching ICMP request to *evil.example.com*
- *Marks as INVALID && Drops the packet*
- Computer 1 never receives this malicious ICMP reply



Stateless vs. Stateful Packet Filters

Consider implementing the typical home network policy:

- Allow inbound traffic in response to an outbound connection

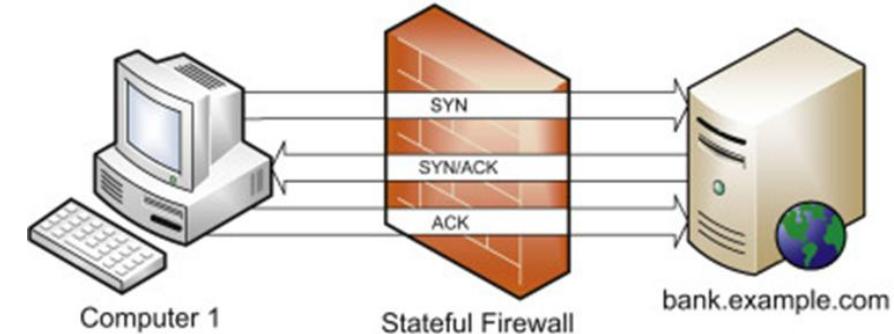
Issue: How do we know what inbound TCP traffic is in response to an outbound connection?

Attempted solutions with **stateless filters**:

- **Rule 1:** Allow outbound TCP (all ports)
- **Rule 2:** Deny inbound TCP WITHOUT ACK flag ← Attempts to block new connection initiations
- **Rule 3:** Allow inbound TCP WITH ACK flag
- **Rule 4:** Deny all other inbound

Problem:

- Attacker can set ACK flag on malicious packets!
 - SRC addr && Port → evil.com:80
 - DST addr && Port → your_ip_addr:12345 [ACK = 1] **&& [RST = 1]**
- Some legitimate packets might be blocked: Server → Client: [FIN = 1] (ACK flag might be 0)



Stateless vs. Stateful Packet Filters

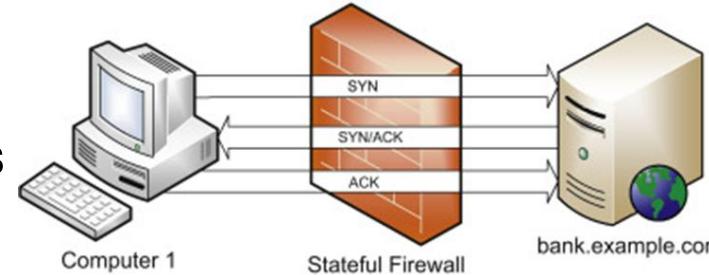
Consider implementing the typical home network policy:

- Allow inbound traffic in response to an outbound connection

Issue: How do we know what inbound traffic is in response to an outbound connection?

Attempted solutions with **stateful filters and state table**:

- **Rule 1:** Check if packet matches existing session → Allow
- **Rule 2:** If not, check if it's starting a new valid session → Check rules
- **Rule 3:** Otherwise → Deny



Session ID	Protocol	SRC IP:Port	DST IP:Port	State	Timeout
#4738	TCP	192.168.1.100:52341	93.184.216.34:443	ESTABLISHED	3600s
#4739	UDP	192.168.1.100:53128	8.8.8.8:53	ESTABLISHED	30s

Legitimate responses: Matched to sessions → allow

Spoofed packets: Don't match any session → block

Unsolicited inbound: No matching session → block

In networking context, **unsolicited traffic** means packets arriving at your network that you did not initiate or ask for

Stateless vs. Stateful Packet Filters

Aspect	Stateless Packet Filters	Stateful Packet Filters
Traffic Evaluation	Individual packets only (no memory)	Tracks and evaluates connection states (maintain state table)
Decision Based On	Static rules: IP addr, ports, protocols, flags	Static rules + connection context (NEW, ESTABLISHED, RELATED, INVALID)
Performance	Faster, less resource-intensive	Slower, more resource-intensive (state table maintenance, memory overhead)
Security	Basic filtering (less secure); vulnerable to spoofing, e.g., ACK flag attacks	Context-aware filtering (more secure); validates connections, prevents spoofing
Use Case	Simple networks, legacy systems	Modern enterprise networks, dynamic connections

Stateless vs. Stateful Packet Filters: Example 1

Web Browsing

Scenario	Stateless Behavior	Stateful Behavior
User visits website	Rule 1: Allow outbound port 443 Rule 2: Allow inbound from any source with ACK flag (insecure)	Rule 1: Allow outbound port 443 Rule 2: Response automatically allowed
Attacker sends packet with ACK flag	Allowed through (insecure)	Checked against state table, blocked

Stateless vs. Stateful Packet Filters: Example 2

ICMP Ping

Scenario	Stateless Behavior	Stateful Behavior
Internal user pings external host	Rule 1: Allow outbound ICMP Rule 2: Allow inbound ICMP from anywhere	Rule1: Allow outbound ICMP Rule 2: Reply automatically allowed
Attacker sends unsolicited ICMP	Allowed through (insecure)	No matching state, blocked

ICMP has no ports, seq #, or connection states like TCP. How stateful firewalls track ICMP “connection”?

Solution: Firewall creates pseudo-connection by tracking ICMP-specific fields

Stateless vs. Stateful Packet Filters: Example 2

Step 1: User pings google.com

192.168.1.100 → 8.8.8.8

Step 2: Firewall creates state table entry

Step 3: Google replies

8.8.8.8 → 192.168.1.100

Step 4: Firewall checks incoming packet

- Protocol: ICMP
- Type: 0 (Echo Reply)
- ICMP Identifier: 1234
- ICMP Sequence: 1
- Source IP: 8.8.8.8
- Destination IP: 192.168.1.100

Attacker sends unsolicited ICMP reply?

- No entry matches this ICMP message

Field	Value	Purpose
Session ID	#4740	Unique identifier for this "connection"
Protocol	ICMP	Protocol type
Type	8 (Echo Request)	ICMP message type sent
Expected Reply Type	0 (Echo Reply)	Expected response type
ICMP Identifier	1234	Unique ID from ping command
ICMP Sequence	1	Sequence number from ping
Source IP	192.168.1.100	Internal host
Destination IP	8.8.8.8	External host (google.com)
State	ESTABLISHED	Waiting for reply
Timestamp	14:30:45.123	When request was sent
Timeout	10 seconds	How long to wait for reply

Sample Question

RULE #	ACTION	SOURCE	DESTINATION	PORT	PROTOCOL	FLAGS/STATE	COMMENT
Rule 1	Allow	*	192.168.1.10	80	TCP	New	Allow new HTTP connections to web server
Rule 2	Allow	192.168.1.10	*	*	TCP	Established	Allow web server HTTP responses
Rule 3	Deny	*	*	*	*	*	Default deny rule

- **Packet to Evaluate:**

- Source IP: 203.0.113.5
- Destination IP: 192.168.1.10
- Destination Port: 443
- Protocol: TCP
- Flags: SYN

- *Question a:* Will this packet be allowed or denied by the policy? Justify your answer with reference to the rules and flags.
- *Question b:* Modify the given policy (if needed) to ensure the packet is handled according to the scenario requirements.

Sample Question

- Question a:
 - The packet will be denied.
 - Justification: The packet violates Rule 1, which only allows HTTP traffic (port 80) to the web server (192.168.1.10) from any source.
- Question b:
 - Add a new rule as follows

RULE #	ACTION	SOURCE	DESTINATION	PORT	PROTOCOL	FLAGS/STATE	COMMENT
Rule 1	Allow	*	192.168.1.10	80	TCP	New	Allow new HTTP connections to web server
Rule NEW	Allow	*	192.168.1.10	443	TCP	New	Allow new HTTPS connections to web server
Rule 2	Allow	192.168.1.10	*	*	TCP	Established	Allow web server HTTP/HTTPS responses
Rule 3	Deny	*	*	*	*	*	Default deny rule

Roadmap

- *Motivation and Security Policies*
- *Firewalls: Packet Filters*
- ***Firewall Limitations and Evasion***

New Challenge: Deep Packet Inspection

What We've Learned So Far:

- Packet filters inspect headers (IP addresses, ports, protocols, flags)
- They *don't examine payload*

From: A	To: B
Seq = 4	
Hello world	



Problem: What if malicious content is hidden inside the **payload**?

- Example scenario:
 - *Firewall Rule:* Deny all packets containing the word **root**
 - *Goal:* Block exploitation attempts that use '**root**' command

From: C	To: D
Seq = 2	
Log in	



Question: How does the firewall detect "root" in the payload?

- Must perform Deep Packet Inspection (DPI)
- Must reconstruct TCP streams to see complete messages

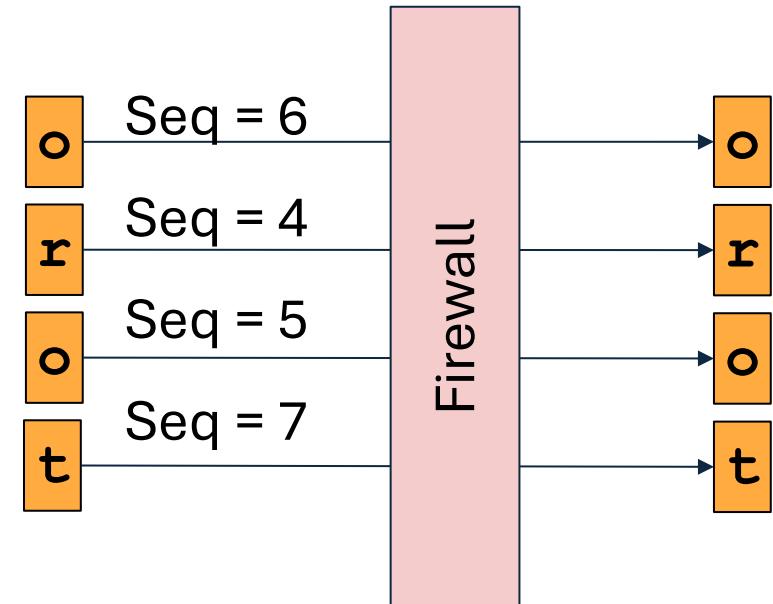
From: C	To: D
Seq = 8	
as root	



Evading Packet Filters

Question: How does the firewall detect "root" in the payload?

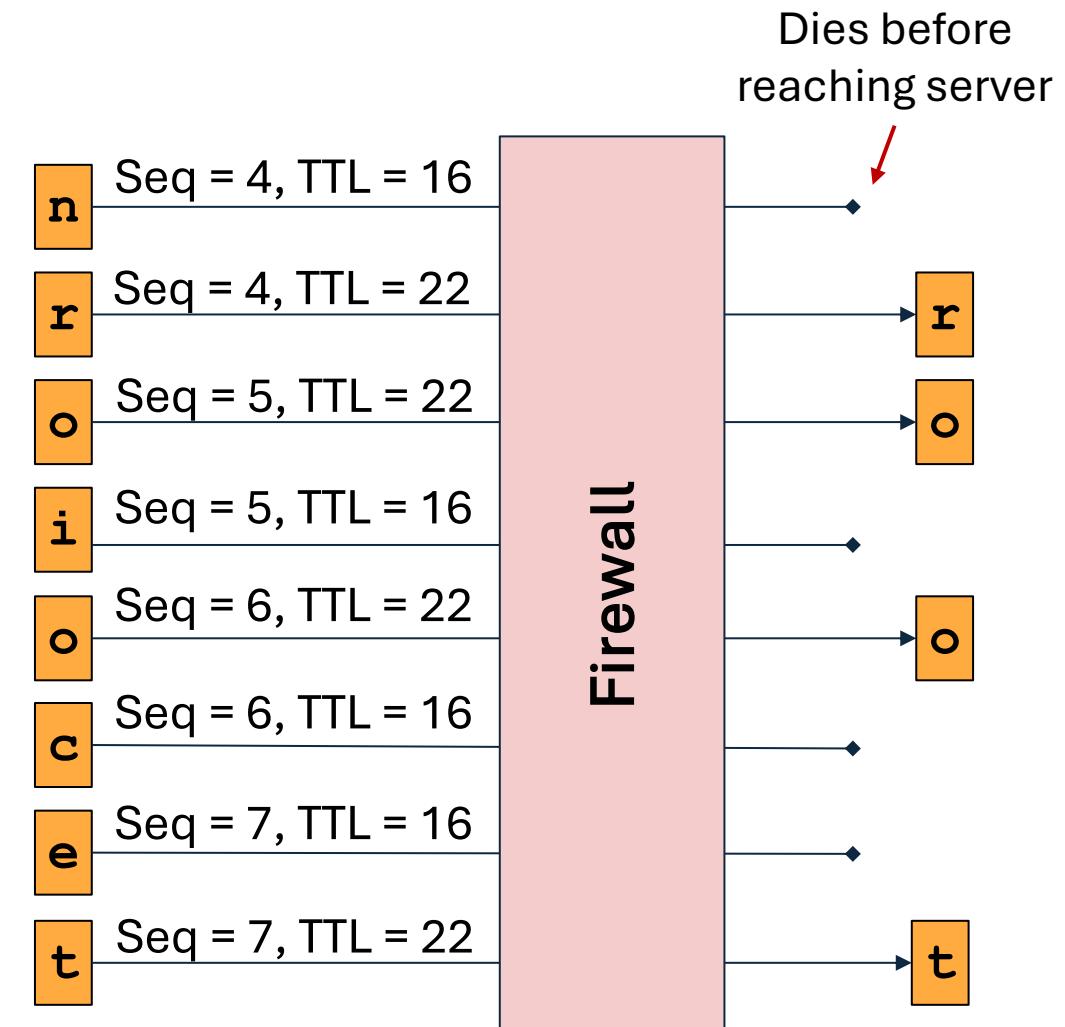
- Recall: TCP
 - Messages are split into packets before being sent
 - Packets can arrive out of order: The application will use sequence numbers to reorder packets
- *Attack 1: Payload Fragmentation*
 - Split the word **root** across multiple packets
 - No single packet contains **root**, so the firewall won't stop any of these packets
 - Destination host reassembles TCP stream → sees complete malicious content



Evading Packet Filters

Question: How does the firewall detect "root" in the payload?

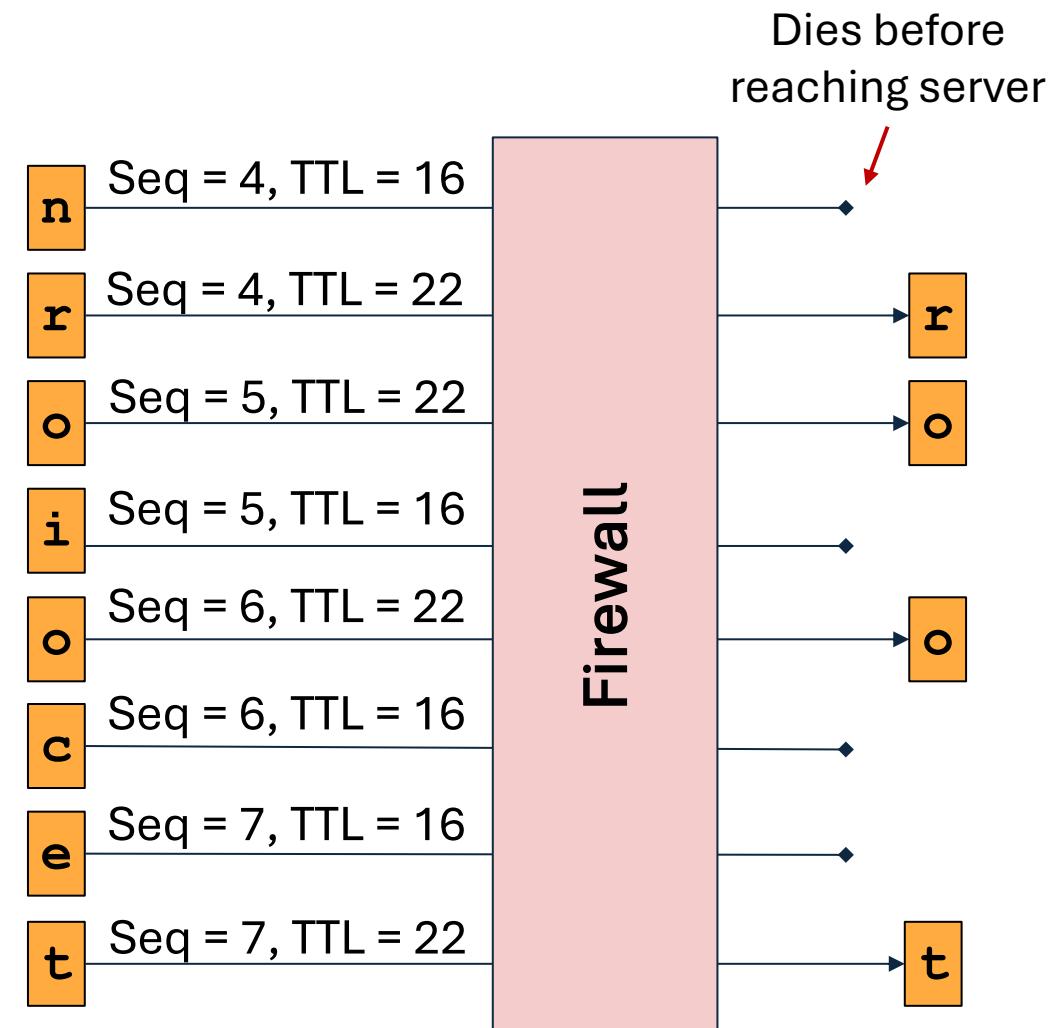
- Recall: IP packets have a time-to-live (TTL)
 - The number of hops a packet may take before the packet is dropped
 - The attacker can easily find how many hops away a given server is → Send ping packets with increasing TTLs until the server responds
- **Attack 2: TTL Evasion Attack**
 - Confuse firewall by sending duplicate packets with different TTLs
 - What the firewall sees: '**nroiocet**'
 - What the server sees: '**root**'



Evading Packet Filters

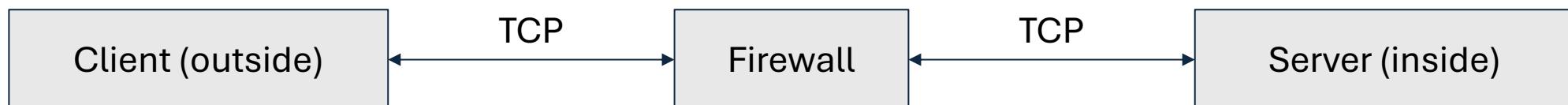
- Defense strategies

- Reconstruct TCP streams
- Inspect reassembled payload, not individual packets
- Apply content filters to complete messages
- Example: next-generation firewall ([NGFW](#))



Other Types of Firewalls

- **Proxy firewall:** Instead of forwarding packets, form two TCP connections: One with the source, and one with the destination
 - Avoids problems with packets, since the firewall has direct access to the TCP byte streams
- **Application proxy firewall:** Certain protocols allow for proxying at the application layer
 - Example: HTTP proxies will make an HTTP request on behalf of the user then return the HTTP response to the client



Summary: Firewalls

- Firewalls: Defend many devices by defending the network
 - Security policies dictate how traffic on the network is handled
- Packet filters: Choose to either forward or drop packets
 - **Stateless packet filters:** Choose depending on the packet only
 - **Stateful packet filters:** Choose depending on the packet and the history of the connection
 - Attackers can subvert packet filters by splitting key payloads or exploiting the TTL
- Proxy firewalls: Create a connection with both sides instead of forwarding packets
- ML-Powered NGFW: <https://www.paloaltonetworks.com/cyberpedia/what-is-an-ml-powered-ngfw>

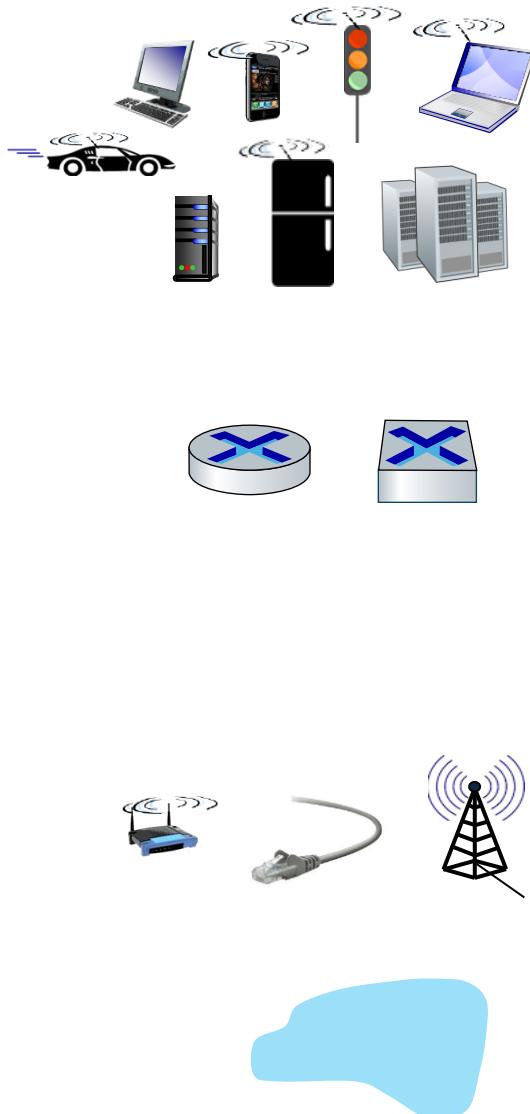
CSC 3511 Security and Networking

Week 2, Lecture 1: Network Structure

Roadmap

- ***Review: What is the Internet? What is a protocol?***
- *Network structure: how to build a “network of networks”*
- *Layers and encapsulation/decapsulation*

What's the “Internet”?



Billions of connected computing *devices*:

- *hosts* = *end systems*
- running *network apps* as “*network edge*”

Packet switches:

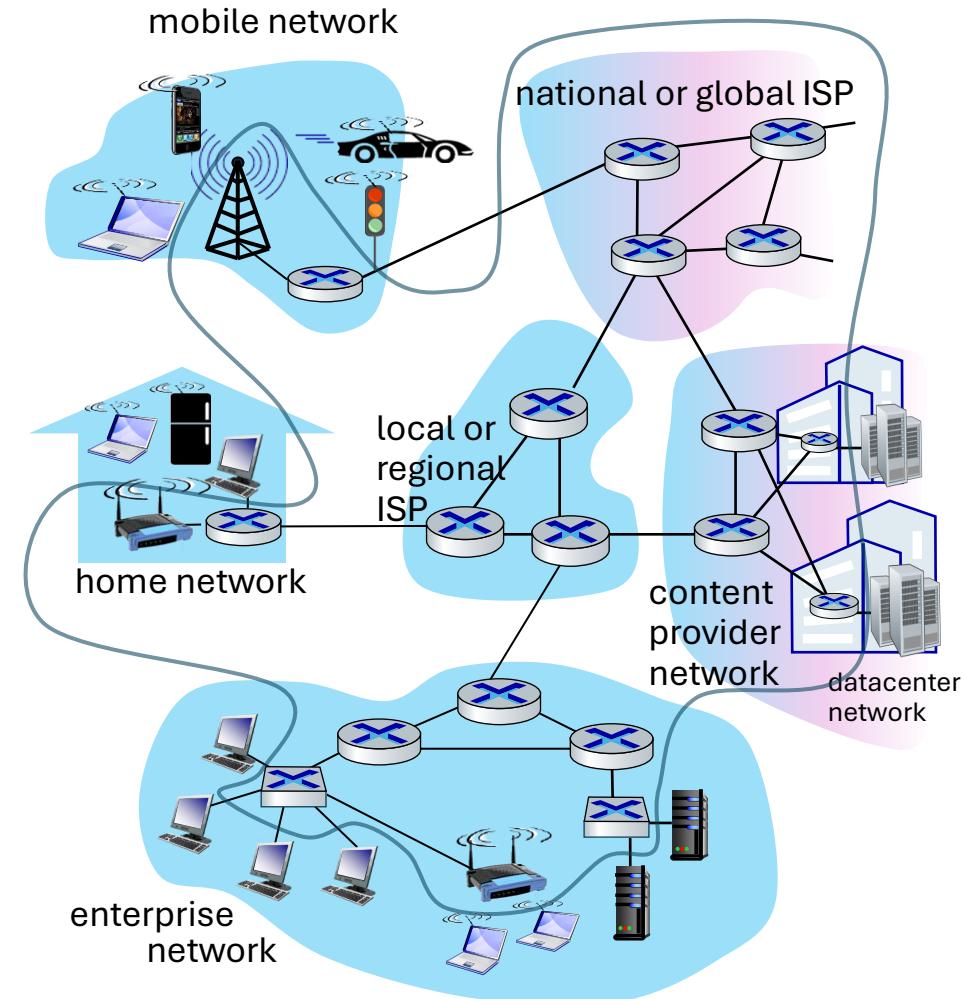
- forward packets (chunks of data) as “*network core*”
- *routers, switches*

Communication links

- fiber, copper, radio, satellite
- transmission rate: *bandwidth*

Networks

- collection of devices, routers, links: managed by an organization

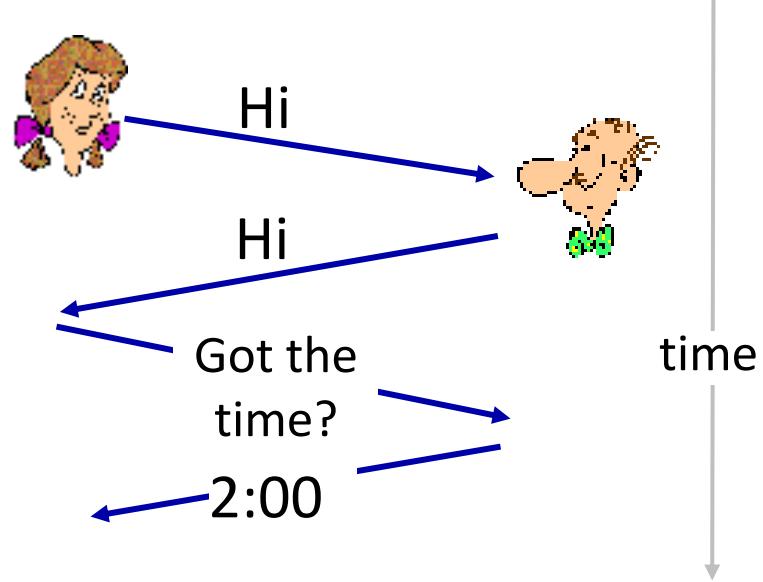


What's a protocol – A Human Analogy

Protocols define the format, order of messages sent and received among network entities, and actions taken on message transmission, receipt

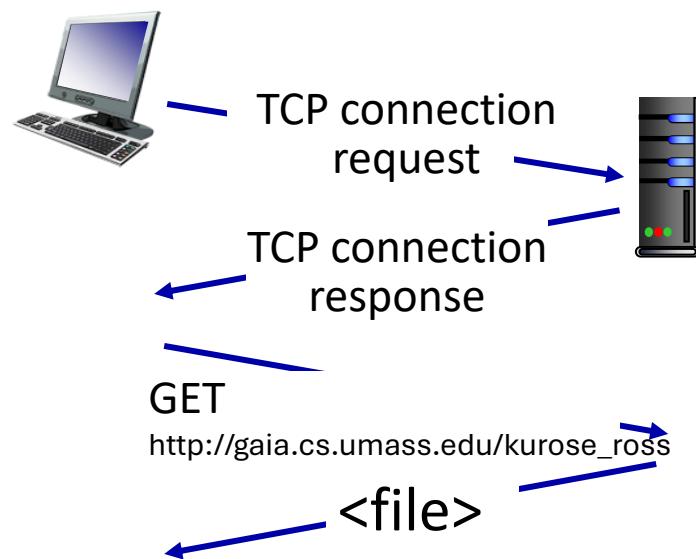
Human protocols:

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



Network protocols:

- computers (devices) rather than humans
- all communication activity in Internet governed by protocols



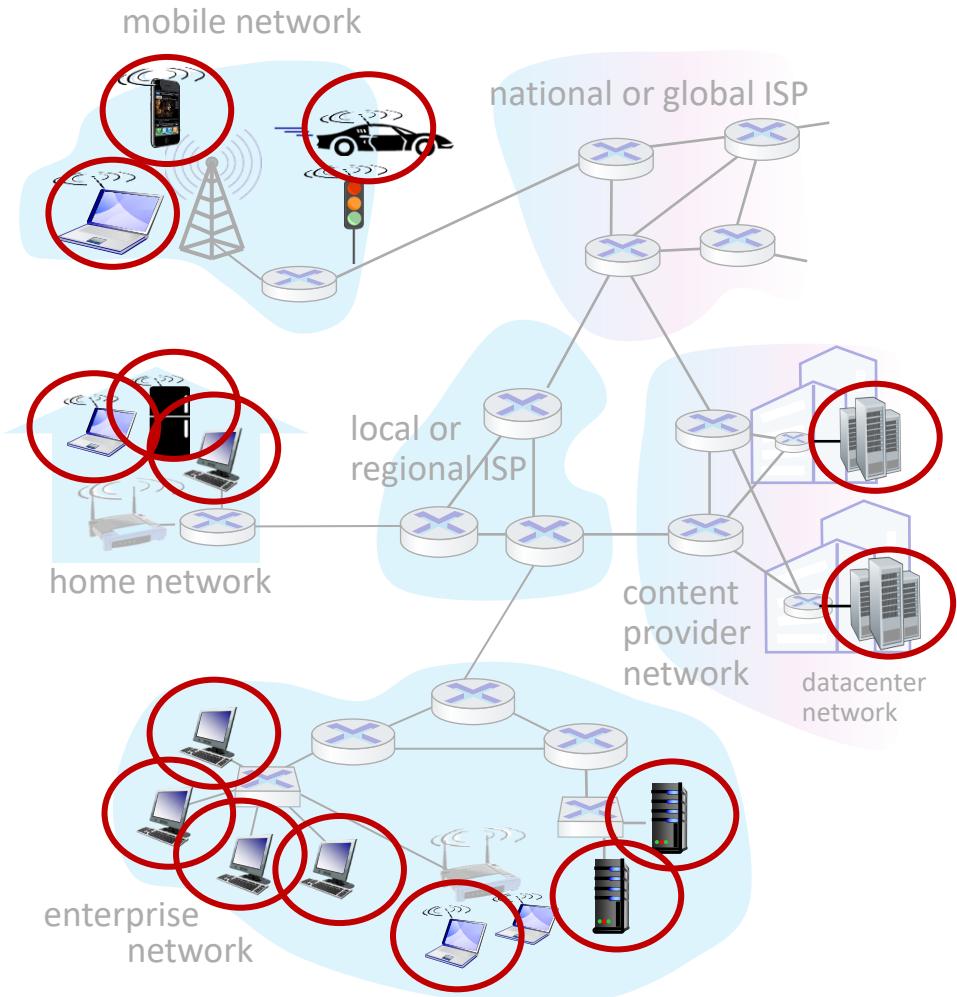
Roadmap

- *Review: What is the Internet? What is a protocol?*
- ***Network structure: how to build a “network of networks”***
- *Layers and encapsulation/decapsulation*

A closer look at Internet structure

Network edge (endpoint):

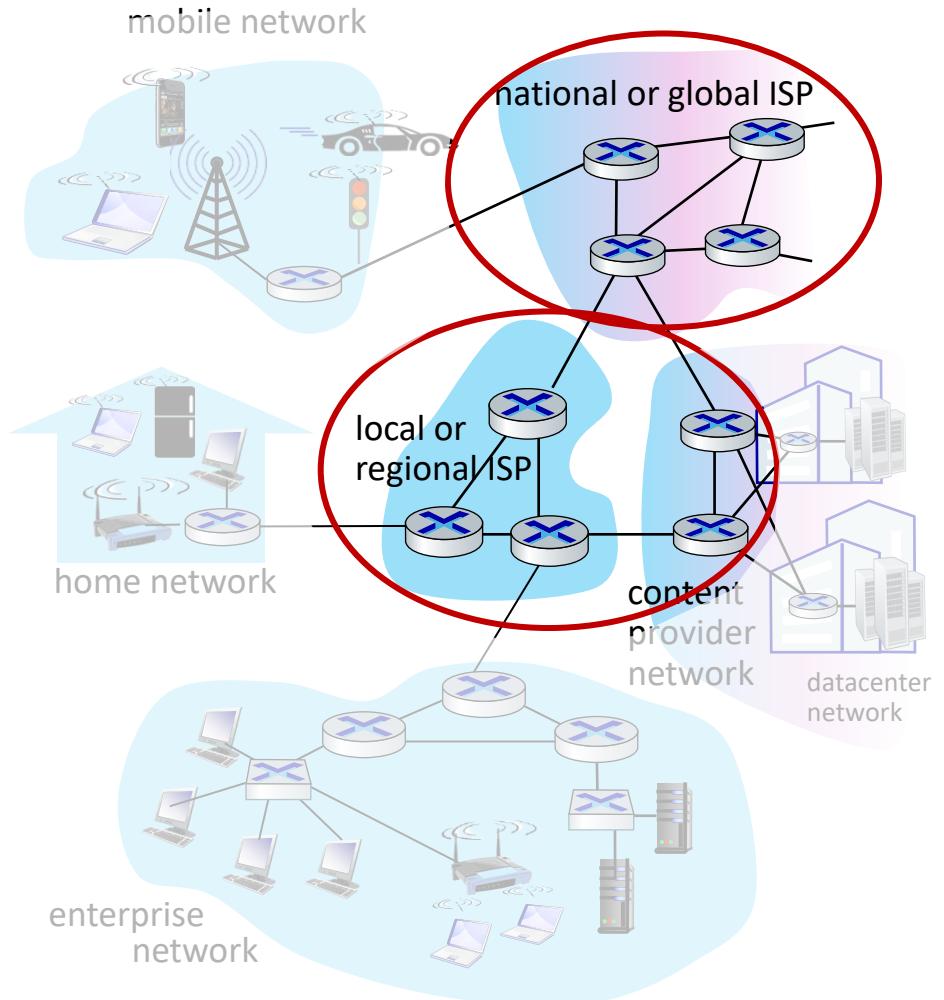
- Personal computers (PCs), adapters, modems, and devices that connect to them



A closer look at Internet structure

Network core:

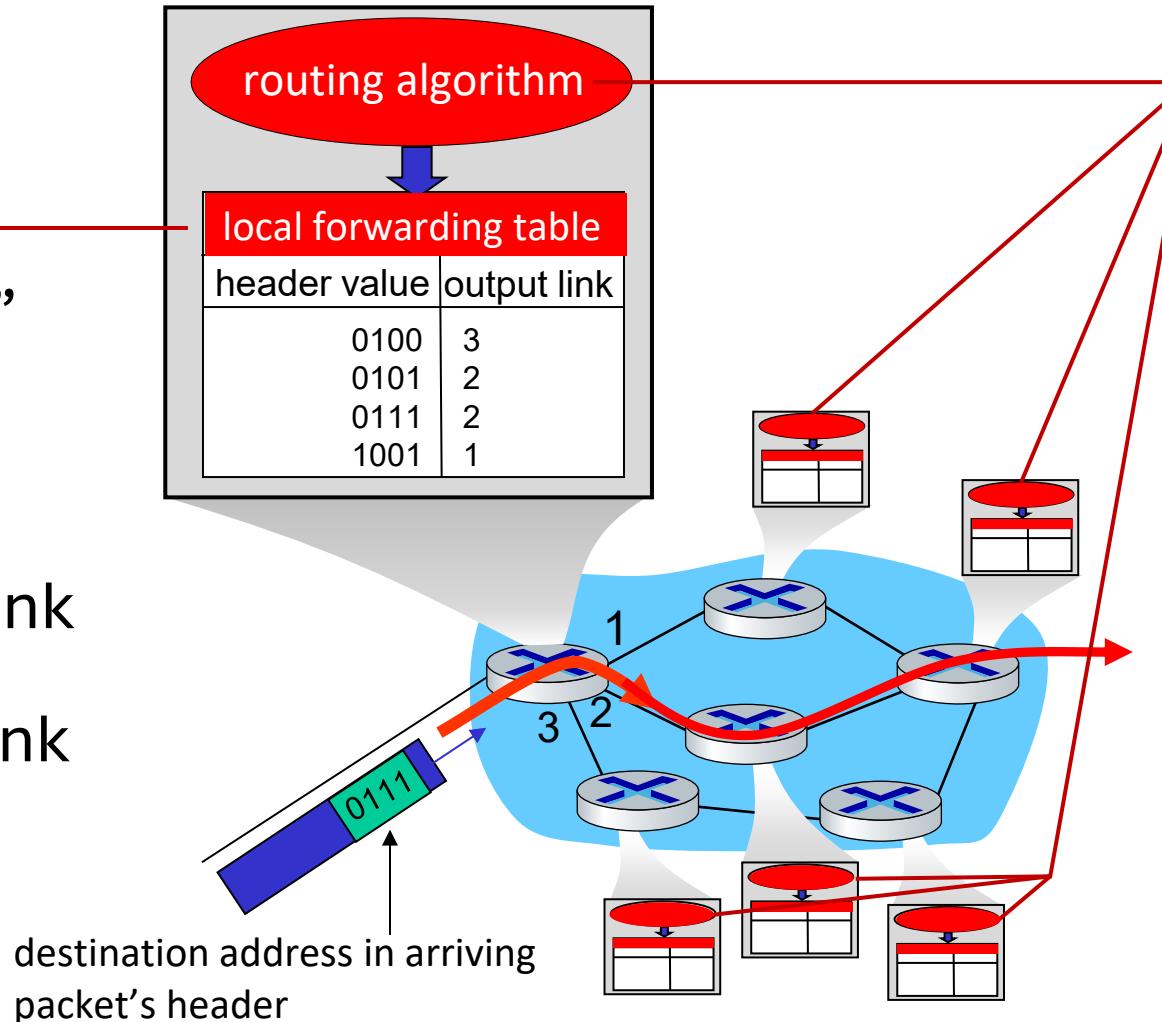
- interconnected routers
- network of networks



Two key network-core functions

Forwarding:

- aka “switching”
- *Local* action:
move arriving
packets from
router’s input link
to appropriate
router output link



Routing:

- Find paths through the network
- *Global* action:
determine source-destination paths taken by packets
- routing algorithms

Postal Analogy

- Imagine you're mailing a package from San Jose, CA to Northampton, MA
- **Routing** is like GPS planning your entire trip before you start:
 - Packages from California to Massachusetts should go through major sorting facilities in Denver, then Chicago, then finally to Massachusetts
 - This is the path determination - done once, updated occasionally
 - Need *routing algorithm*



- traceroute command:
 - PowerShell command:
 - tracert google.com
 - tracert 8.8.8.8
 - Linux command: traceroute google.com
 - Web-based tool:
<https://tools.keycdn.com/traceroute>

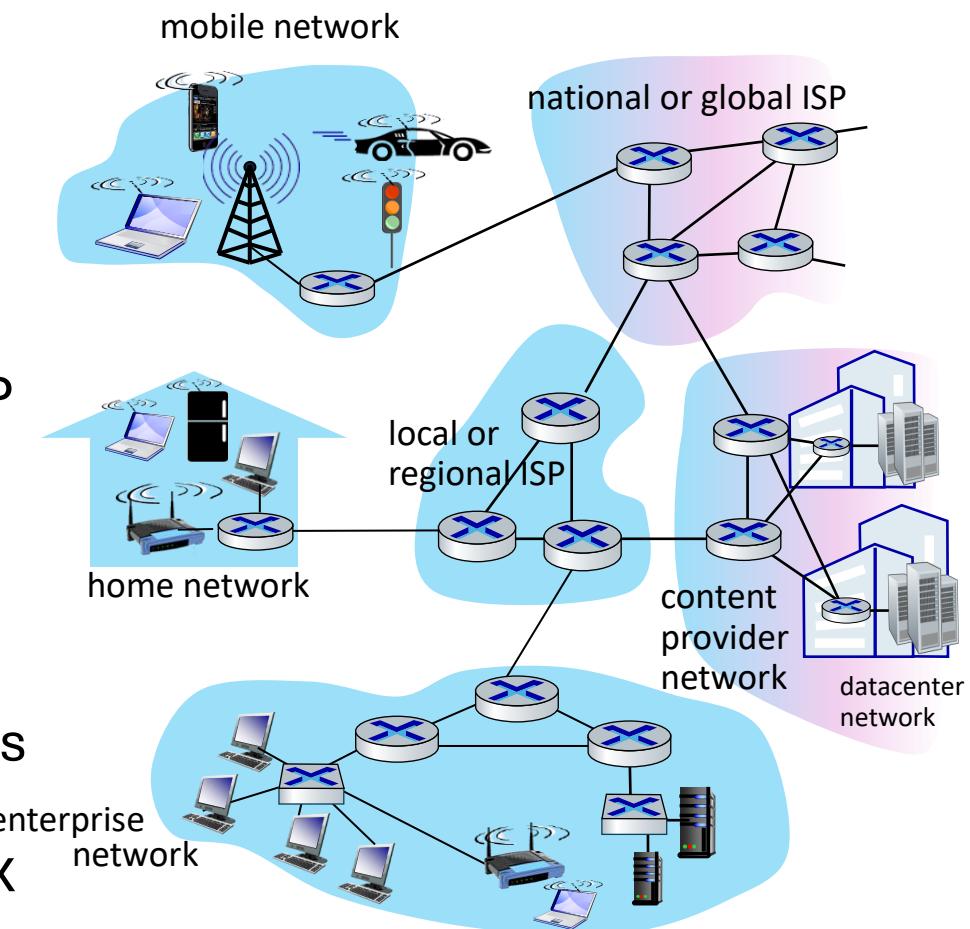
Postal Analogy

- Imagine you're mailing a package from San Jose, CA to Northampton, MA
- **Forwarding** is what you do at each highway intersection:
 - When you reach Sacramento, you see the intersection signs. You check your next-hop instruction: “*Destination: Northampton... take I-80 East*”
 - You don't recalculate the entire route → you just follow the sign for your next highway
 - A simple look up → This happens at every stop: a quick, local decisions based on **a simple lookup table**



Internet structure: a “network of networks”

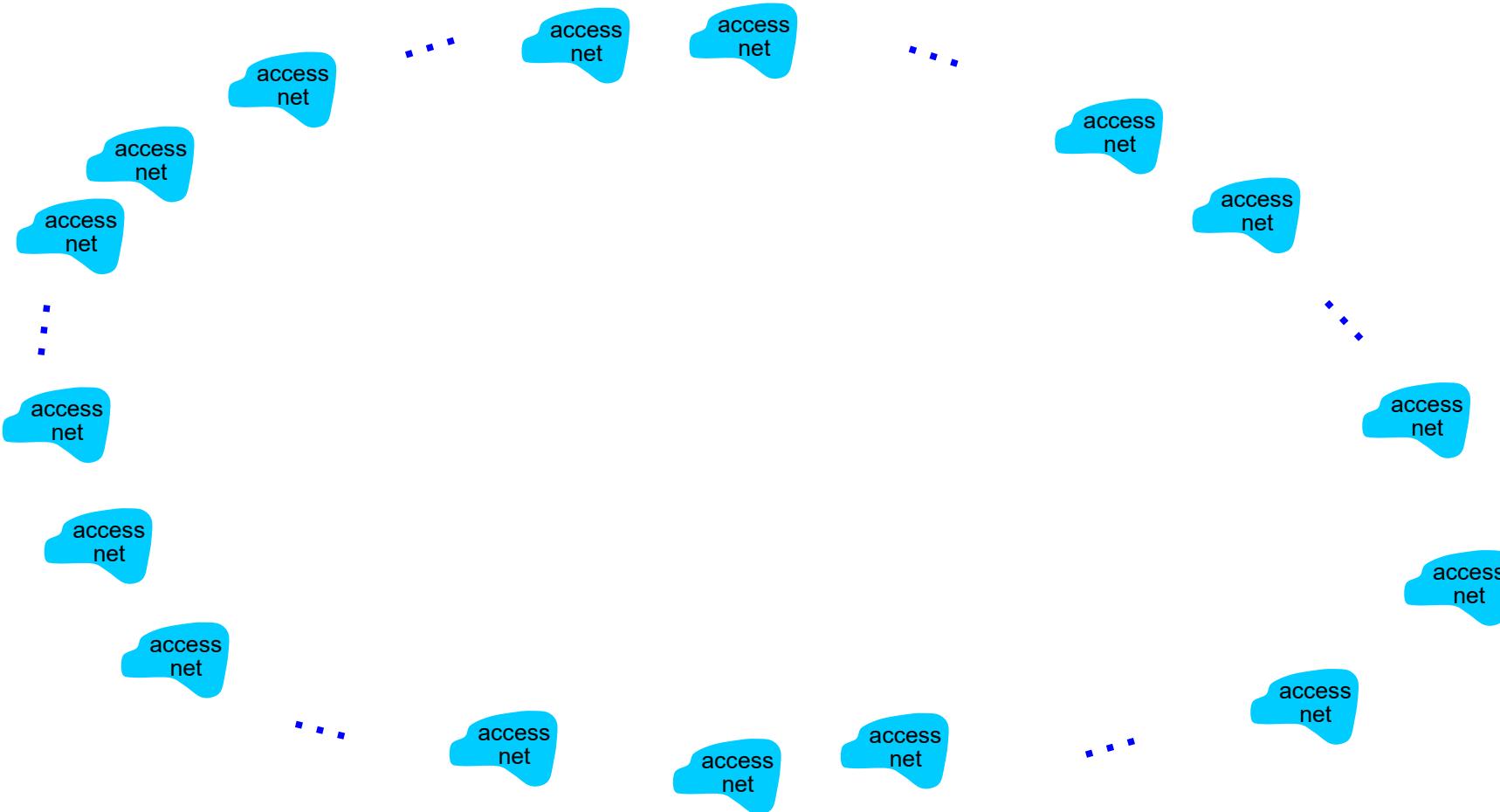
- Hosts connect to Internet via access Internet Service Providers (**ISPs**):
 - Residential: AT&T, Spectrum, Verizon
 - Mobile: T-Mobile, Verizon Wireless, AT&T Mobility
- What is an ISP?
 - What they do: Provide the physical connection + IP address assignment
 - What you pay for: Bandwidth (50 Mbps, 1 Gbps), often with data caps
- Access ISPs in turn must be interconnected
 - So that *any*two hosts (*anywhere!*) can send packets to each other
- Resulting network of networks is very complex



Let's take a stepwise approach to describe current Internet structure

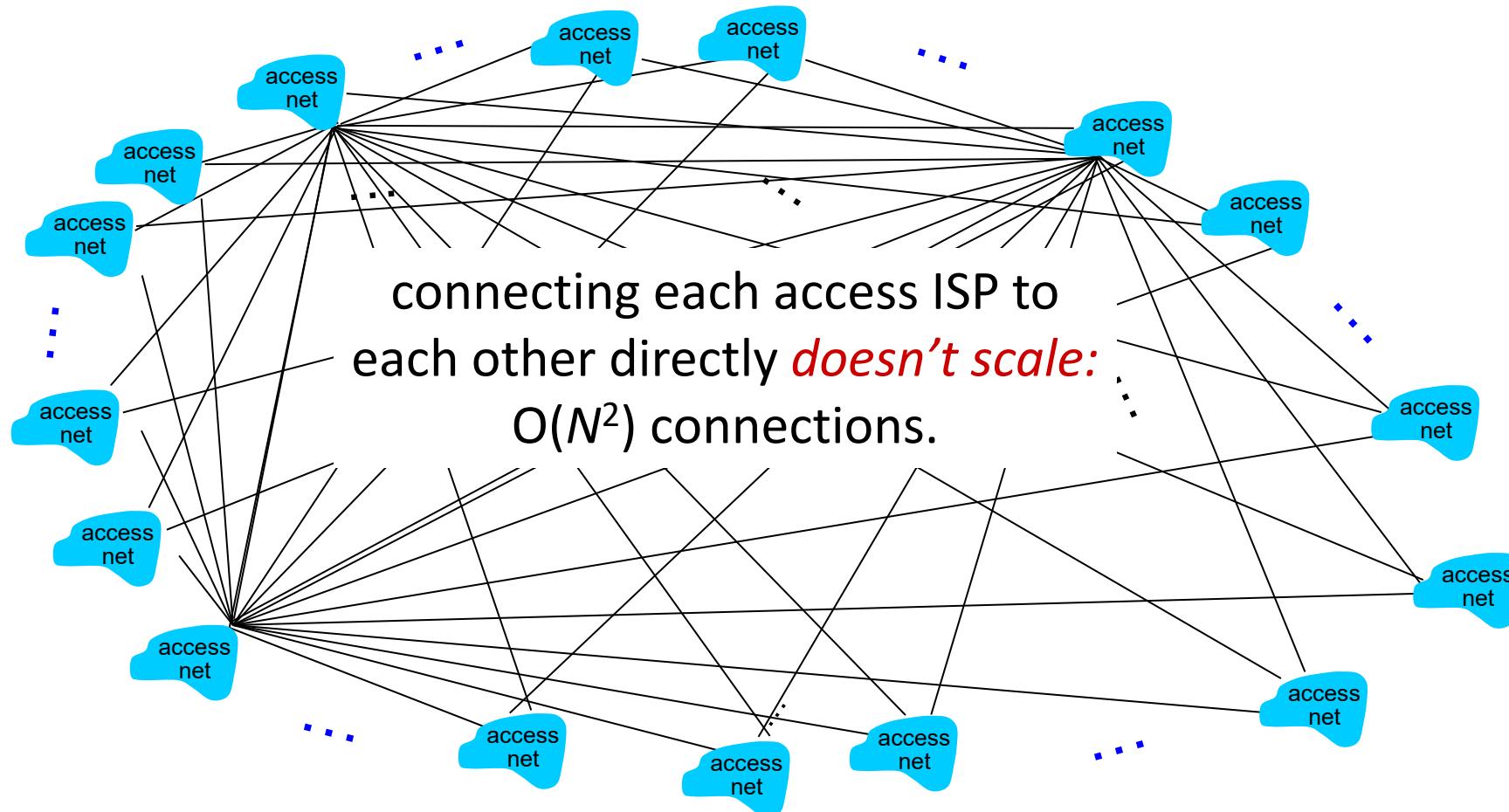
Internet structure: a “network of networks”

Question: given *millions* of access Internet Service Providers (ISPs), how to connect them together?



Internet structure: a “network of networks”

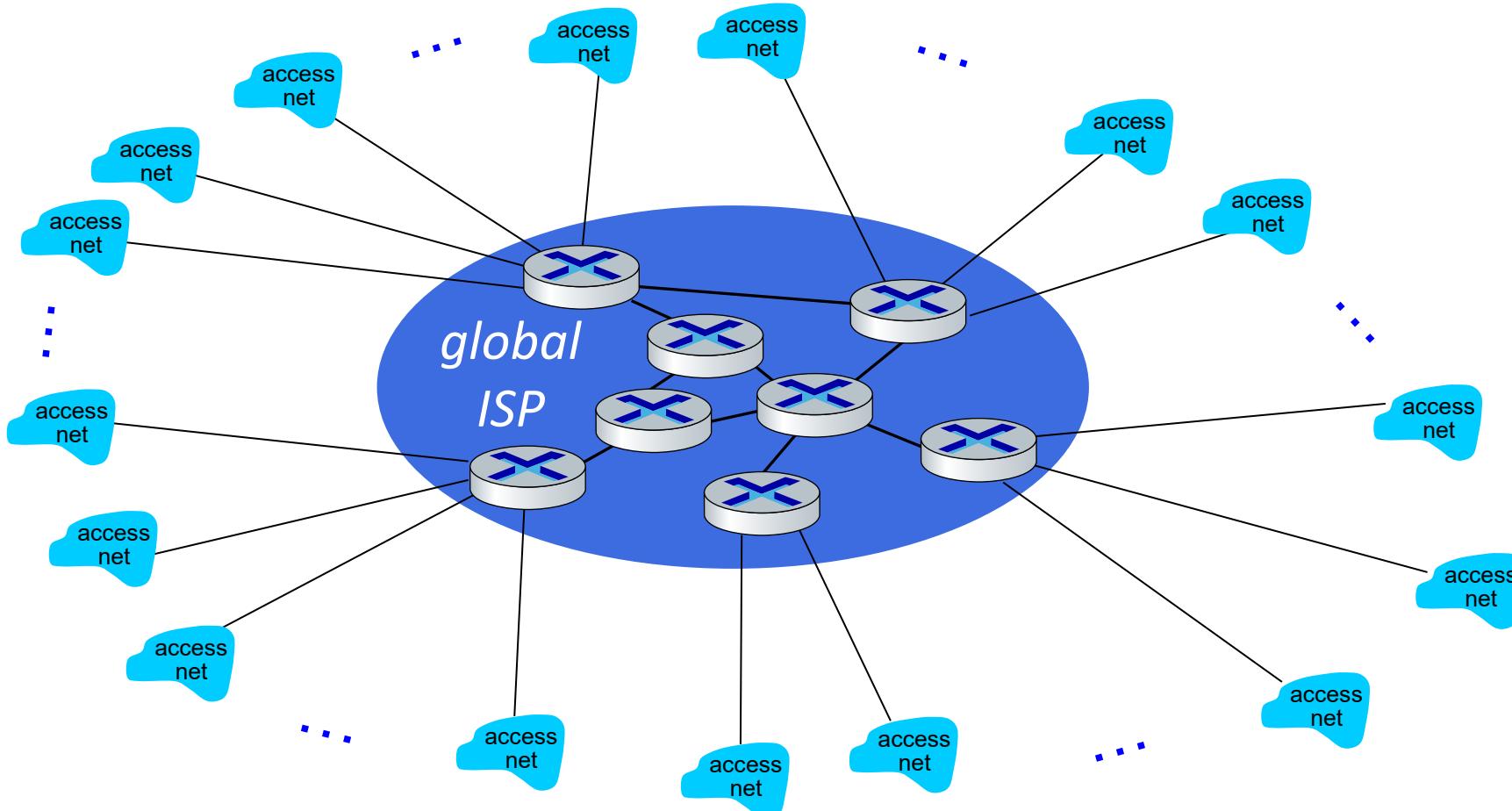
Question: given *millions* of access ISPs, how to connect them together?



Internet structure: a “network of networks”

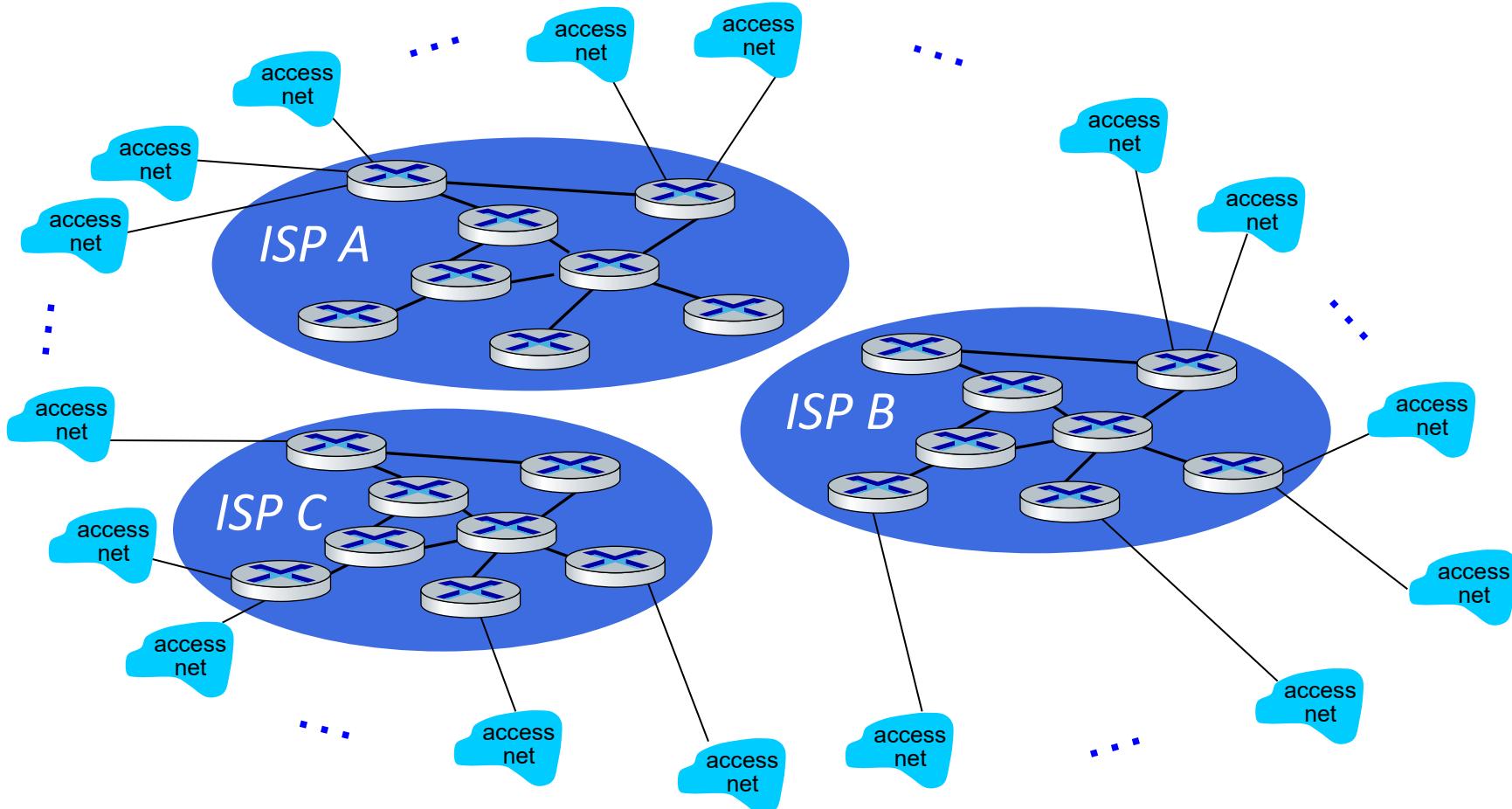
Option: connect each access ISP to one global transit ISP?

Customer and provider ISPs have economic agreement.



Internet structure: a “network of networks”

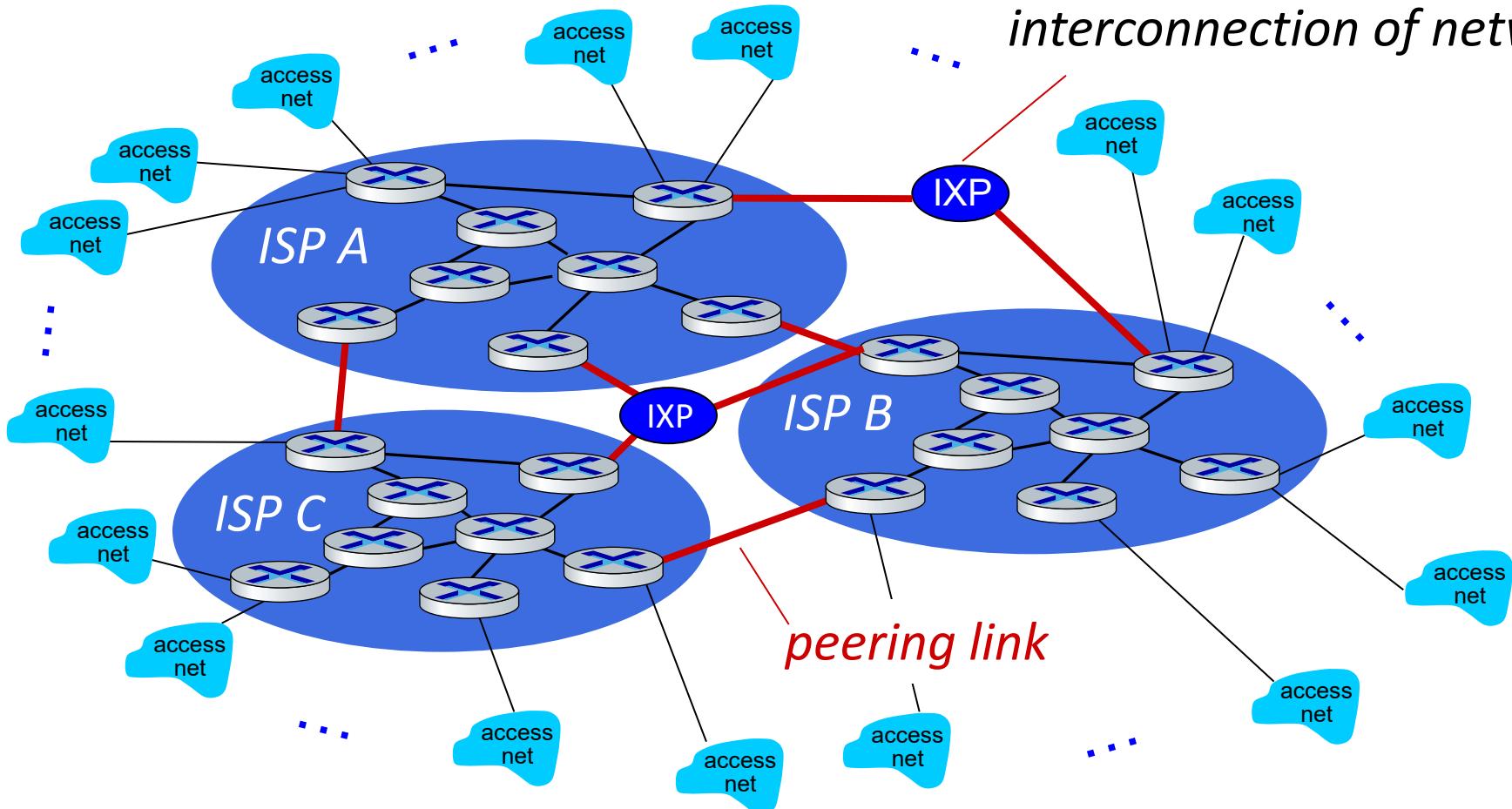
But if one global ISP is viable business, there will be competitors



Internet structure: a “network of networks”

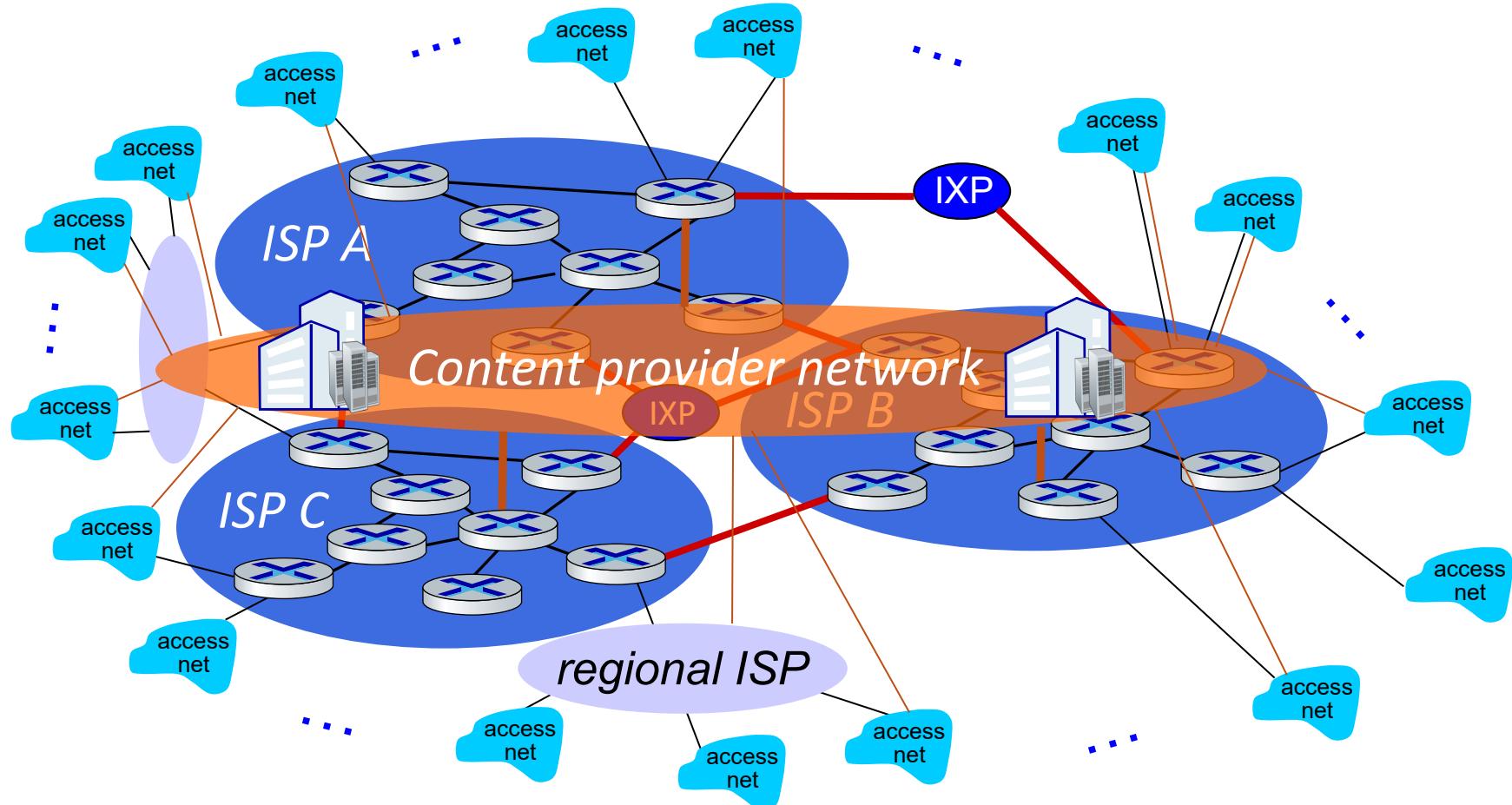
But if one global ISP is viable business, there will be competitors ... who will want to be connected

Internet exchange point: a direct interconnection of networks

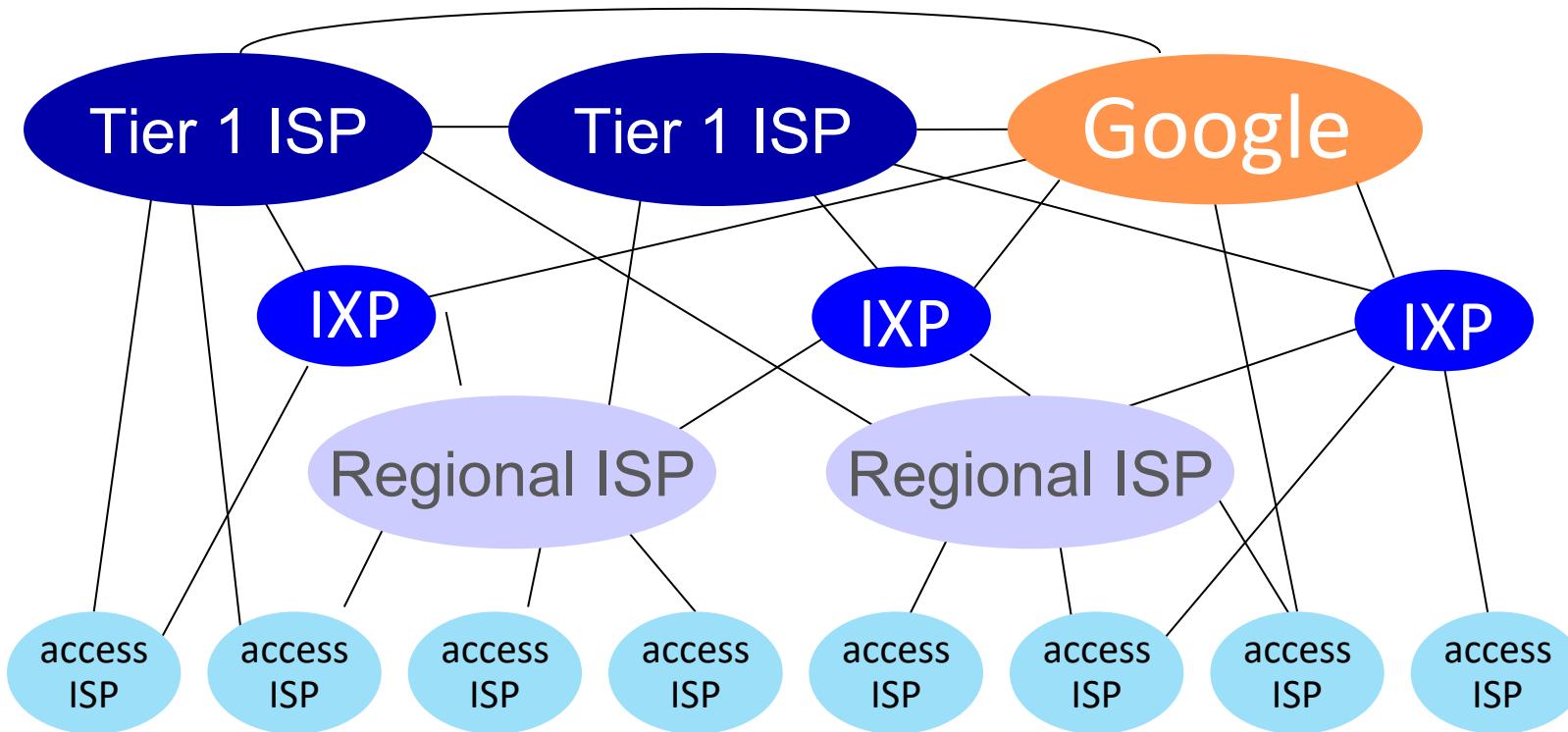


Internet structure: a “network of networks”

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

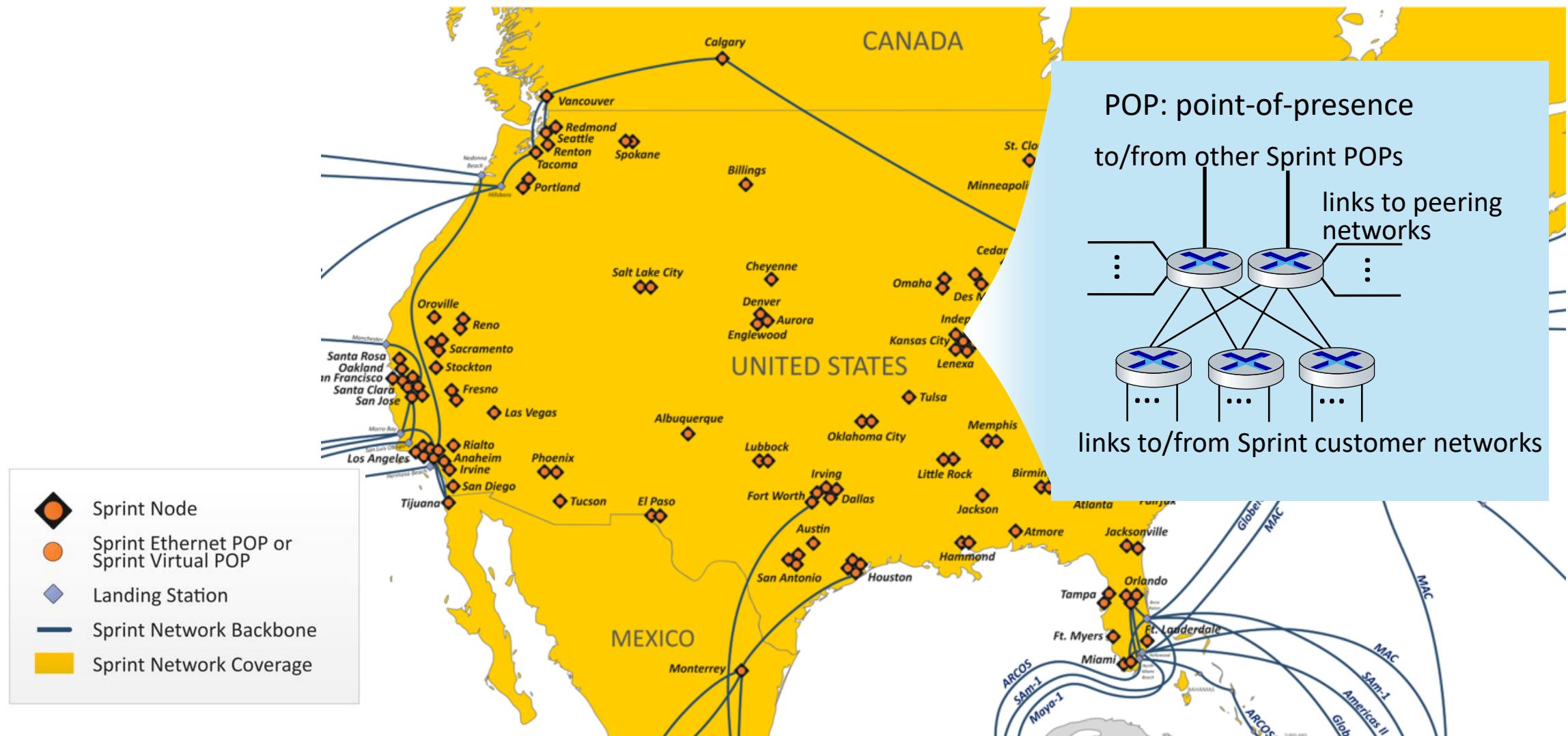


Internet structure: a “network of networks”



- At “center”: small # of well-connected large networks
 - “tier-1” commercial ISPs (e.g., Sprint, AT&T, Verizon), national & international coverage
 - content provider networks (e.g., Google, Facebook): private network that connects its data centers to Internet, often bypassing tier-1, regional ISPs
 - It's messier than a pure hierarchy - economics and politics shape it

Tier-1 ISP Network map: Sprint (2019)

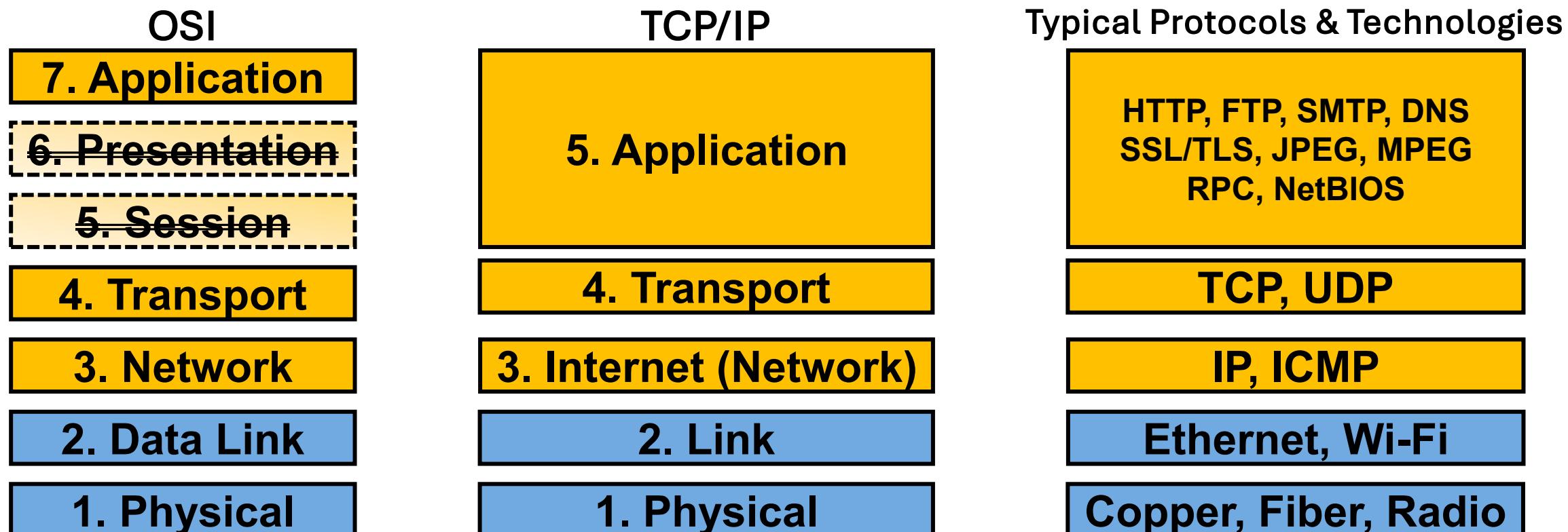


Roadmap

- *Review: What is the Internet? What is a protocol?*
- *Network structure: how to build a “network of networks”*
- ***Layers and encapsulation/decapsulation***

Internet layering: OSI vs. TCP/IP model

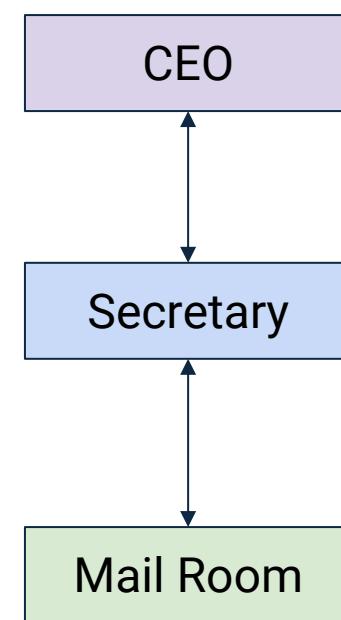
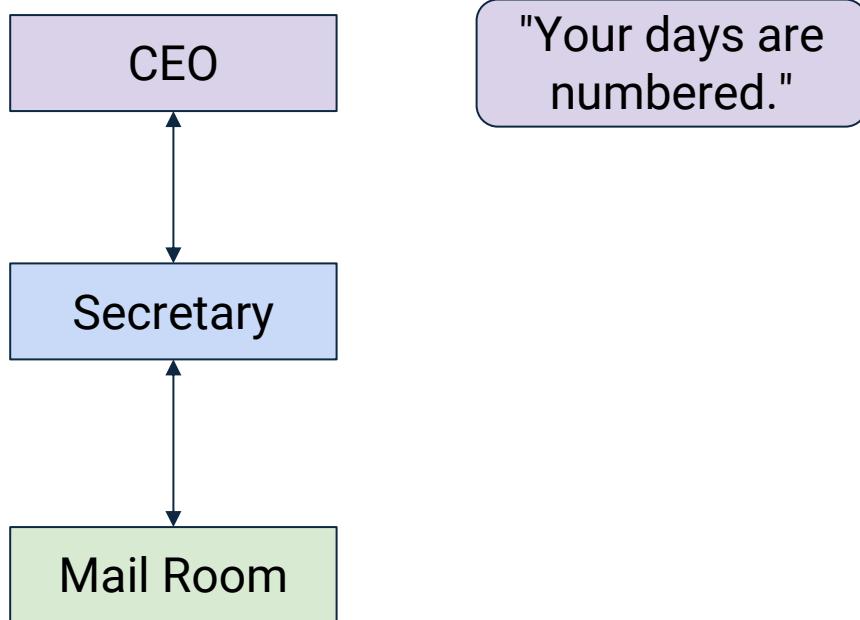
- Transmission Control Protocol/Internet Protocol (TCP/IP) model:
 - Practical model used on today's Internet → Layers 5 (Session) + 6 (Presentation) are usually bundled into Application in TCP/IP
- Different models, same idea → networking is layered abstraction, where each layer provides services to the one above
- We follow the TCP/IP model (aka the Internet protocol suite model) to organize course content



Layer of the Network – Postal Analogy

CEO Alice wants to send a message to CEO Bob

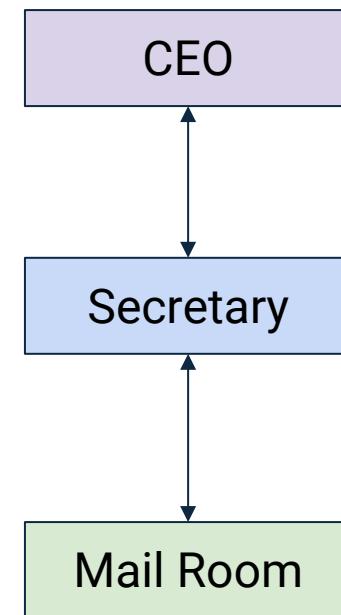
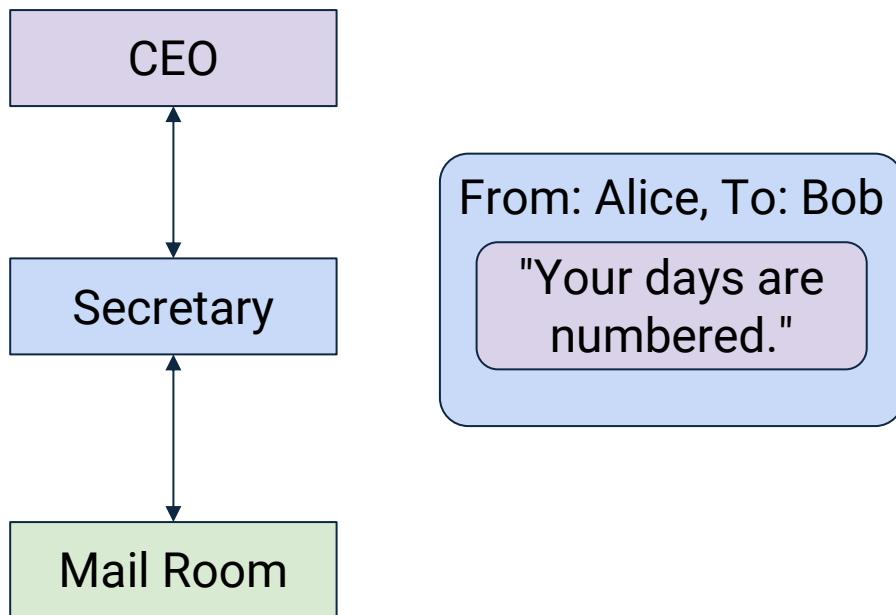
Alice writes a letter



Layer of the Network – Postal Analogy

Alice passes the letter down to her secretary

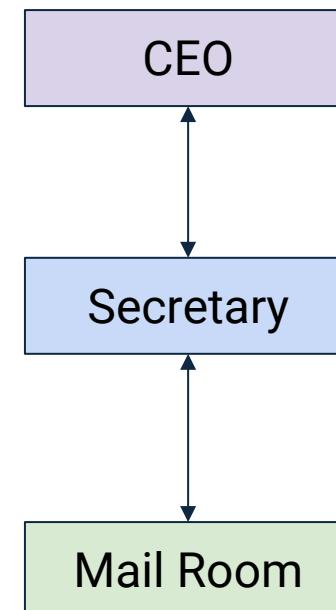
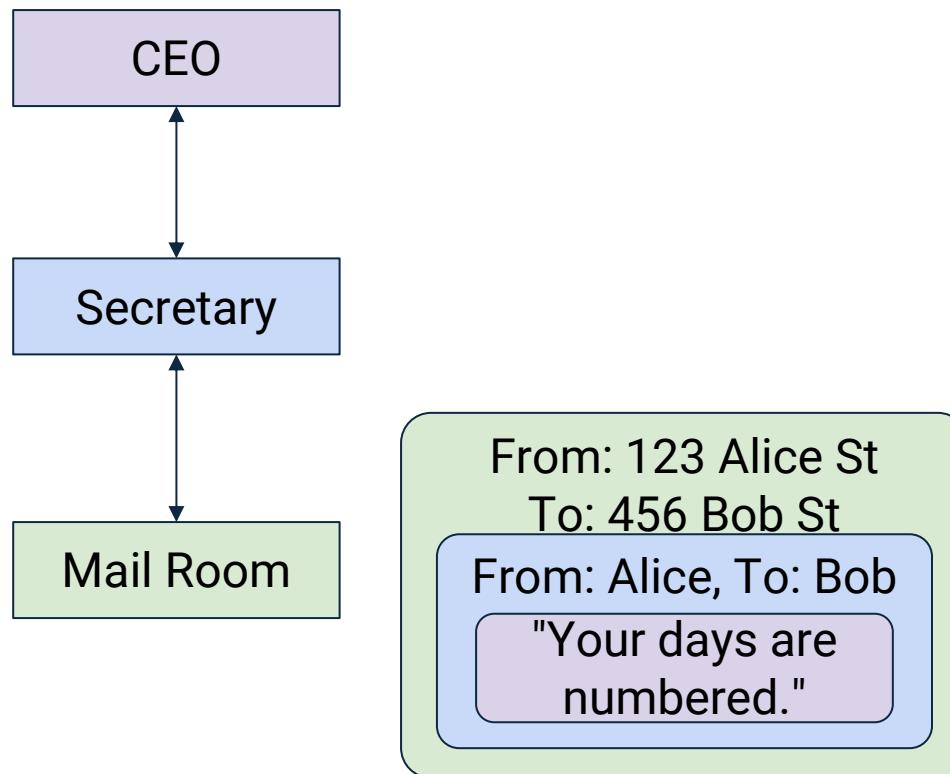
Her secretary puts the letter in an envelope



Layer of the Network – Postal Analogy

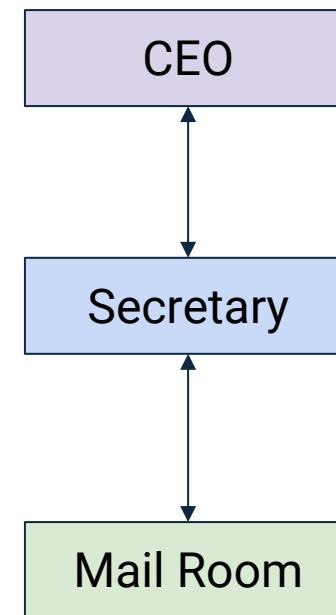
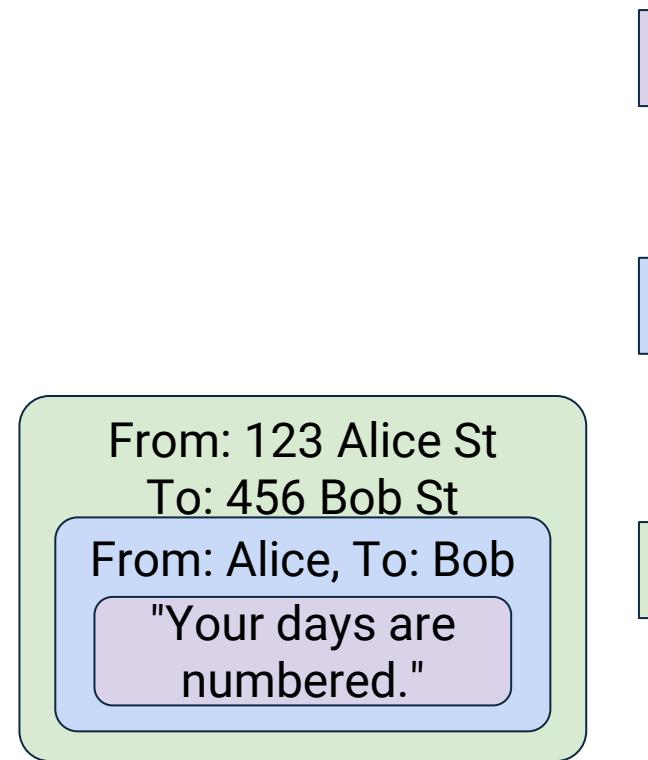
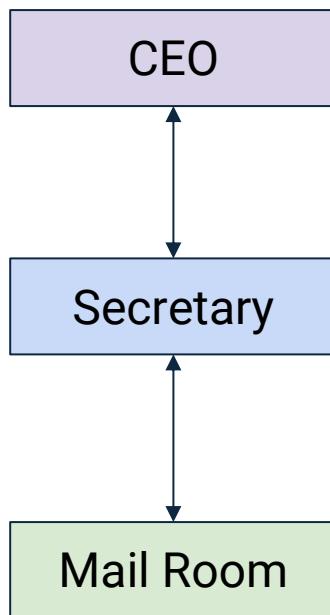
Her secretary passes the letter down to the mailman

The mailman puts the envelope in a box



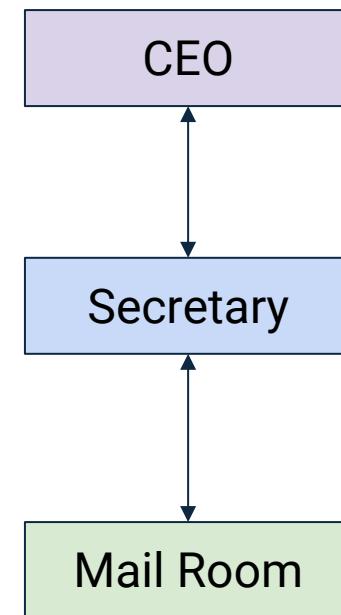
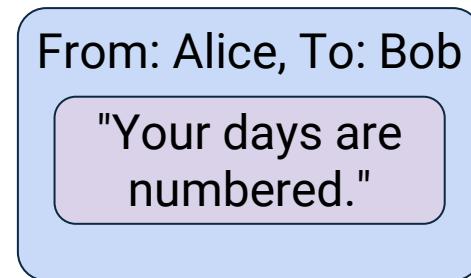
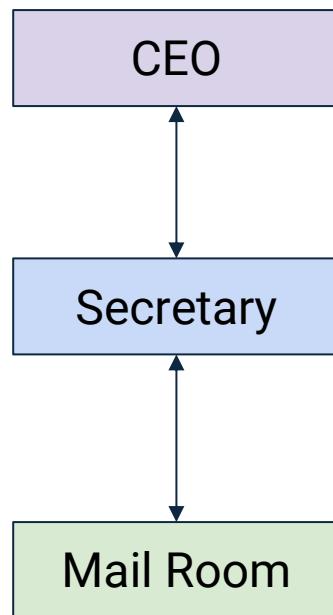
Layer of the Network – Postal Analogy

The packet travels through the postal system, to Bob's building



Layer of the Network – Postal Analogy

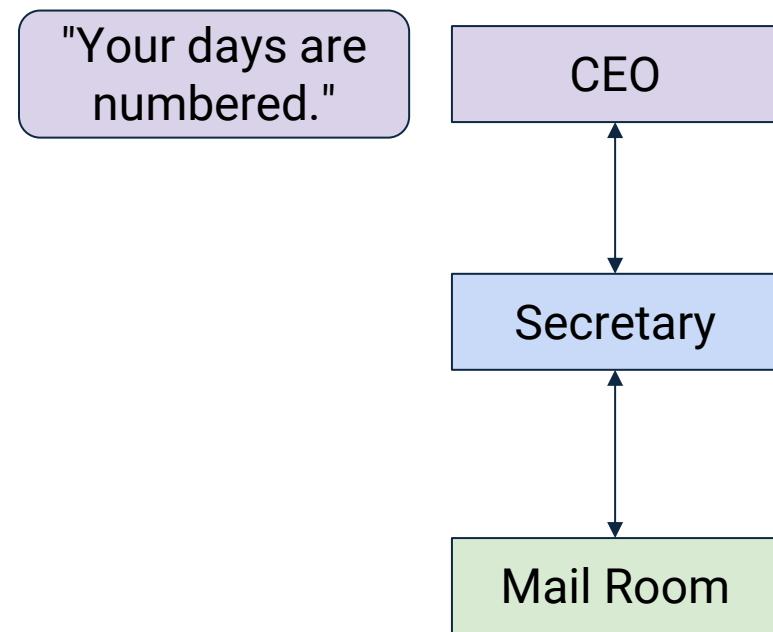
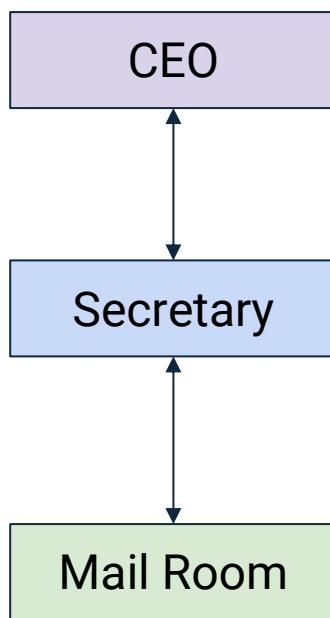
The mailman unwraps the box, revealing the envelope inside
The mailman passes the envelope up to the secretary



Layer of the Network – Postal Analogy

The secretary unwraps the envelope, revealing the letter inside.

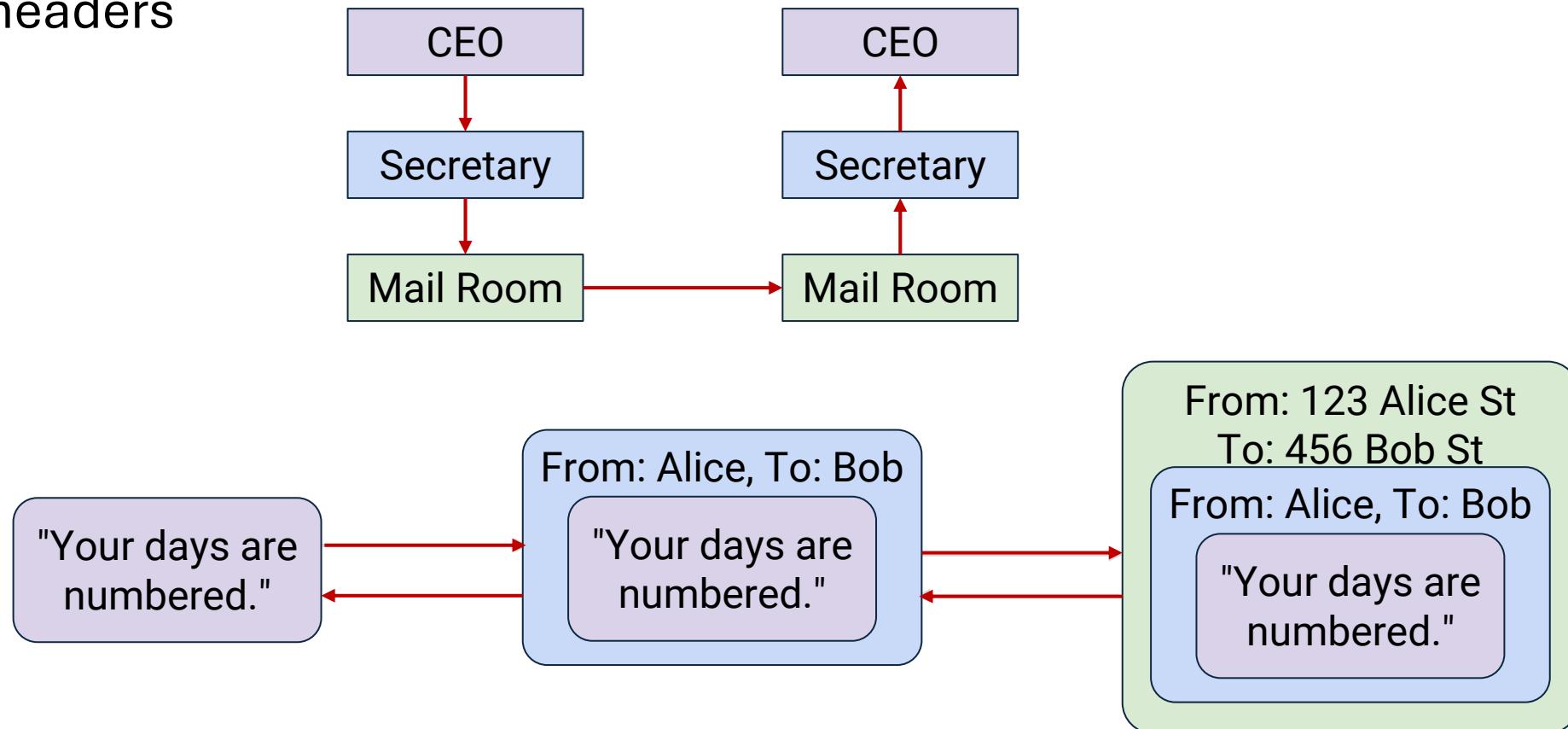
The secretary passes the letter up to Bob.



Layer of the Network – Postal Analogy

As we move to lower layers, we wrap additional **headers** (and sometimes a trailer) around the packet → **Encapsulation**

As we move to higher layers, we peel off headers (**decapsulation**), revealing the inner headers



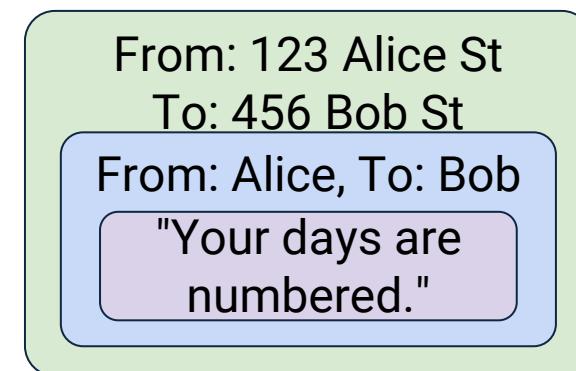
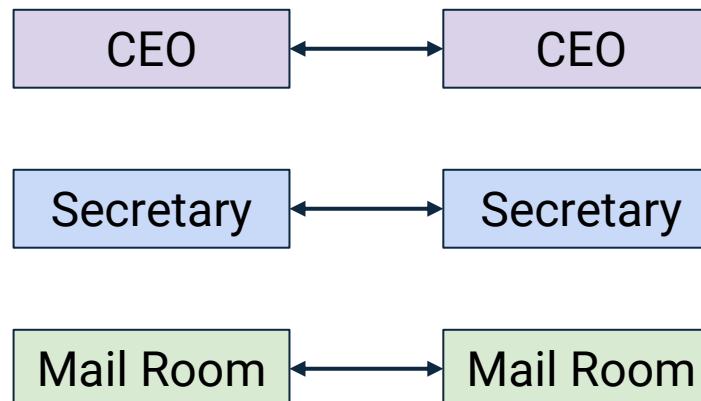
Layer of the Network – Postal Analogy

Each person only cares about the headers at their layer

- Mailman reads the green header, ignores all the payload inside

Each person communicates with its peers at the same layer

- Alice's secretary writes the blue header, for Bob's secretary to read
- A protocol at a specific layer only makes sense to people at that layer



Mailman only cares about this.

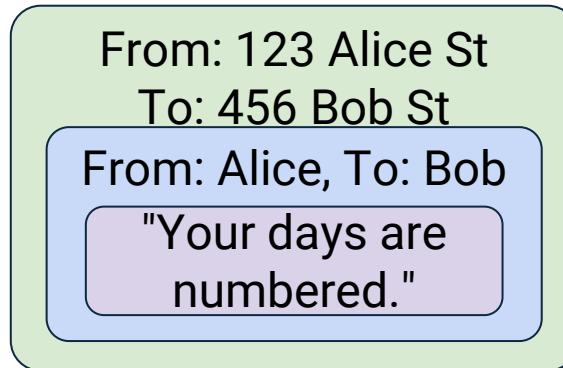
Secretary only cares about this.

CEO only cares about this.

Layer of the Network – Postal Analogy

Notice: Different layers use different **addressing schemes**

- Inside a building: “329 Diercks Hall.”
- In the postal system: “1025 N Milwaukee St, Milwaukee, WI 53202”

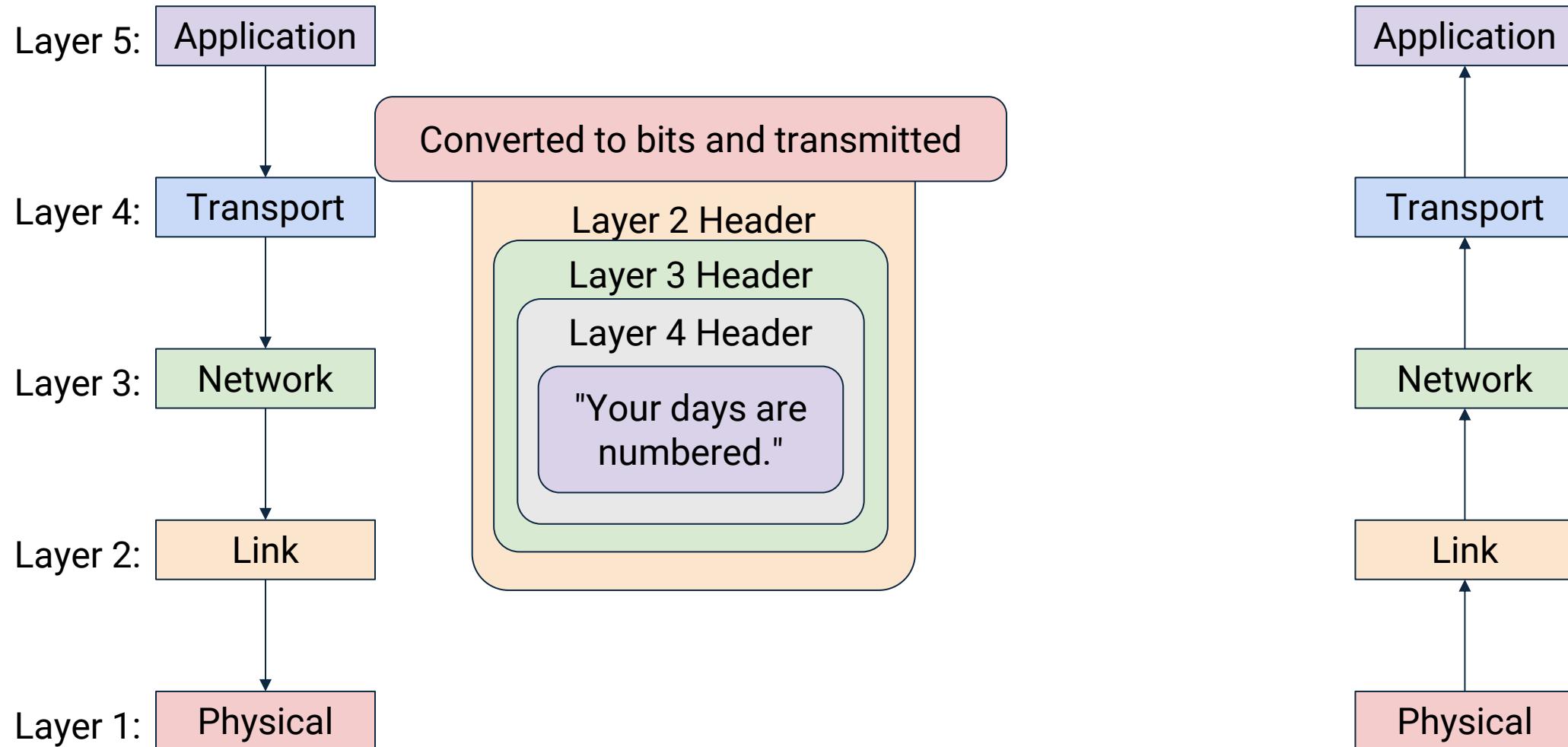


These addresses make sense to the mailman

These names make sense to the secretary

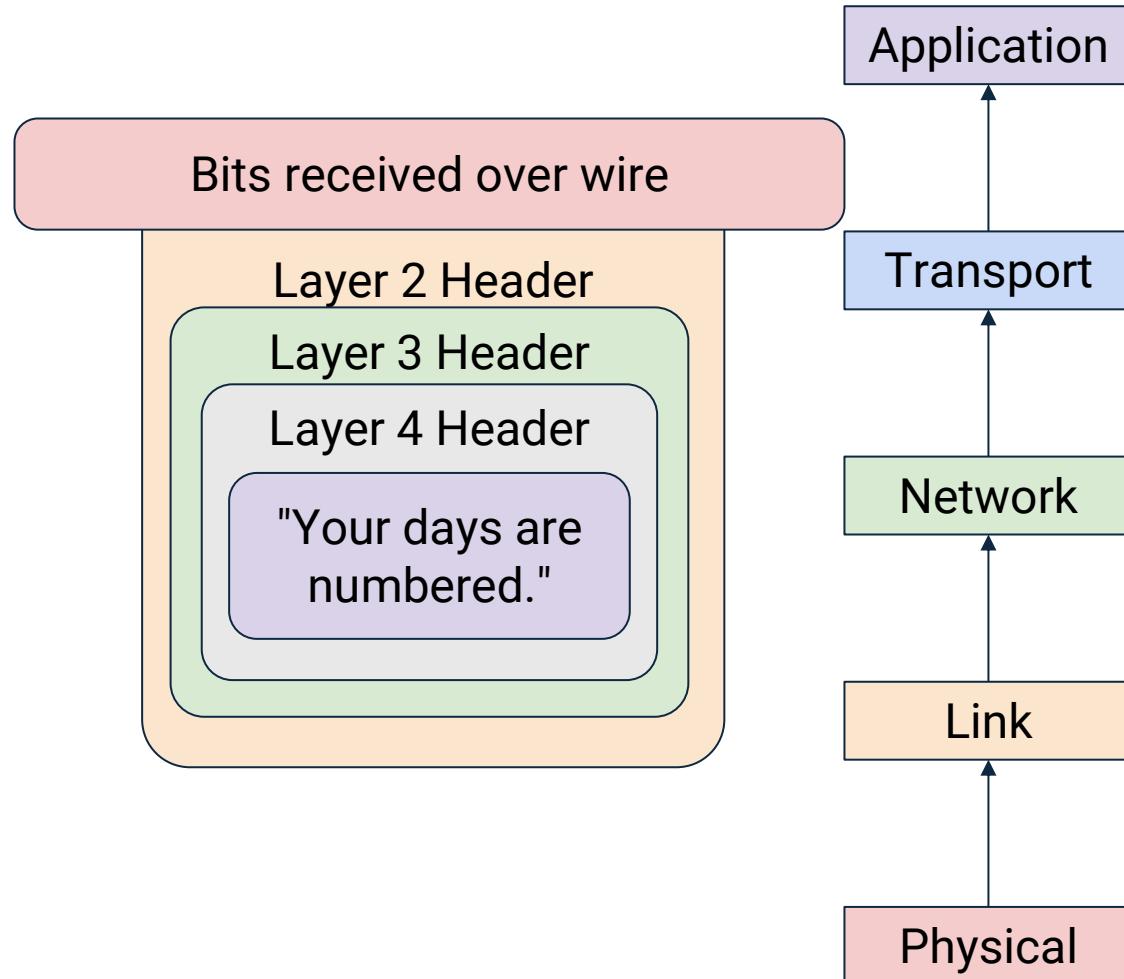
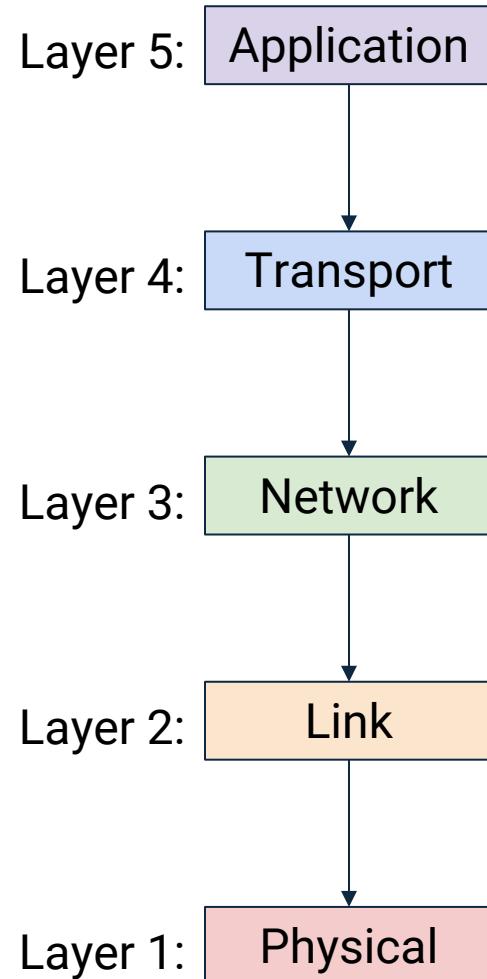
Internet Headers and Encapsulation

As we move to lower layers, we wrap additional headers around the packet



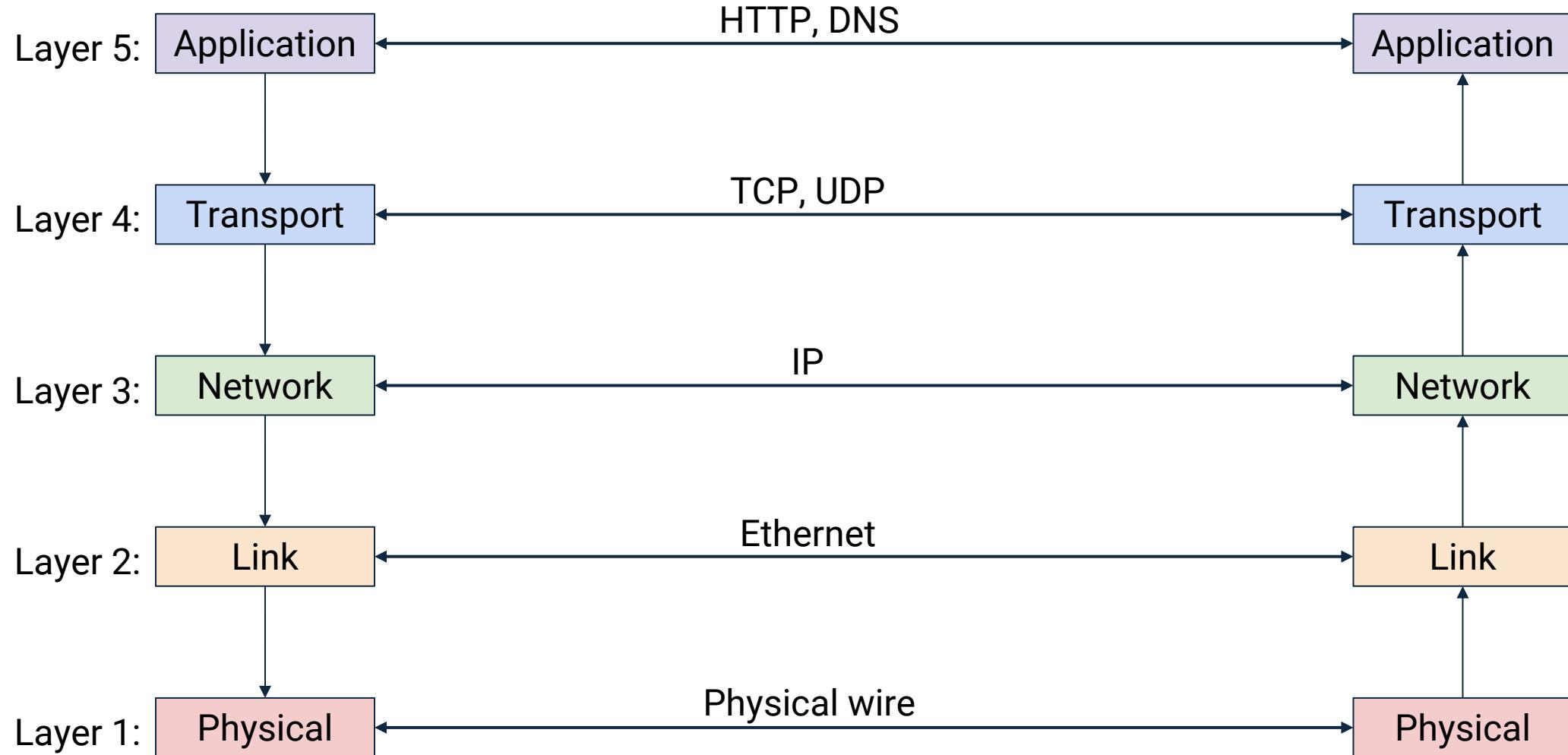
Internet Headers and Decapsulation

As we move to higher layers, we peel off headers, revealing the inner headers



Layers and Protocols

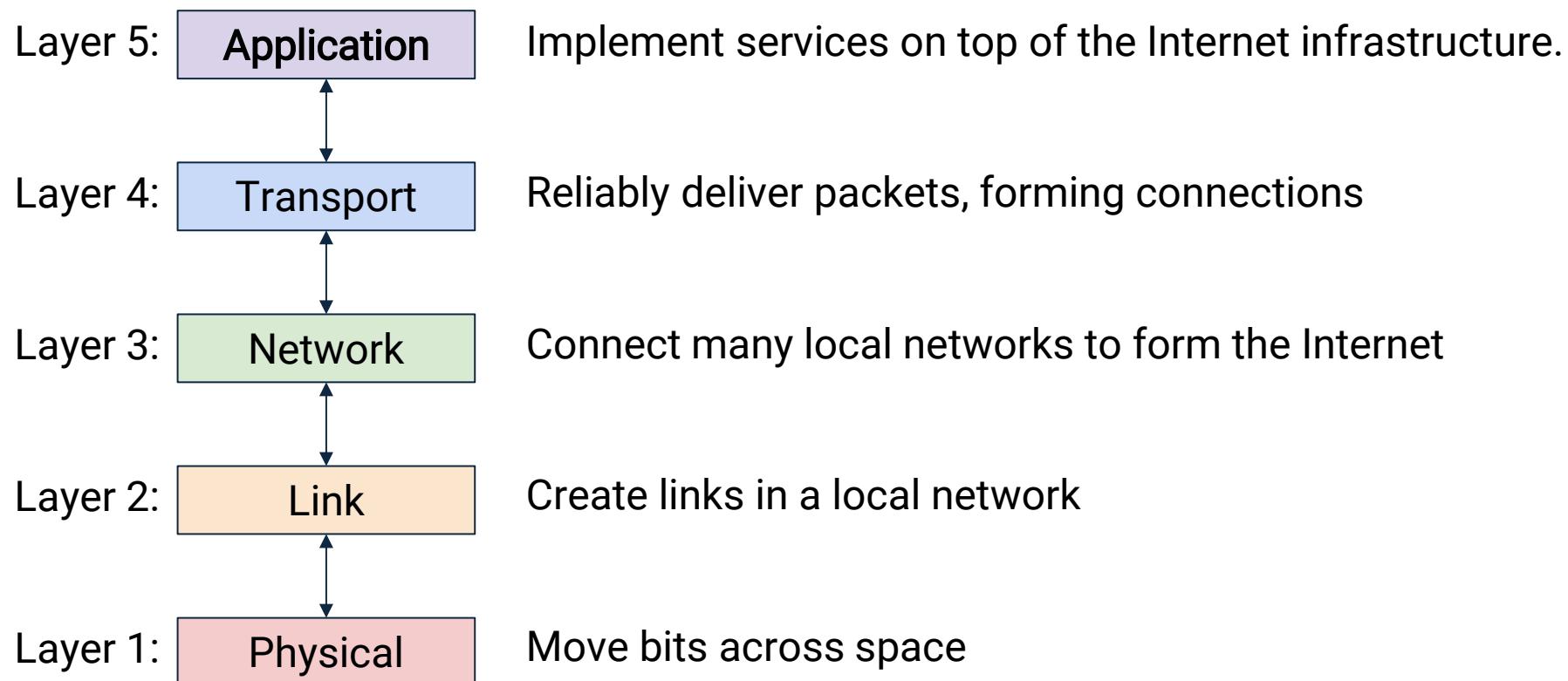
Peers at the same layer communicate with each other using the header at that layer



Application Layer: Next Lecture

Application layer builds services (e.g. websites, video streaming) on top of transport layer

- This design lets us build different services, all on the same infrastructure.



Summary

We've covered a “ton” of material!

- Internet overview
- what's a protocol?
- network edge, access network, core
 - Routing versus packet forwarding
 - Internet structure
- layering, service , encapsulation
- networks under attack

You now have:

- context, overview, vocabulary, “feel” of networking
- more depth, detail, *and fun to follow!*

CSC 3511 Security and Networking

Week 2, Lecture 2: Application Layer and HTTP

Roadmap

- *Principles of network applications*
- *Web and HTTP*

Application Layer: Overview

Our goals:

- Conceptual & implementation aspects of application-layer protocols
- Learn about protocols by examining popular application-layer protocols and infrastructure
 - HTTP
 - SMTP, IMAP
 - DNS
- Build network applications
 - socket API

5. Application

HTTP, FTP, SMTP, DNS
SSL/TLS, JPEG, MPEG
RPC, NetBIOS

4. Transport

TCP, UDP

3. Network

IP, ICMP

2. Data Link

Ethernet, Wi-Fi

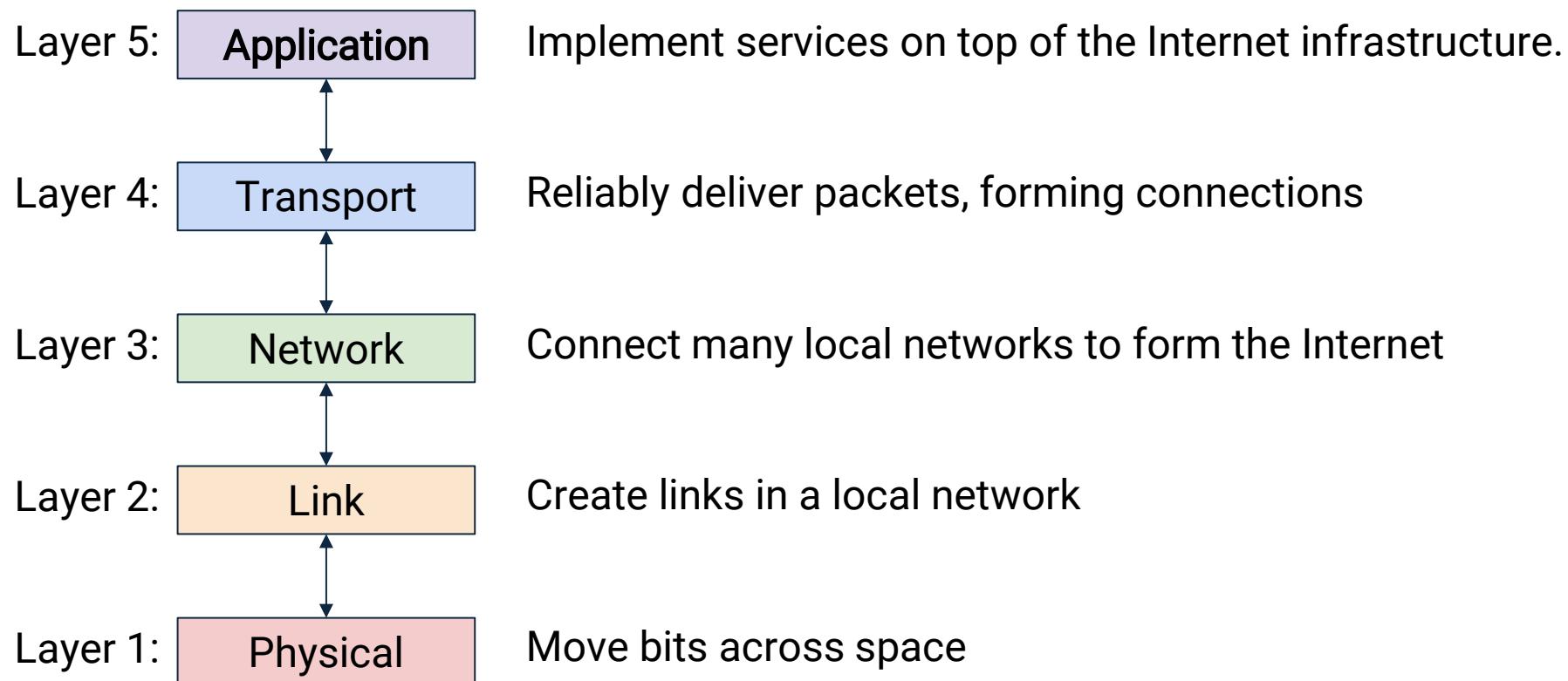
1. Physical

Copper, Fiber, Radio

Application Layer: Overview

Application layer builds services (e.g. websites, video streaming) on top of transport layer

- This design lets us build different services, all on the same infrastructure.



Some Network Apps

- Social networking
- Web
- Text messaging
- E-mail
- Multi-user network games
- Streaming stored video
(YouTube, Hulu, Netflix)
- P2P file sharing
- Voice over IP (e.g., Skype)
- Real-time video conferencing (e.g., Zoom)
- Internet search
- Remote login
- ...

Q: How do apps communicate?

A: Processes & Protocols

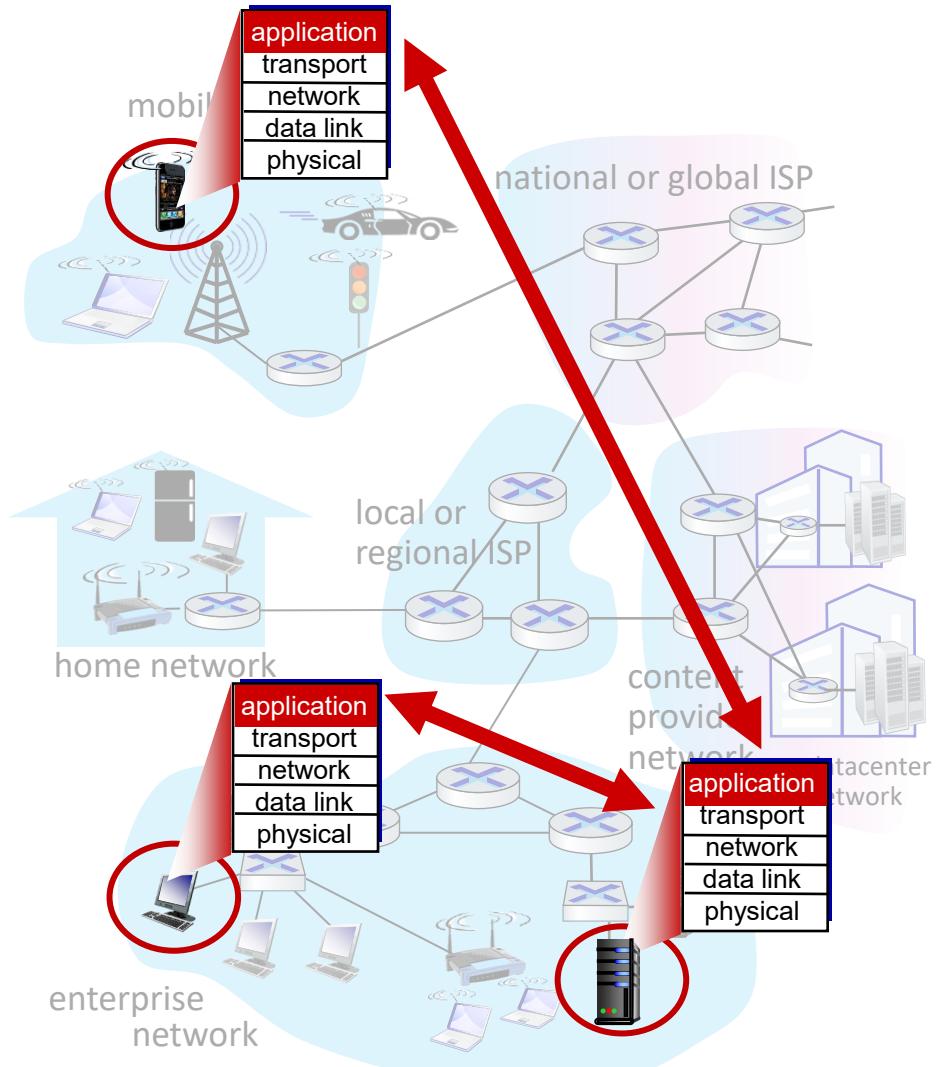
Creating a Network App

Write programs that:

- Run on (different) end systems
- Communicate over network
- E.g., web server software communicates with browser software

No need to write software for network-core devices

- Network-core devices do not run user applications
- Applications on end systems allows for rapid app development, propagation



Processes communicating

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

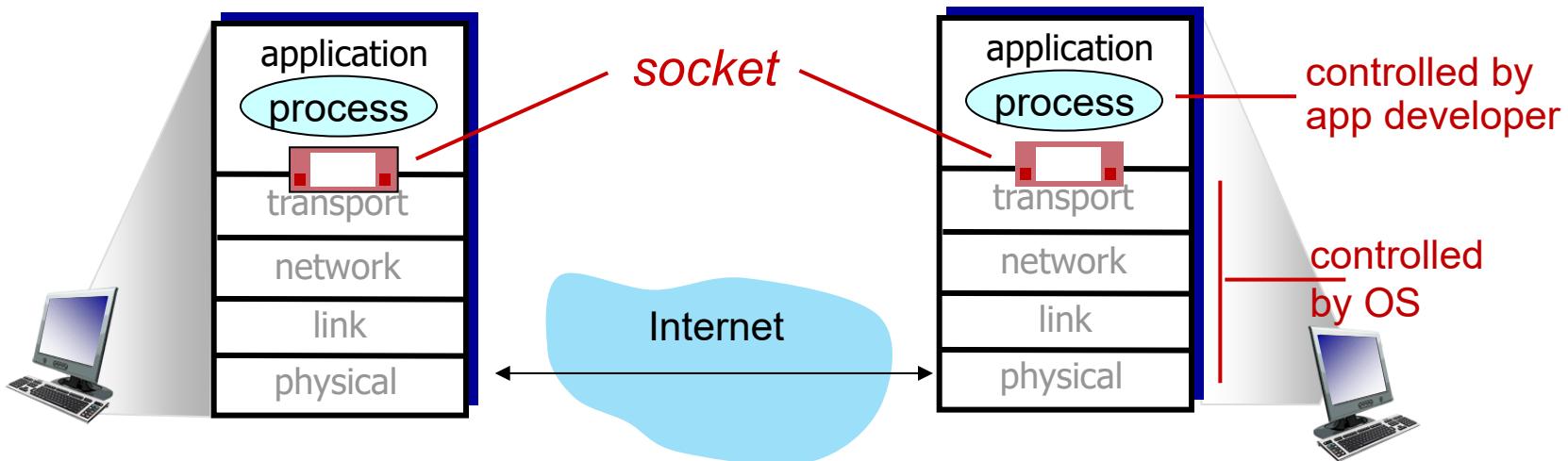
Clients, Servers

Client process: process that initiates communication

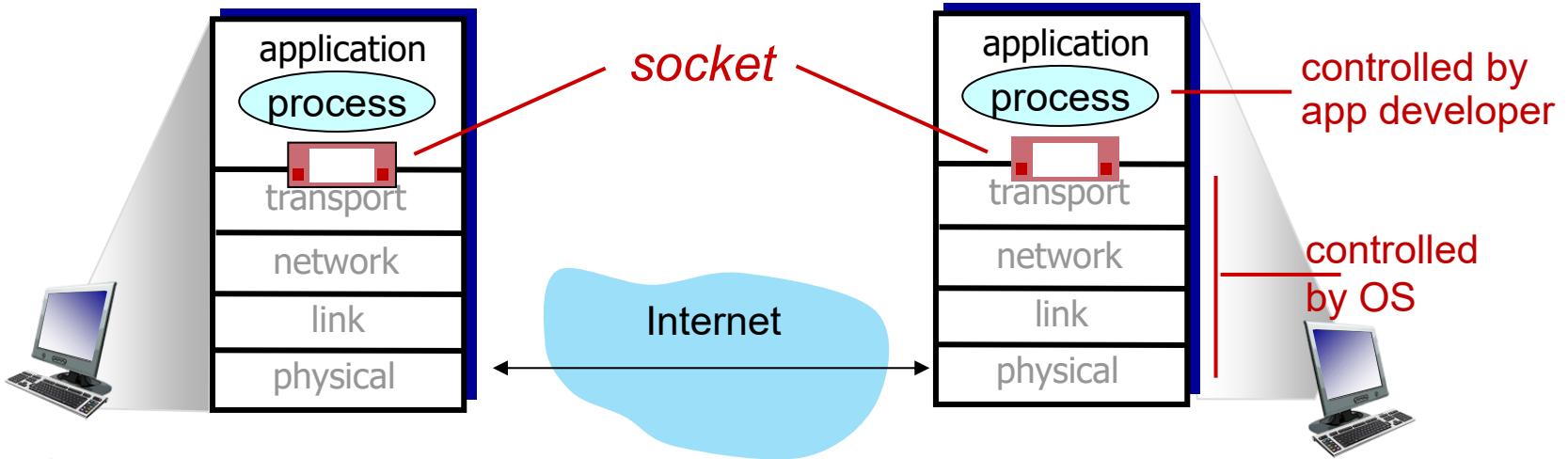
Server process: process that waits to be contacted

Sockets

- Process sends/receives messages to/from its (network) socket
- Socket analogous to door
 - Sending process shoves message out the door
 - Sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
- Two sockets involved: one on each side



Sockets



Client side

```
sock = socket.socket()  
sock.connect(("www.google.com", 80))  
sock.send("GET / HTTP/1.1\r\n...")
```

Server side

```
server = socket.socket()  
server.bind("", 80)  
server.listen()
```

- Let's See Real Sockets
 - Try this command on your Windows PowerShell: \$ netstat -an | findstr :443
 - It displays all network connections specifically related to port 443 (HTTPS)
 - Transport layer protocols: week 3

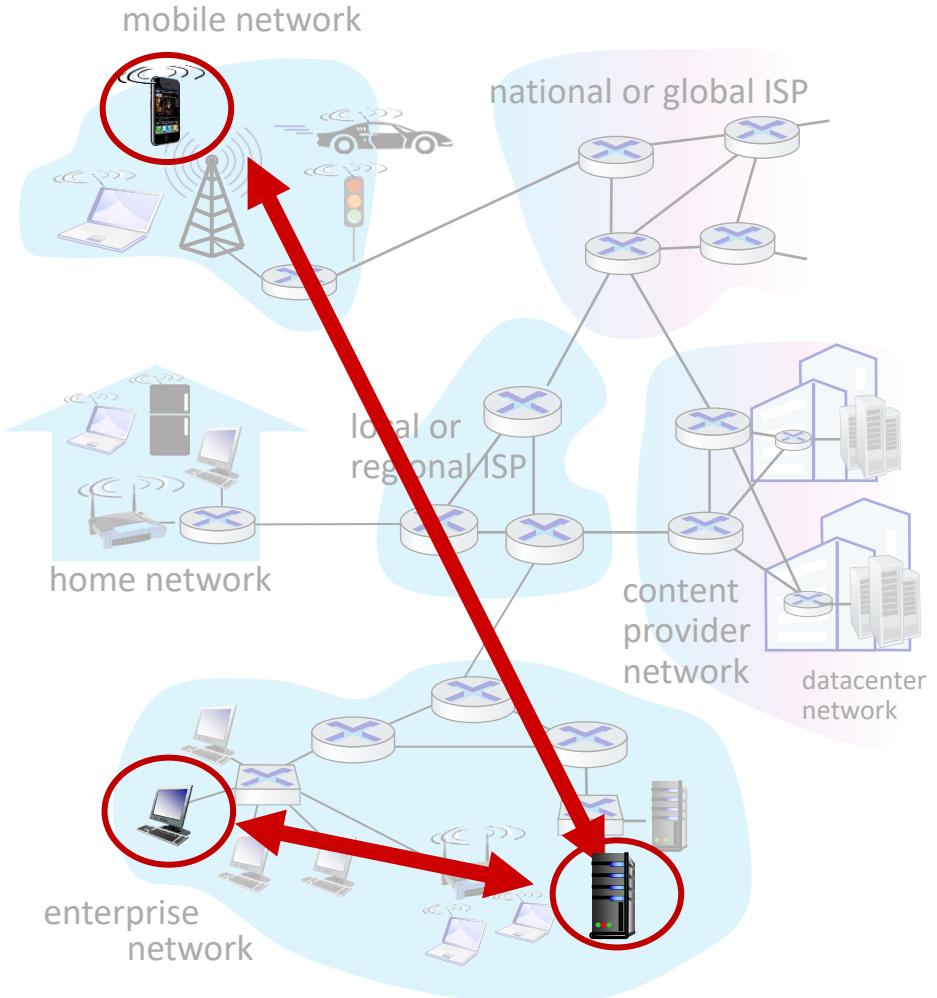
Client-server paradigm

Server:

- *Always-on* host
- Permanent (**static**) IP address
- Often in data centers, for scaling

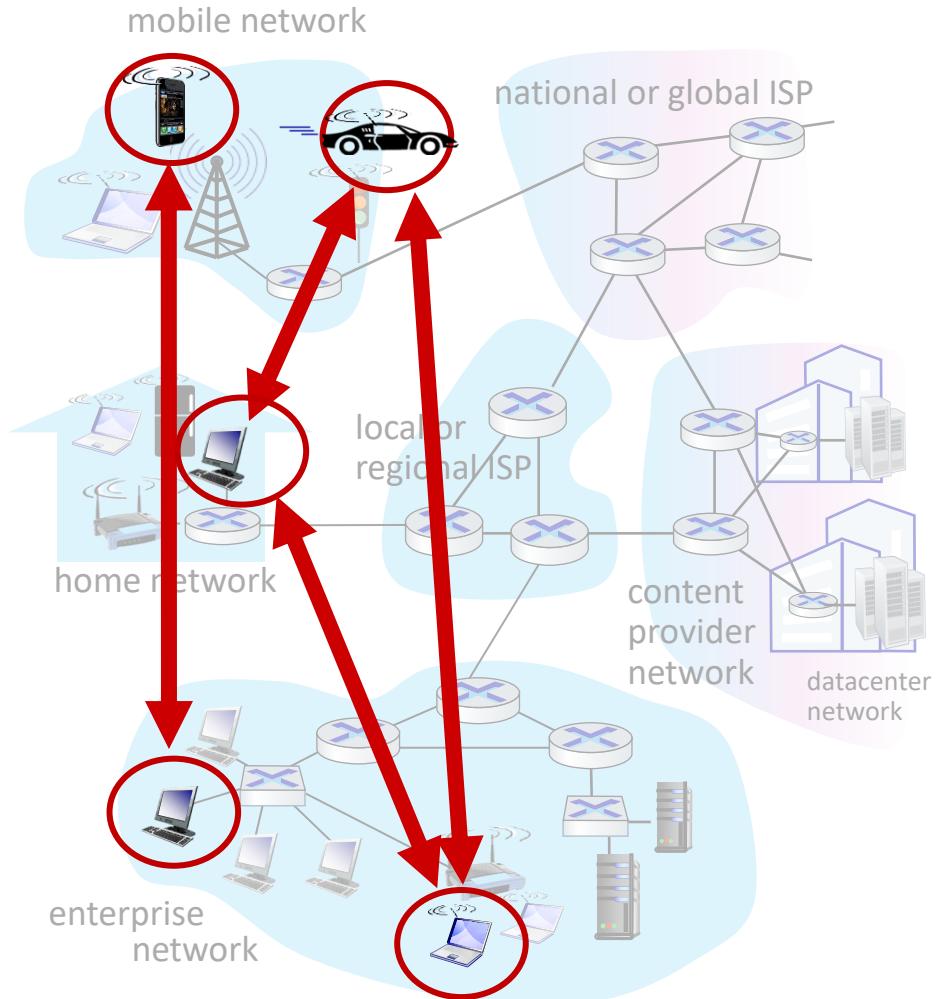
Clients:

- Contact, communicate with **server**
- May be intermittently connected
- May have dynamic IP addresses
- Do *not* communicate directly with each other
- Examples: HTTP, IMAP, FTP



Peer-peer architecture

- 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
- Example: P2P file sharing [BitTorrent]
 - Note: applications with P2P architectures have client processes & server processes



Addressing Processes

- To receive messages, process must have *identifier*
- Host device has unique 32-bit IP address (IPv4)
- *Q:* Does IP address of host on which process runs suffice for identifying the process?
- *A: No*, many processes can be running on same host
- *Identifier* includes both **IP address** and **port numbers** associated with process on host.
- Example port numbers:
 - HTTP server: 80
 - mail server: 25
- To send HTTP message to www.msoe.edu web server:
 - IP address: 69.20.113.171
 - port number: 80
 - Try \$ *nslookup www.msoe.edu*
- More shortly...

An Application-layer Protocol Defines:

- Types of messages exchanged:
 - E.g., request, response
 - Message syntax:
 - What fields in messages & how fields are delineated
 - Message semantics
 - Meaning of information in fields
 - Rules for when and how processes send & respond to messages
- Open protocols:
 - Defined in RFCs, everyone has access to protocol definition
 - Allows for interoperability
 - E.g., HTTP, SMTP
 - Proprietary protocols:
 - E.g., Skype, Zoom

Roadmap

- *Principles of network applications*
- **Web and HTTP**

From Theory to Practice: HTTP

Let's dive into a real-world example - the World Wide Web and HTTP

- Web page consists of *objects*, each of which can be stored on different Web servers
- Object can be HTML file, JPEG image, Java applet, audio file,...
- Web page consists of *base HTML-file* which includes *several referenced objects, each* addressable by a *URL*, e.g.,

www.msoe.edu/computerScience/rosie.jpg

host name

path name

- *Q: What protocol standardizes how browsers request the HTML file and referenced objects from different servers?*
- *A: HTTP*



Apply to MSOE!

Ready to learn more about Milwaukee School of Engineering and our computer science degree? Filling out the application is fast and easy!

[Let's get started >](#)

Base HTML file:

<https://www.msoe.edu/academics/undergraduate-degrees/engineering/computer-science/>



https://msoe.s3.amazonaws.com/files/callouts/square_xsml_android-app-class-thumbnail-2.jpg

Brief History of HTTP

- Development initiated by Tim Berners-Lee at European Organization for Nuclear Research (CERN) in 1989
 - 1991: Initial specification, HTTP/0.9, drafted
 - 1996: Standardized as HTTP/1.0
 - 1997: Updated to **HTTP/1.1** (persistent HTTP)
 - 2015: HTTP/2.0 was introduced
 - 2022: HTTP/3.0 was introduced
- Driven by a need to share information between scientists
- Needed a mechanism to transfer hypertext pages, with links to other pages
- Resulting protocol: HyperText Transfer Protocol (HTTP)
- You can still view [the first website ever made](#)

HTTP Basics

HTTP is a **client-server** protocol

- One user is the client (e.g. your web browser, your PowerShell terminal)
- One user is the server (e.g. the Google data center, Rosie)

HTTP is a **request-response** protocol

- Client sends one request, and receives exactly one response

HTTP is “**stateless**”

- Server maintains no information about past client requests

HTTP runs over TCP

- Client and server run a TCP handshake and send data over the bytestream
- No need to worry about packets being reordered, dropped, etc.
- Server listens for HTTP on well-known port 80 (*A later secure version uses 443*)



HTTP Requests

The request syntax is in human-readable plaintext (can be typed by a human).

Version: What HTTP version we're using

URL (Uniform Resource Locator): The *resource* we want to interact with

- Intuition: The filepath of a file on some remote server

Method: What we want to do with that resource.

- **GET:** Send me this resource. Originally, this was the only method
- **POST:** Send data to the server (e.g. user submits a form)
- Other methods for *manipulating* content on the server, not just retrieving it:
 - PUT, CONNECT, DELETE, OPTIONS, PATCH, TRACE, etc.

Method	URL	version
GET	/projects/project1.html	HTTP/1.1

\r\n
carriage return character
line-feed character

ends with
a newline

HTTP Responses

Version: What HTTP version we're using.

Status code: A number, telling us what happened with the request

Description: A description of the status code

Content: The resource the user requested!

HTTP/1.1	200	OK	<html>Project 1 Spec...</html>
version	status code	description	content

Status codes are classified into various categories, according to numeric value:

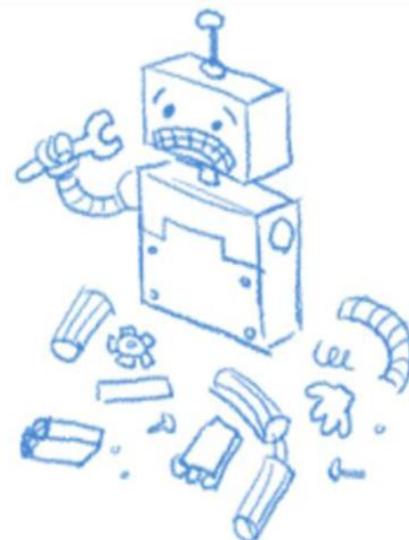
- 100s: Informational responses
- 200s: Successful responses
- 300s: Redirection messages
- 400s: Client error
- 500s: Server error



404. That's an error.

The requested URL /doesnotexist was not found on this server. That's all we know.

Try on your browser: <http://neverssl.com/>



Speeding Up HTTP

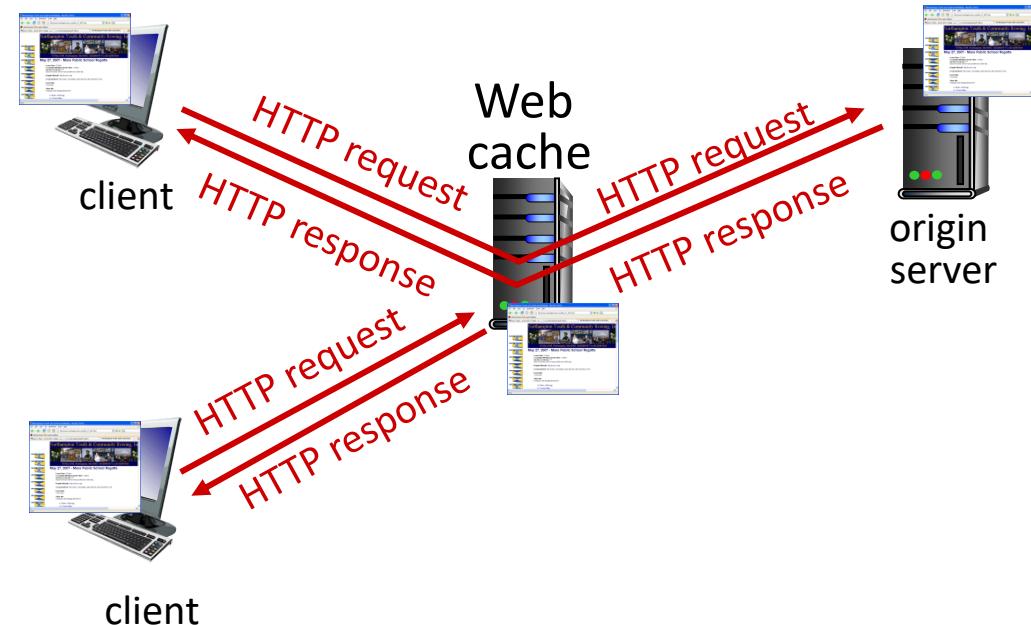
- Loading a single website can require multiple HTTP requests
 - One request for the HTML (text/formatting) of the page
 - Separate requests for every picture
 - Separate requests for scripts to make the page interactive
- Naive approach: Separate *TCP* connection with remote server for each request
 - Server is far away → delay/RTT is high
 - We have to do a 3-way handshake for every request → Time and resource consuming



Web caches

Goal: satisfy client requests without involving origin server

- User configures browser to point to a (local) *Web cache*
- Browser sends all HTTP requests to cache
 - *If* object in cache: cache returns object to client
 - *Else* cache requests object from origin server, caches received object, then returns object to client

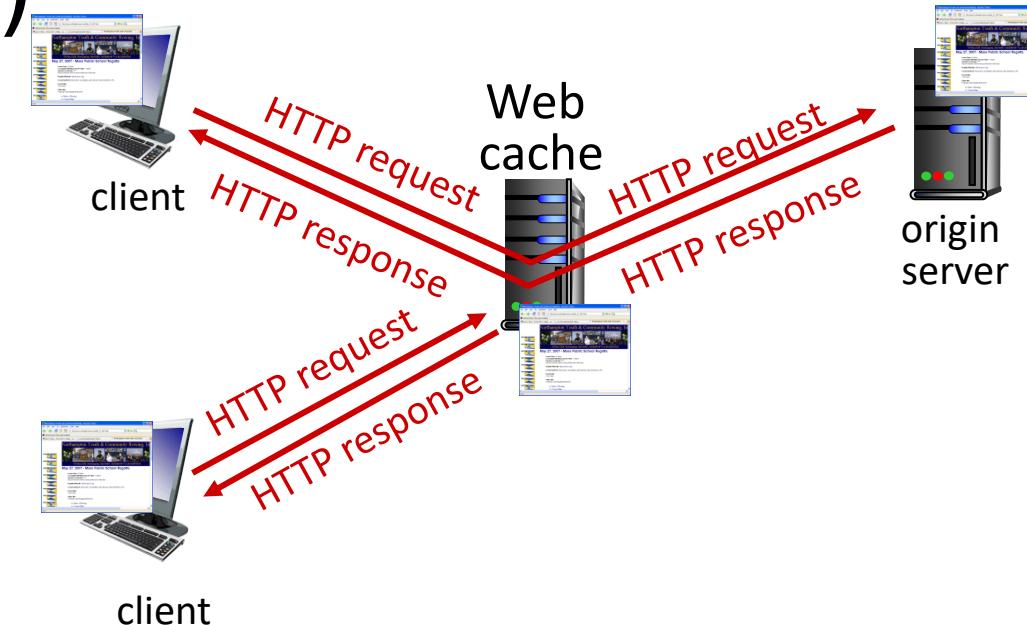


Web caches (aka proxy servers)

- Web cache acts as both client and server
 - Server for original requesting client
 - Client to origin server
- Server tells cache about object's allowable caching in response header:

```
Cache-Control: max-age=<seconds>
```

```
Cache-Control: no-cache
```



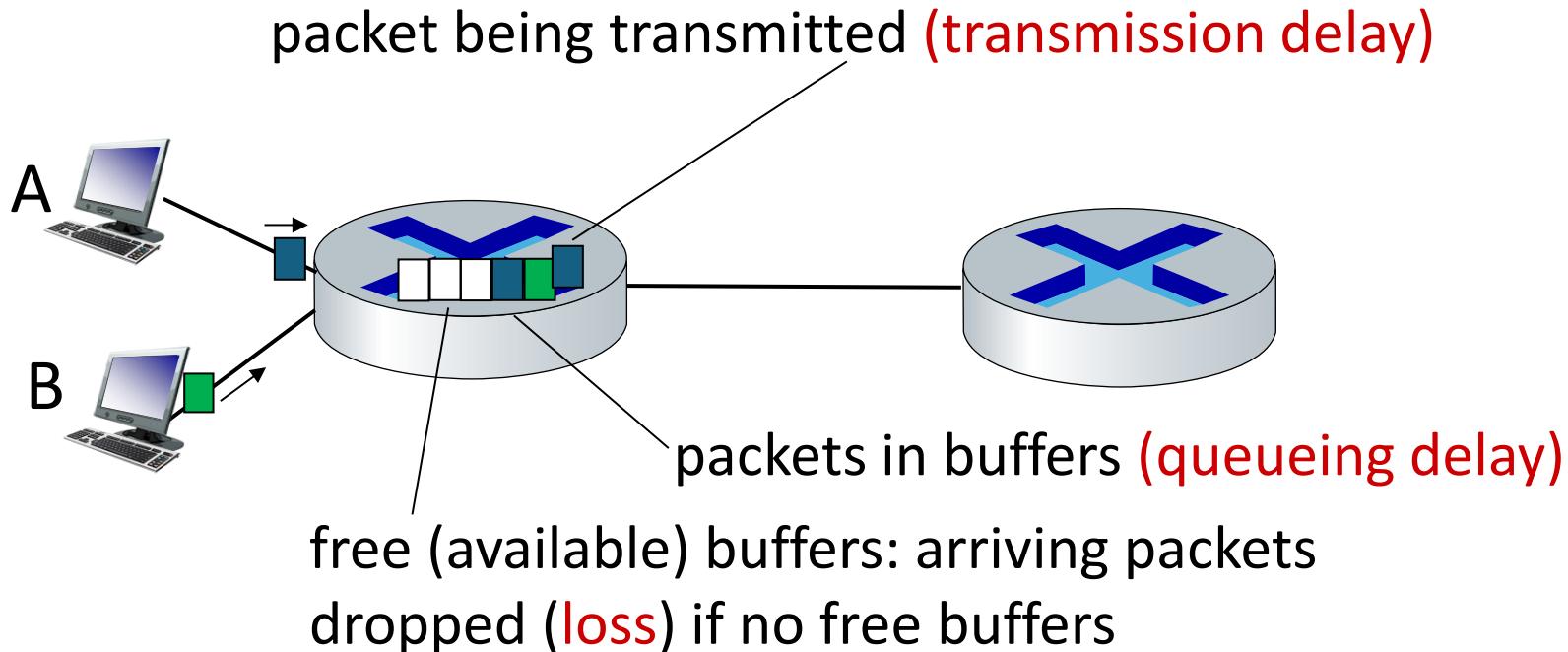
Why Web caching?

- **Reduce response time** for client request (why?)
 - Cache is closer to client
- **Reduce traffic** on an institution's access link

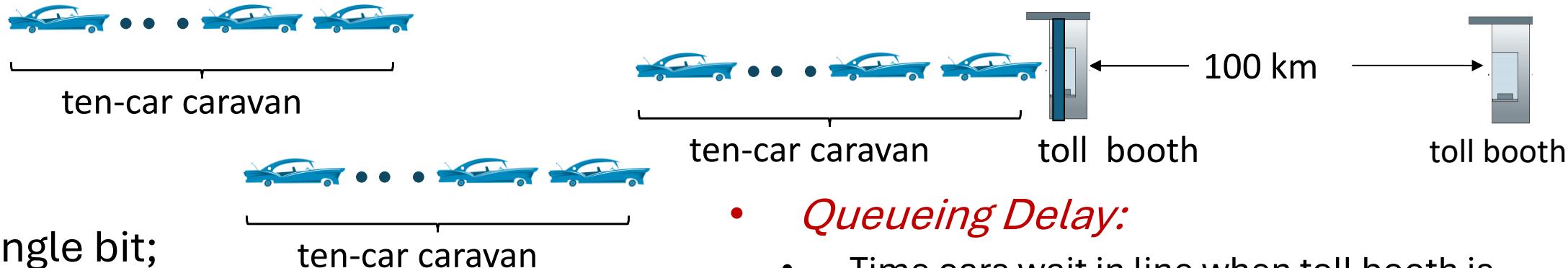
Network Performance: Delay, Loss, Throughput

How do packet delay and loss occur?

- Packets queue in router buffers, waiting for turn for transmission
- Queue length grows when arrival rate to link (temporarily) *exceeds* output link capacity
- **Packet loss** occurs when memory to hold queued packets fills up



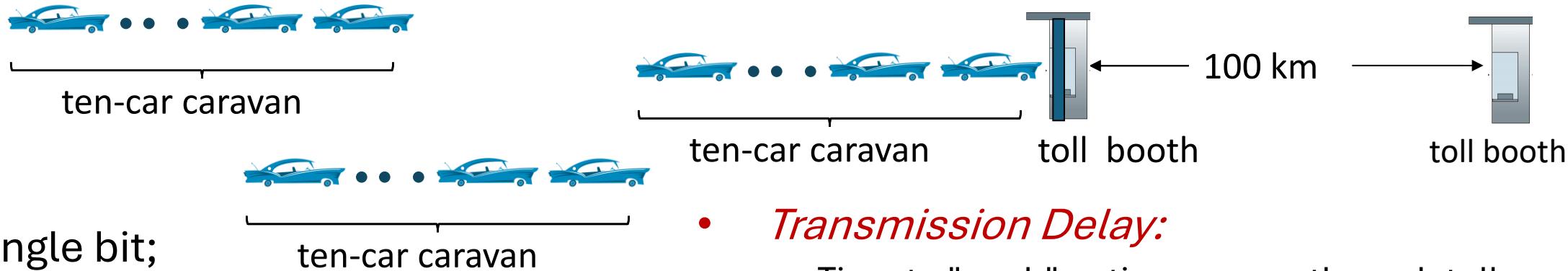
Network Performance: Caravan analogy



- A car \approx single bit;
- A 10-car caravan \approx a packet burst;
- Multiple caravans \approx packets from different origins using same router
- Toll service \approx Router
- Toll booth operator takes 1 sec to check each car's documents (processing delay)
- Toll booth takes 12 sec to service each car (transmission rate)
- Cars travel at 100 km/hr between toll booths (propagation speed)

- ***Queueing Delay:***
 - Time cars wait in line when toll booth is busy with other traffic
 - E.g., our caravan arrives, but 5 cars from another route are already waiting. The queueing time is: $5 \times 12 = 60$ sec for the first car of our caravan
 - **In network:** Packets waiting in router buffers when link is busy
- ***Processing Delay:***
 - Time for toll booth operator to check each car's documents
 - **In network:** Router examining packet headers, making forwarding decisions

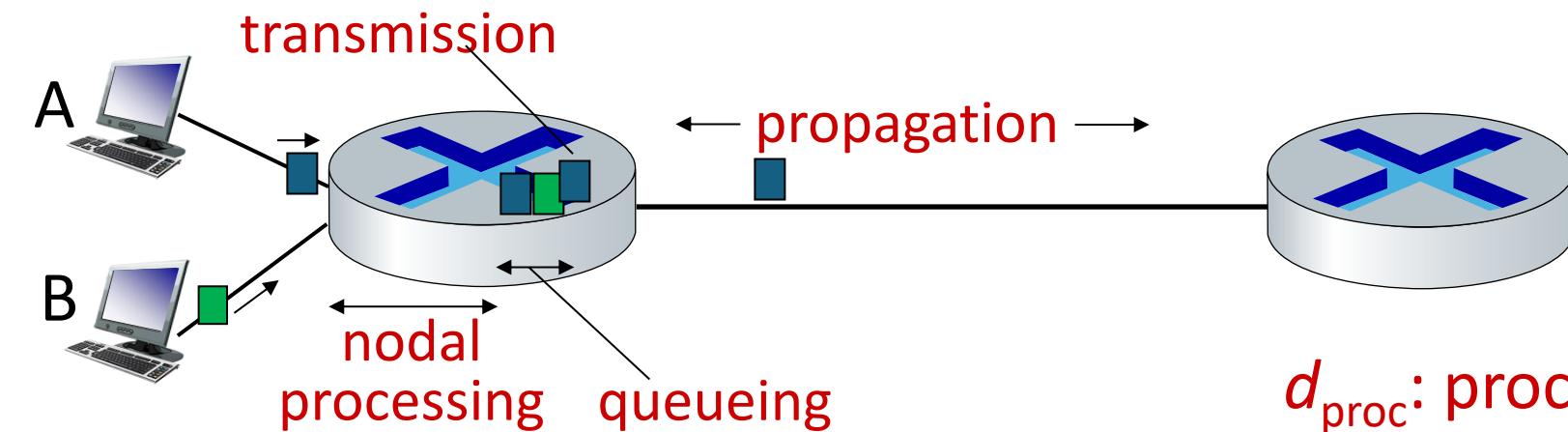
Network Performance: Caravan analogy



- A car \approx single bit;
- A 10-car caravan \approx a packet burst;
- Multiple caravans \approx packets from different origins using same router
- Toll service \approx Router
- Toll booth operator takes 1 sec to check each car's documents (processing delay)
- Toll booth takes 12 sec to service each car (transmission rate)
- Cars travel at 100 km/hr between toll booths (propagation speed)

- ***Transmission Delay:***
 - Time to "push" entire caravan through toll booth. E.g., $10 \text{ cars} \times 12 \text{ sec} = 120 \text{ seconds}$
 - **In network:** Time to push all packet bits onto the link (*packet length/ link transmission rate*)
- ***Propagation Delay:***
 - Time for a car to travel from 1st to 2nd toll booth
 - E.g., $100 \text{ km} \div 100 \text{ km/hr} = 60 \text{ minutes}$
 - **In network:** Time for signal to travel through physical medium (*length of physical link/propagation speed*)

Network Performance: Delay



$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

d_{trans} : transmission delay:

- L : packet length (bits)
- R : link transmission rate (bps)

$$\bullet \quad d_{\text{trans}} = L/R$$

d_{trans} and d_{prop}
very different

d_{queue} : queueing delay

- Time waiting at output link for transmission
- Depends on congestion level of router

d_{proc} : processing delay

- Nodal processing: check bit errors, determine output link, etc.
- Typically, < microsecs

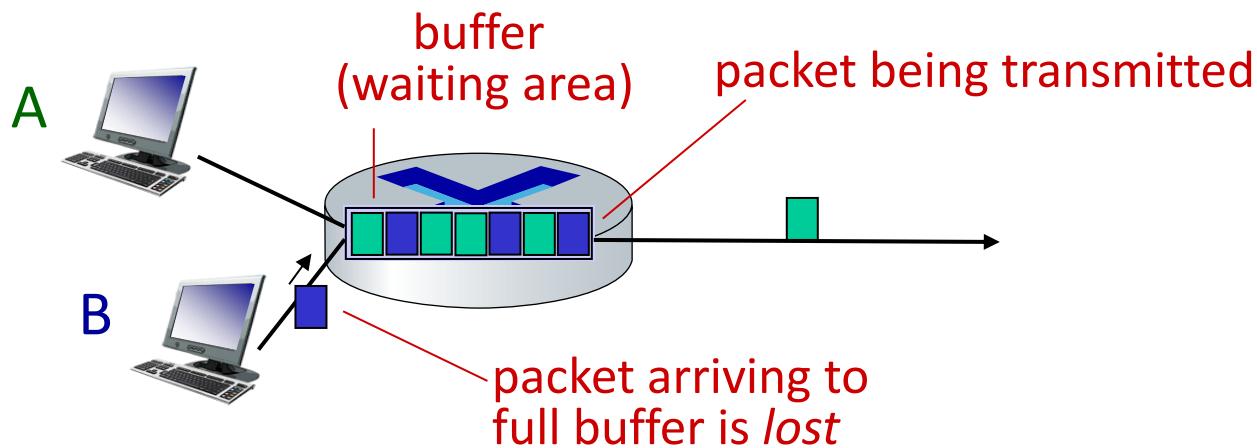
d_{prop} : propagation delay:

- d : length of physical link
- s : propagation speed ($\sim 2 \times 10^8$ m/sec)

$$\bullet \quad d_{\text{prop}} = d/s$$

Network Performance: Packet Loss

- Router buffers have limited capacity (finite queue space)
- Buffer overflow: When packets arrive faster than they can be transmitted)
- **Drop policy:** New packets dropped when buffer is full → *Packet Loss*

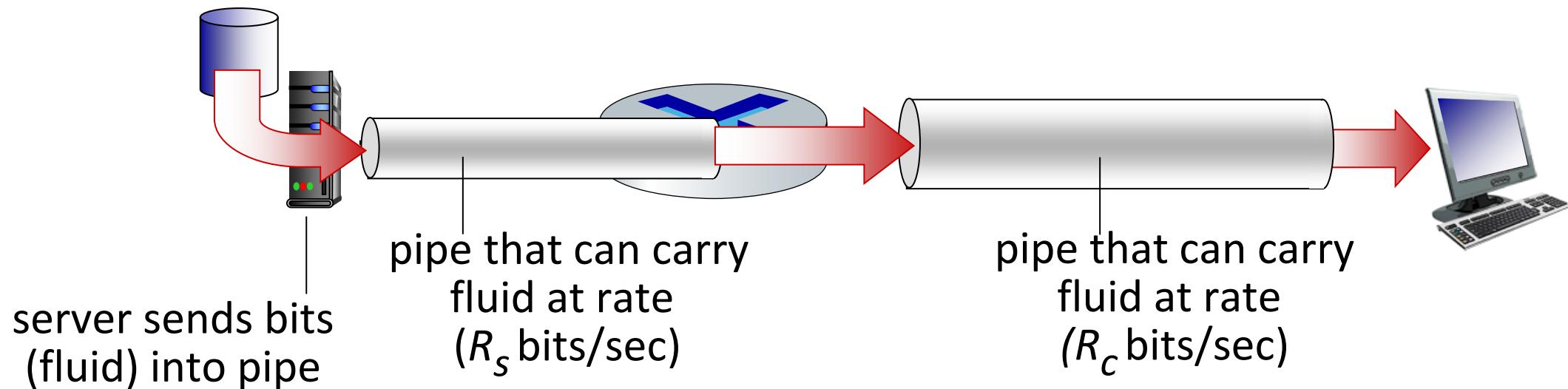


What Happens to Lost Packets?

- Depends on protocol: TCP retransmits, UDP does not (we will see later on)
- Retransmission location: Previous router, or source end system
- Application impact: Delay (TCP) or missing data (UDP)
- Real-world impact: Web browsing slows down, file downloads pause, video stutters, gaming lag spikes

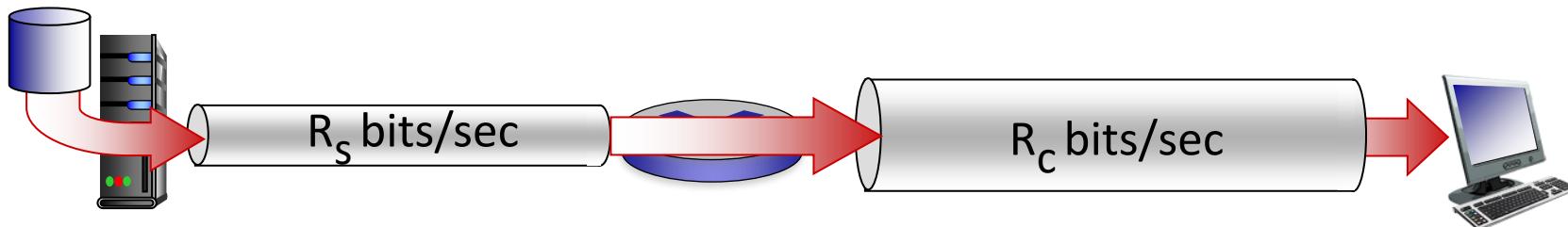
Network Performance: Throughput

- *Throughput*: rate (bits/time unit) at which bits are being sent from sender to receiver
 - *Instantaneous*: rate at given point in time
 - *Average*: rate over longer period of time

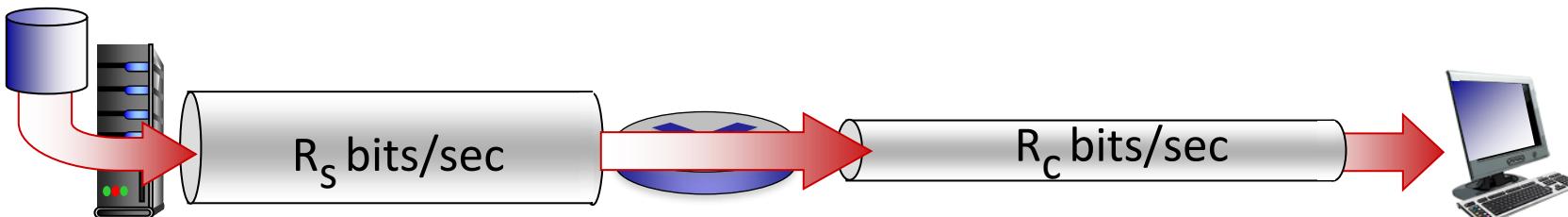


Network Performance: Throughput

$R_s < R_c$ What is average end-end throughput?



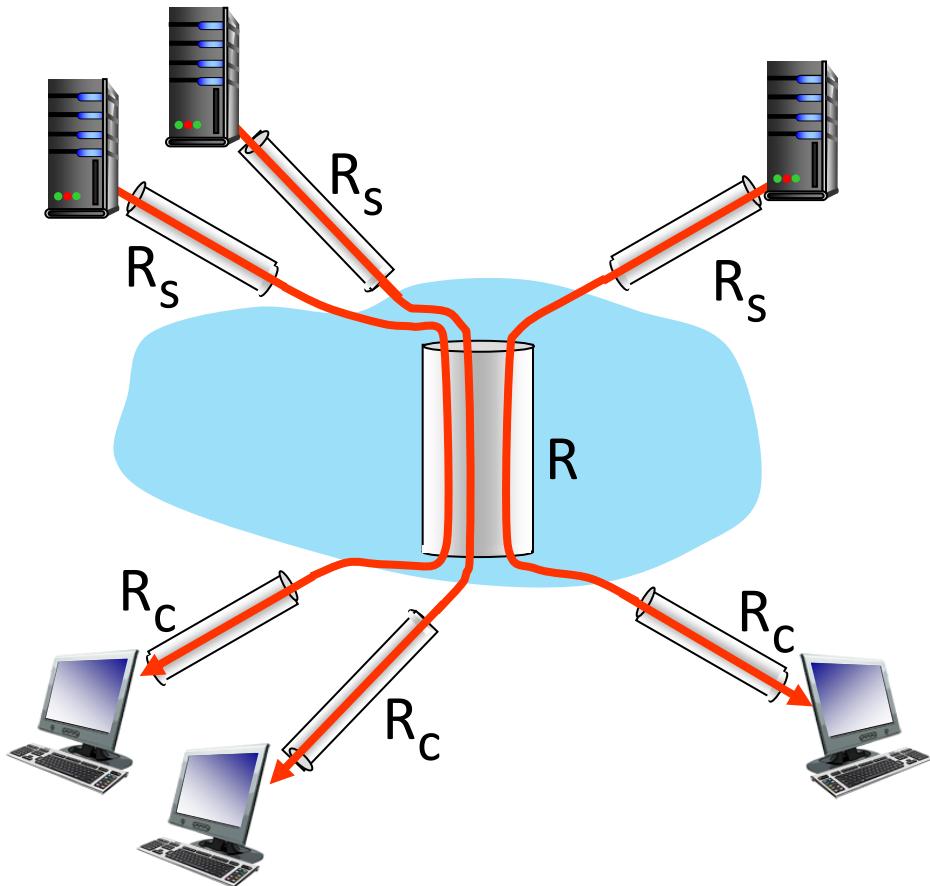
$R_s > R_c$ What is average end-end throughput?



— *bottleneck link* —

link on end-end path that constrains end-end throughput

Throughput: network scenario



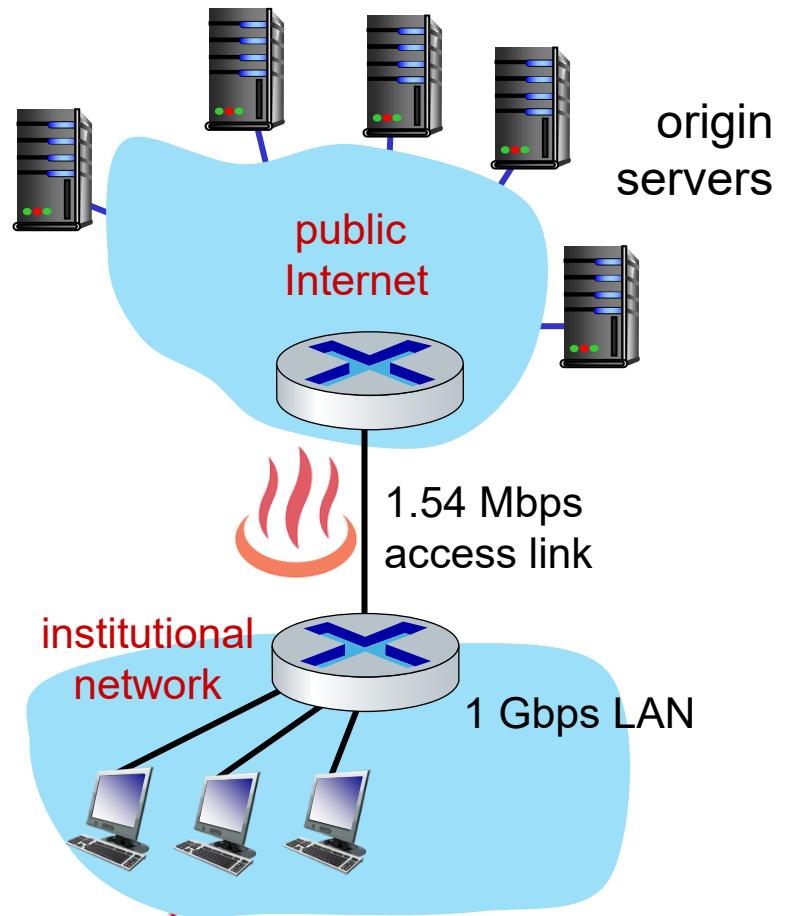
10 connections (fairly) share
backbone bottleneck link R bits/sec

- Per-connection end-end throughput: $\min(R_c, R_s, R/10)$
- In practice: R_c or R_s is often bottleneck

Caching example

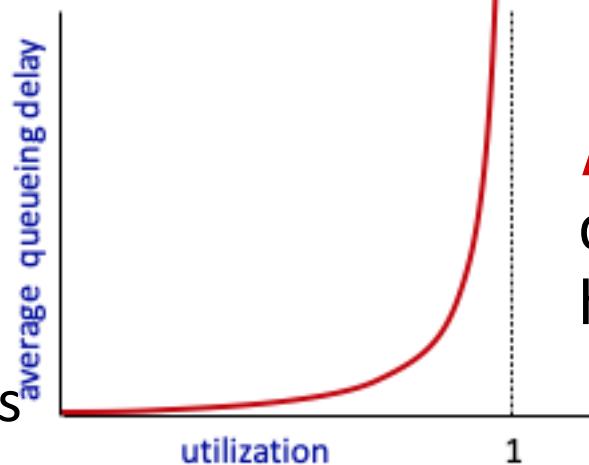
Scenario:

- Access link rate: 1.54 Mbps (Megabits per second)
- RTT from institutional router to server: 2 sec
- Round-trip time (RTT): time for a small packet to travel from client to server and back
- Web object size: 100K bits
- Average request rate from browsers to origin servers: 15/sec
 - Avg data rate to browsers: 1.50 Mbps



Performance:

- Access link utilization = $1.5/1.54 = .97$
- LAN utilization: 0.0015
- End-end delay = RTT +
access link delay + LAN delay
= 2 sec + minutes + microseconds



Problem: large queueing delays at high utilization!

Option 1: buy a faster access link

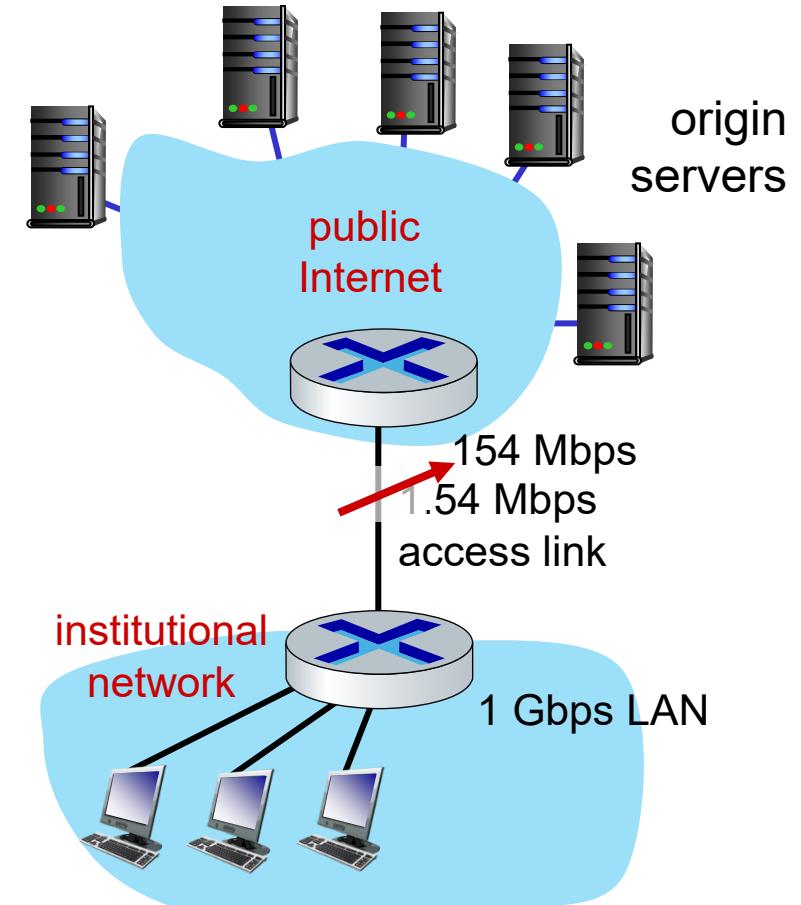
Scenario:

- access link rate: ~~1.54~~ Mbps
- RTT from institutional router to server: 2 sec
- web object size: 100K bits
- average request rate from browsers to origin servers: 15/sec
 - avg data rate to browsers: 1.50 Mbps

Performance:

- access link utilization = ~~.97~~ → .0097
- LAN utilization: .0015
- end-end delay = Internet delay +
access link delay + LAN delay
 $= 2 \text{ sec} + \cancel{\text{minutes}} + \cancel{\text{microseconds}}$

Cost: faster access link (**expensive!**) → msecs



Option 2: install a web cache

Solution:

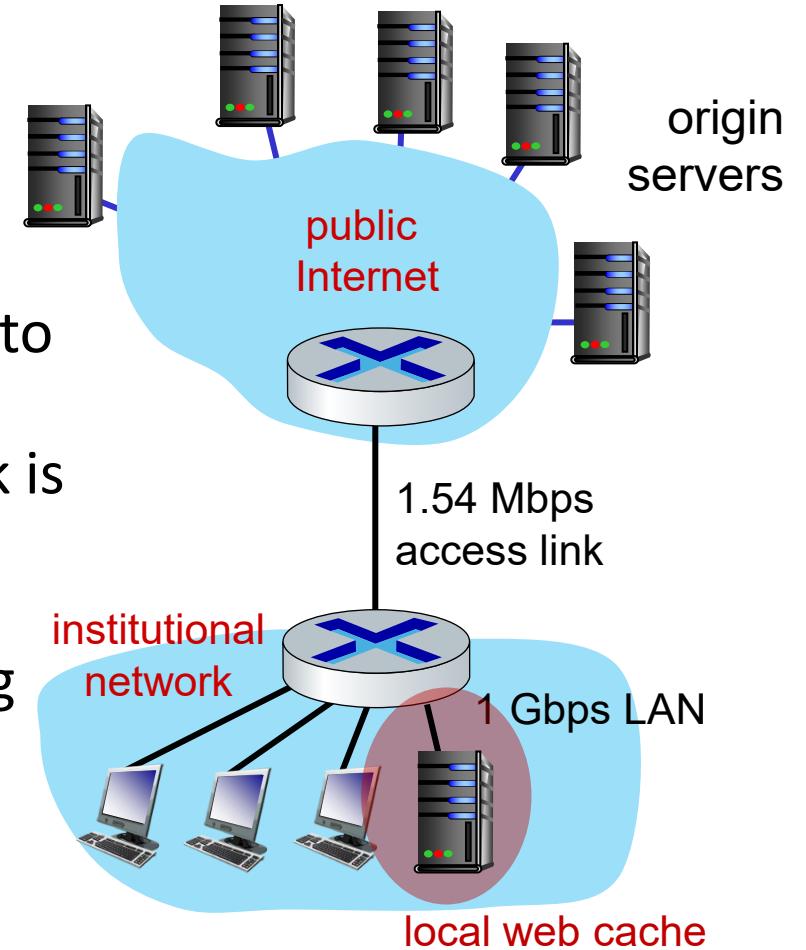
- **Installed a Local Web Cache:** Added a local web cache within the institutional network to store frequently accessed web content closer to the users.

Why it works:

- Store frequently accessed data locally, reducing the need to fetch it repeatedly from a remote server
 - Reduce Load on Access Link: The institutional network is a bottleneck when multiple users request data simultaneously.
 - Improve Data Access Speed & Lower Latency: Fetching data from the local web cache is much faster

Benefits:

- Reduce Internet Latency
- Faster Response Time & Improved Performance



Maintaining user/server state: cookies

Web sites and client browser use ***cookies*** to maintain some state between transactions

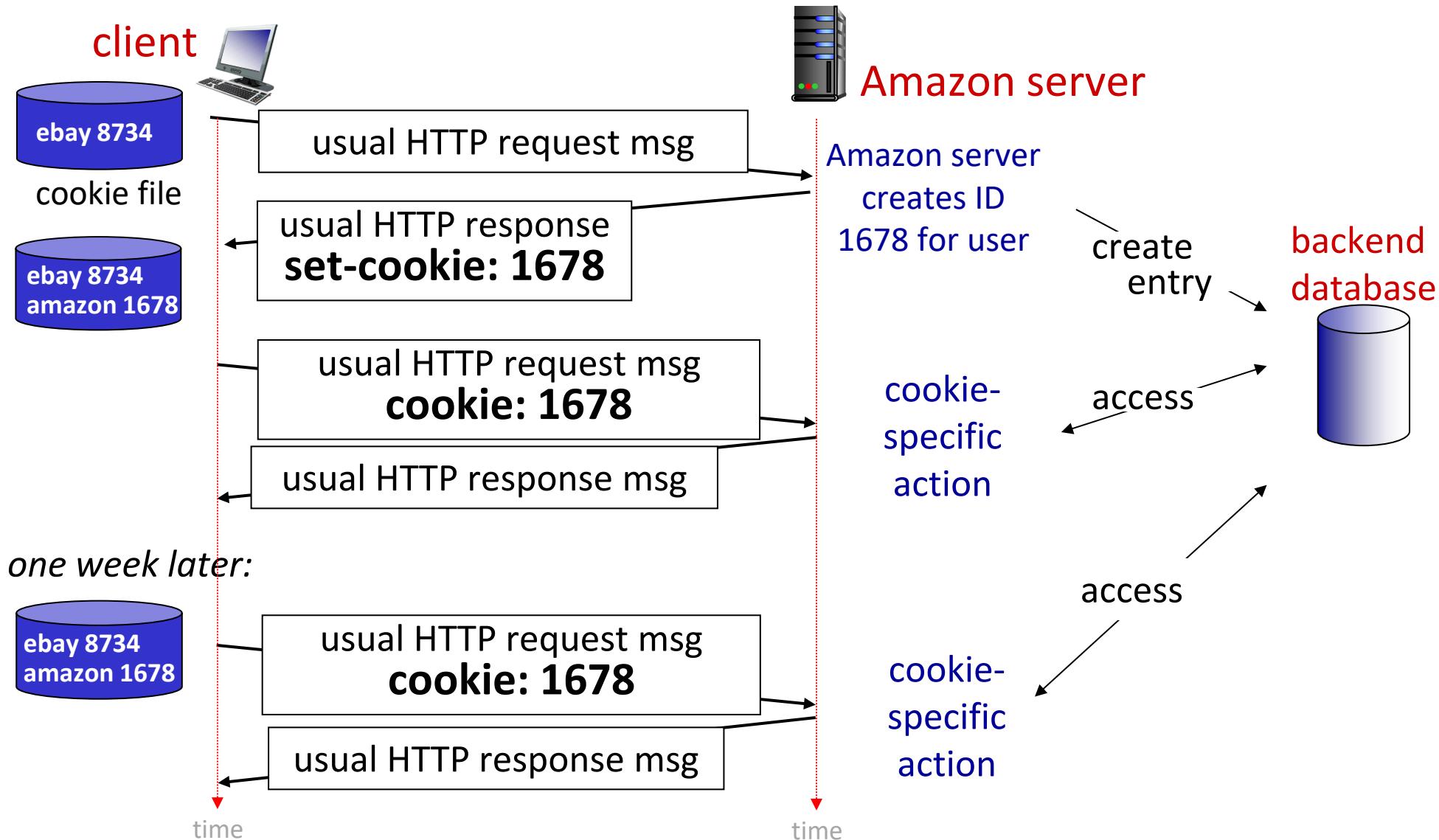
four components:

- 1) cookie header line of HTTP ***response*** message
- 2) cookie header line in ***next HTTP request*** message
- 3) cookie file kept on user's host (client), managed by user's browser
- 4) cookie also kept on back-end database at Website

Example:

- Susan uses browser on laptop, visits specific e-commerce site for first time
- when initial HTTP request arrives at site, site creates:
 - Unique ID (aka "cookie")
 - Entry in backend database for ID
- Subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

Maintaining user/server state: cookies



HTTP Cookies: Tracking a User's Browsing Behavior

What cookies can be used for:

- User session state (Web, e-mail)
- Authorization
- Shopping carts
- Recommendations

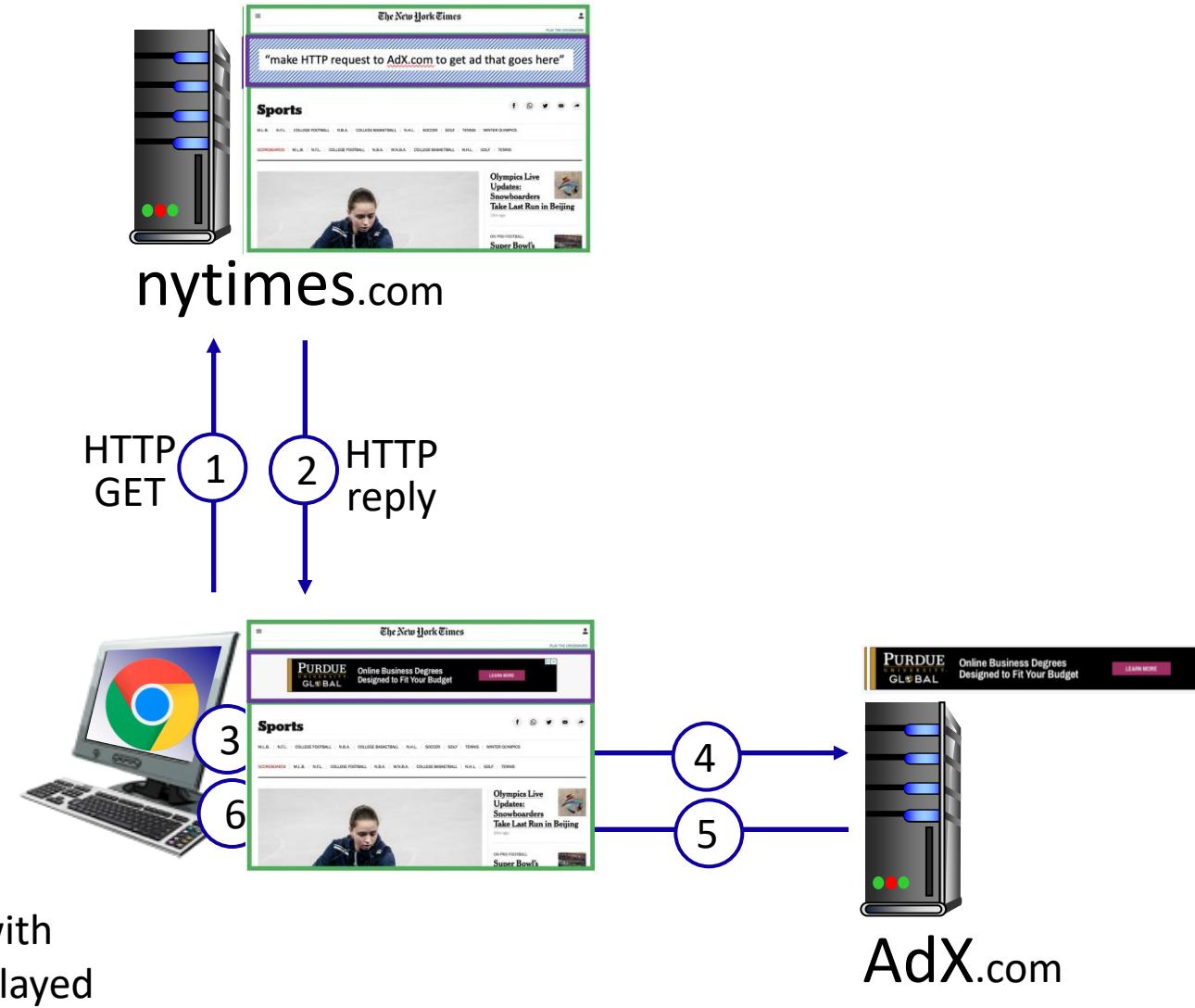
aside

cookies and privacy:

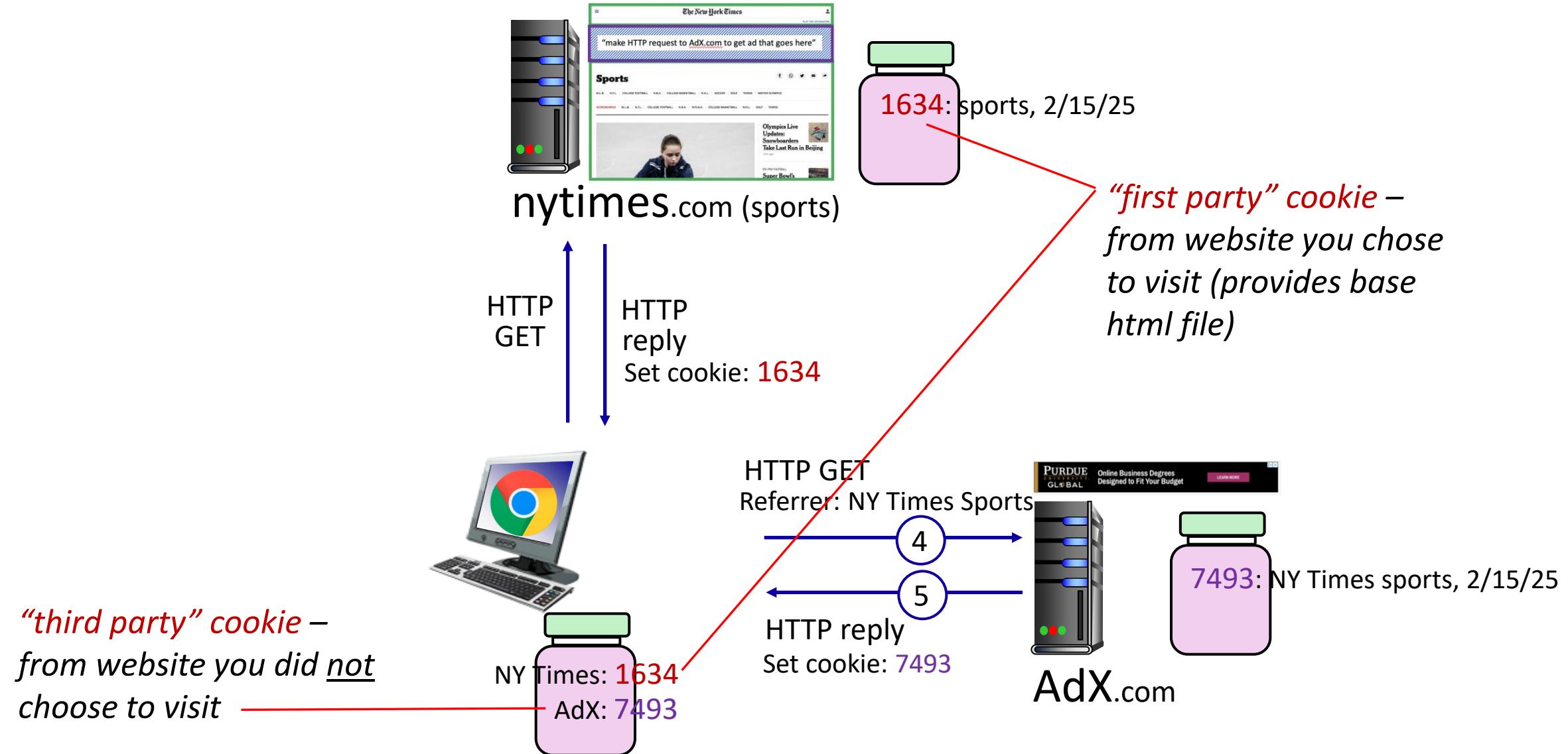
- Cookies permit sites to *learn* a lot about you on their site
- Third party persistent cookies (tracking cookies) allow common identity (cookie value) to be *tracked across multiple web sites*
- *Third party tracking via cookies:*
 - *Disabled by default in Firefox, Safari browsers*
 - *Disabled in Chrome browser since 2023*

Example: displaying a NY Times web page

- 1 GET base html file from nytimes.com
- 2
- 4 fetch ad from AdX.com
- 5
- 7 display composed page

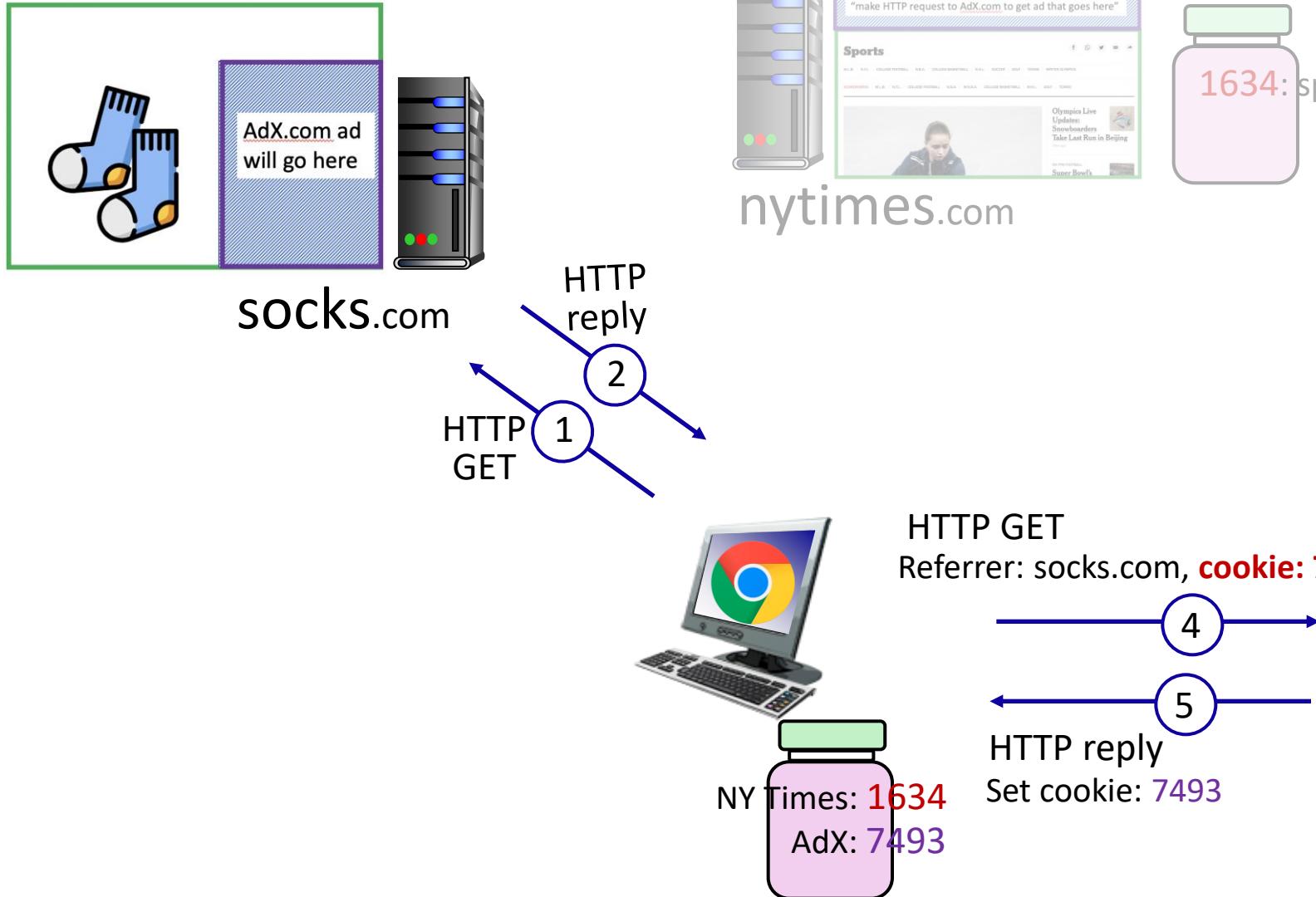


Cookies: tracking a user's browsing behavior



Cookies: tracking a user's browsing behavior

On 2/16/25, I visit socks.com



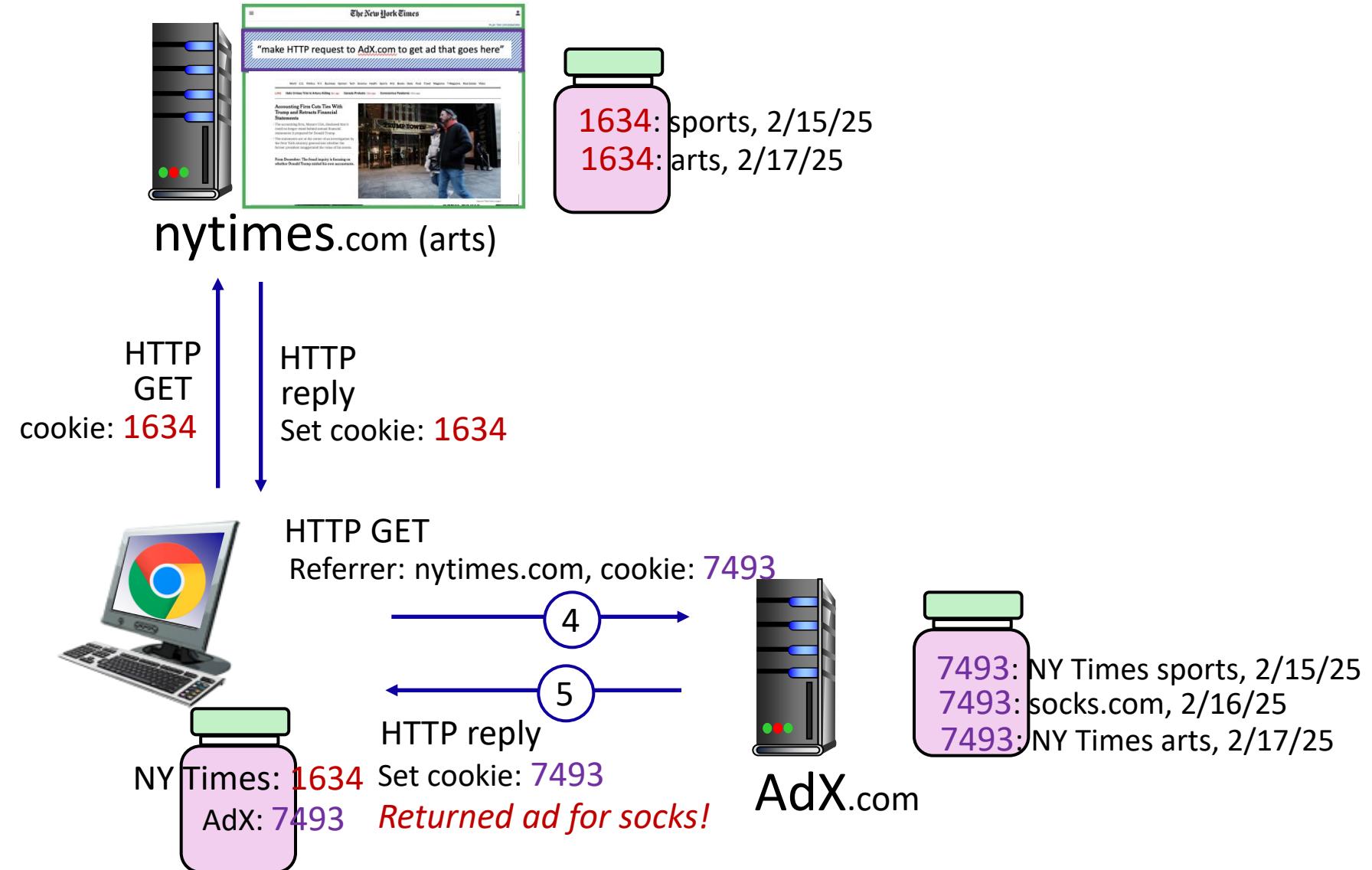
AdX:

- *tracks my web browsing* over sites with AdX ads
- can return targeted ads based on browsing history

7493: NY Times sports, 2/15/25
7493: socks.com, 2/16/25

AdX.com

Cookies: tracking a user's browsing behavior (one day later)



CSC 3511 Security and Networking

Week 3, Lecture 1: Domain Name System

Roadmap

- ***What is DNS for?***
- *Design and implementation*

DNS: Domain Name System

People: many identifiers:

- SSN, name, passport #

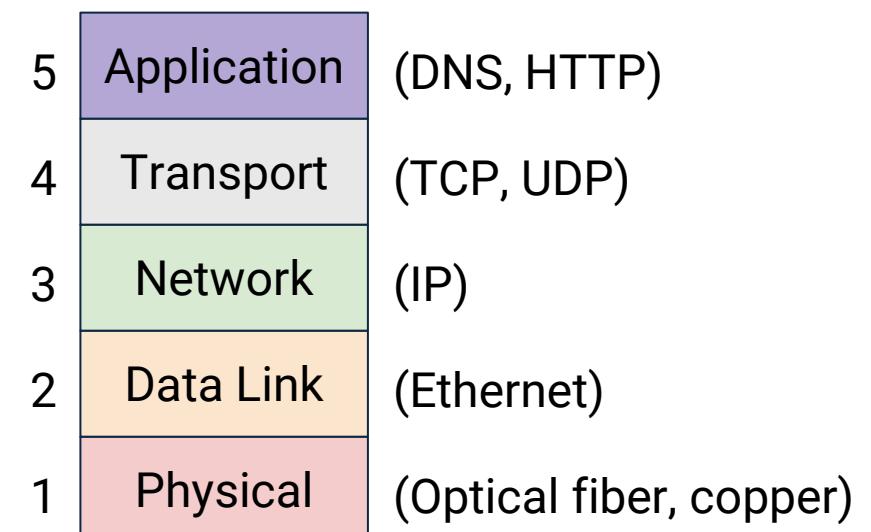
Internet hosts, routers:

- Human-readable names (host name), e.g., www.msoe.edu - contains no relevant routing information
- IP address (8.8.8.8) - used for machines to route datagrams but **Hard to remember**

Q: how to map between name and IP address, and vice versa ?

Domain Name System (DNS):

- *Application-layer protocol:* hosts, DNS servers communicate to *resolve* names (address/name translation)
- A distributed database implemented in hierarchy of many name servers



DNS: Definition

DNS (Domain Name System): An application layer protocol for translating human-readable domain names to IP addresses

Usage:

- You want to send a packet to a certain domain (e.g. you type a `www.google.com` into your browser)
- Your computer performs a **DNS lookup** to translate the domain name to an IP address
- Your computer sends the packet to the corresponding IP address



Goals of DNS

DNS must be **scalable**

- Many hosts/names; Many lookups; Many updates

DNS must be **highly available**

- No single point of failure → replication?

DNS must be **lightweight** and **fast**

- Connections often start with a name lookup (e.g. user typing domain in browser)
- If DNS is slow, every connection is slow

Roadmap

- *What is DNS for?*
- ***Design and implementation***

The First Trial: A Single DNS Name Server

DNS is a client-server, request-response protocol.

- To perform a DNS lookup, you (the client) send a DNS query:
"What is the IP address of www.google.com?"
- The name server sends a DNS response with the answer:
"The IP address of www.google.com is 74.125.25.99."

Name server: A server on the Internet responsible for answering DNS requests

- Name servers have domain names and IP addresses too
- Example: There's a name server with name a.edu-servers.net and IP 192.5.6.30.

Issues:

- Too many requests: one name server can't handle every DNS request from the entire Internet
- Single point of failure
- If there are many name servers, how do you know which one to contact?

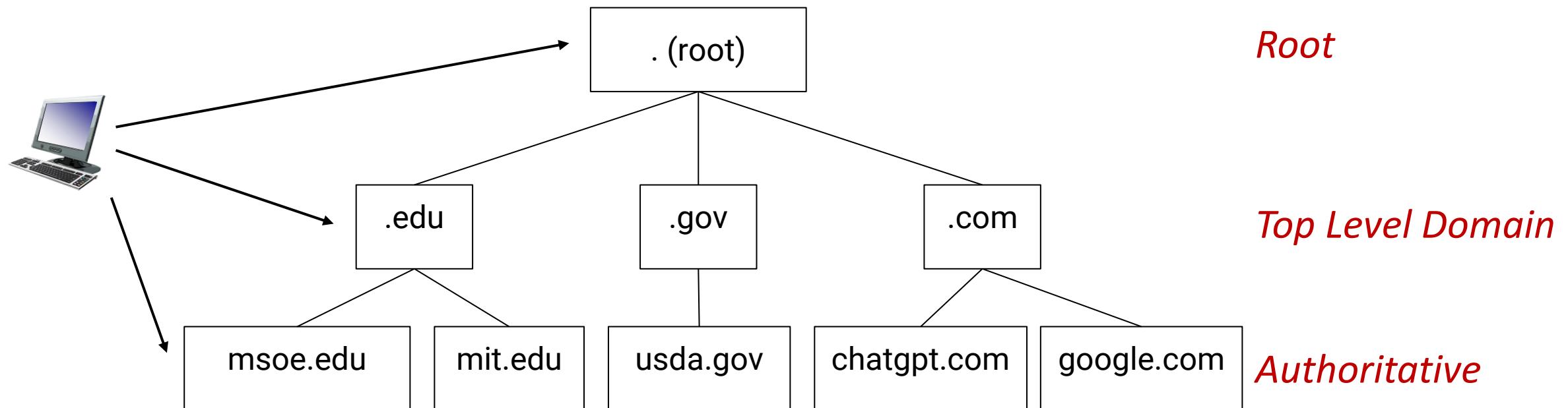
DNS Name Server Hierarchy

Idea #1: If one name server doesn't know the answer to your query, the name server can direct you to another name server

- Analogy: If I don't know the answer to your question, I will direct you to a friend who can help.

Idea #2: Arrange the name servers in a tree hierarchy

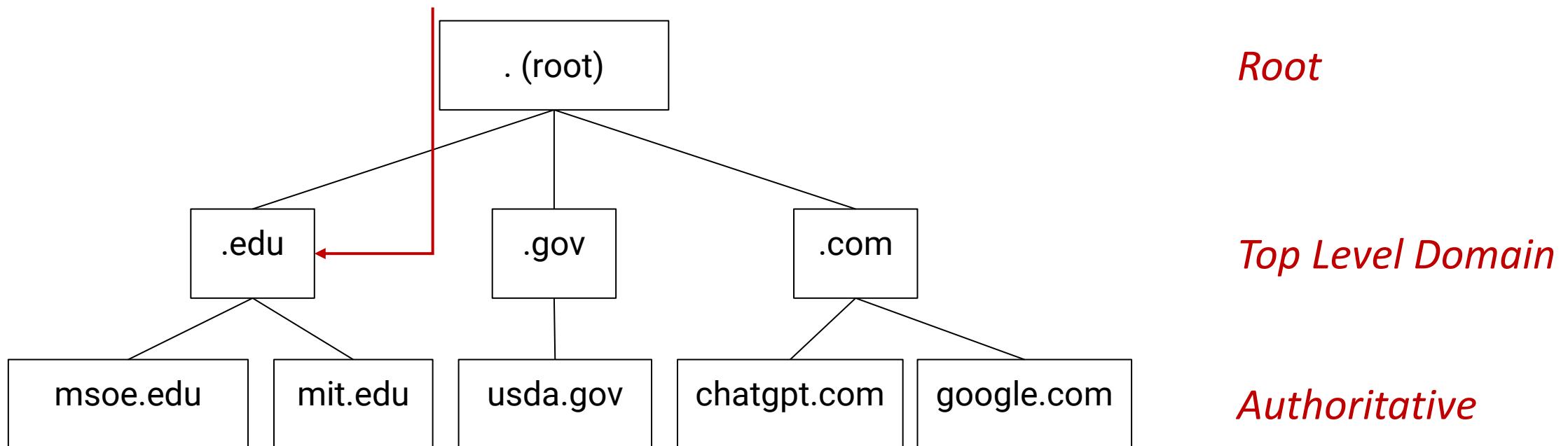
- Intuition: Name servers will direct you down the tree until you receive the answer to your query



DNS Name Server Hierarchy

Each box is a name server. The label represents which queries the name server is responsible for answering

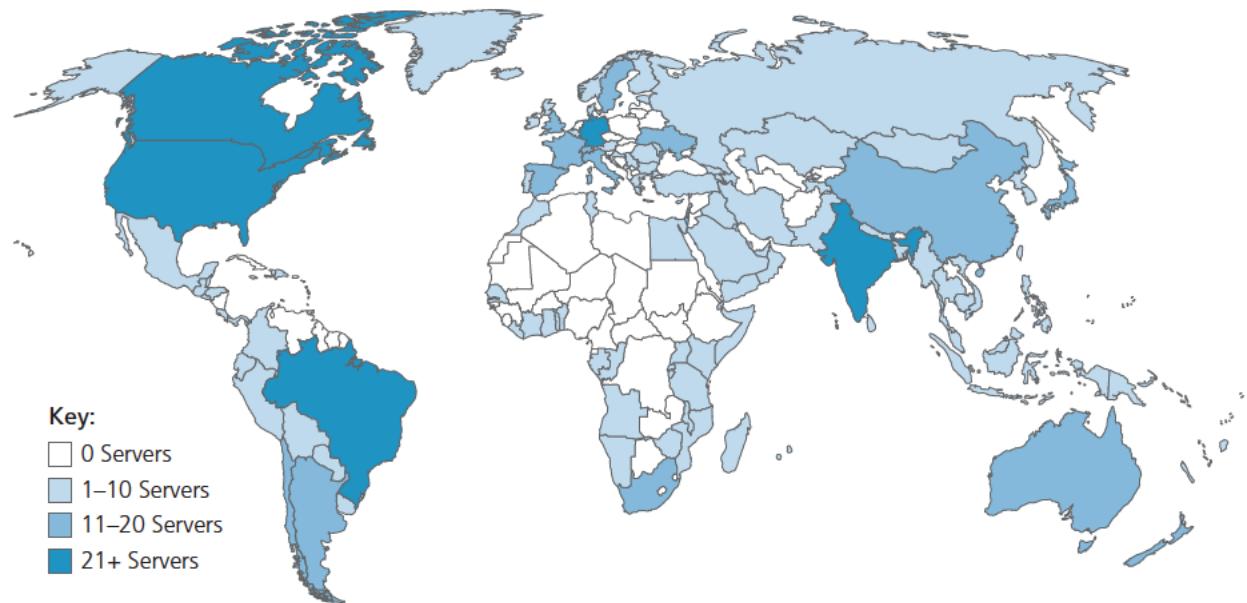
This name server is responsible for .edu queries like www.msoe.edu, but not a query like mail.google.com.



DNS: root name servers

- Official, contact-of-last-resort by name servers that can not resolve name
- *Incredibly important* Internet function
 - Internet couldn't function without it!
 - DNSSEC – provides security (authentication, message integrity)
- ICANN (Internet Corporation for Assigned Names and Numbers) manages root DNS domain

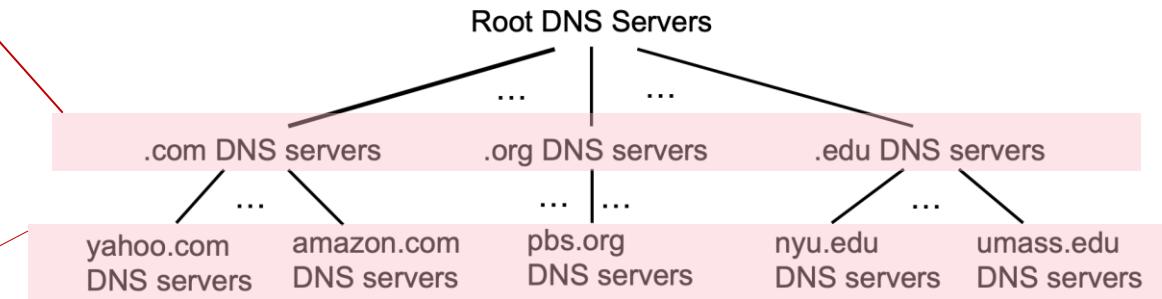
13 logical root name “servers” worldwide each “server” replicated many times (~200 servers in US)



Top-Level Domain, and authoritative servers

Top-Level Domain (TLD) servers:

- Responsible for managing the top-level domains such as .com, .org, .net, .edu, .aero, .jobs, .museums, and all top-level country domains, e.g.: .cn, .uk, .fr, .ca, .jp
- Maintain a directory of which DNS servers are authoritative for each domain under their TLD
- Network Solutions: authoritative registry for .com, .net TLD
- Educause: .edu TLD



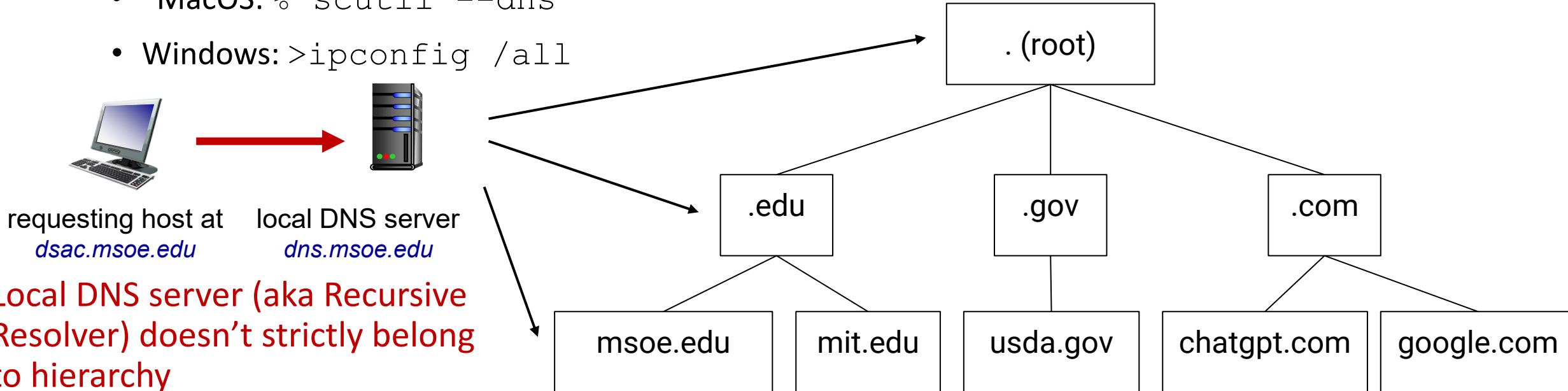
Authoritative DNS servers:

- Organization's own DNS server(s), providing authoritative hostname to IP mappings for organization's named hosts
- Can be maintained by organization or service provider

Local DNS name servers

When host makes DNS query, it is sent to its *local* DNS server

- Local DNS server returns reply:
 - From its local **cache** of recent name-to-address translation pairs (possibly out of date!)
 - Forwarding request into DNS hierarchy for resolution. Once resolved, the local DNS server **caches** the response for future queries to speed up access for you and other users on the network.
- Each ISP has local DNS name server; to find yours:
 - MacOS: % scutil --dns
 - Windows: >ipconfig /all

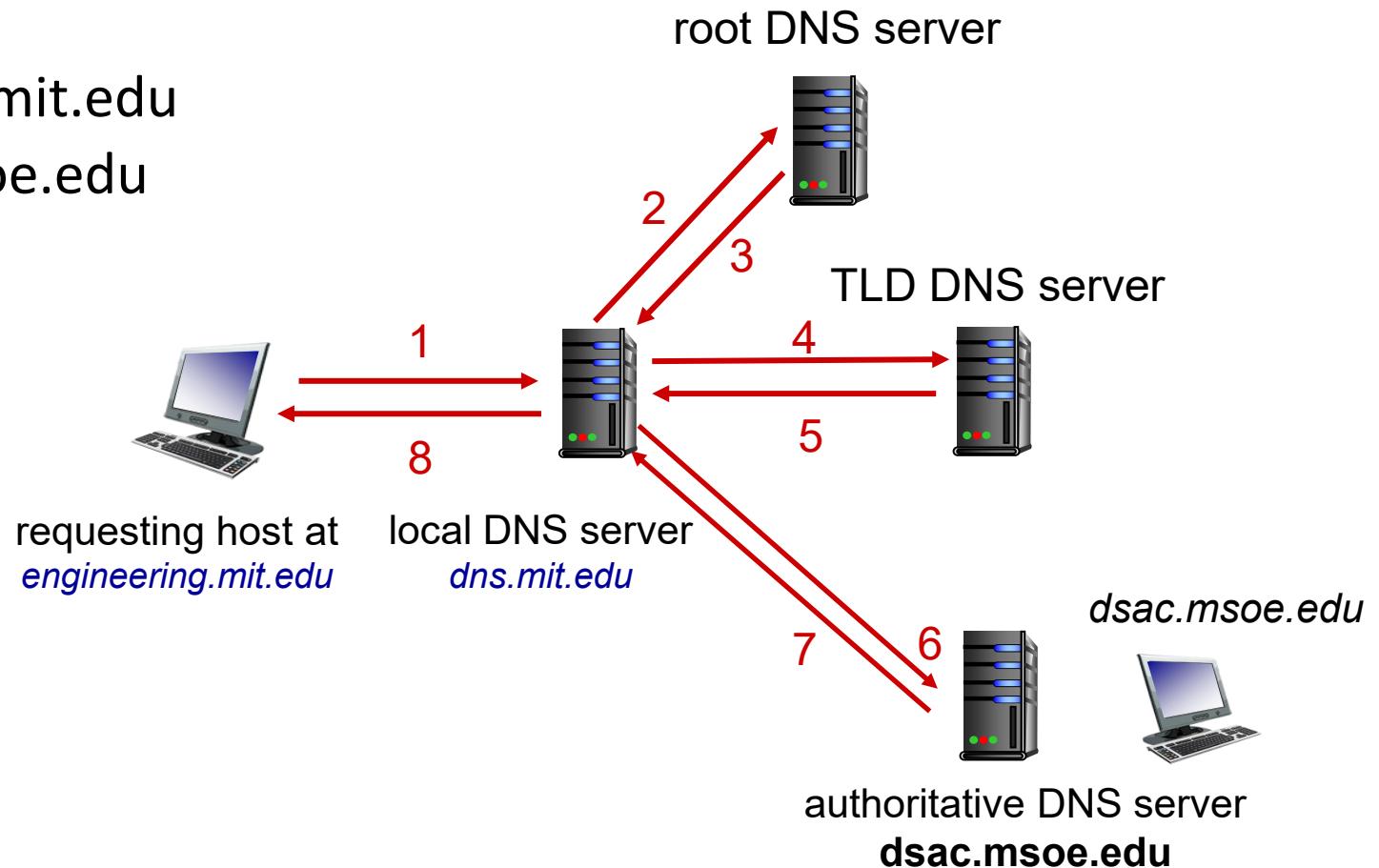


DNS Name Resolution: Iterated Query

Example: host at engineering.mit.edu wants IP address for dsac.msoe.edu

Iterated query:

- Contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”



Caching DNS Information

- Once (any) name server learns mapping, it *caches* mapping, and *immediately* returns a cached mapping in response to a query
 - Caching improves response time
 - Cache entries timeout (disappear) after some time (time to live -TTL)
 - Top-Level Domain (TLD) servers typically cached in local name servers
- Cached entries may be *out-of-date*
 - If named host changes IP address, may not be known Internet-wide until all TTLs expire!

DNS Records

DNS: distributed database storing resource records (RR)

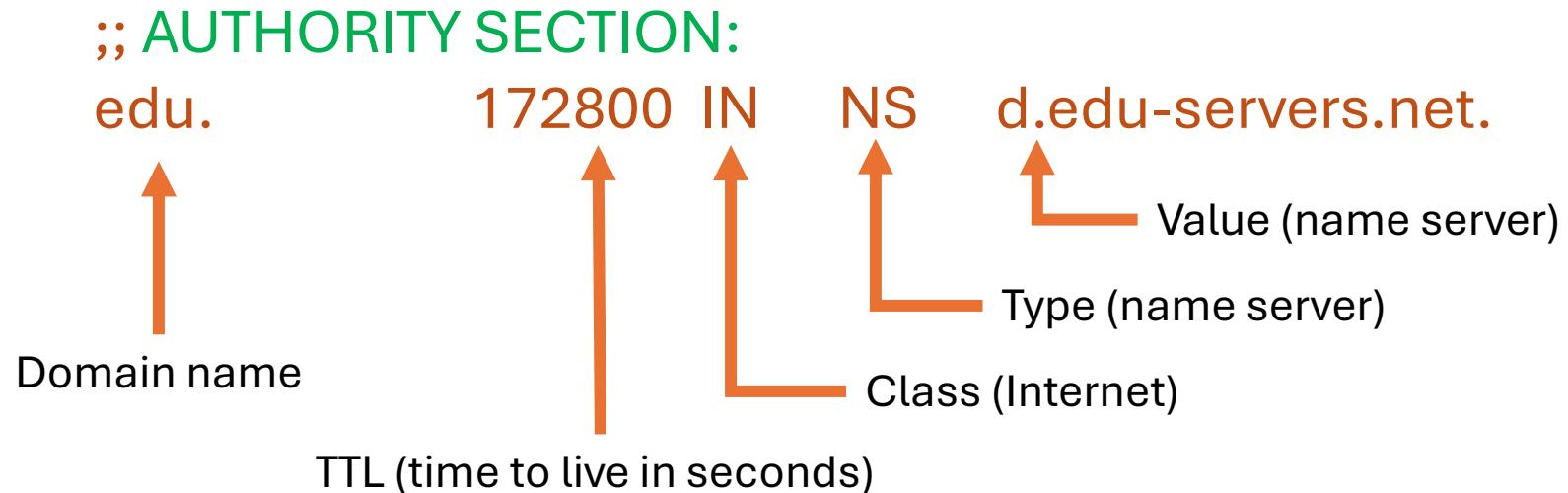
RR format: (name, [TTL], class, value, type)

type=A ("Address" Record)

- name is hostname
- value is IP address

type=NS ("Name Server" Record)

- name is domain (e.g., foo.com)
- value is hostname of authoritative name server for this domain



class: DNS was designed to support different **types** of networks

- CH = Chaos (used for some diagnostic queries)
- HS = Hesiod (used by some Unix systems)
- NONE = Used in dynamic updates
- **99.9% of DNS records use "IN" (Internet class)**

DNS Lookup Walkthrough

```
$ dig +norecurse www.msoe.edu @198.41.0.4
```

```
; <>> DiG 9.16.28 <>> +norecurse www.msoe.edu @198.41.0.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7193
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
```

;; QUESTION SECTION:

```
;www.msoe.edu.      IN   A
```

;; AUTHORITY SECTION:

```
edu.        172800 IN  NS  d.edu-servers.net.
edu.        172800 IN  NS  b.edu-servers.net.
edu.        172800 IN  NS  f.edu-servers.net.
.....
```

;; ADDITIONAL SECTION:

```
d.edu-servers.net. 172800 IN  A  192.31.80.30
d.edu-servers.net. 172800 IN  AAAA 2001:500:856e::30
b.edu-servers.net. 172800 IN  A  192.33.14.30
.....
```

```
;; Query time: 20 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Mon Sep 16 16:44:43 Central Daylight Time 2024
;; MSG SIZE rcvd: 836
```

- DNS queries always start with a request to the root name server

The IP address of the root name server is usually hard-coded into local DNS servers

You can try this at home! Use the dig utility in your terminal

Remember to set the **+norecurse** flag so you can traverse the name server hierarchy yourself

Here's the DNS response from the root name server

DNS Lookup Walkthrough

```
; <>> DiG 9.16.28 <>> +norecurse www.msoe.edu @198.41.0.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7193
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
```

Here's the DNS header

;; QUESTION SECTION:

```
;www.msoe.edu.      IN   A
```

;; AUTHORITY SECTION:

```
edu.        172800 IN   NS   d.edu-servers.net.
edu.        172800 IN   NS   b.edu-servers.net.
edu.        172800 IN   NS   f.edu-servers.net.
```

.....

- id: 7193 → 16-bit ID number
- flags: qr → This is a query record
- QUERY, ANSWER, AUTHORITY, ADDITIONAL → record counts

;; ADDITIONAL SECTION:

```
d.edu-servers.net. 172800 IN   A   192.31.80.30
d.edu-servers.net. 172800 IN   AAAA  2001:500:856e::30
b.edu-servers.net. 172800 IN   A   192.33.14.30
```

.....

```
;; Query time: 20 msec
```

```
;; SERVER: 198.41.0.4#53(198.41.0.4)
```

```
;; WHEN: Mon Sep 16 16:44:43 Central Daylight Time 2024
```

```
;; MSG SIZE rcvd: 836
```

DNS Lookup Walkthrough

```
; <>> DiG 9.16.28 <>> +norecurse www.msoe.edu @198.41.0.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7193
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
```

;; QUESTION SECTION:

```
;www.msoe.edu.      IN  A
```

;; AUTHORITY SECTION:

```
edu.        172800 IN  NS  d.edu-servers.net.
edu.        172800 IN  NS  b.edu-servers.net.
edu.        172800 IN  NS  f.edu-servers.net.
.....
```

;; ADDITIONAL SECTION:

```
d.edu-servers.net. 172800 IN  A  192.31.80.30
d.edu-servers.net. 172800 IN  AAAA 2001:500:856e::30
b.edu-servers.net. 172800 IN  A  192.33.14.30
.....
```

```
;; Query time: 20 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Mon Sep 16 16:44:43 Central Daylight Time 2024
;; MSG SIZE rcvd: 836
```

Here's the DNS payload. It's a collection of resource records (one per line)

DNS Lookup Walkthrough

```
; <>> DiG 9.16.28 <>> +norecurse www.msoe.edu @198.41.0.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7193
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
```

;; QUESTION SECTION:

```
;www.msoe.edu. IN A
```

;; AUTHORITY SECTION:

```
edu. 172800 IN NS d.edu-servers.net.
edu. 172800 IN NS b.edu-servers.net.
edu. 172800 IN NS f.edu-servers.net.
.....
```

;; ADDITIONAL SECTION:

```
d.edu-servers.net. 172800 IN A 192.31.80.30
d.edu-servers.net. 172800 IN AAAA 2001:500:856e::30
b.edu-servers.net. 172800 IN A 192.33.14.30
.....
```

```
; Query time: 20 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Mon Sep 16 16:44:43 Central Daylight Time 2024
;; MSG SIZE rcvd: 836
```

The answer section is blank, because the root name server did not return the answer we're looking for.

We can confirm this by checking the header, which says there are 0 records in the answer section.

Here's the question section. The name is **www.msoe.edu**, the type is **A**, and the value is blank. It shows that we are looking for the IP address of **www.msoe.edu**.

type=A ("Address" Record)

- name is hostname
- value is IP address

DNS Lookup Walkthrough

```
; <>> DiG 9.16.28 <>> +norecurse www.msoe.edu @198.41.0.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7193
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
```

;; QUESTION SECTION:

```
;www.msoe.edu.      IN  A
```

;; AUTHORITY SECTION:

```
edu.        172800 IN  NS  d.edu-servers.net.
edu.        172800 IN  NS  b.edu-servers.net.
edu.        172800 IN  NS  f.edu-servers.net.
.....
```

;; ADDITIONAL SECTION:

```
d.edu-servers.net. 172800 IN  A  192.31.80.30
d.edu-servers.net. 172800 IN  AAAA 2001:500:856e::30
b.edu-servers.net. 172800 IN  A  192.33.14.30
.....
```

```
; Query time: 20 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Mon Sep 16 16:44:43 Central Daylight Time 2024
;; MSG SIZE rcvd: 836
```

The authority and additional sections tell the local DNS server where to look next

Note that there are multiple .edu name servers for redundancy

DNS Lookup Walkthrough

```
; <>> DiG 9.16.28 <>> +norecurse www.msoe.edu @198.41.0.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7193
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
```

;; QUESTION SECTION:

```
;www.msoe.edu.      IN  A
```

;; AUTHORITY SECTION:

```
edu.        172800 IN  NS  d.edu-servers.net.
edu.        172800 IN  NS  b.edu-servers.net.
edu.        172800 IN  NS  f.edu-servers.net.
.....
```

;; ADDITIONAL SECTION:

```
d.edu-servers.net. 172800 IN  A  192.31.80.30
d.edu-servers.net. 172800 IN  AAAA 2001:500:856e::30
b.edu-servers.net. 172800 IN  A  192.33.14.30
.....
```

```
;; Query time: 20 msec
;; SERVER: 198.41.0.4#53(198.41.0.4)
;; WHEN: Mon Sep 16 16:44:43 Central Daylight Time 2024
;; MSG SIZE rcvd: 836
```

This A record helpfully tells us the IP address of the next name server we mean to contact.

DNS sample query (using dig utility)

```
$ dig +norecurse www.msoe.edu @198.41.0.4
```

```
; <>> DiG 9.16.28 <>> +norecurse www.msoe.edu @198.41.0.4
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7193
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27
```

;; QUESTION SECTION:

```
;www.msoe.edu. IN A
```

;; AUTHORITY SECTION:

```
edu. 172800 IN NS d.edu-servers.net.
edu. 172800 IN NS b.edu-servers.net.
edu. 172800 IN NS f.edu-servers.net.
.....
```

;; ADDITIONAL SECTION:

```
d.edu-servers.net. 172800 IN A 192.31.80.30
d.edu-servers.net. 172800 IN AAAA 2001:500:856e::30
b.edu-servers.net. 172800 IN A 192.33.14.30
.....
```

```
;; Query time: 20 msec
```

```
;; SERVER: 198.41.0.4#53(198.41.0.4)
```

```
;; WHEN: Mon Sep 16 16:44:43 Central Daylight Time 2024
```

```
;; MSG SIZE rcvd: 836
```

```
C:\Users\liao>dig +norecurse www.msoe.edu @192.33.14.30
; <>> DiG 9.16.28 <>> +norecurse www.msoe.edu @192.33.14.30
;; global options: +cmd
R, id: 19697
4, ADDITIONAL: 1
```

DNS queries always start with a request to the root name server. The IP address of the root name server is usually hard-coded into recursive resolvers

```
msoe.edu. 172800 IN NS ns1-05.azure-dns.com.
msoe.edu. 172800 IN NS ns2-05.azure-dns.net.
msoe.edu. 172800 IN NS ns3-05.azure-dns.org.
msoe.edu. 172800 IN NS ns4-05.azure-dns.info.

;; Query time: 14 msec
;; SERVER: 192.33.14.30#53(192.33.14.30)
;; WHEN: Mon Sep 16 16:59:31 Central Daylight Time 2024
;; MSG SIZE rcvd: 178
```

```
C:\Users\liao>dig +norecurse www.msoe.edu @ns1-05.azure-dns.com.

; <>> DiG 9.16.28 <>> +norecurse www.msoe.edu @ns1-05.azure-dns.com.
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1086
;; flags: qr aa ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

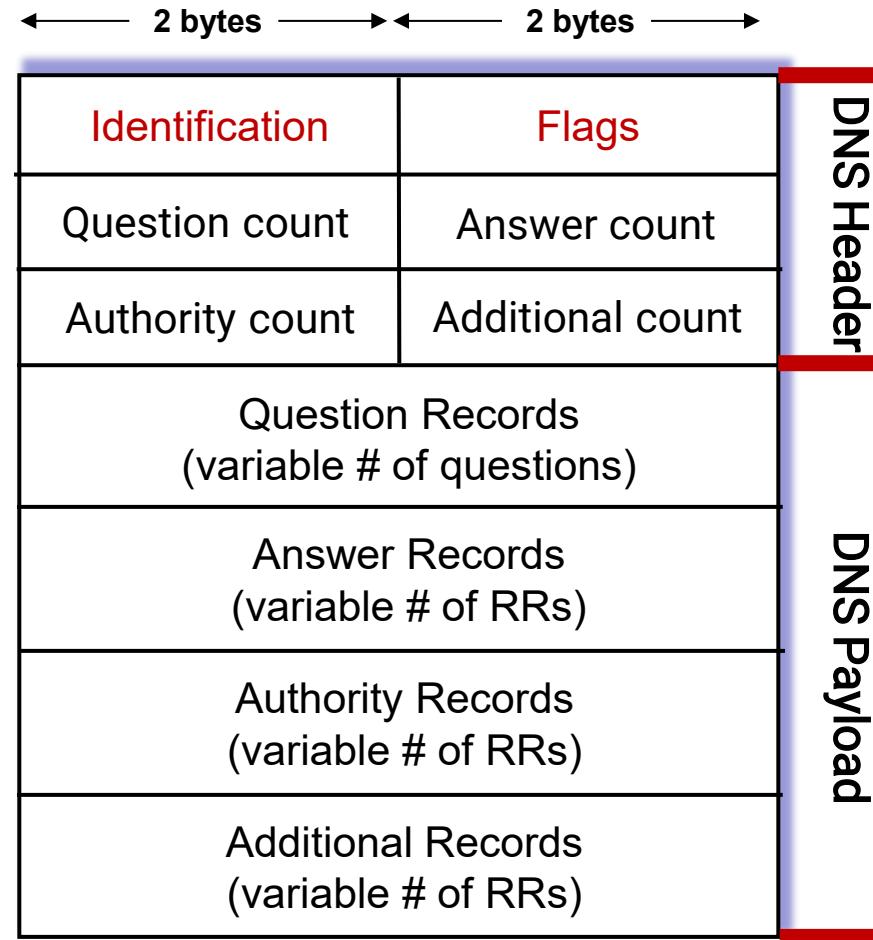
;; OPT PSEUDOSECTION:
;; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;www.msoe.edu. IN A

;; ANSWER SECTION:
www.msoe.edu. 3600 IN CNAME msoe.edu.
msoe.edu. 3600 IN A 69.20.113.171

;; Query time: 6 msec
;; SERVER: 13.107.236.5#53(13.107.236.5)
;; WHEN: Mon Sep 16 16:59:44 Central Daylight Time 2024
;; MSG SIZE rcvd: 71
```

DNS Protocol Messages

DNS *query* and *reply* messages, both have same *format*:



Identification (ID number):

- 16-bit number for query, reply to query uses **same number**

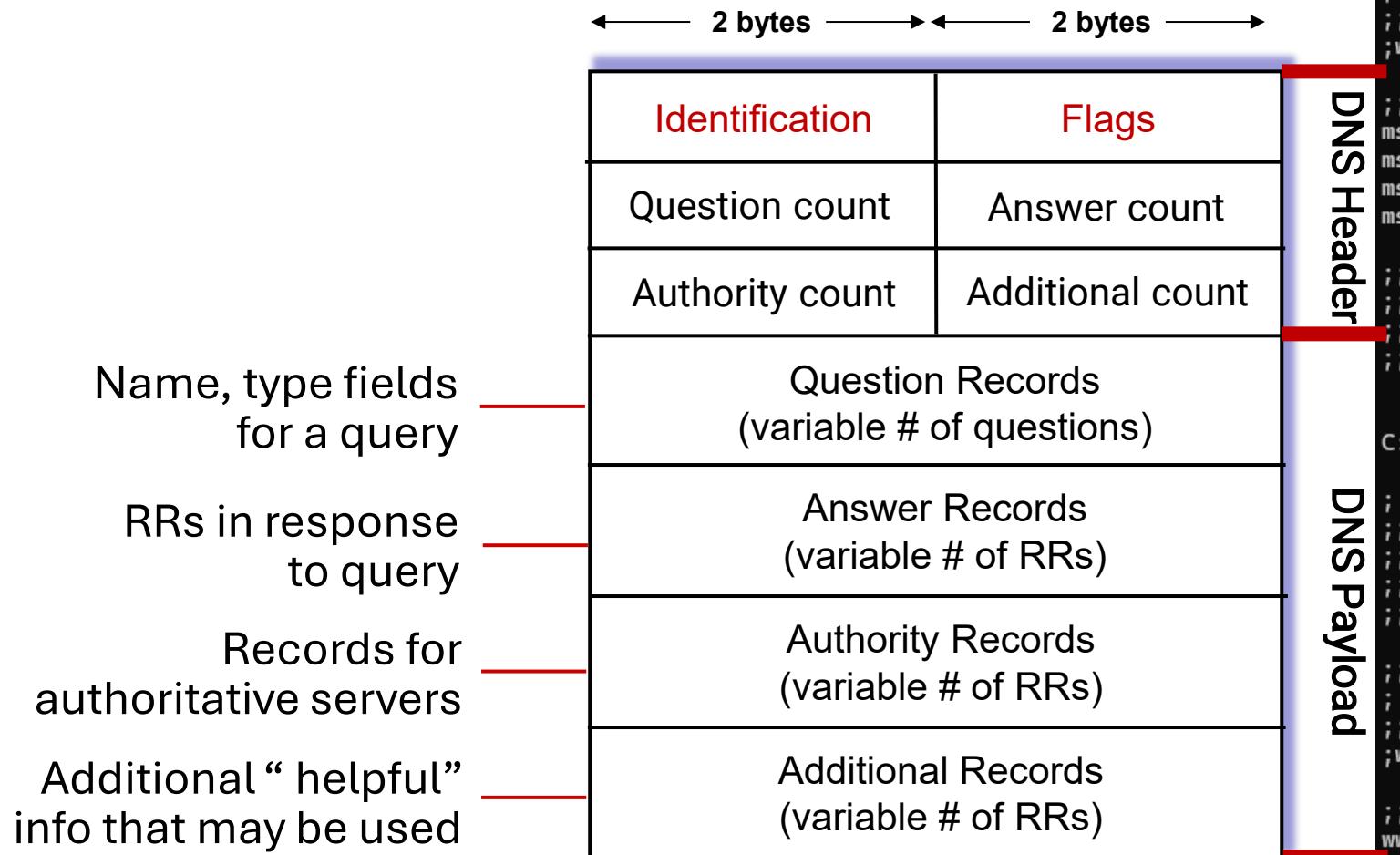
Flags:

- **QR bit**: 0 in queries, 1 in responses
- RD bit: 1 if we want the recursive resolver to do the lookup for us, and 0 otherwise

Counts:

- The number of records of each type in the DNS payload

DNS Protocol Messages



```
C:\Users\liao>dig +norecurse www.msue.edu @192.33.14.30
; <>> DiG 9.16.28 <>> +norecurse www.msue.edu @192.33.14.30
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 19697
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.msue.edu.           IN      A

;; AUTHORITY SECTION:
msue.edu.          172800  IN      NS      ns1-05.azure-dns.com.
msue.edu.          172800  IN      NS      ns2-05.azure-dns.net.
msue.edu.          172800  IN      NS      ns3-05.azure-dns.org.
msue.edu.          172800  IN      NS      ns4-05.azure-dns.info.

;; Query time: 14 msec
;; SERVER: 192.33.14.30#53(192.33.14.30)
;; WHEN: Mon Sep 16 16:59:31 Central Daylight Time 2024
;; MSG SIZE rcvd: 178

C:\Users\liao>dig +norecurse www.msue.edu @ns1-05.azure-dns.com.
; <>> DiG 9.16.28 <>> +norecurse www.msue.edu @ns1-05.azure-dns.com.
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 1086
;; flags: qr aa ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;www.msue.edu.           IN      A

;; ANSWER SECTION:
www.msue.edu.        3600    IN      CNAME   msue.edu.
msue.edu.            3600    IN      A       69.20.113.171

;; Query time: 6 msec
;; SERVER: 13.107.236.5#53(13.107.236.5)
;; WHEN: Mon Sep 16 16:59:44 Central Daylight Time 2024
;; MSG SIZE rcvd: 71
```

DNS Security

DDoS attacks

- Bombard root servers with traffic
 - Not successful to date
 - Traffic filtering
 - Local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
 - Potentially more dangerous

Spoofing attacks

- intercept DNS queries, returning bogus replies
 - DNS cache poisoning
 - RFC 4033: DNSSEC authentication services

We will discuss this topic in Week 9

Application Layer Summary

Our study of network application layer is now complete!

- Application architectures
 - Client-server
 - P2P
- Network performance:
 - Delay, packet loss, throughput
- Cache
- Cookie
- Specific protocols:
 - HTTP
 - DNS

Next:

- Transport layer service
 - Connection-oriented, reliable: TCP
 - Unreliable, datagrams: UDP
- Socket programming:
 - TCP, UDP sockets

CSC 3511 Security and Networking

Week 3, Lecture 2: Transport Layer and UDP

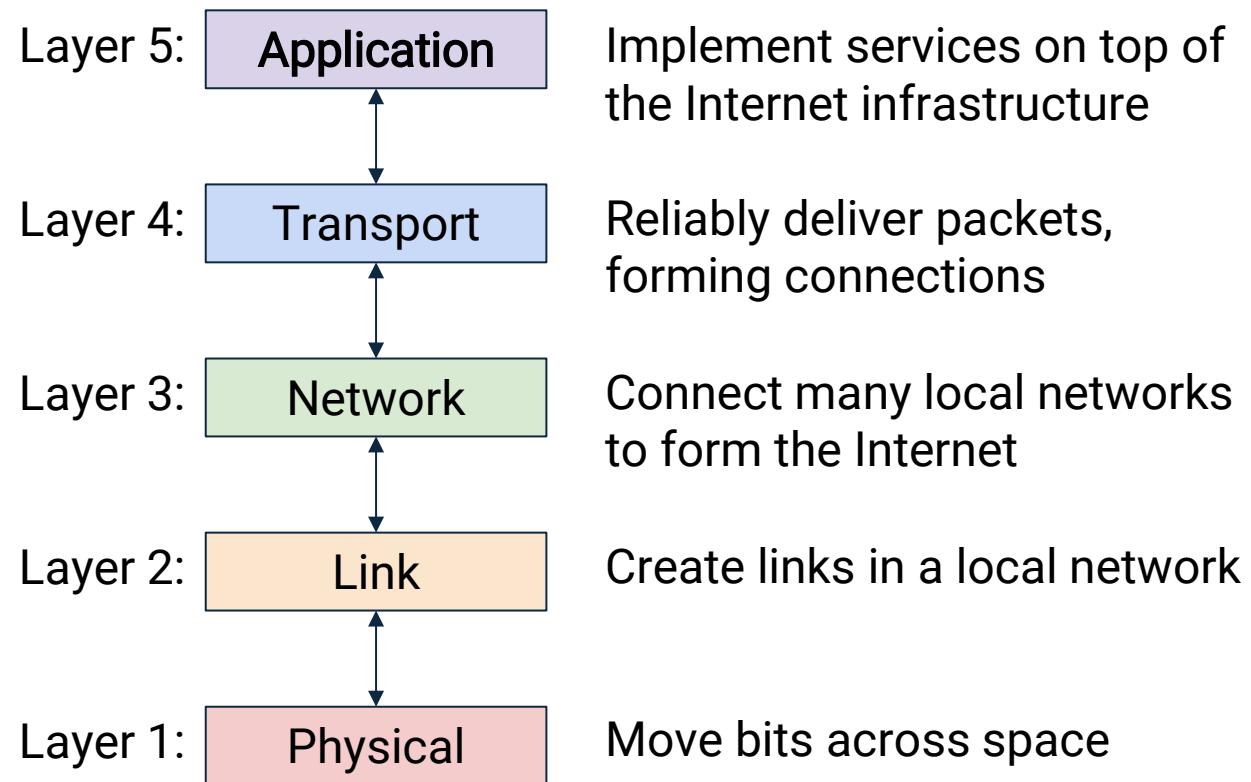
Roadmap

- ***Transport-layer services overview***
- *Connectionless transport: UDP*

Internet Layer: Review

Layer 3 (Network) gave us the ability to send packets anywhere on the Internet.

- Abstraction level: Packets individually sent through the network.
- Problem: IP offers ***best-effort*** delivery
 - Packets can be **lost**, corrupted, reordered, **delayed**, or duplicated
- Problem: Applications don't want to think about packets and best-effort
 - Programmers want to think in terms of a more convenient abstraction

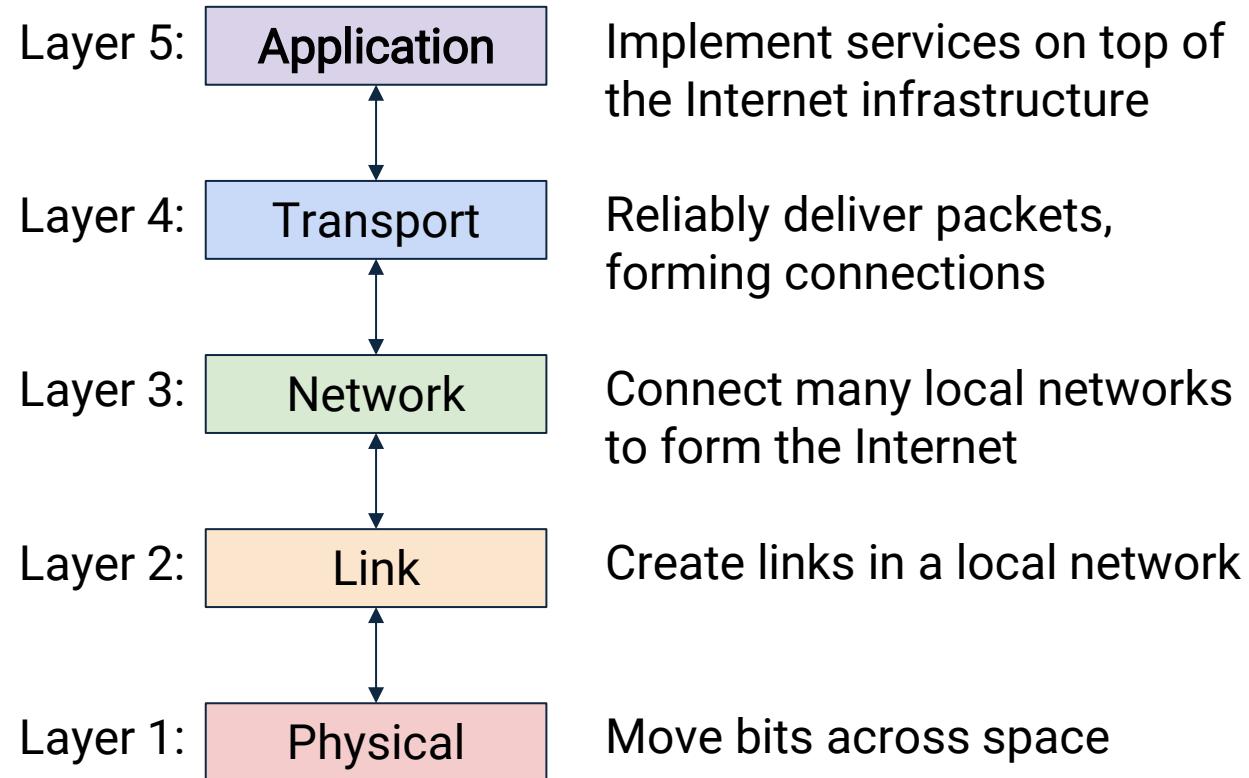


Transport Layer: Overview

Layer 4 (Transport) builds extra features on top of Layer 3

- **TCP** (Transmission Control Protocol): Adds *de-multiplexing* and *reliability*
- **UDP** (User Datagram Protocol): Adds *de-multiplexing* only
- Both protocols provide a more useful abstraction to programmers

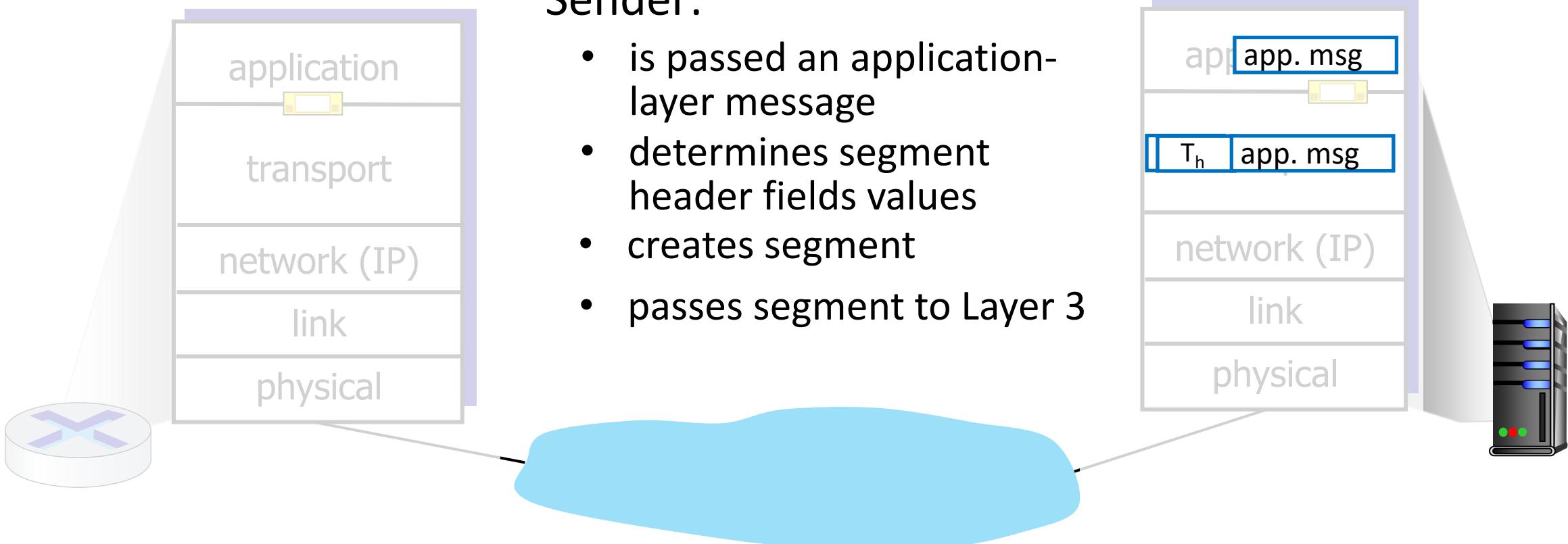
Let's check out each of these features



Transport Layer Actions

Transport protocols actions in end systems:

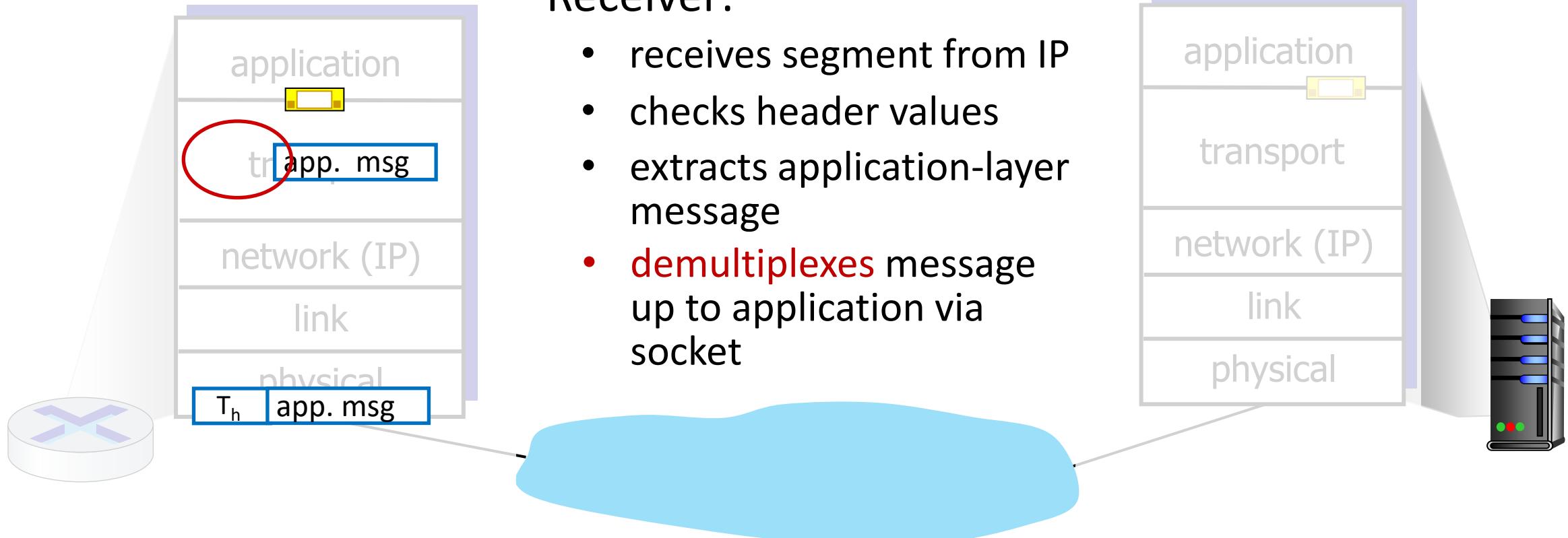
- Sender: breaks application messages into *segments*, passes to network layer
- Receiver: reassembles segments into messages, passes to application layer



Transport Layer Actions

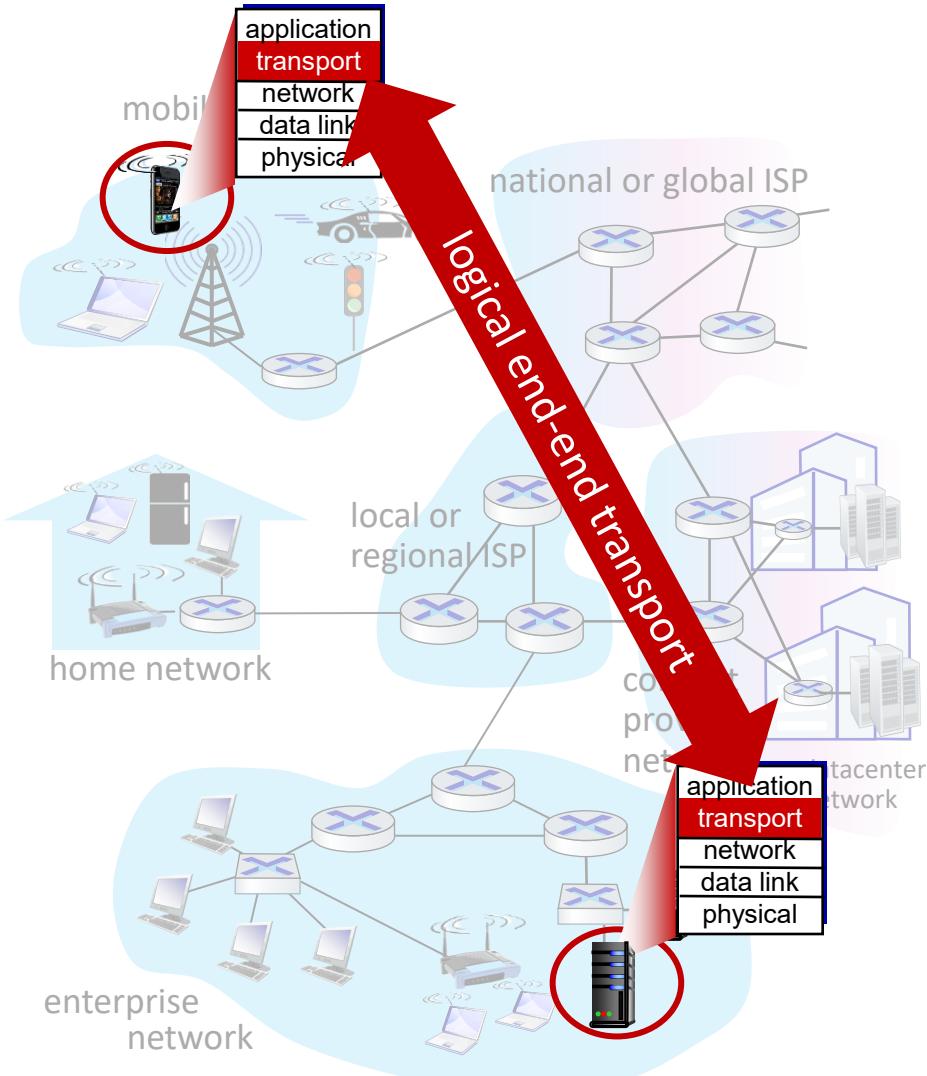
Transport protocols actions in end systems:

- Sender: breaks application messages into *segments*, passes to network layer
- Receiver: reassembles segments into messages, passes to application layer



Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
 - Reliable, in-order delivery
 - Congestion control
 - Flow control
 - Connection setup
- **UDP:** User Datagram Protocol
 - Unreliable, unordered delivery
 - No-frills extension of “best-effort” IP
- Services *not* available:
 - Delay guarantees
 - Bandwidth guarantees



Roadmap

- *Transport-layer services overview*
- ***Connectionless transport: UDP***

UDP: User Datagram Protocol

- “no frills,” “bare bones” Internet transport protocol
- “best effort” service, UDP segments may be:
 - Lost
 - Delivered out-of-order to app
- *Connectionless:*
 - No **handshaking** between UDP sender, receiver
 - Each UDP segment handled independently of others

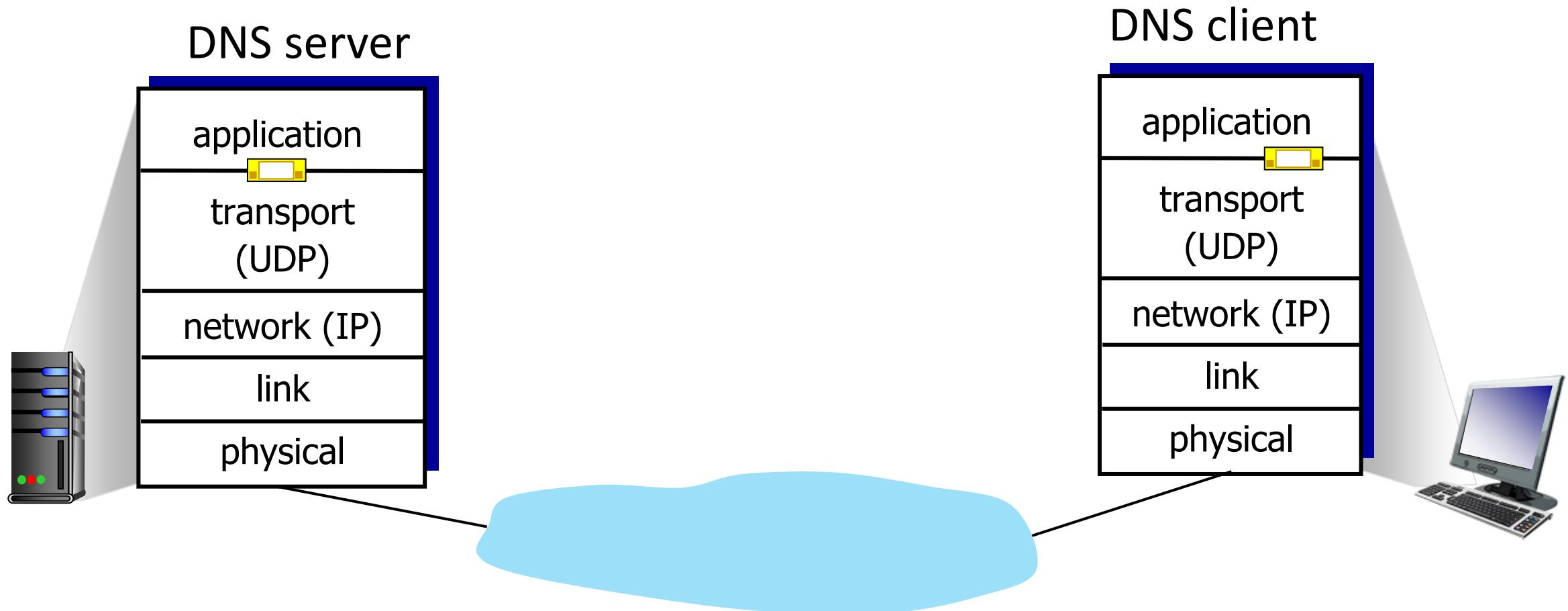
Why is there a UDP?

- No connection establishment (which can add RTT delay)
- Simple: no connection state at sender, receiver
- Small header size
- No congestion control
 - UDP can blast away as fast as desired!
 - Can function in the face of congestion

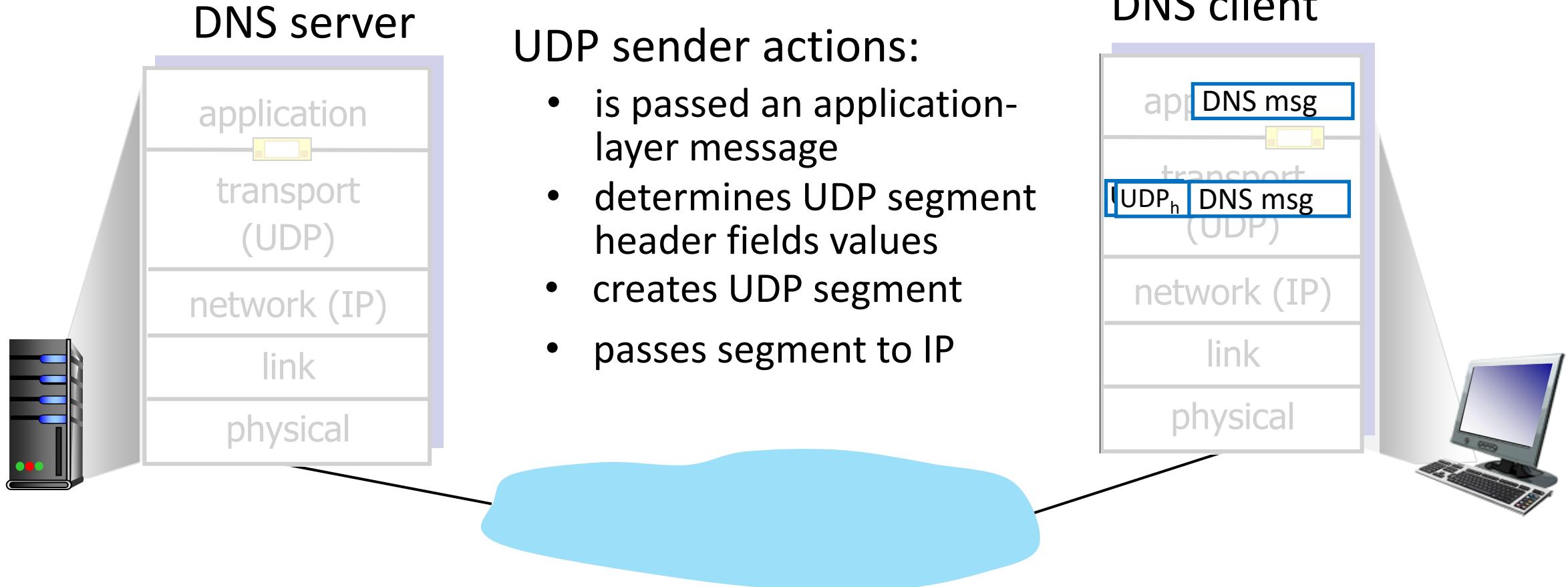
UDP: User Datagram Protocol

- UDP use:
 - **streaming multimedia apps** (loss tolerant, rate sensitive)
 - **DNS**
 - **SNMP**
 - **HTTP/3**
- If reliable transfer needed over UDP (e.g., HTTP/3):
 - Add needed reliability at application layer
 - Add congestion control at application layer

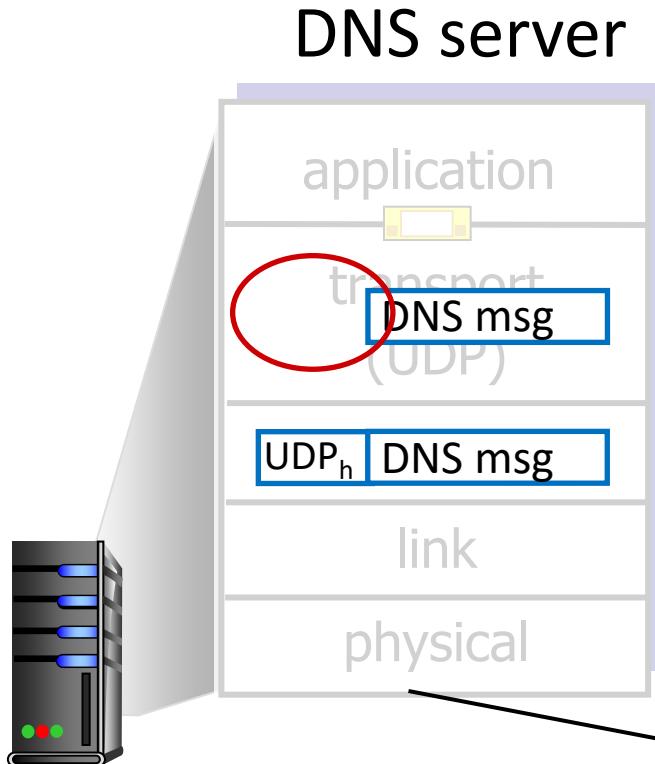
UDP: Transport Layer Actions



UDP: Transport Layer Actions



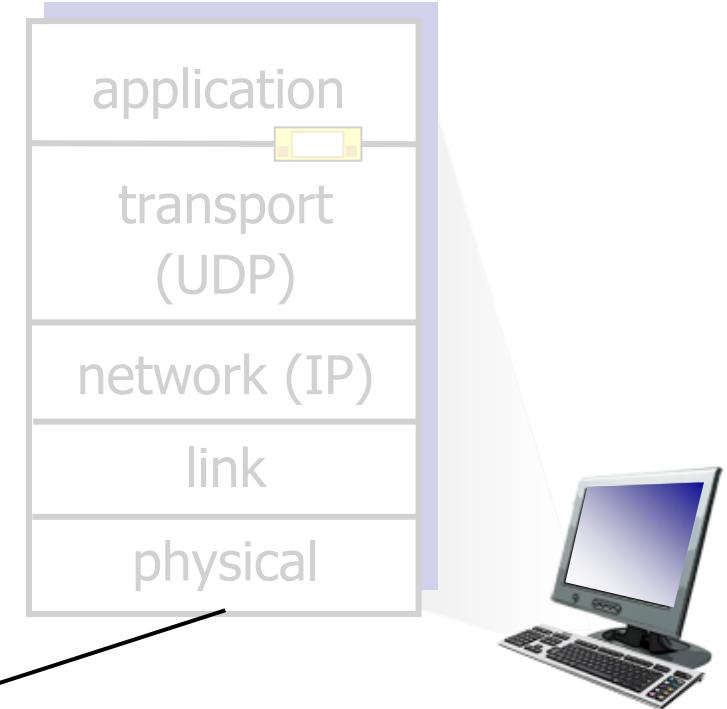
UDP: Transport Layer Actions



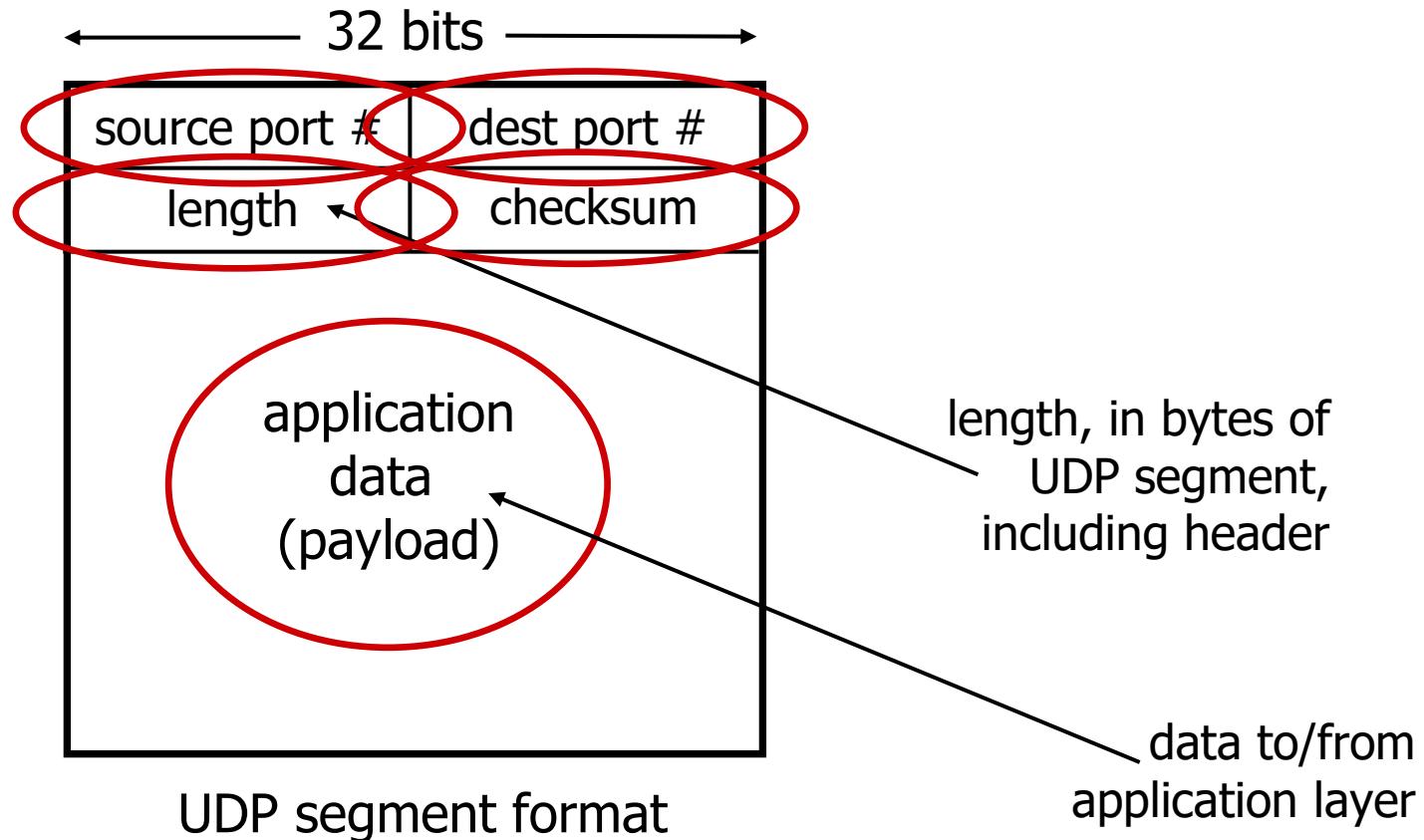
UDP receiver actions:

- receives segment from IP
- checks UDP checksum header value
- extracts application-layer message
- **demultiplexes** message up to application via socket

DNS client



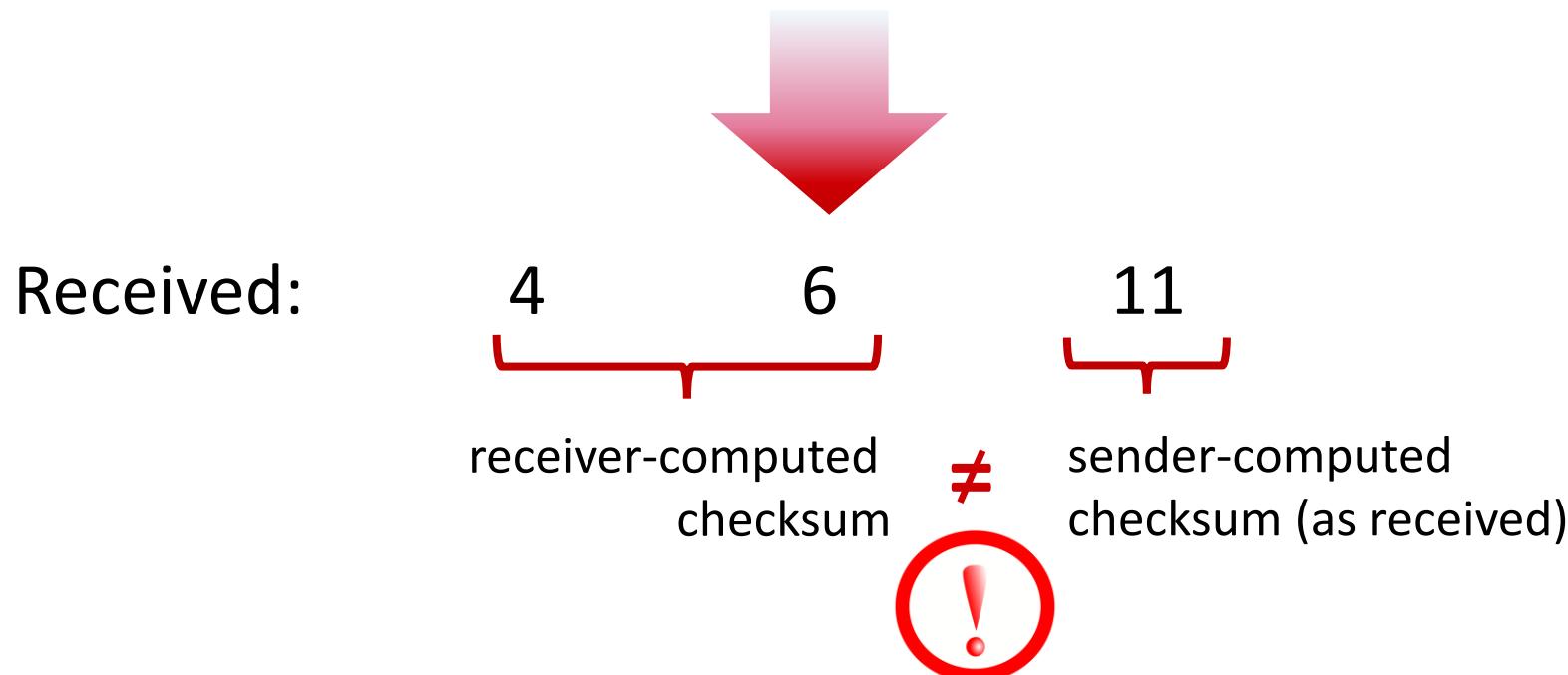
UDP segment header



UDP checksum

Goal: detect errors (*i.e.*, flipped bits) in transmitted segment

	1 st number	2 nd number	sum
Transmitted:	5	6	11



Internet checksum

Goal: detect errors (*i.e.*, flipped bits) in transmitted segment

sender:

- Treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- **Checksum:** addition (one's complement sum) of segment content
- Checksum value put into UDP checksum field

receiver:

- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
 - not equal - error detected
 - equal - no error detected. *But maybe errors nonetheless?* More later

Internet checksum: an example

example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
<hr/>																
wraparound (carryout bit)	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1
<hr/>																
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1
<hr/>																

The **carryout** bit is added back into the LSB of the result

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

* Check out the online interactive exercises for more examples:
http://gaia.cs.umass.edu/kurose_ross/interactive/internet_checksum.php

UDP Checksum Calculation at the Sender

- Prepare the UDP packet:
 - Header fields: source port, destination port, checksum (initialized as 0)
 - Data: The actual payload being sent (e.g., DNS query).
- Create the pseudo-header:
 - Source IP, destination IP, protocol number (17 for UDP), total length (header & data)
- Compute & insert checksum:
 - Concatenate pseudo-header, UDP header, and data as a unified structure
 - Interpret the above structure as 16-bit integers to compute the checksum
 - Insert the checksum into the UDP header
 - **A special case:** if the computed checksum is zero; it's transmitted as 0xFFFF

UDP Checksum Verification at the Receiver

- Receive the UDP packet:
 - Extract source port, destination port, length, checksum, and payload data
- Rebuild the pseudo-header:
 - Source IP, destination IP, protocol number (17), total length (header & data)
- Verify checksum:
 - Re-create the same structure as the sender for checksum verification
 - Verify checksum: all 16-bit words, **including the received checksum**
 - Apply wraparound to carry out bits (same as sender)
 - Verify if the result is all 1s (1111 1111 1111 1111)

Internet checksum: weak protection!

example: add two 16-bit integers

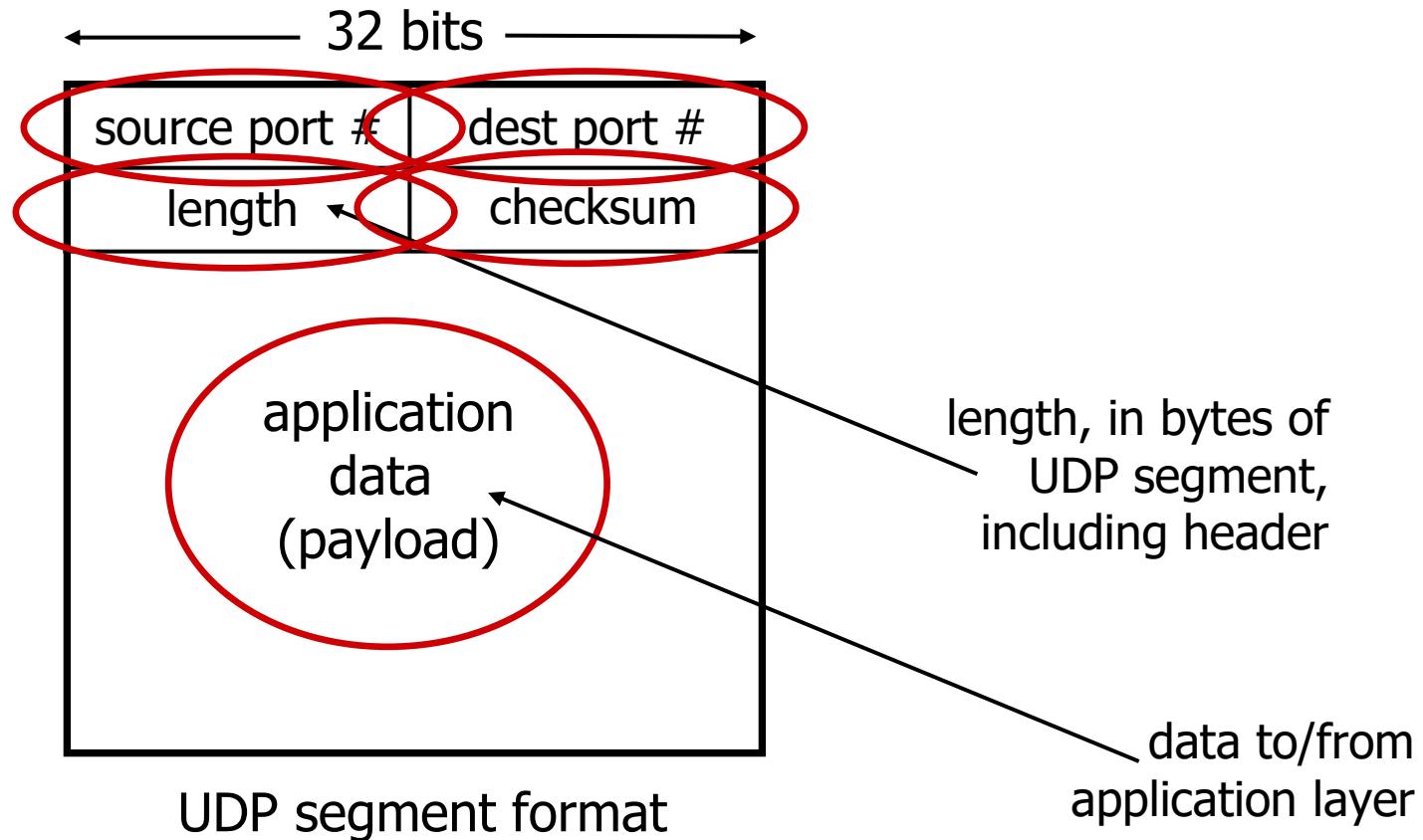
	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0	1
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0	1
<hr/>																			
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	0	1
sum	1	0	1	1	1	0	1	1	1	0	1	1	1	1	1	0	0	0	0
checksum	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	1	0	0

Even though numbers have changed (bit flips), **no** change in checksum!

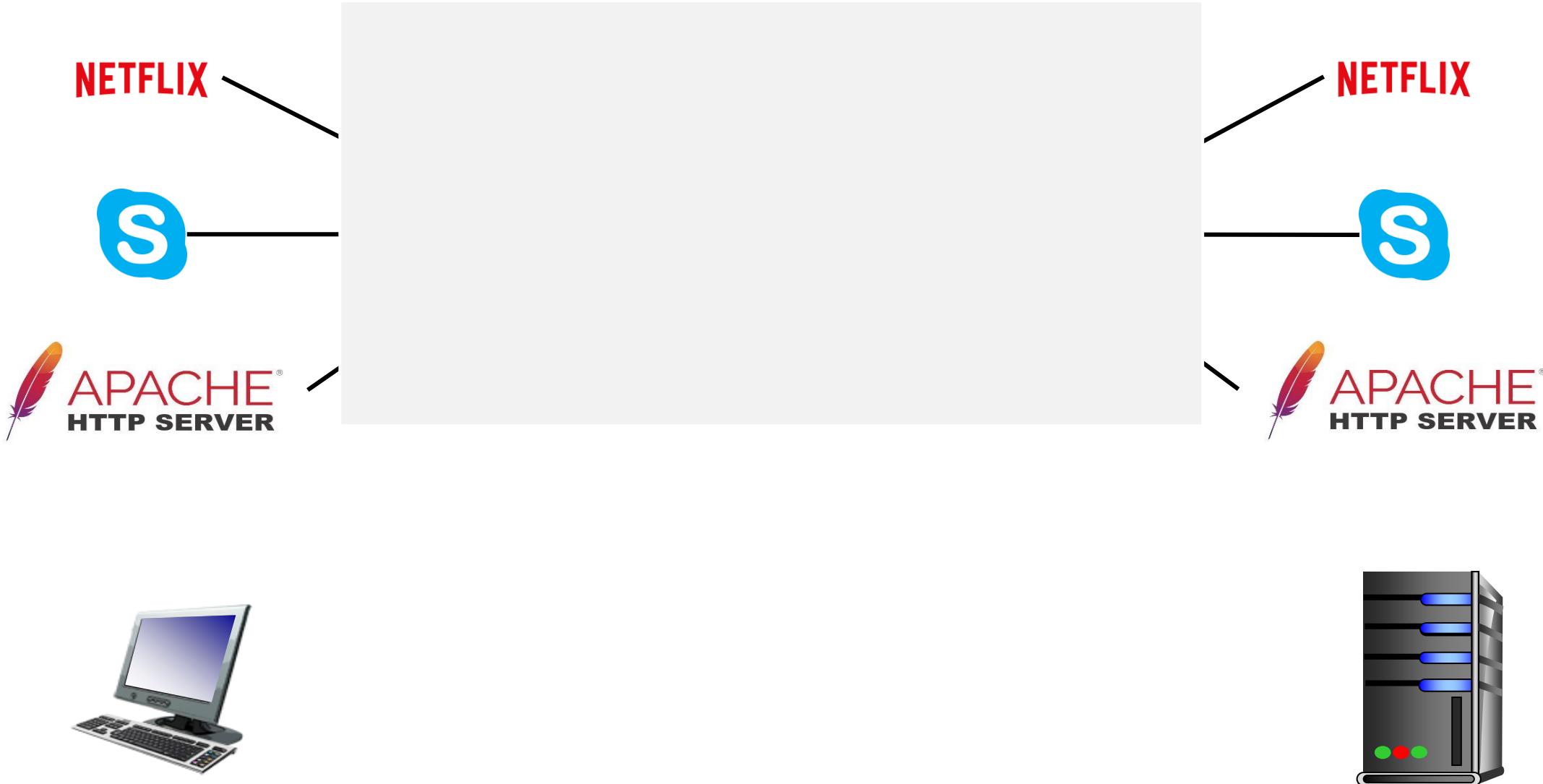
Summary: UDP

- “no frills” protocol:
 - Segments may be lost, delivered out of order
 - Best effort service: “send and hope for the best”
- UDP has its plusses:
 - No setup/handshaking needed (no RTT incurred)
 - Can function when network service is compromised
 - Helps with reliability (checksum)
- Build additional functionality on top of UDP in application layer (e.g., HTTP/3)

Challenge question 1: Where is the IP Address in a UDP Segment?



Challenge question 2: How does the computer know which incoming data packets belong to Netflix vs Skype vs Apache?



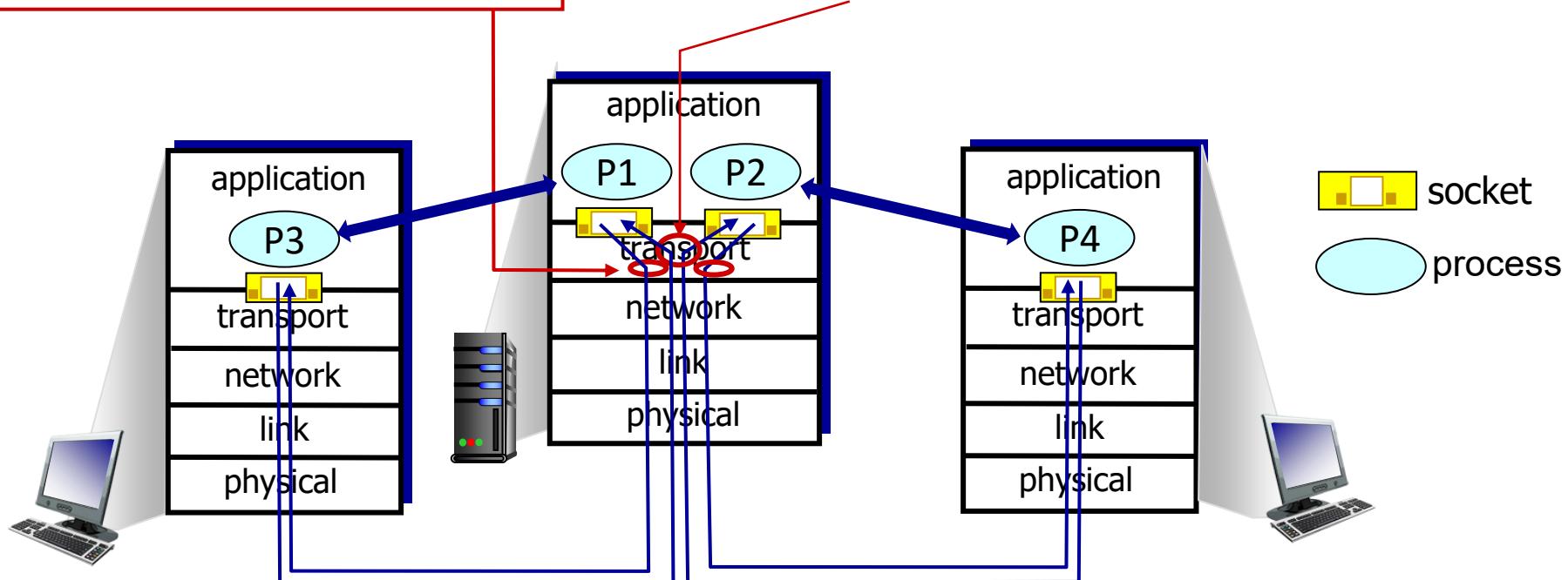
Multiplexing/demultiplexing

multiplexing as sender:

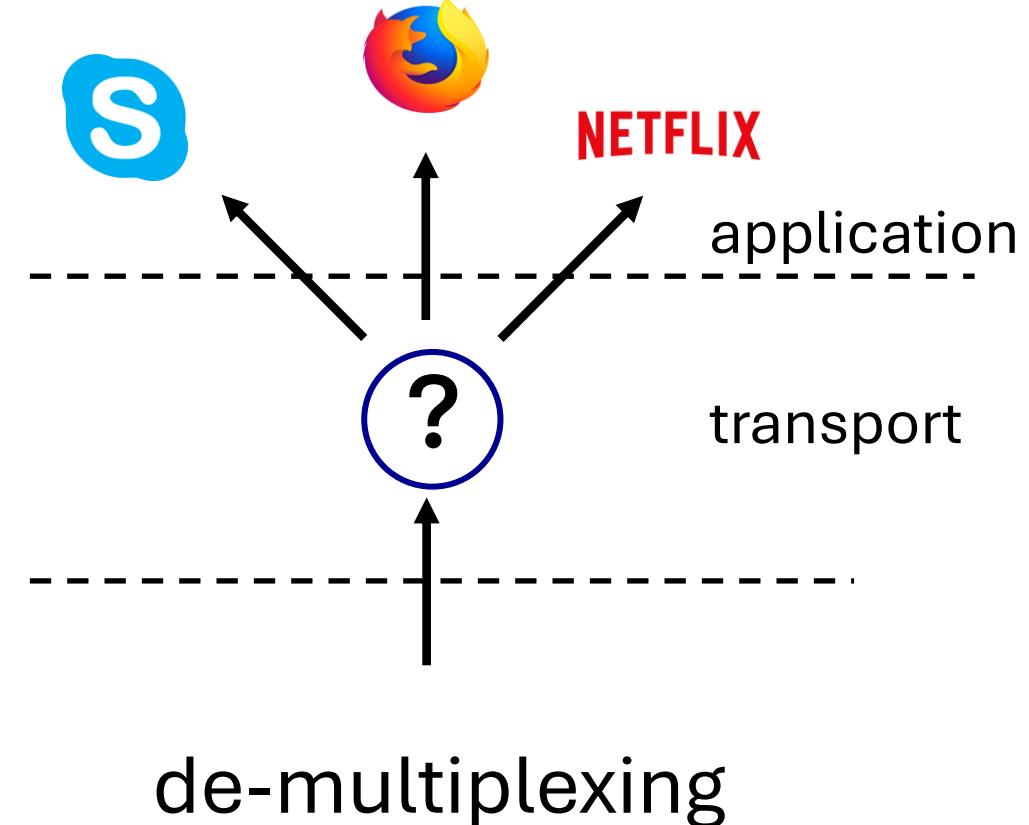
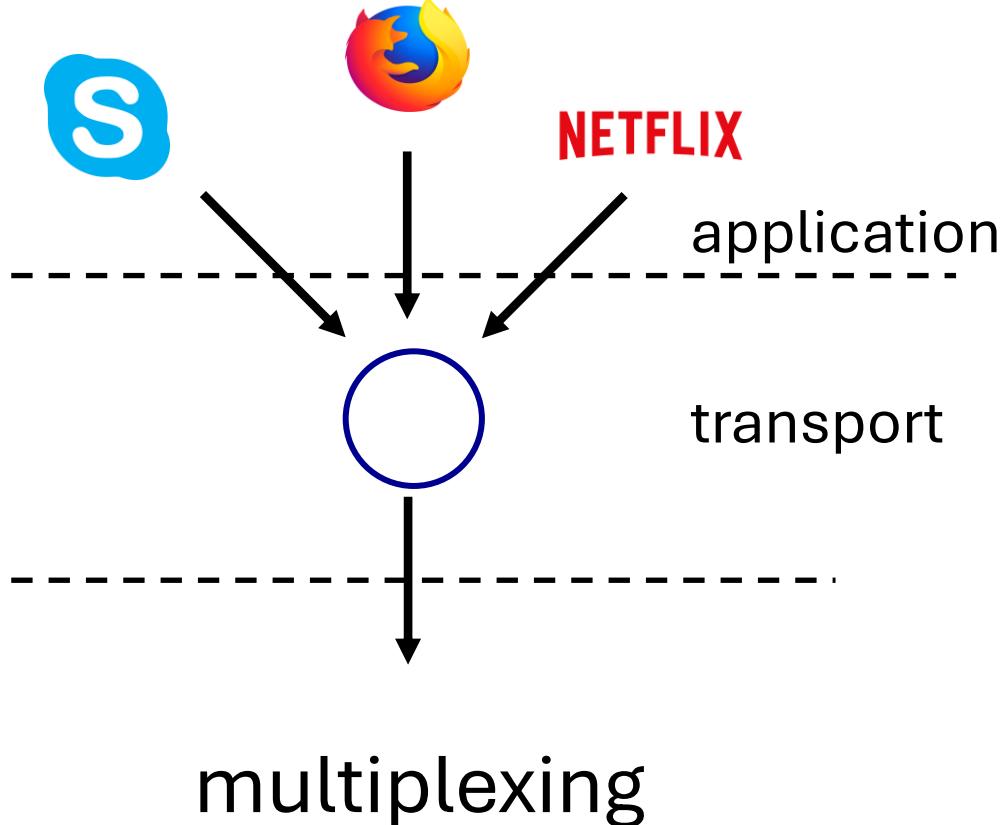
handle data from multiple sockets, add transport header (later used for demultiplexing)

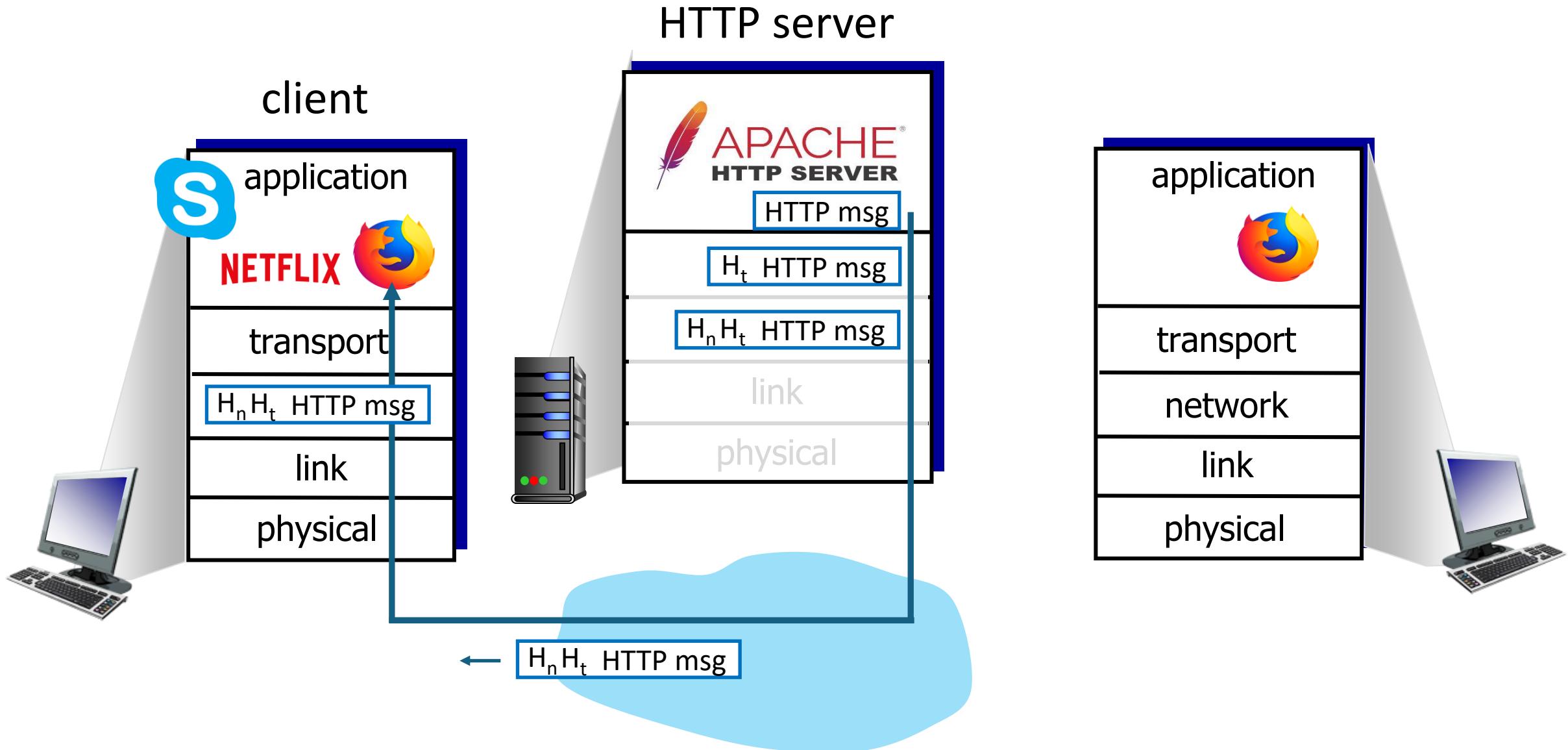
demultiplexing as receiver:

use header info to deliver received segments to correct socket



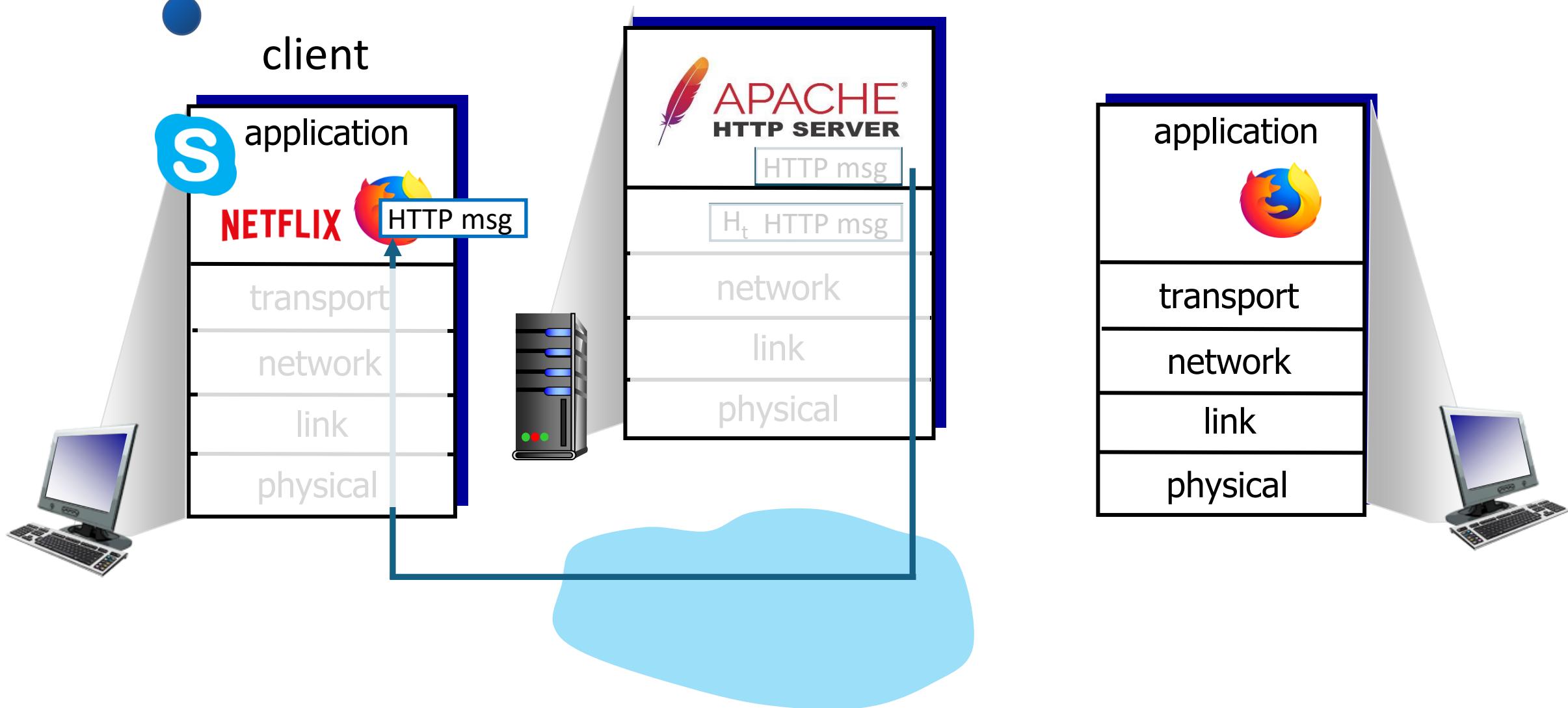
Multiplexing/demultiplexing





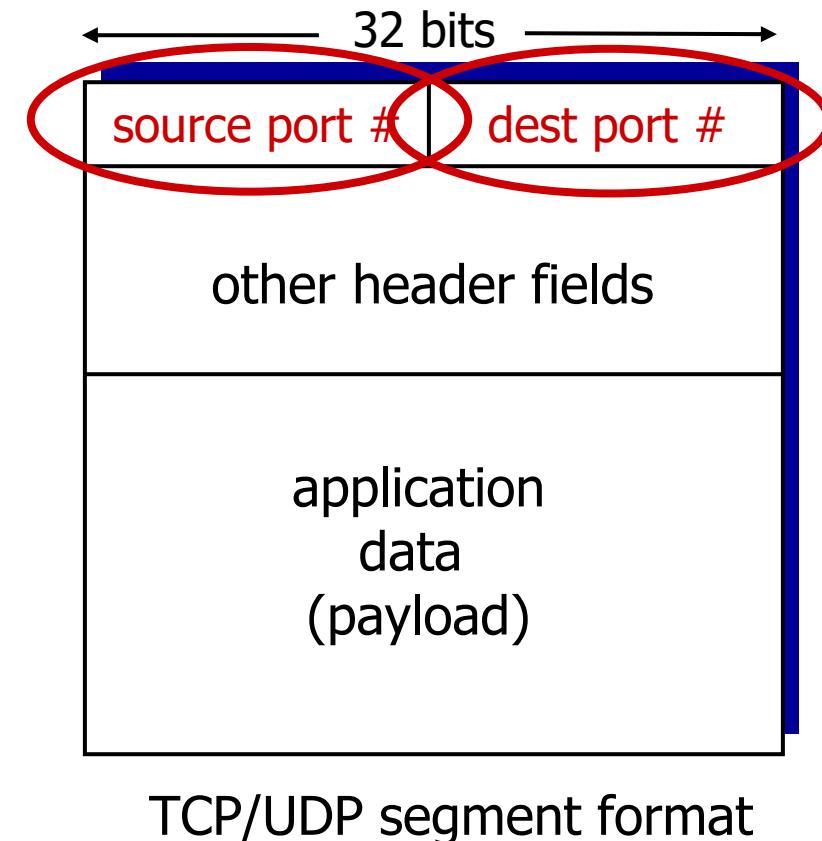


Q: how did transport layer know to deliver message to Firefox browser process rather than Netflix process or Skype process?



How demultiplexing works

- Receiving host receives IP datagrams:
 - Each datagram has source IP address, destination IP address
 - Each datagram carries one transport-layer segment
 - Each segment has source, destination port number
- Receiving host uses *IP addresses & port numbers* to direct segment to appropriate socket



Connectionless demultiplexing

Recall:

- When creating socket, must specify *host-local* port #:

```
serverSocket = socket(AF_INET, SOCK_DGRAM)  
serverSocket.bind(("0.0.0.0", serverPort))
```

- When creating datagram to send into UDP socket, must specify
 - Destination IP address
 - Destination port #

```
clientSocket.sendto(message.encode(),  
(serverName, serverPort))
```

When receiving host receives UDP segment:

- Checks destination port # in segment
- Directs UDP segment to socket with that port #

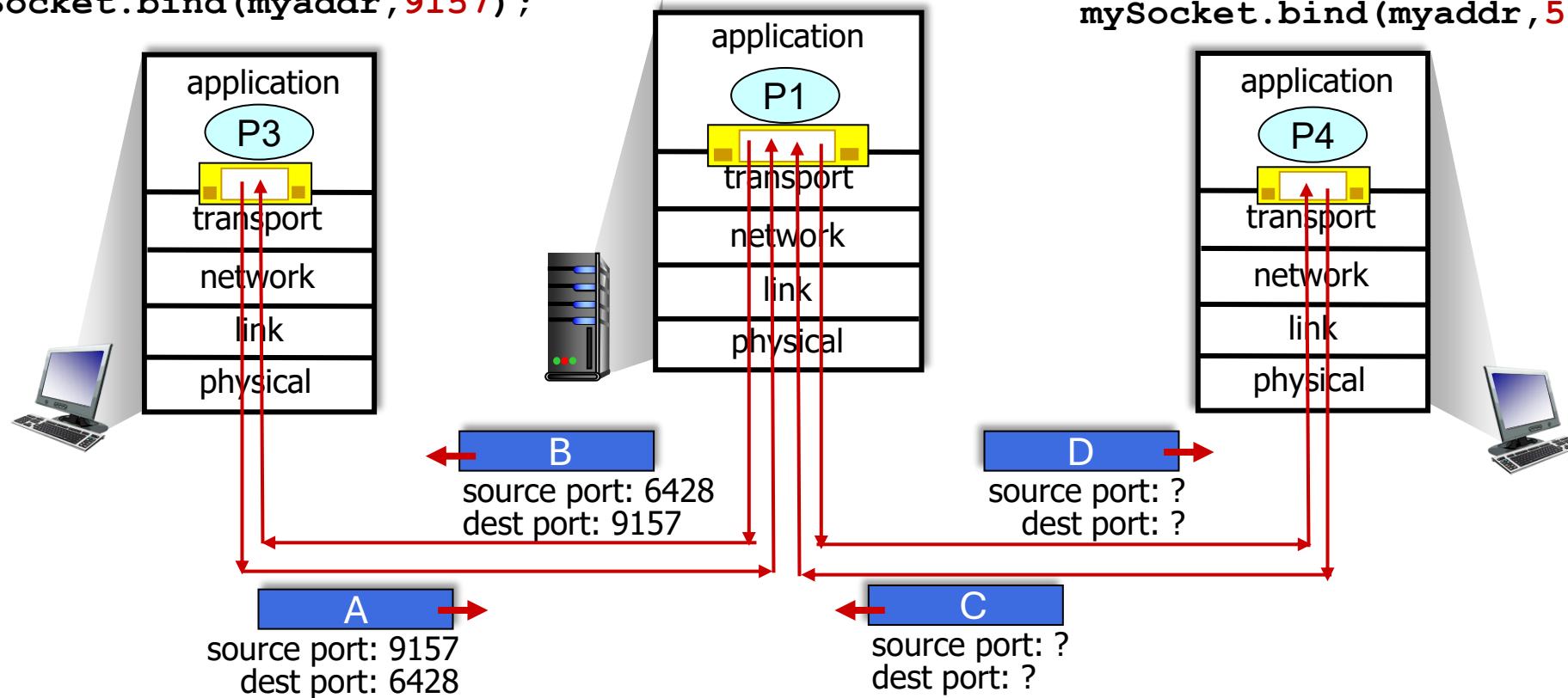


IP/UDP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

Connectionless demultiplexing: an example

```
mySocket =  
    socket(AF_INET, SOCK_DGRAM)  
mySocket.bind(myaddr, 6428);
```

```
mySocket =  
    socket(AF_INET, SOCK_DGRAM)  
mySocket.bind(myaddr, 9157);
```



```
mySocket =  
    socket(AF_INET, SOCK_DGRAM)  
mySocket.bind(myaddr, 5775);
```

UDP demultiplexing uses only the **destination port** to route packets to the correct socket, regardless of source IP/port

CSC 3511 Security and Networking

Week 4, Lecture 1: Transport Layer and TCP

Roadmap

- ***Reliable data transmission***
 - *TCP implementation: TCP connection setup*
 - *TCP implementation: RTT estimation*
 - *TCP implementation: Byte notation, segments, sequence numbers & ACK number*

Transport Layer Protocols

Layer 3 (Network/IP) gave us the ability to send packets anywhere on the Internet

- Abstraction level: Packets individually sent through the network
- Problem: IP offers best-effort delivery
 - Packets can be *lost, corrupted, reordered, delayed, or duplicated*
- Problem: Applications don't want to think about packets and best-effort
 - Programmers want to think in terms of a more convenient abstraction

Layer 4 (Transport) builds extra features on top of Layer 3

- **UDP** (User Datagram Protocol): Adds *de-multiplexing* only
- **TCP** (Transmission Control Protocol): Adds *de-multiplexing* and ***reliability***
- Both protocols provide a more useful abstraction to programmers

Let's check out each of these features

Defining Reliability

We have to deal with **5 problems** from the best-effort service model:

1. Packets can be dropped
2. Packets can be corrupted
3. Packets can be delayed
4. Packets can be duplicated
5. Packets can be reordered

3 different levels of reliability:

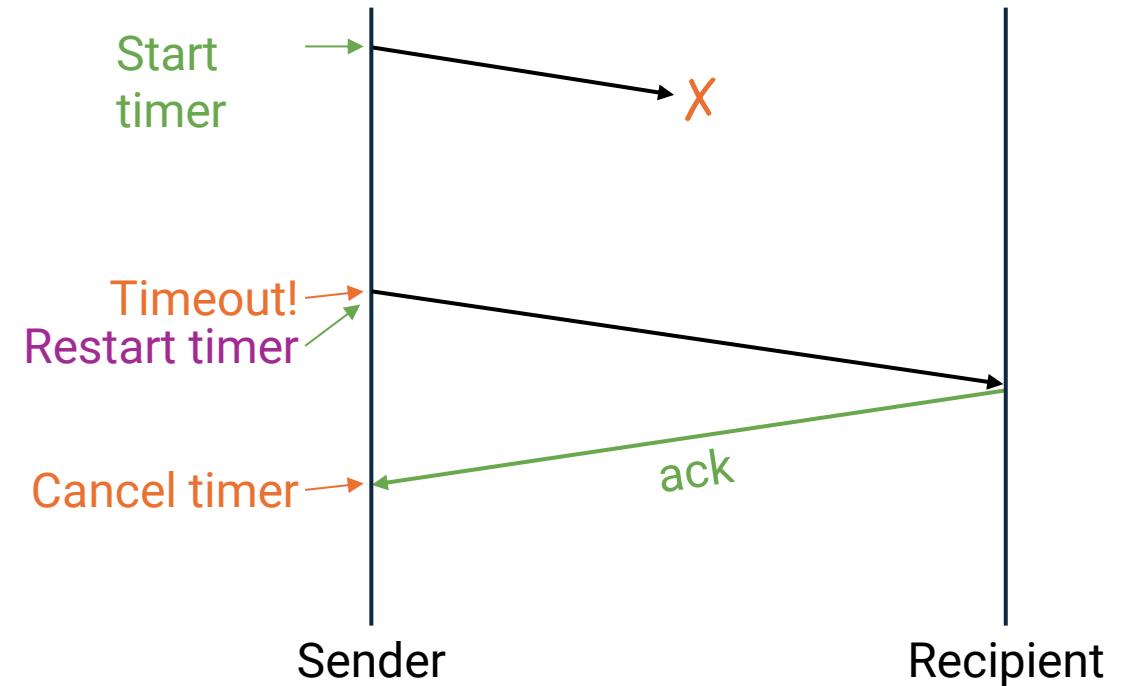
- Best-effort delivery
- *At-least-once* delivery: Every packet arrives, but some might arrive more than once (duplicates)
- *Exactly-once* delivery: Every packet arrives exactly once

Our plan for bridging the abstraction gap:

- At transport layer: Use IP (best-effort delivery) to build ***at-least-once*** delivery
 - Then, get rid of duplicates and pass the result to the application
- This gives the application ***exactly-once* delivery**

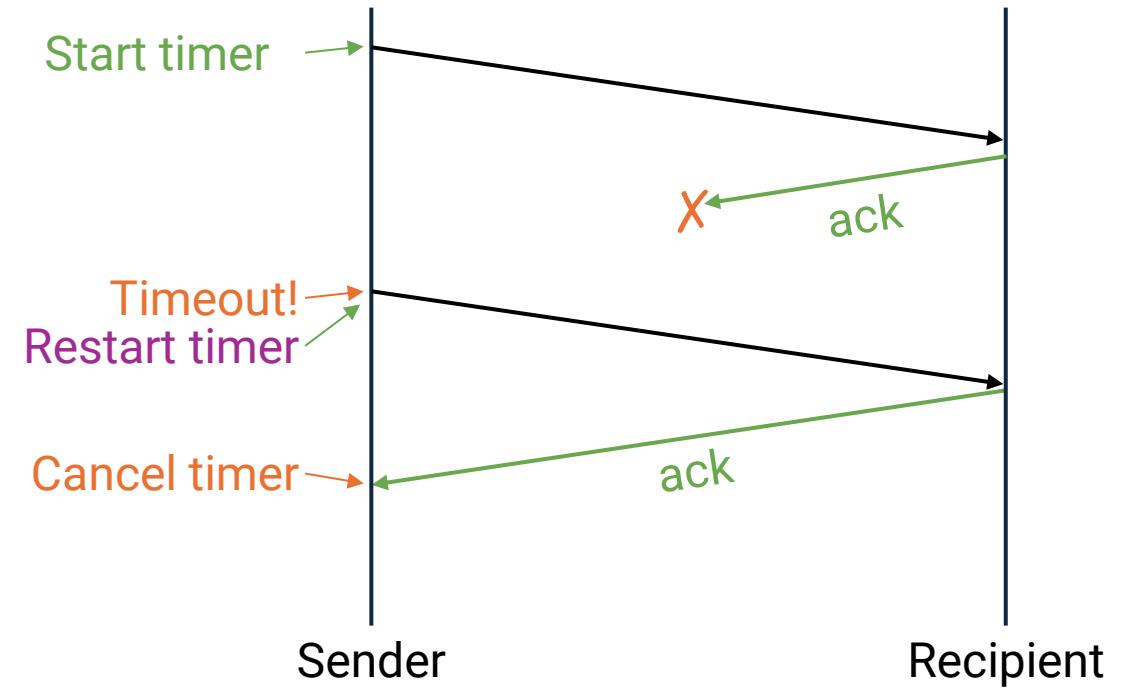
Designing Reliability for a Single Packet

- **Problem 1** - Packets can be dropped:
 - The packet is sent. How does the sender know if it was received successfully?
- **Solution:** The recipient replies with an acknowledgment (**ack**)
 - Set a timer when we send the packet
 - If the timer expires and we don't get the ack, resend the packet
 - When we get the ack, cancel the timer



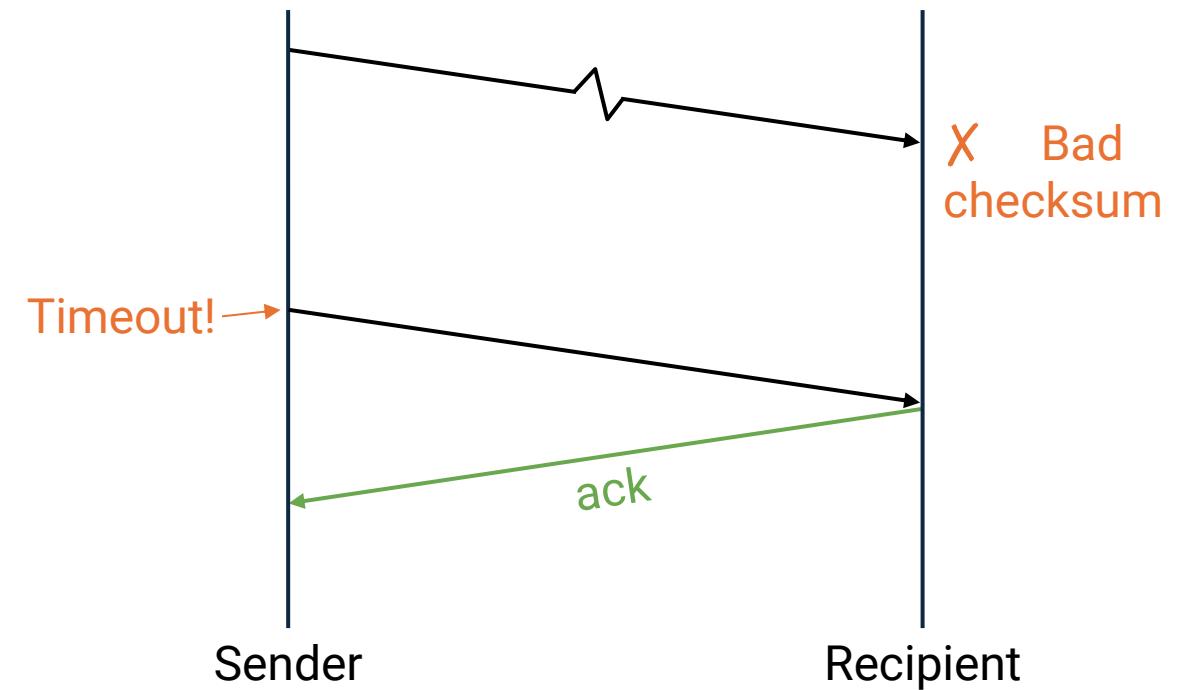
Designing Reliability for a Single Packet

- Problem 1 - Packets can be dropped:
 - What if the **ack** gets dropped
- The timer expires, and the packet gets resent. Repeat until the ack arrives.
- The destination received the packet twice, but that's okay (*at-least-once delivery*).



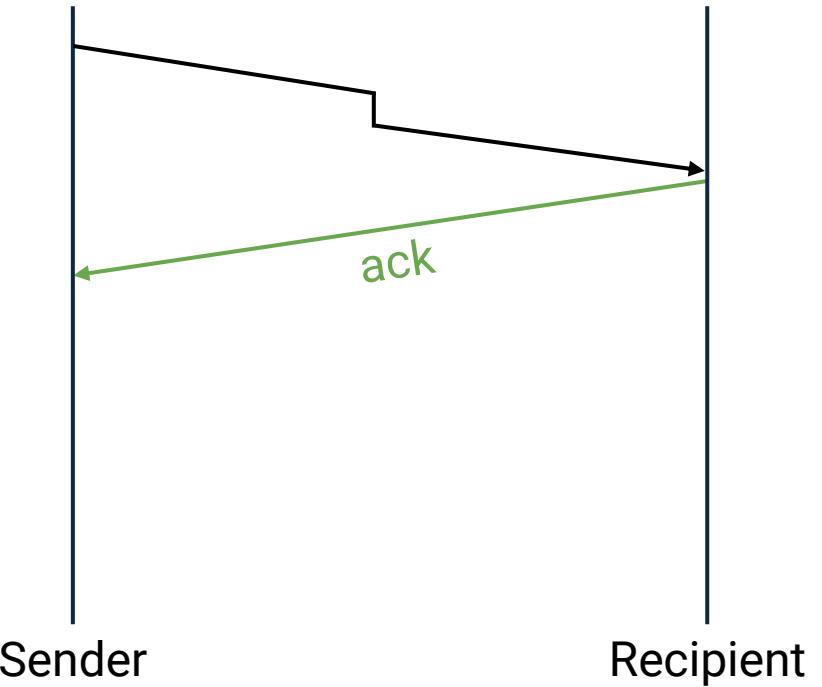
Designing Reliability for a Single Packet

- **Problem 2 -** Packets can be corrupted:
 - The packet is sent. How does the sender know if it was received successfully?
- **Solution:** The recipient calculates the checksum
 - When you get a corrupt packet, ignore it (don't send the ack)
 - Sender times out and resends
 - Alternative solution: When you get a corrupt packet, send a negative acknowledgement (nack)



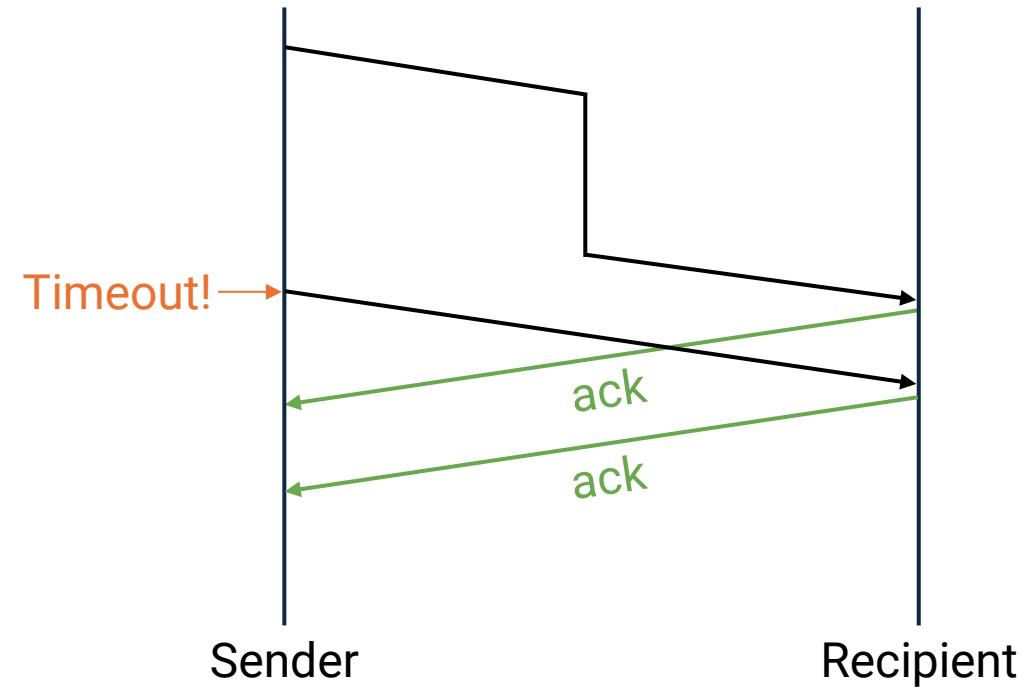
Designing Reliability for a Single Packet

- **Problem 3 -** Packets can be delayed:
 - Minor delay: No problem. The ack still arrives before the timeout.



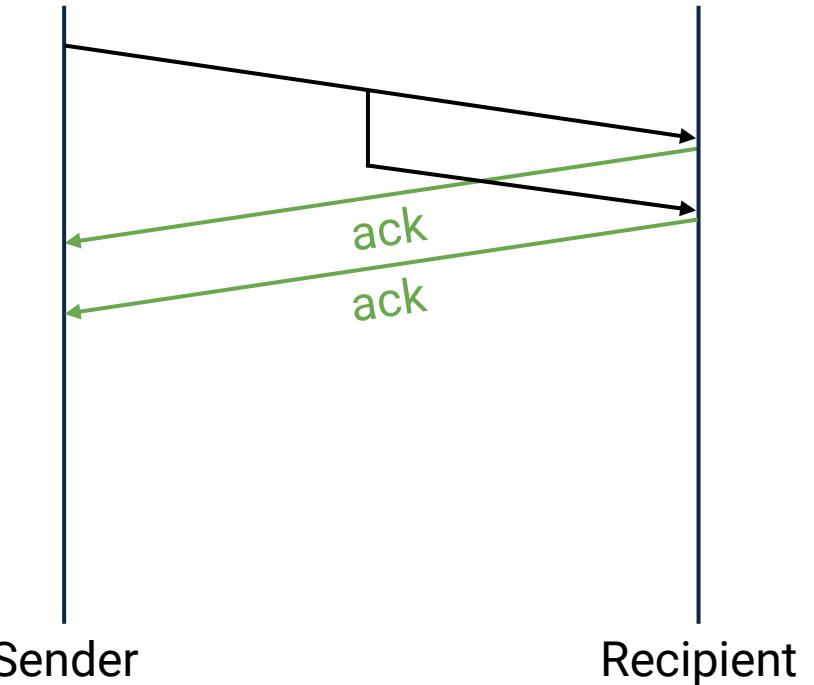
Designing Reliability for a Single Packet

- **Problem 3 -** Packets can be delayed:
 - Longer delay: Sender times out and resends
 - Aka, premature timeout
 - The sender receives two acks. That's fine → The recipient will handle duplicate packets



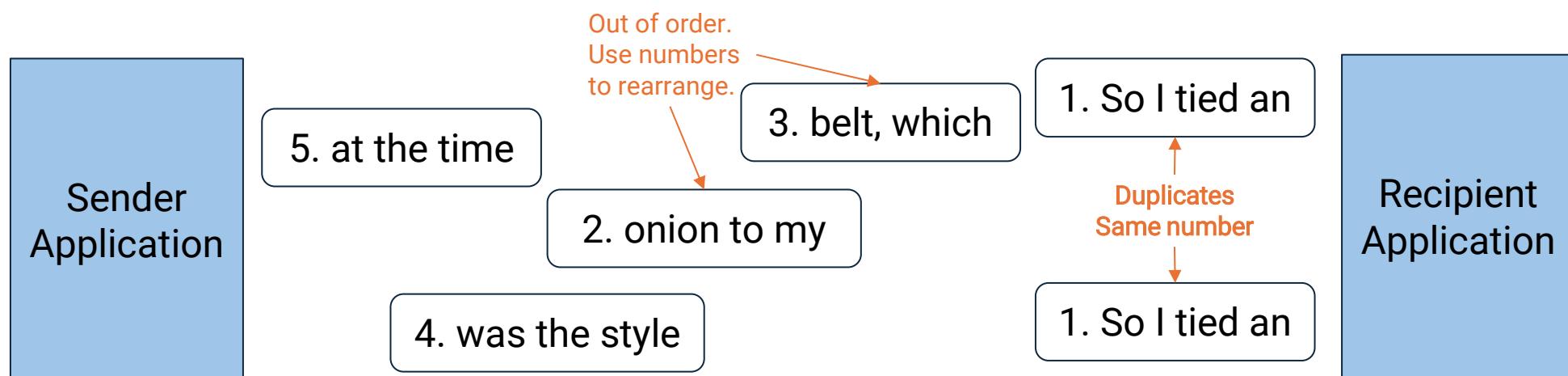
Designing Reliability for a Single Packet

- **Problem 4 -** Packets can be duplicated:
 - The recipient gets the packet twice, and the sender receives two acks
 - Why would the network even duplicate a packet? Usually, because of link-level reliability gone wrong (very rare)
 - The recipient will handle duplicate packets



Designing Reliability for Multiple Packets

- Problem 5 - Packets can be reordered (not relevant in single-packet case):
 - For problem 1 to 4: We can use the single-packet solution repeatedly to support sending multiple packets
 - One extra design component: **sequence numbers**
 - Label each packet with a unique, increasing number
 - Label each ack with which numbered packet is being acked
 - Solve reordering – use numbers to reorder packets
 - **Solve duplication – use numbers to identify duplicate packets**



Roadmap

- *Reliable data transmission*
- ***TCP implementation: TCP connection setup***
- *TCP implementation: RTT estimation*
- *TCP implementation: Byte notation, segments, sequence numbers & ACK number*

TCP State

Reliability requires maintaining state at the end hosts

- **Sender** has to remember:
 - Which packets have been sent and not acked?
 - How much longer on the timer before I resend a packet?
- **Receiver** has to buffer the out-of-order packets

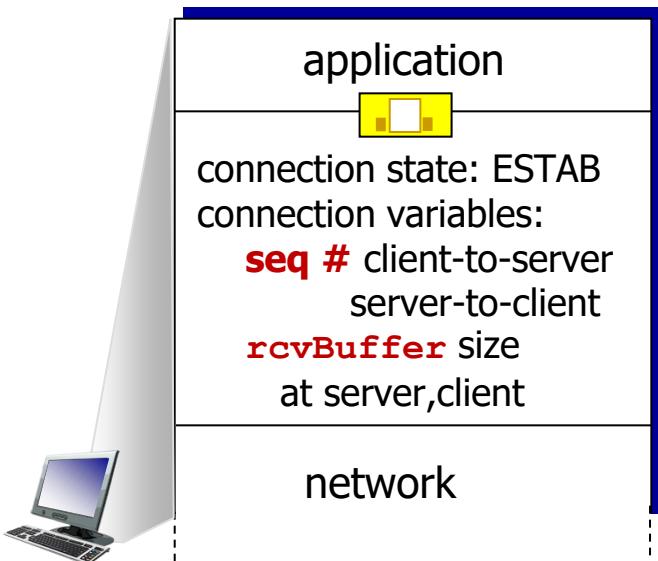
State is maintained at the end hosts (sender and receiver), not in the network

- In each separate connection, both end hosts need to maintain state
- How to build a connection?
 - TCP 3-way handshake

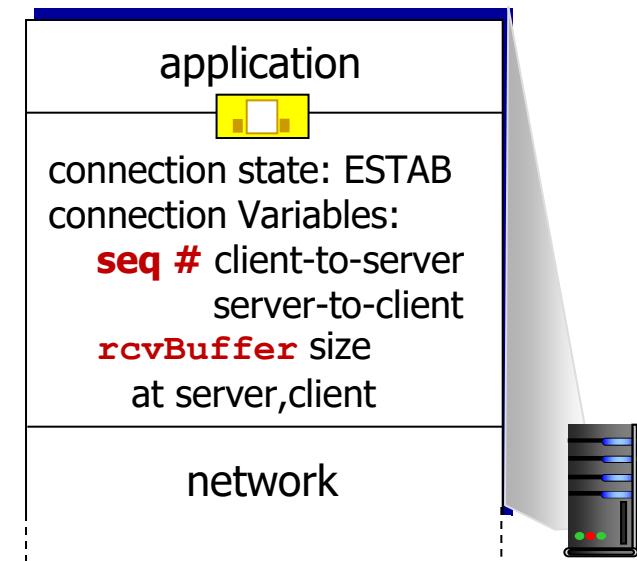
TCP Connection Setup: Three-Way Handshake

Before exchanging data, sender/receiver “handshake”:

- Agree to establish connection (each knowing the other willing to establish connection)
- Agree on connection parameters (e.g., initial sequence number (ISN))



```
Socket clientSocket =  
    newSocket("hostname", "port number");
```



```
Socket connectionSocket =  
    welcomeSocket.accept();
```

TCP Connection Setup: Three-Way Handshake

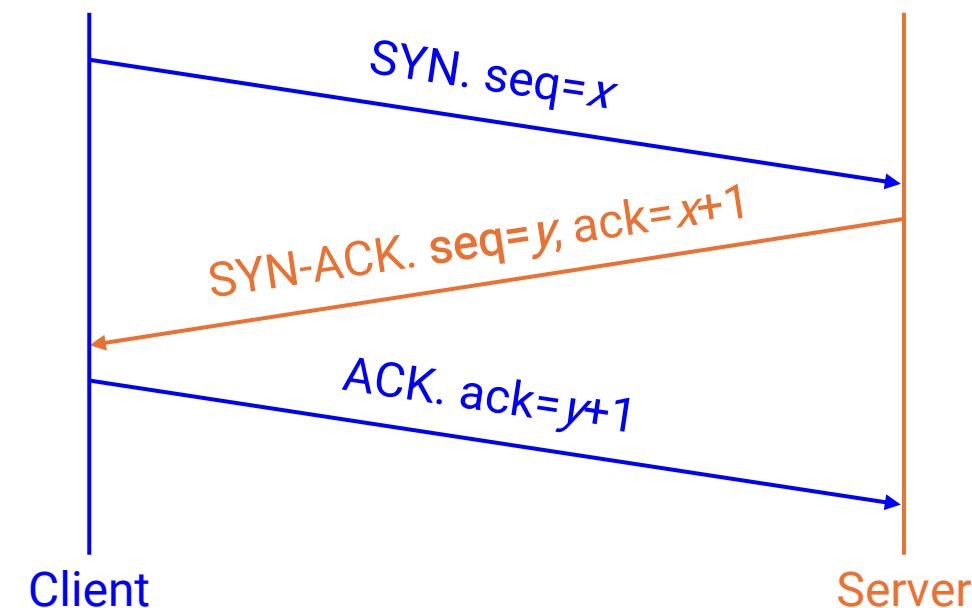
Goal: Each host tells its ISN to the other host

1. **SYN.** Client says: "My ISN is x ."
2. **SYN-ACK.** Server says: "I received x (expecting $x+1$ next). Also, my ISN is y ."

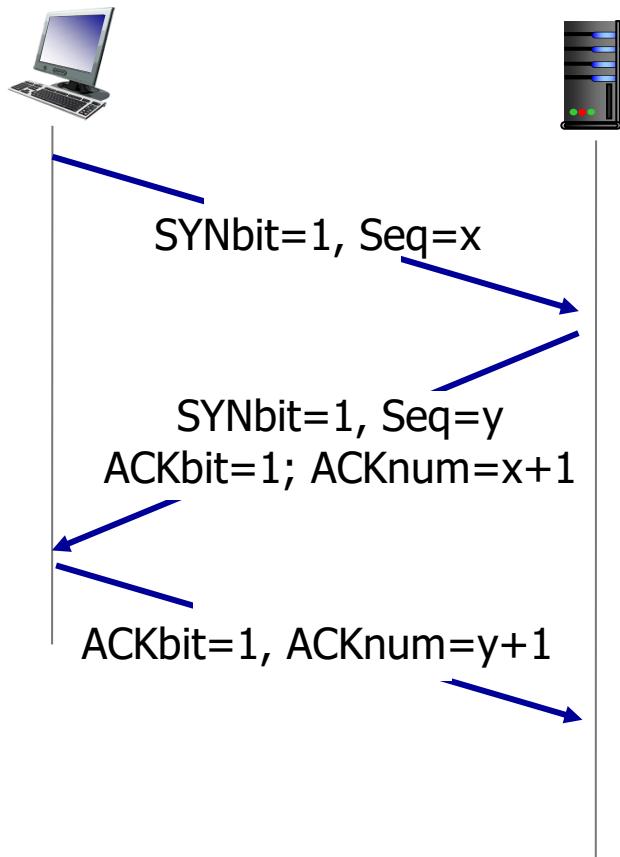
Connections in TCP are full-duplex:

- Both hosts can send data, and both hosts can receive data
 - A can send to B, and B can send to A, simultaneously, in the same connection
1. **ACK.** Client says: "I received y (expecting $y+1$ next)."

After the three-way handshake, both sides can start sending data



TCP Connection Setup: Three-Way Handshake

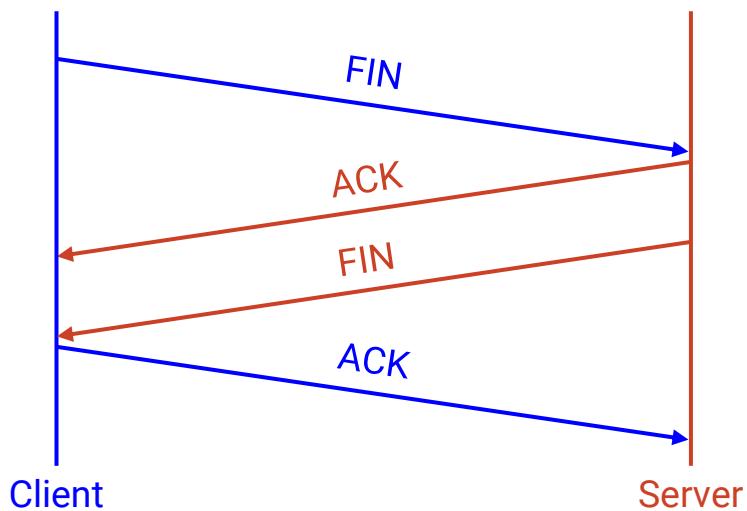


- The final ACK (3rd) acts as a confirmation mechanism to ensure that:
 - Both client and server are synchronized and aware of the connection state
 - No data is transmitted before the connection is fully established, avoiding the acceptance of duplicate data
 - The server only considers the connection fully established once the client sends the final ACK
 - Out-of-sync states are avoided because both sides explicitly confirm that the connection setup is complete

TCP Connection Teardown

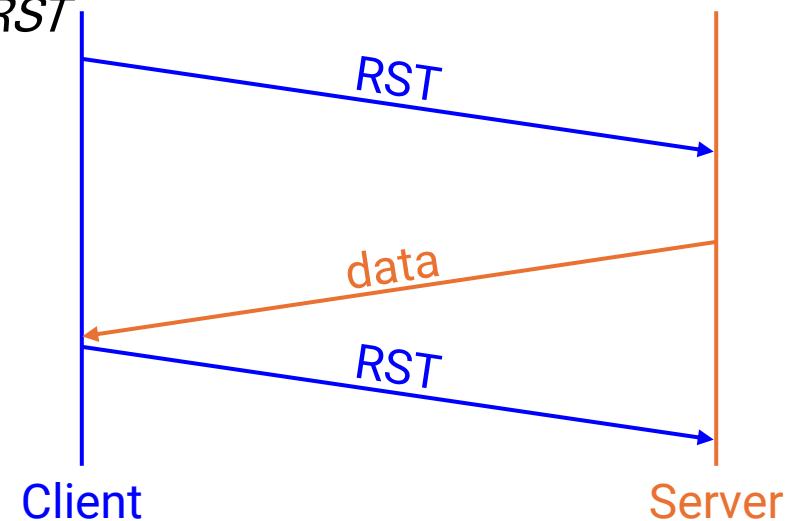
Normal termination:

- Each side sends a FIN packet to say: "I'm done sending, but will keep receiving."
- FIN packets must be acked, just like any other data.
- When only one side has sent FIN, the connection is *half-closed*.
- When both sides have sent FIN (both done sending), the connection is closed.



Just for fun: Abrupt termination can be used instead (e.g. in case of error):

- *Send a RST (reset) to say: "I will no longer send or receive data."*
- *RST packets do not need to be acked*
- *Any data in flight is lost*
- *If the RST sender receives more data later, send another RST*



Roadmap

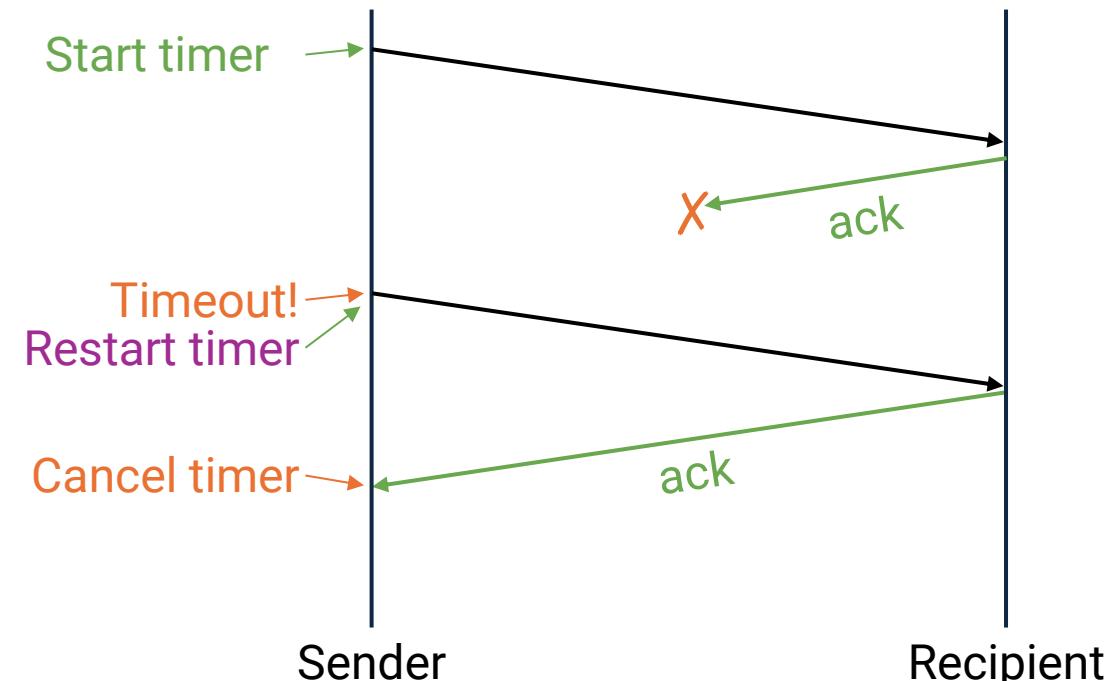
- *Reliable data transmission*
- *TCP implementation: TCP connection setup*
- ***TCP implementation: RTT estimation***
- *TCP implementation: Byte notation, segments, sequence numbers & ACK number*

TCP timeout and round trip time (RTT)

- Problem 1 - Packets can be dropped:
- Solution: Sender sets a timer; The recipient replies with an **ack**
- *How to set TCP timeout value?*
- *Solution: Use RTT (round trip time) to estimate*

Q: how to estimate RTT?

- **Too short:** premature timeout, unnecessary retransmissions
- **Too long:** slow reaction to segment loss



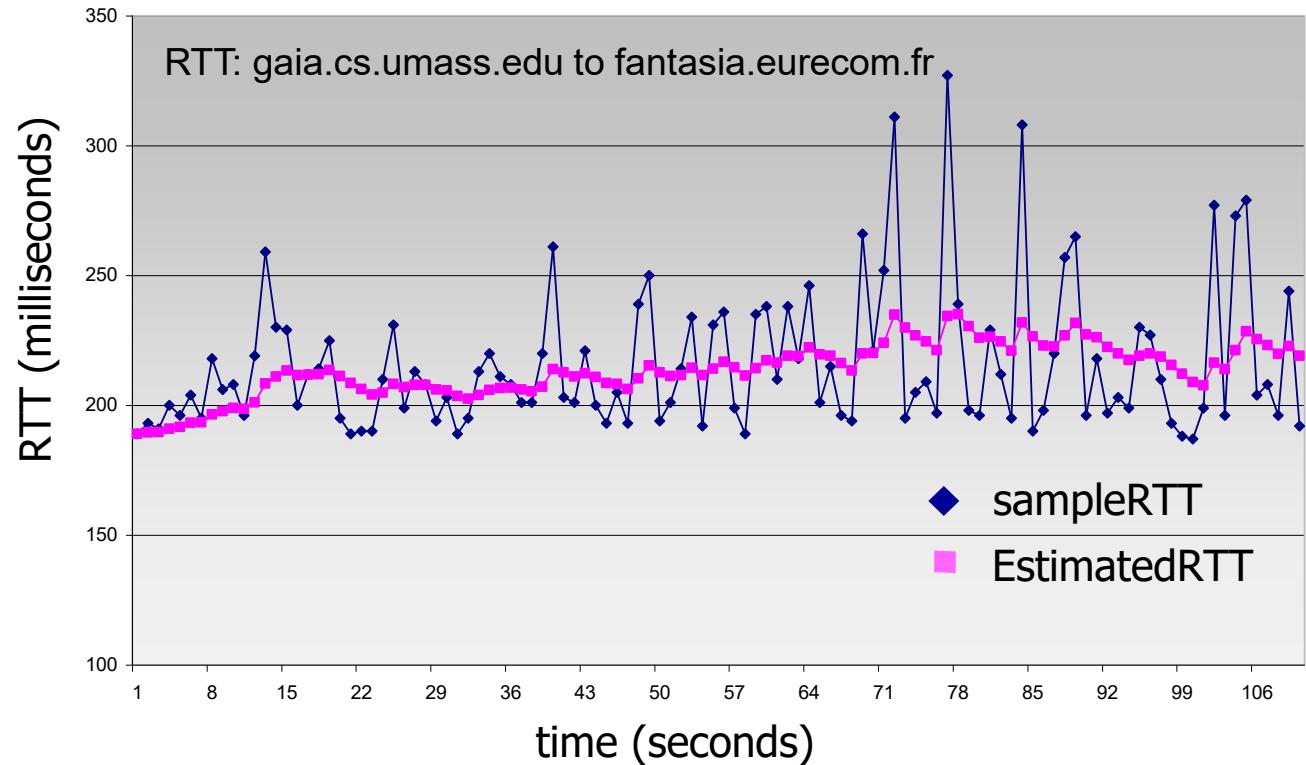
Timeout:

- Waiting time for an ACK
- **Detect lost segments:** the sender relies on the timeout to notice the absence of an ACK
- **Ensure timely retransmission**

TCP timeout and round trip time (RTT)

Q: how to estimate RTT?

- **SampleRTT**: measured time from segment transmission until ACK receipt
 - Ignore retransmissions
- SampleRTT will vary, want estimated RTT “smoother”
 - **Average** several *recent* measurements, not just current SampleRTT



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

- Exponential weighted moving average (EWMA)
- Influence of past sample decreases exponentially fast
- Typical value: $\alpha = 0.125$

TCP timeout and round trip time (RTT)

- Timeout interval: **EstimatedRTT** plus “safety margin”
 - Large variation in **EstimatedRTT**: want a larger safety margin

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT} \text{ (RTT deviation)}$$



estimated RTT

“safety margin”

- **DevRTT**: EWMA of **SampleRTT** deviation from **EstimatedRTT**:

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/TCP_RTT.php

Roadmap

- *Reliable data transmission*
- *TCP implementation: TCP connection setup*
- *TCP implementation: RTT estimation*
- ***TCP implementation: Byte notation, segments, sequence numbers & ACK number***

Notation Change: Bytes vs. Packets

So far, we've used *packets* as the primary unit of data:

- Each packet has a number
- Acks reference packet numbers

TCP is implemented with *bytes* as the primary unit of data

- Each byte has a number → Packets (number) are defined by the number of the **first byte** inside
- ACKs reference byte numbers

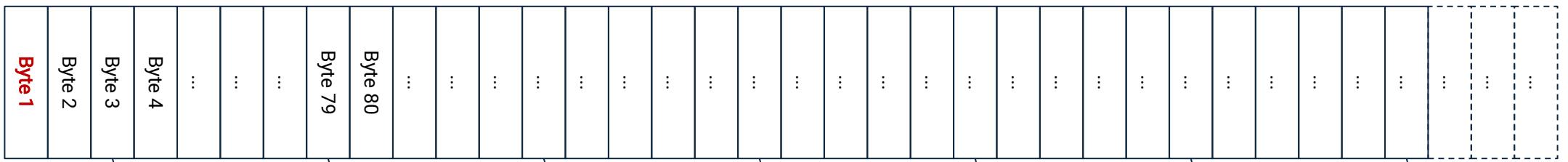
You should be prepared to reason in terms of either.

TCP Segments

TCP provides a reliable, in-order **bytestream**

We have to split this bytestream into packets

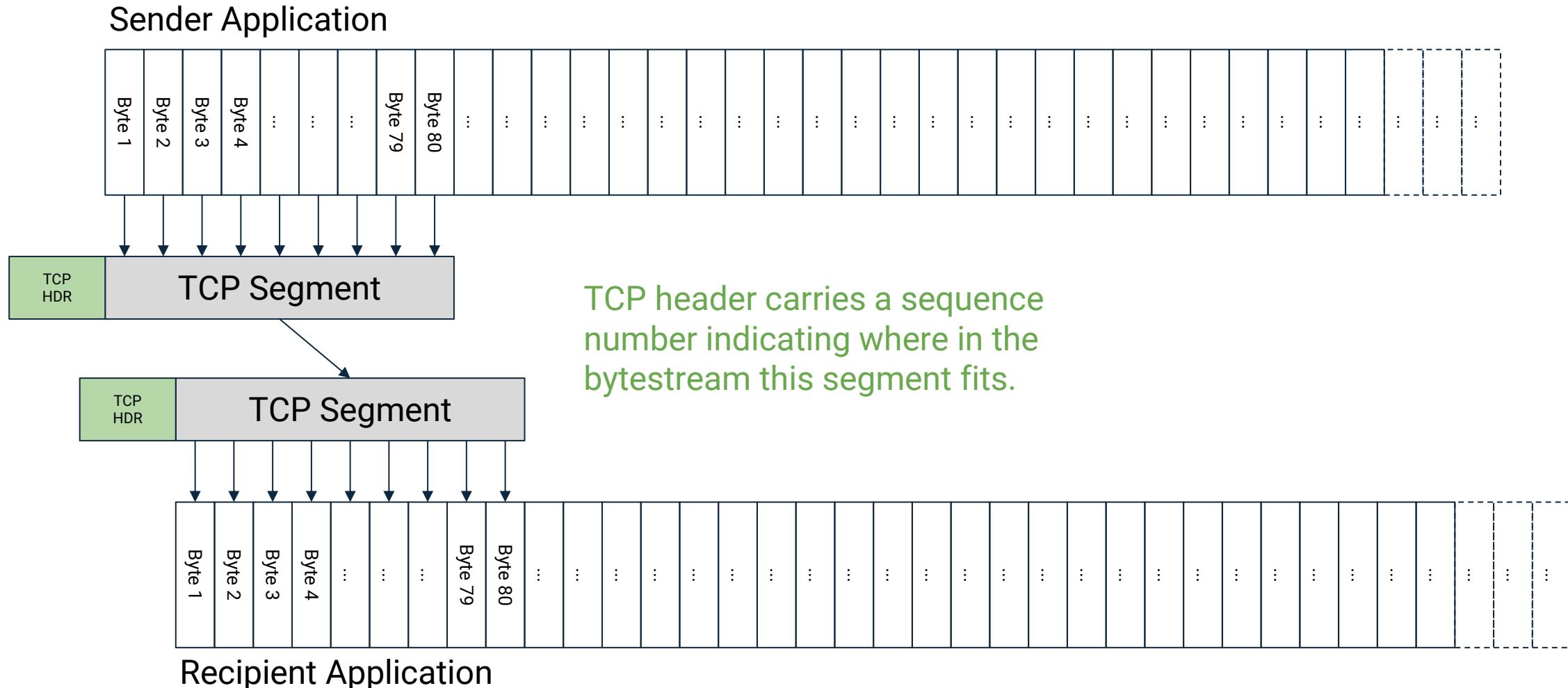
Sender Application



Recipient Application

TCP Segments

A segment is sent when the segment is full (max segment size), or when the segment is not full, but times out waiting for more data

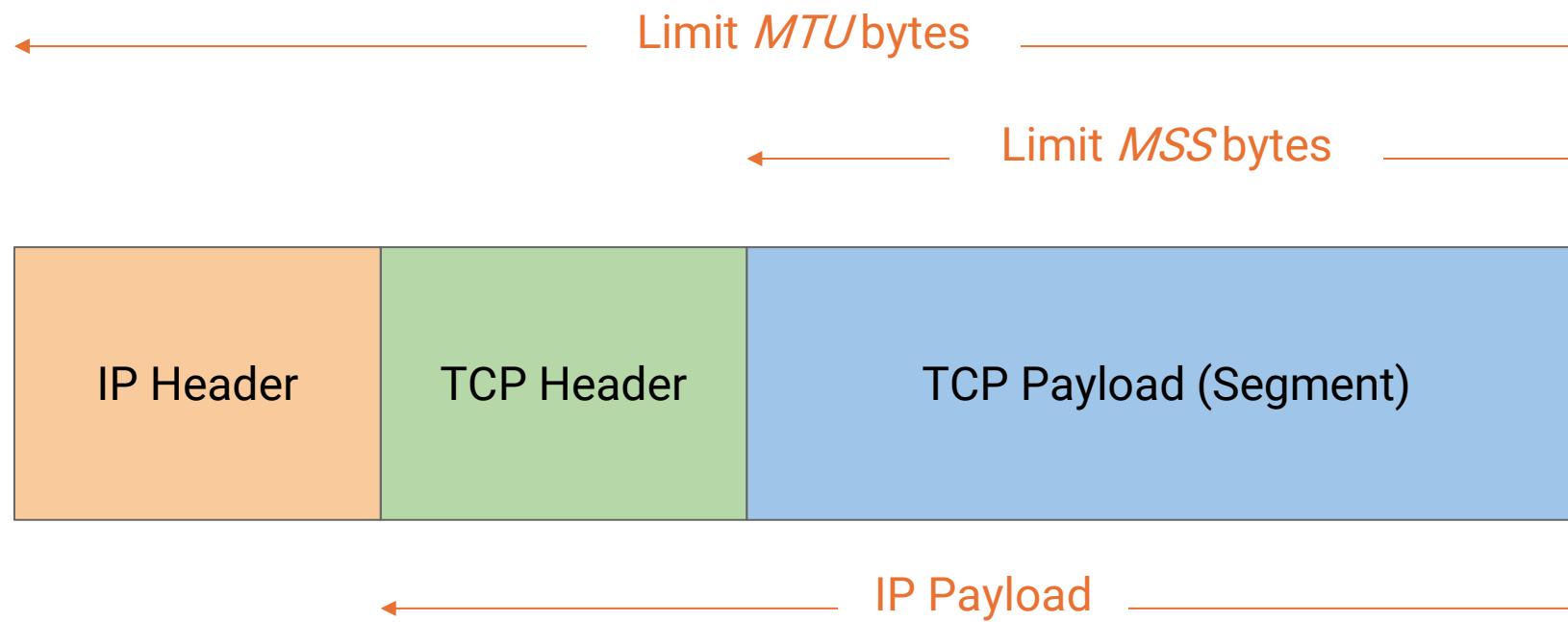


TCP Segments

TCP/IP packet: IP packet with TCP header and TCP data inside

Size limits ([RFC 879 standard](#)):

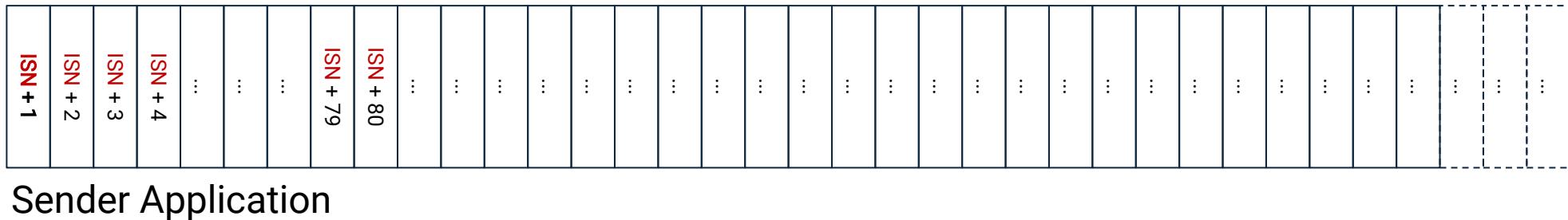
- IP packet: Maximum transmission unit (MTU, e.g., 1500 bytes)
- TCP segment: Maximum segment size (MSS)
- $MSS = MTU - (IP \ header) - (TCP \ header)$



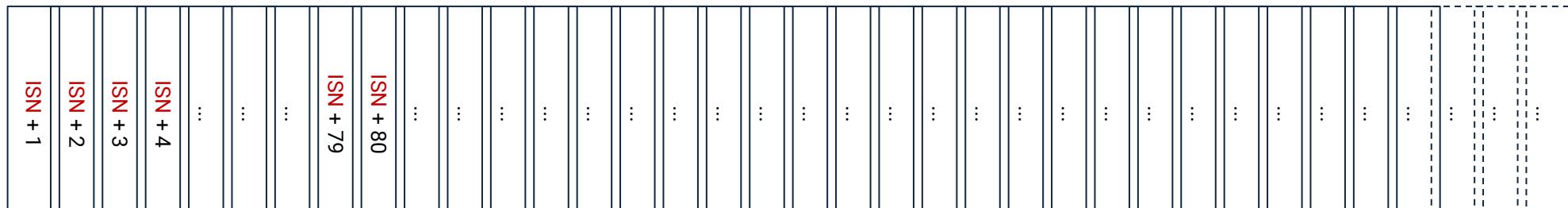
TCP Sequence Numbers

Numbering starts at a **randomly-generated Initial Sequence Number (ISN)**

- First byte is $/SN+1$, then $/SN+2$, etc.
- Starting at a randomly-chosen ISN is very important for security!



Sender Application

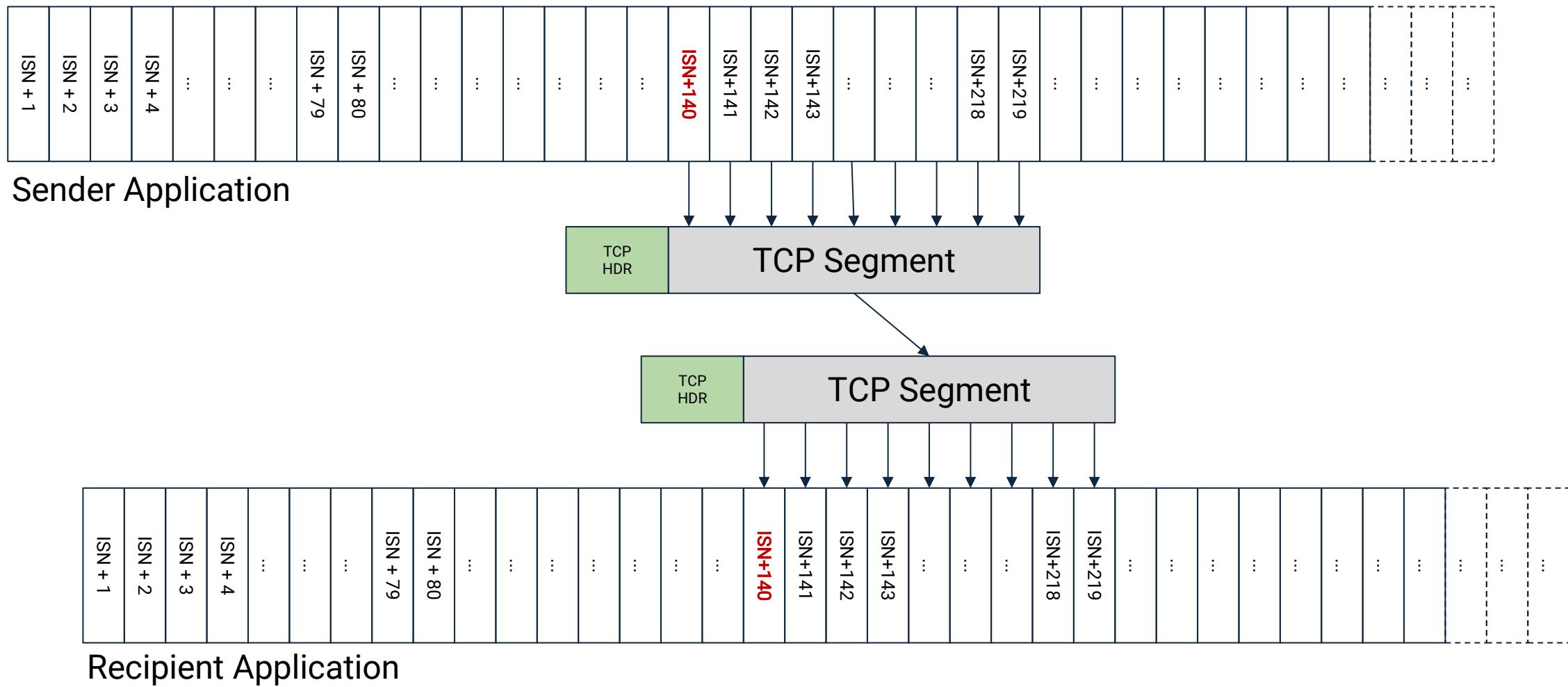


Recipient Application

TCP Sequence Numbers

The ***sequence number*** of a segment is the number of the first byte in the segment

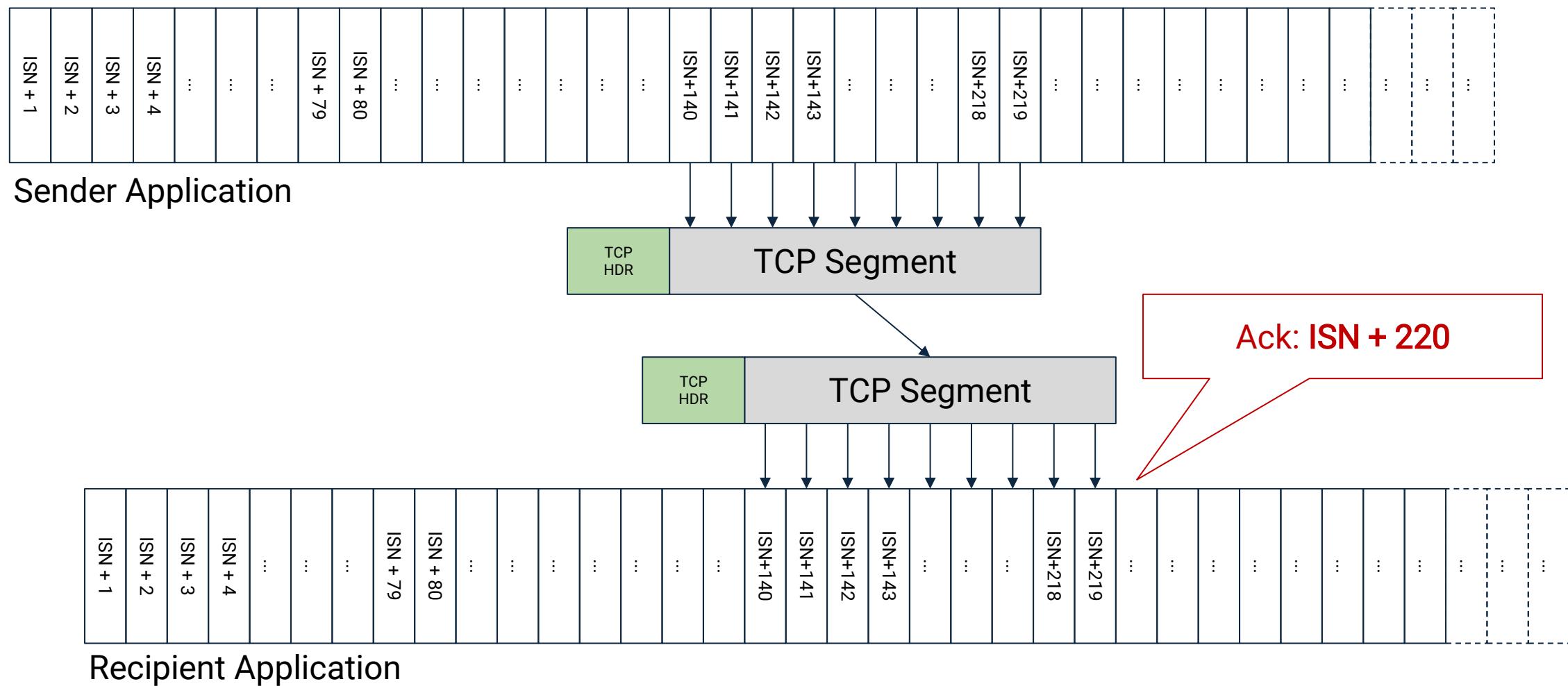
Example: In the segment below, the sequence number is ISN + 140



TCP ACK Numbers

The ack number indicates the next expected byte (i.e. the first unreceived byte)

Example: All bytes up to (and including) ISN + 219 have been received, so the next unreceived byte is **ISN + 220**



TCP Sequence and ACK Numbers

The **sequence number** of a segment is the number of the first byte in the segment

- Sender sends a packet with sequence number j
- The packet contains B bytes
- Bytes in the packet are numbered: $j, j+1, j+2, j+3, \dots, j+B-1$

Recipient sends a **cumulative ack** (number of highest byte received, plus one)

- If all prior data is received, ack number is $j+B$.
- Think of this as the next expected byte, or the first unreceived byte.
- If earlier data before this packet is missing, the ack number will be lower (Retransmit)

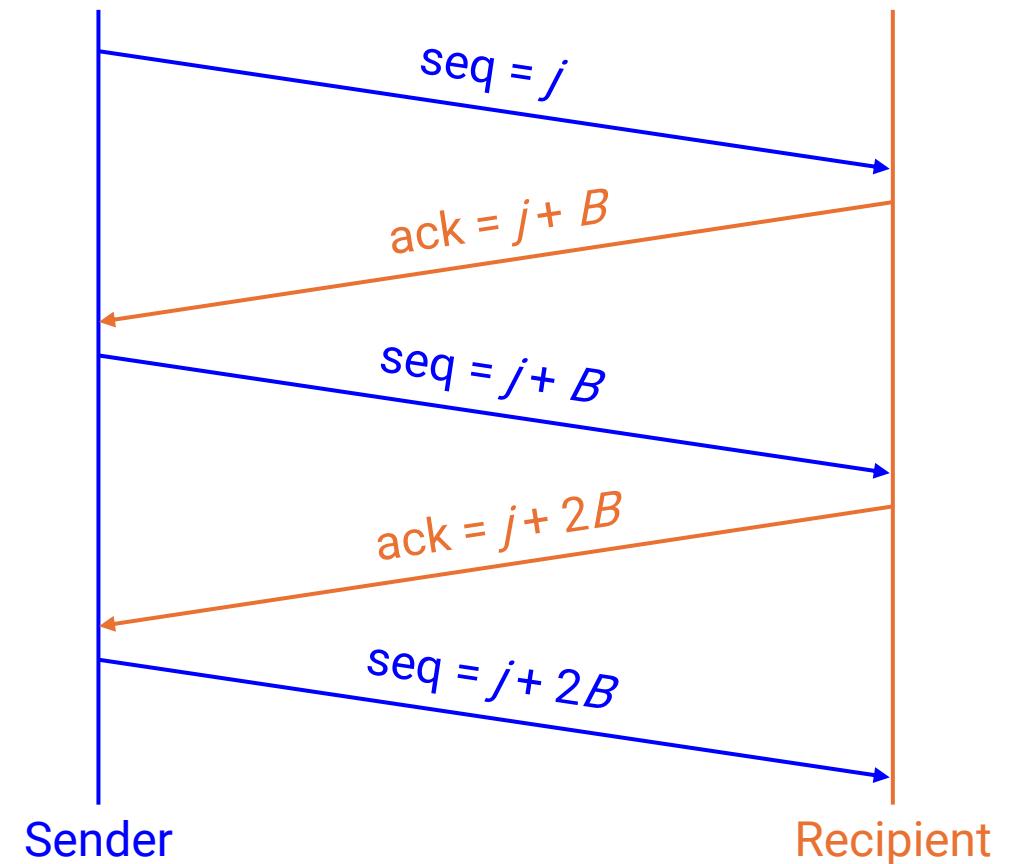
TCP Sequence and ACK Numbers

Assuming only one packet in flight, all packets length B , and no loss:

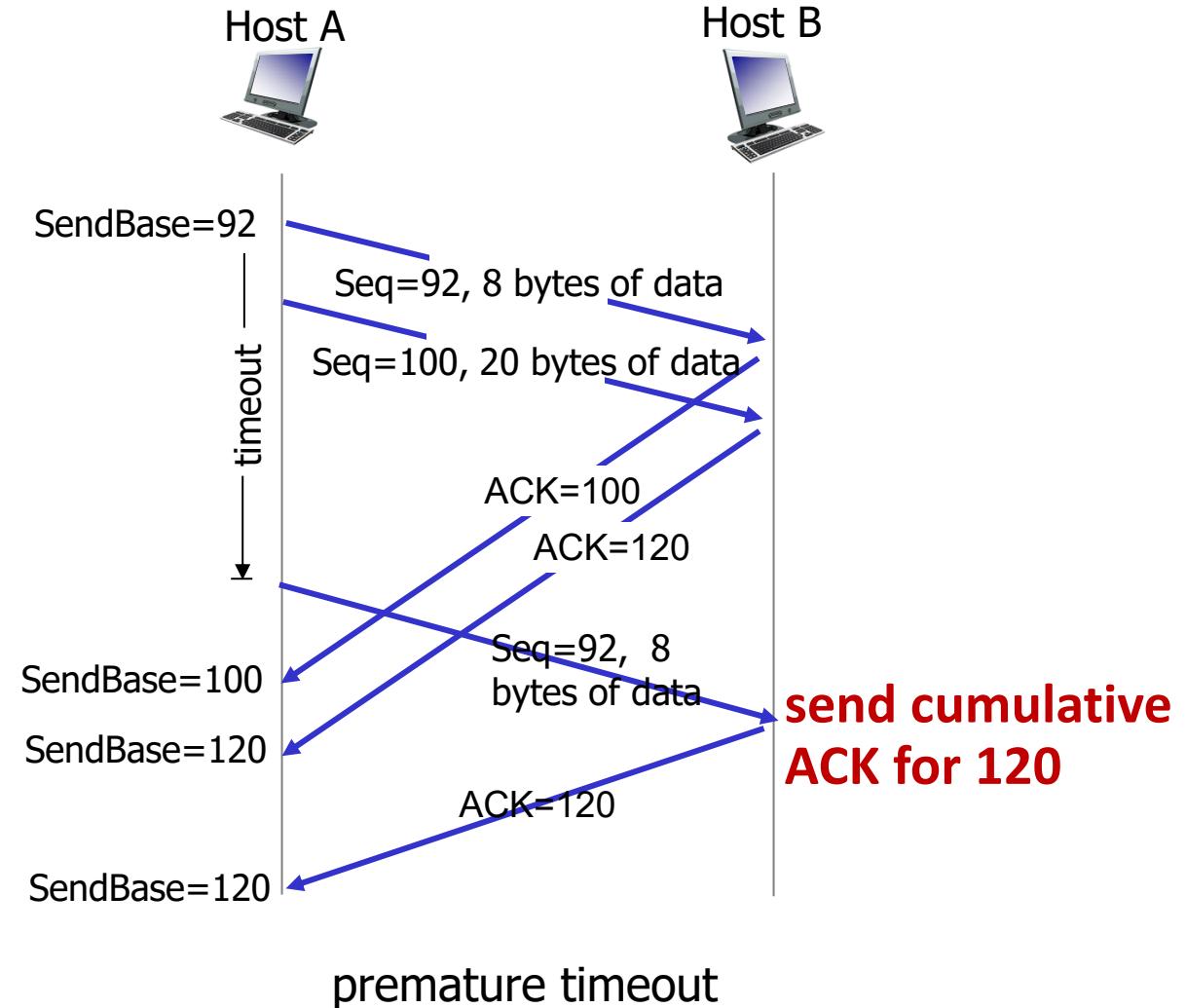
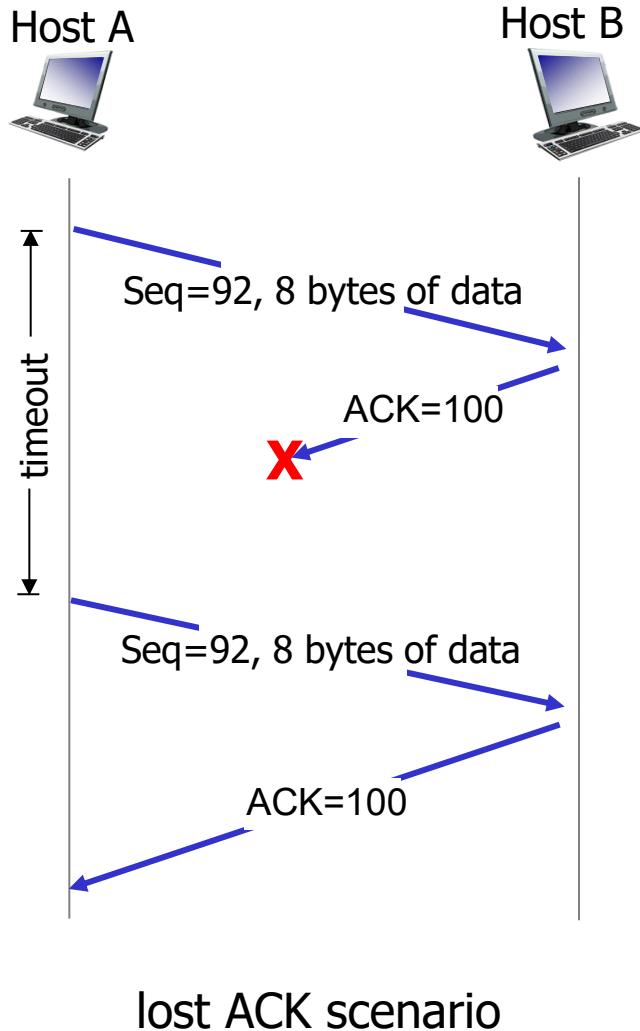
- The last ack number is equal to the next sequence number.
- "I expect $j + B$ next." \rightarrow "I'm sending $j + B$."

Q: What if the packet loss happens?

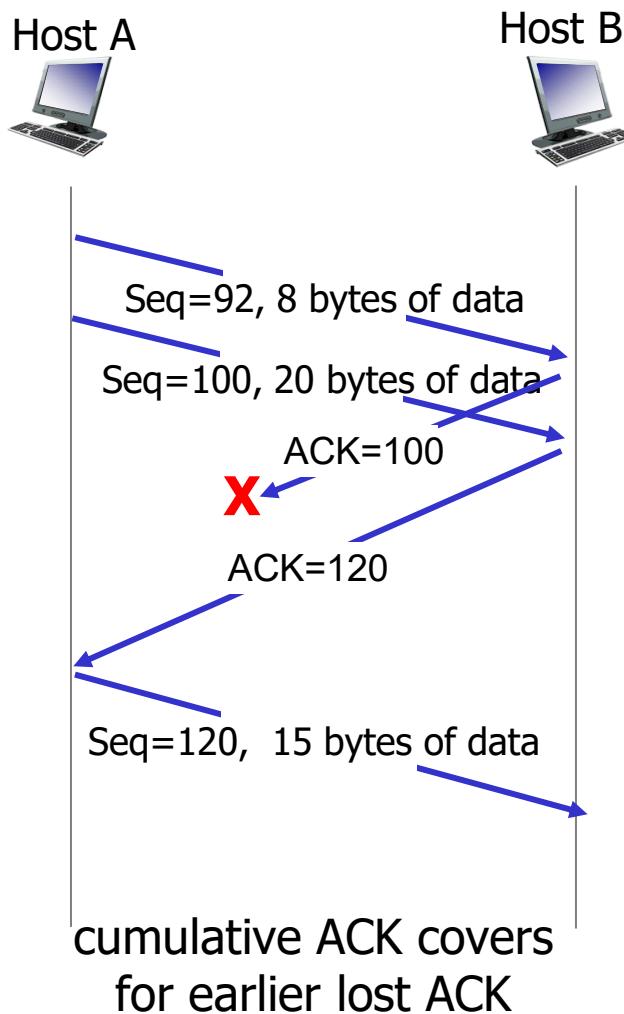
- Sender's packet is lost
- ACK is lost



TCP: Retransmission Scenarios



TCP: Retransmission Scenarios



TCP Fast Retransmit

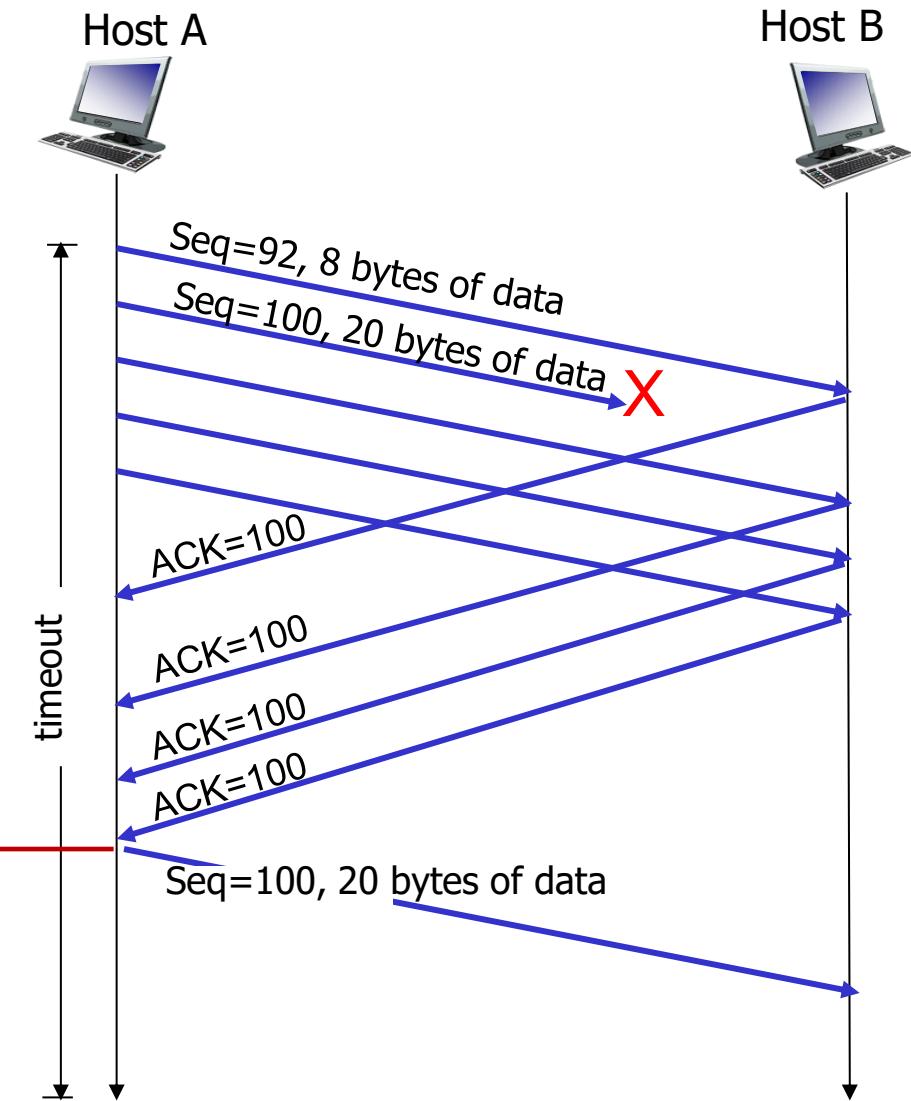
TCP fast retransmit

if sender receives 3 additional ACKs for same data (“triple duplicate ACKs”), resend unACKed segment with smallest seq #

- likely that unACKed segment lost, so don’t wait for timeout



Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!



CSC 3511 Security and Networking

Week 4, Lecture 2: Flow Control and Congestion Control

Roadmap

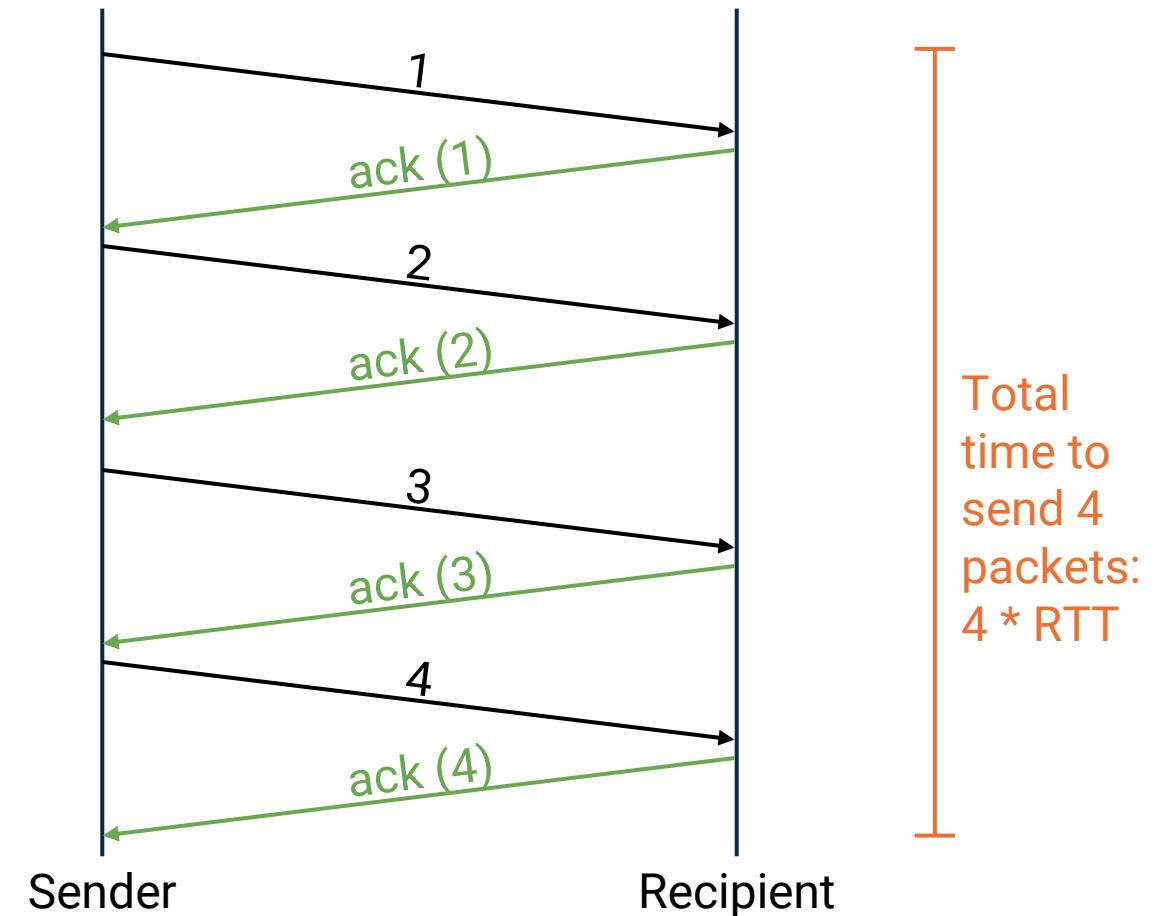
- ***TCP implementation: sliding window***
 - *Flow control*
 - *Congestion control*
 - *TCP header*

Reliably Delivering Multiple Packets

Problem: How to reliably deliver multiple packets (by using ACKs)

Naive approach: Wait for packet i to be acked before sending packet $i+1$.

- This is correct, but really slow
- Throughput: One packet per RTT



Reliably Delivering Multiple Packets

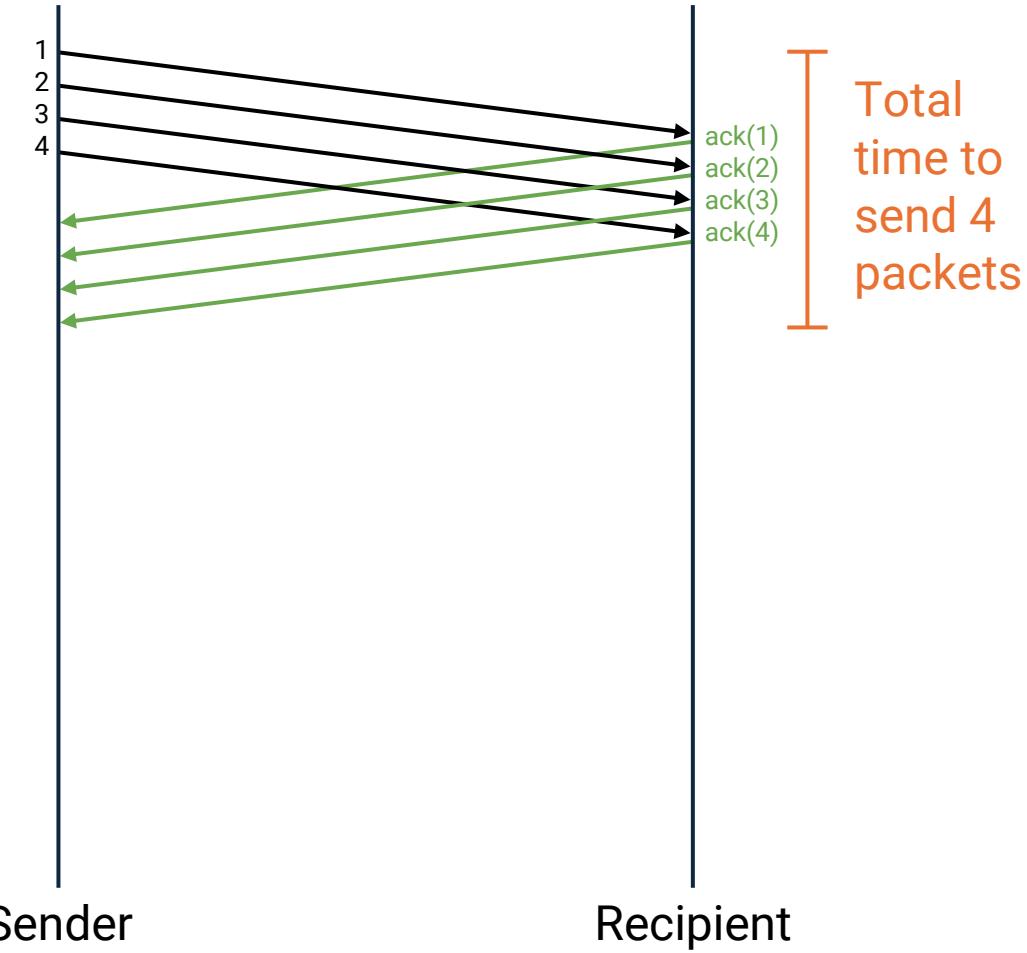
A better approach: Send more packets while waiting for acks

- Have multiple packets **in-flight** (sent, but has not acked) simultaneously

Problem: What is the maximum number of packets can be in-flight at any time?

Answer: Use *window-based algorithm* to limit the number of in-flight packets

- W is the **size of the window**
- To stay inside the window limit:
 - Start: Send W packets
 - Each time a packet gets acked, send the next packet in line



Window-Based Algorithms

Start: Send $W= 4$ packets

When ack(1) arrives: Send 5

When ack(2) arrives: Send 6

In-flight packets: {1, 2, 3, 4}

In-flight packets: {2, 3, 4, 5}

In-flight packets: {3, 4, 5, 6}

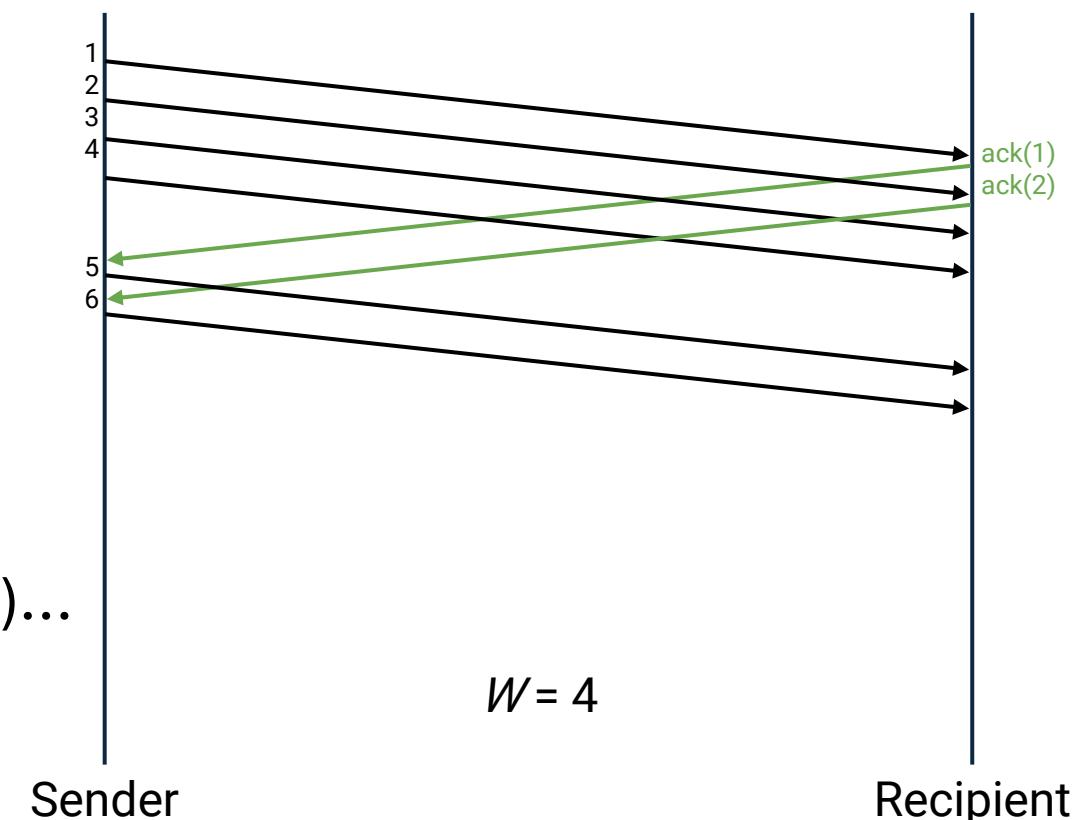
W is the **size of the window**. To stay inside the window limit:

- Start: Send W packets
- Each time a packet gets acked, send the next packet in line

Problem: How big should the window be?

Pick window size W to balance three goals:

1. Take advantage of network capacity ("fill the pipe")...
2. ...but don't overload the recipient (**flow control**)...
3. ...and don't overload links (**congestion control**)



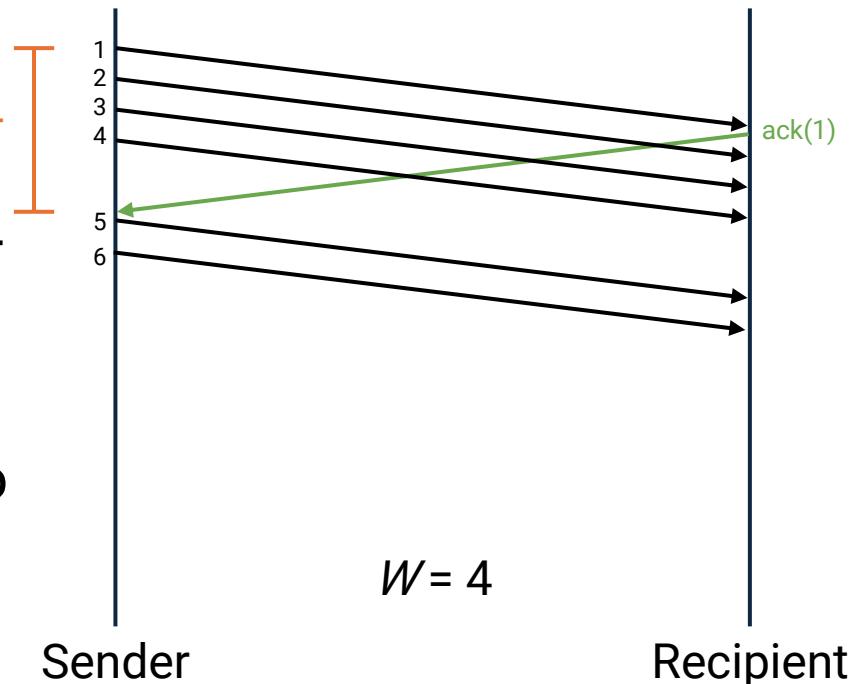
Setting Window Size (1/3): Filling the Pipe

First goal: Take advantage of network capacity ("fill the pipe").

- The sender should never be sitting idle
- Set W large enough to keep the sender constantly busy

How to compute this W ?

- Suppose $RTT = 6$ seconds, and **bandwidth** = 1 packets per second; (*Bandwidth means how fast packets can be sent without clogging the network*)
- Let's focus on the very first RTT: For the first 6 seconds, no acks arrive; We want the sender to be constantly busy
- How many packets can be sent in 5 seconds?
 - $(6 \text{ seconds}) \times (1 \text{ packets per second}) = 6 \text{ packets}$
 - Thus, $W= 6$ packets keeps the sender busy



More generally: $W= RTT \times \text{bandwidth}$

Setting Window Size (1/3): Filling the Pipe

$$W = \text{RTT} \times \text{bandwidth}$$

- The sender is busy the entire RTT sending $W = 6$ packets; ack(1) arrives just as the last of the 6 packets leaves

What exactly is the bandwidth?

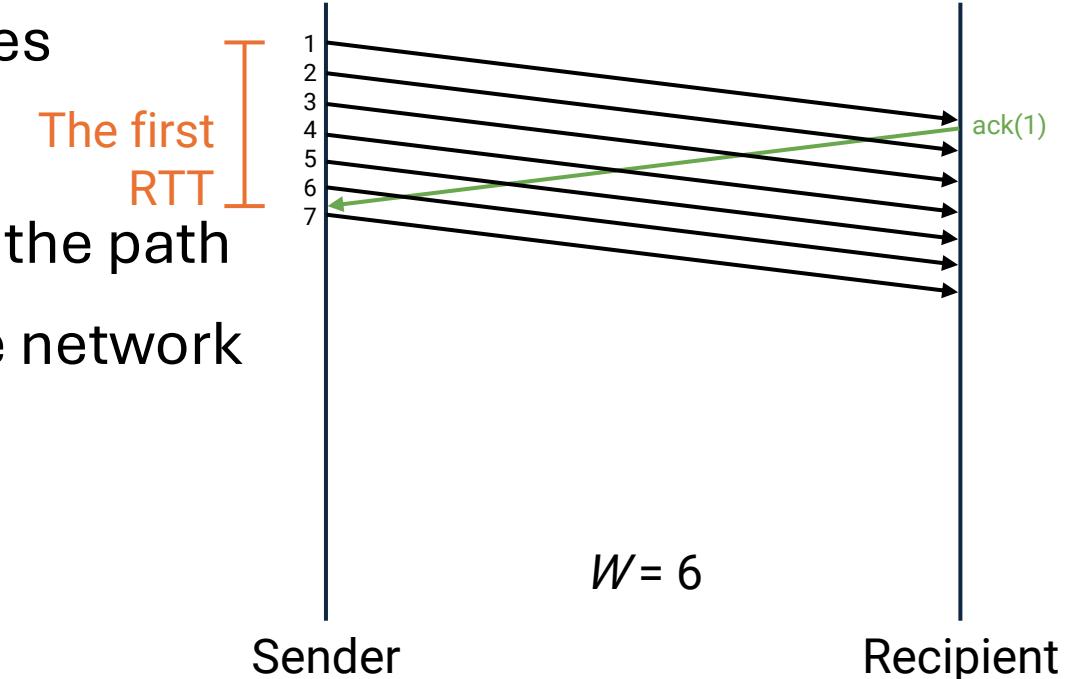
- The minimum ("bottleneck") link bandwidth along the path
- How fast packets can be sent without clogging the network

Rewrite the above formula this in terms of bytes:

- $W \times (\text{packet size}) = \text{RTT} \times \text{bandwidth}$
- $W = \text{window size, in packets}$
- Bandwidth now measured in **bits/second**

This gives us an ***upper bound*** on the desired size of W

- The other two conditions (*don't overload recipient and network*) might impose stricter limits and decrease W .



TCP Sliding Window with Sequence Number

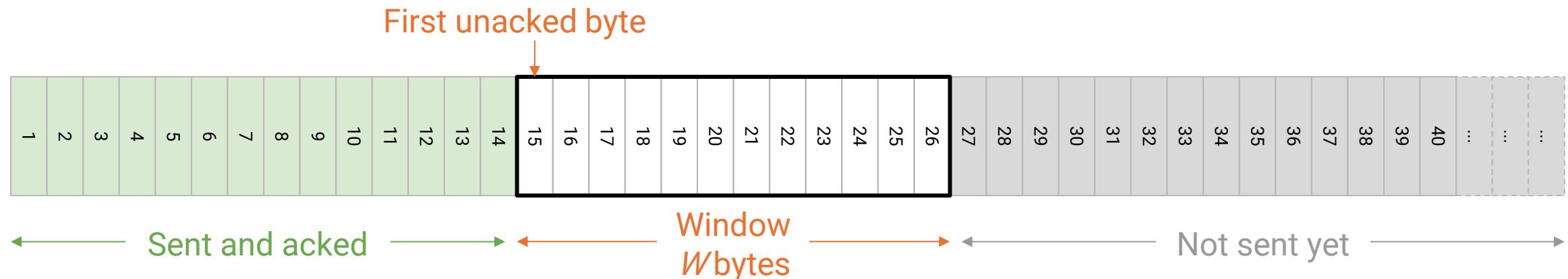
When we measured in packets, W was the maximum number of **packets** in flight

When we measure in bytes, W is the maximum number of *contiguous bytes* in flight

- The window is a range of W contiguous bytes, starting at the **first unacked byte**
- Only these W bytes are allowed to be in flight.

The window slides right if and only if its **leftmost bytes** are acked

- Example: When 15–18 arrive, we can add 27–30



TCP Sliding Window with Sequence Number

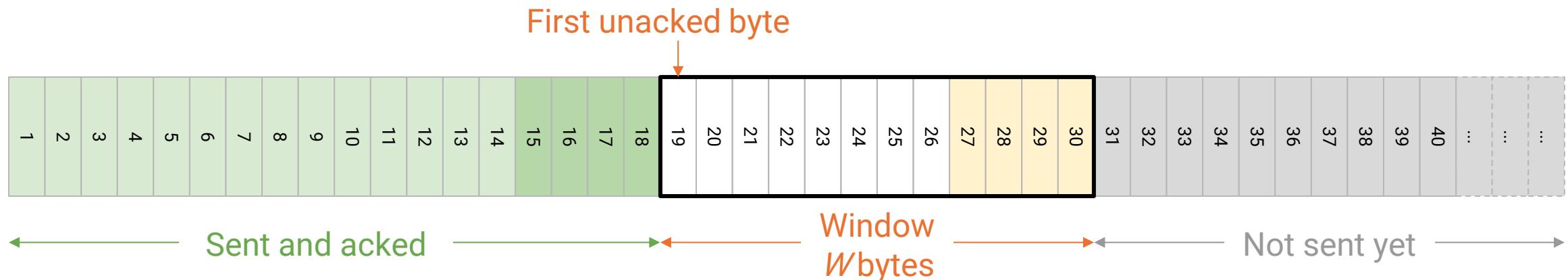
When we measured in packets, W was the maximum number of **packets** in flight

When we measure in bytes, W is the maximum number of *contiguous bytes* in flight

- The window is a range of W contiguous bytes, starting at the **first unacked byte**
- Only these W bytes are allowed to be in flight.

The window slides right if and only if its **leftmost bytes** are acked

- Example: When 15–18 arrive, we can add 27–30



TCP Sliding Window and ACK

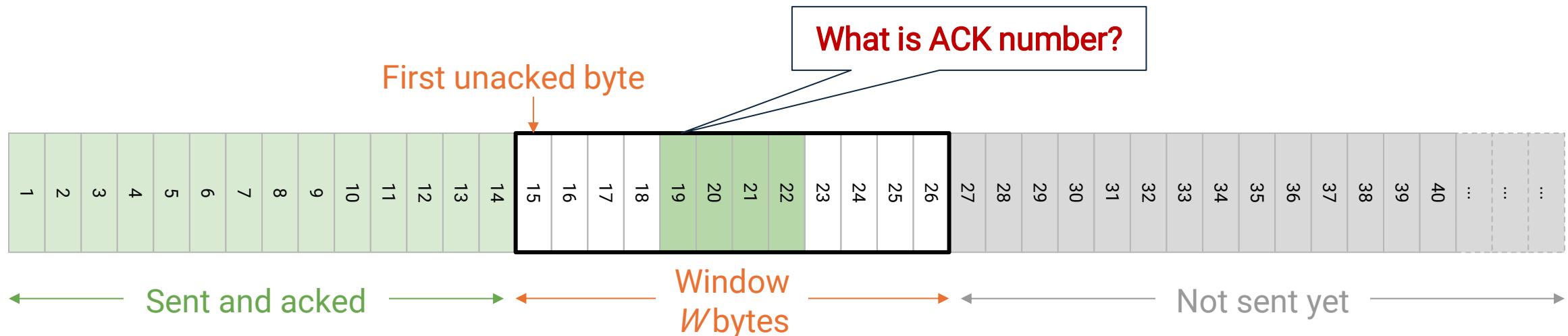
When we measured in packets, W was the maximum number of **packets** in flight

When we measure in bytes, W is the maximum number of *contiguous bytes* in flight

- The window is a range of W contiguous bytes, starting at the **first unacked byte**
- Only these W bytes are allowed to be in flight.

The window slides right if and only if its **leftmost bytes** are acked

- Acking non-leftmost bytes in the window (e.g. 19–22) does not slide the window
- The window is determined by the **first unacked byte** (e.g. still 15)

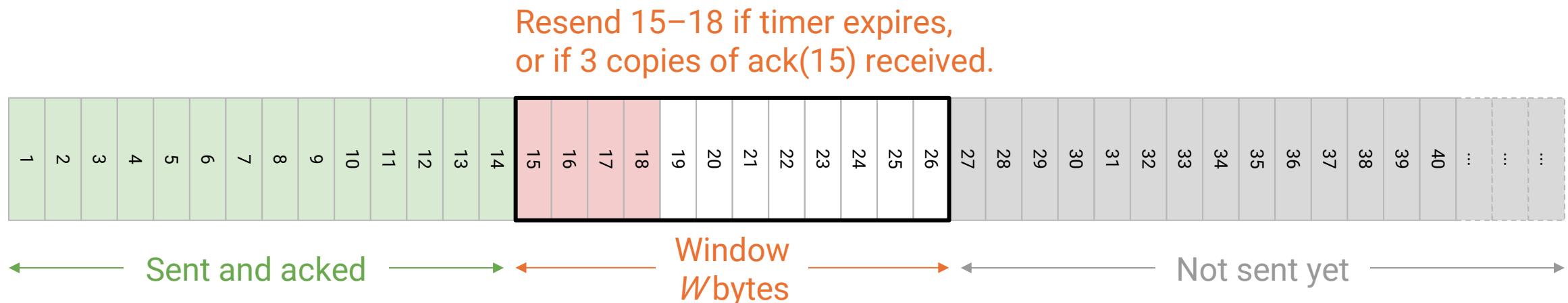


TCP Sliding Window and Loss Detection

Two ways to detect loss and resend. We resend if either condition is true.

In both cases, we always **resend the first unacked packet** (leftmost part of window)

1. Ack-based: If 3 duplicate acks are received, resend.
2. Timer-based: Keep a single timer. If the timer expires, resend.
 - Set timer by estimating RTT



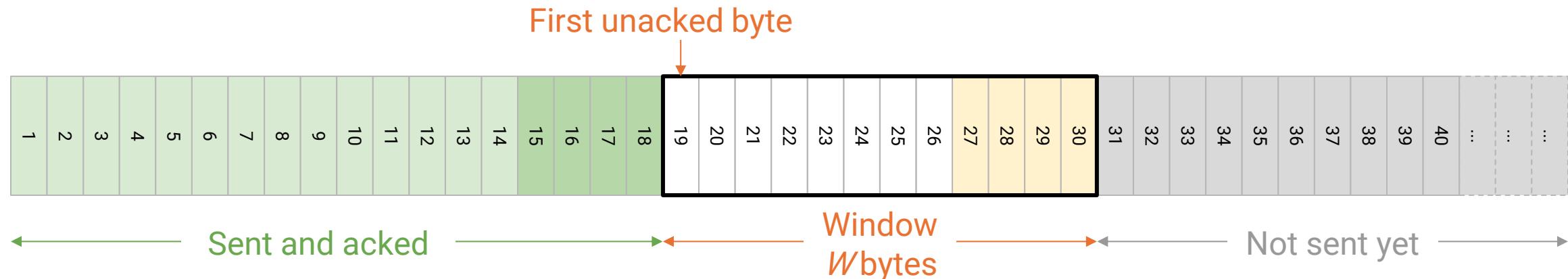
TCP Sliding Window with Additional Restrictions

TCP uses cumulative acks and a sliding window

- Cumulative ack: "I have received everything up to (not including) 15."
- Thus, first unacked byte is 15, and the sliding window is $[15, 15 + W - 1]$.

In practice, W is set as the **minimum of two values**:

- Advertised window ($rwnd$, recipient reports their remaining buffer space)
- Congestion window ($cwnd$, sender magically finds value to avoid network overload)



Roadmap

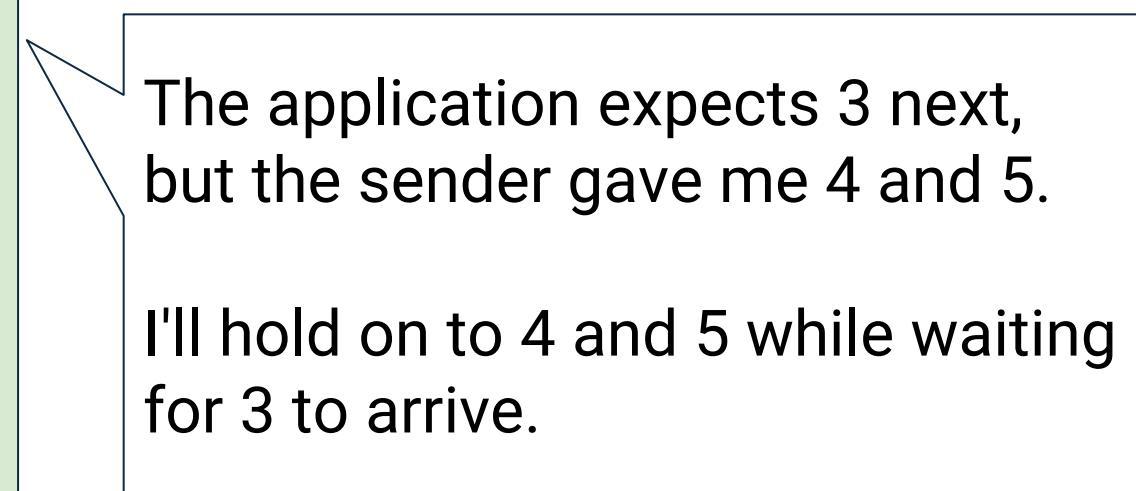
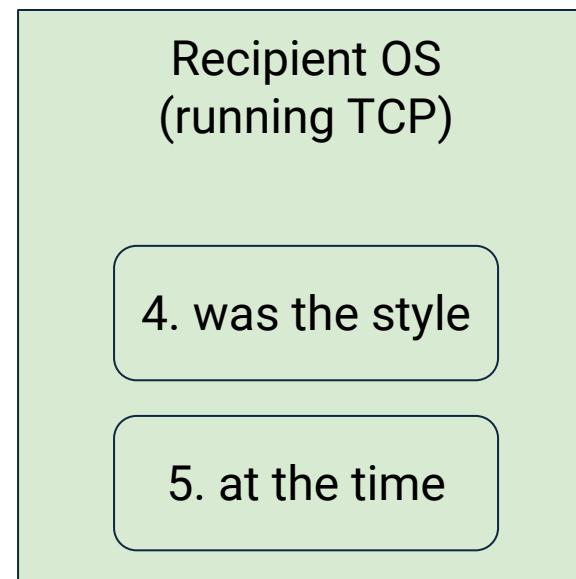
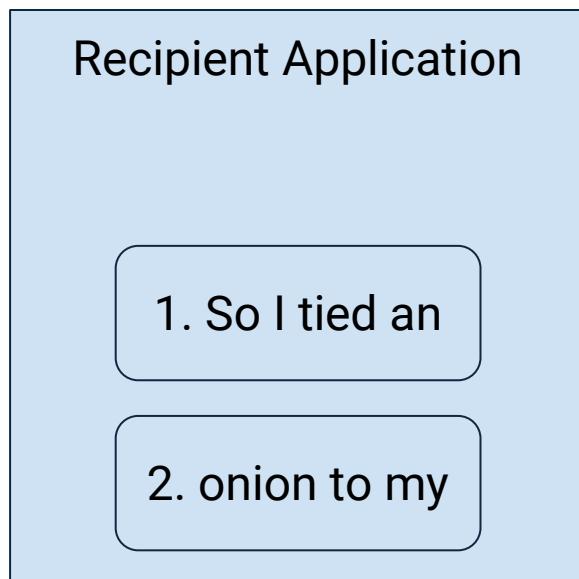
- *TCP implementation: sliding window*
- **Flow control**
- *Congestion control*
- *TCP header*

Setting Window Size (2/3): Don't Overload Recipient (Flow Control)

Consider the recipient:

- TCP might receive packets out-of-order, but can only deliver packets to the **application** in order
- The recipient must **buffer** incoming packets that are out of order
- Packets stay in the buffer until all "missing" packets arrive

We have to make sure the recipient doesn't run out of buffer space



Setting Window Size (2/3): Don't Overload Recipient (Flow Control)

Flow control ensures the recipient has enough buffer space for out-of-order packets

- **Recipient** tells the sender how much space it has left
- The size of remaining space is called the *advertised window*
- This value is carried in **ACK** messages (*rwnd* field in TCP header)
- Sender adjusts its window accordingly
- Number of packets in flight must be less than the recipient's advertised window

Recipient Application

1. So I tied an

2. onion to my

Recipient OS
(running TCP)

4. was the style

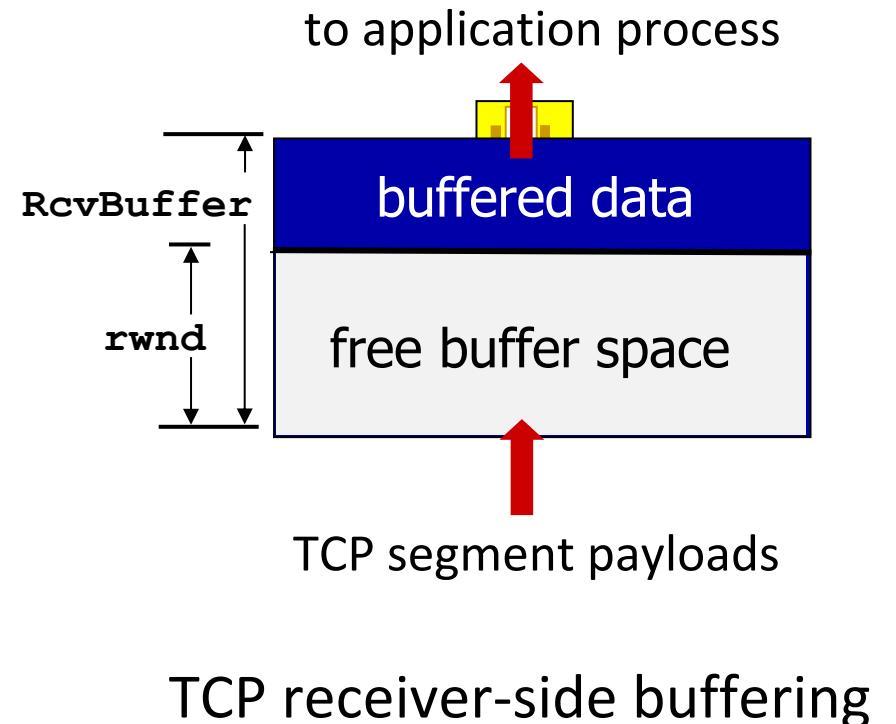
5. at the time

The application expects 3 next,
but the sender gave me 4 and 5.

I'll hold on to 4 and 5 while waiting
for 3 to arrive.

Setting Window Size (2/3): Don't Overload Recipient (Flow Control)

- Recipient “advertises” free buffer space in **rwnd** field in TCP header
 - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
 - Many operating systems auto-adjust **RcvBuffer**
- Sender limits amount of unACKed (“in-flight”) data to received **rwnd**
- Guarantees receive buffer will not overflow



flow control —

Receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

Roadmap

- *TCP implementation: sliding window*
- *Flow control*
- ***Congestion control***
- *TCP header*

Setting Window Size (3/3): Don't Overload Network (Congestion Control)

Previously, we set W to fully consume the bottleneck link bandwidth.

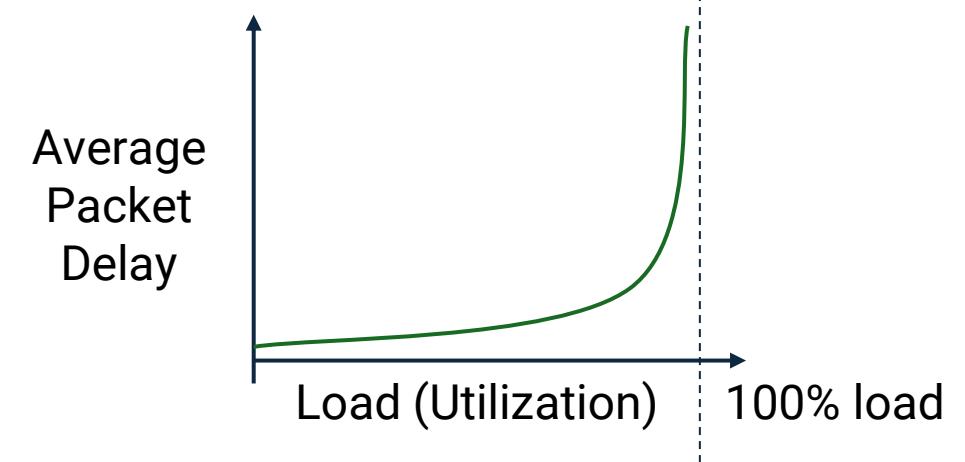
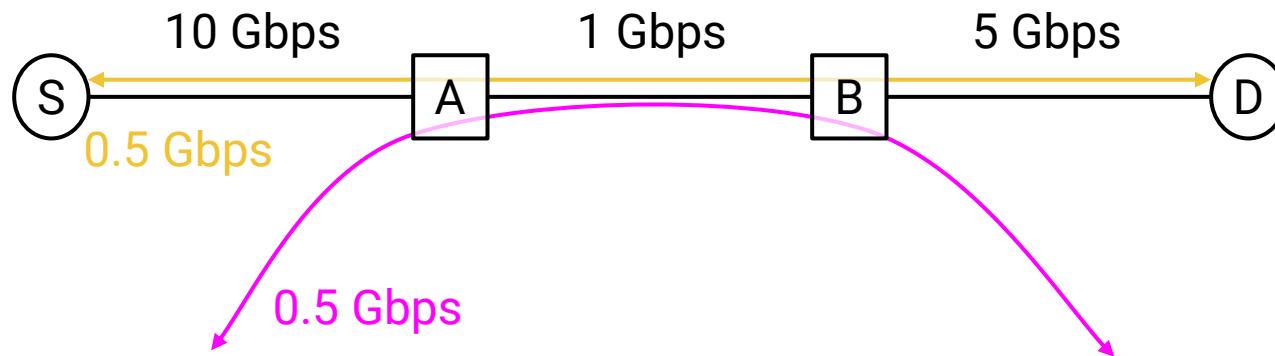
- Example: S–D would set W to send data at 1 Gbps

In practice, the bottleneck is shared with other flows

- Example: With the red flow, S–D should only send at 0.5 Gbps
- Each sender should only consume *its share* of the bandwidth → Should reduce window size W
- But, how do we update the window size?

Recall: $W = RTT \times \text{bandwidth}$

So W and bandwidth are proportional (roughly speaking)



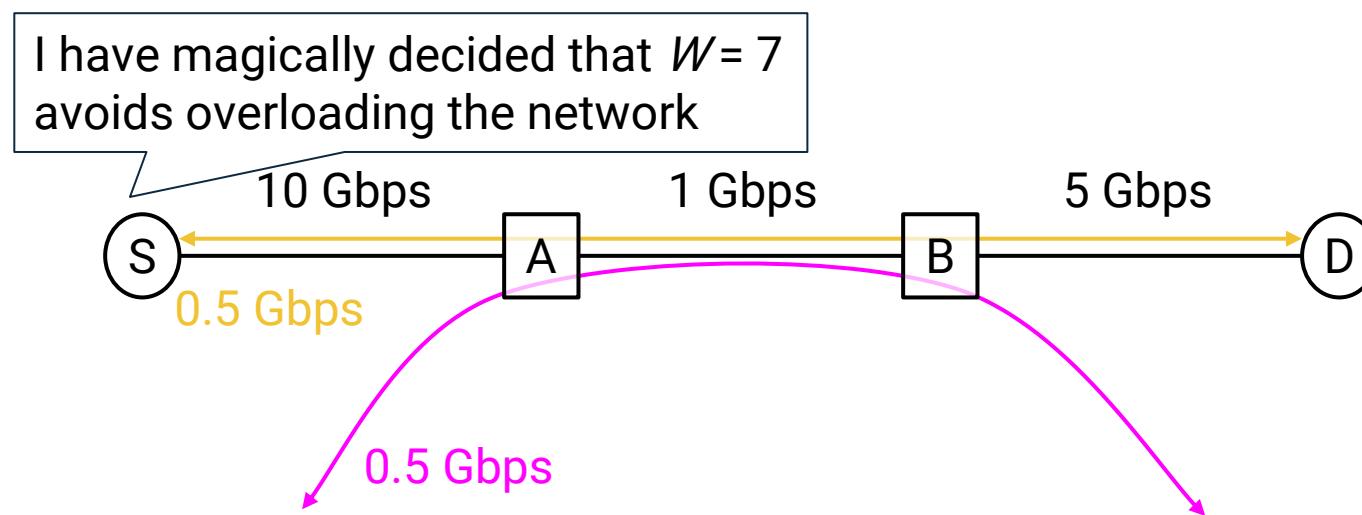
Setting Window Size (3/3): Don't Overload Network (Congestion Control)

The sender's TCP code implements a **congestion control algorithm** that dynamically computes the sender's share of bandwidth

- The algorithm balances a few goals (e.g. performance, avoid overload, fairness)
- Dynamic adjustment: Learn the current congestion level, and adjust rate accordingly
- The output of the algorithm is a **congestion window** (cwnd)

There are network-assisted congestion control algorithms (TCP ECN, ATM, DECbit, etc.):

- Routers provide direct feedback to sender/receiver



Recall: $W = RTT \times \text{bandwidth}$
So W and bandwidth are proportional (roughly speaking)

Setting Window Size (3/3): Don't Overload Network (Congestion Control)

Congestion Control Algorithm Sketch

Every host *independently* runs the same algorithm:

1. Pick an initial rate R (*How do we pick the initial rate?*)
2. Try sending at rate R for some period
 - Did I experience congestion in this time period? (*How do we detect congestion: loss or delay?*)
 - If no, increase R
 - If yes, reduce R (*How much do we increase/decrease by?*)
 - Repeat

Components of a solution:

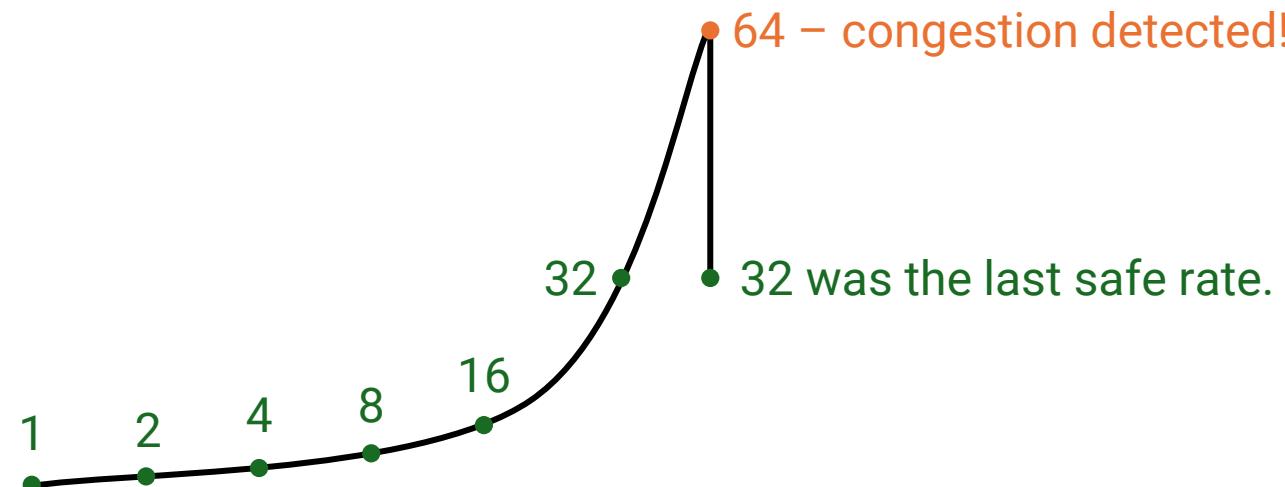
- Discovering an initial rate
- Detecting congestion
- Reacting to congestion (or lack thereof) by increasing/decreasing rate

Setting Window Size (3/3): Don't Overload Network (Congestion Control)

Discovering an Initial Rate: Slow Start

Slow start is an algorithm for discovering the initial rate:

- Start with a small rate (could be much less than actual bandwidth)
- Increase exponentially (e.g. double rate each time) until we detect loss
- A safe rate is the most recent rate before detecting loss:
 - $0.5 \times$ rate when loss occurred



Setting Window Size (3/3): Don't Overload Network (Congestion Control)

Reacting to Congestion

Rate adjustment: How much do we increase/decrease by?

- Critical part of congestion control design
- Determines how quickly a host adapts to changes in available bandwidth

Goals for rate adjustment:

- Efficiency: High utilization of link bandwidth.
- Fairness: Each flow gets an equal share of bandwidth.

Setting Window Size (3/3): Don't Overload Network (Congestion Control)

Reacting to Congestion

At a high level, we can either adjust quickly or slowly:

- Fast: **Multiplicative** changes.
 - Increase by doubling: $R \rightarrow 2R$
 - Decrease by halving: $R \rightarrow R/2$
- Slow: **Additive** changes.
 - Increase by adding: $R \rightarrow R + 1$
 - Decrease by subtracting: $R \rightarrow R - 1$

We can combine these rates in four ways:

- AIAD: Additive Increase,
Additive Decrease
- AIMD: **Additive Increase,**
Multiplicative Decrease
(TCP uses this)
- MIAD: Multiplicative Increase,
Additive Decrease
- MIMD: Multiplicative Increase,
Multiplicative Decrease

Setting Window Size (3/3): Don't Overload Network (Congestion Control)

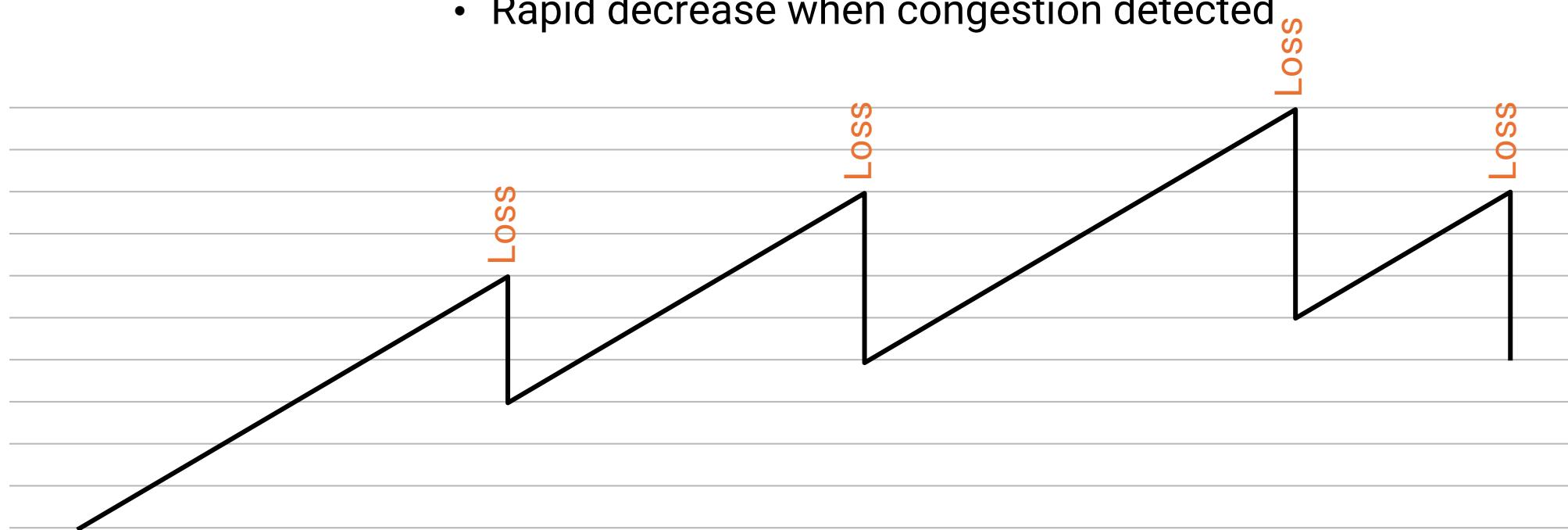
Reacting to Congestion

Why AIMD? Intuition: Sending too much is worse than sending too little

- Sending too much: Congestion, packets dropped and retransmitted
- Sending too little: Somewhat lower throughput

General approach:

- Gentle increase when uncongested (exploration)
- Rapid decrease when congestion detected

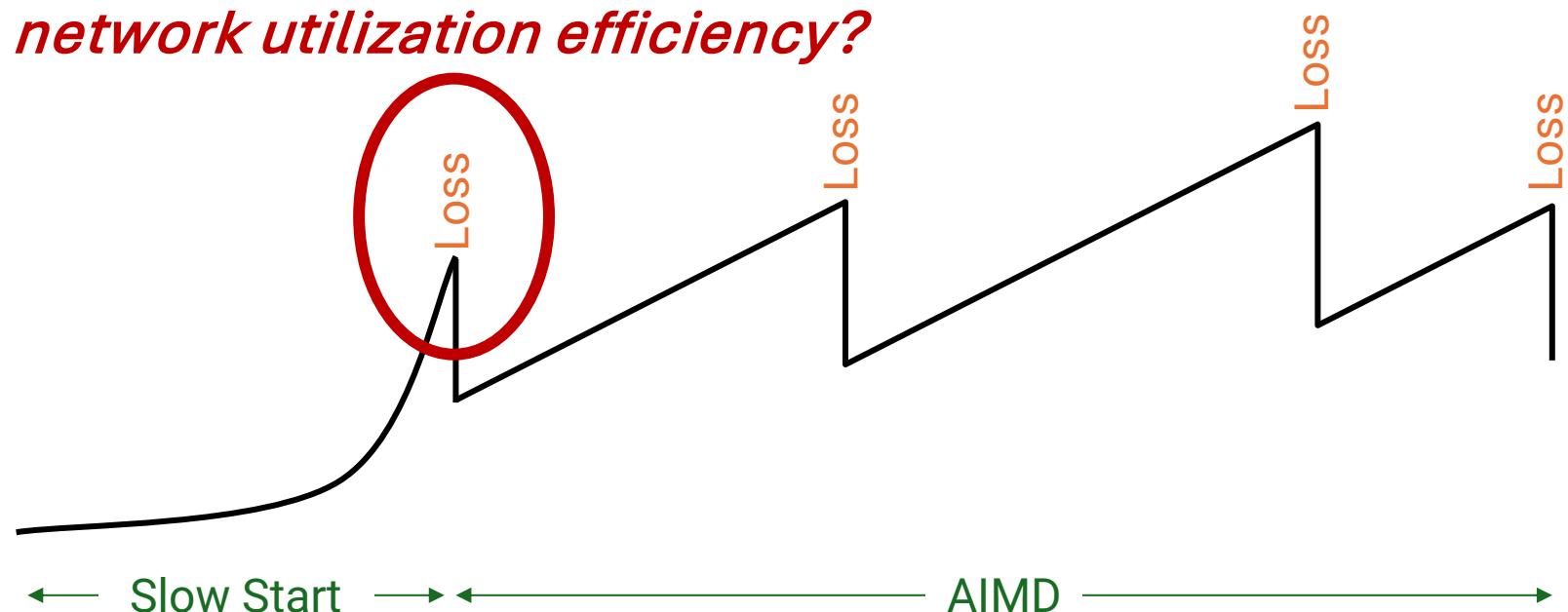


Setting Window Size (3/3): Don't Overload Network (Congestion Control)

1. Pick an initial rate R (*How do we pick the initial rate? → Slow start*)
2. Try sending at rate R for some period ***Q1: How long? → RTT? or event-driven***
 - Did I experience congestion in this time period? (*How do we detect congestion: Packet loss*)
 - If no, increase R (*How much do we increase by? → Additive increase*)
 - If yes, reduce R (*How much do we decrease by? → Multiplicative decrease*)
 - Repeat

Q2: How does this loss affect network utilization efficiency?

This algorithm creates a sawtooth-shaped graph as the rate is adjusted



Setting Window Size (3/3): Don't Overload Network (Congestion Control)

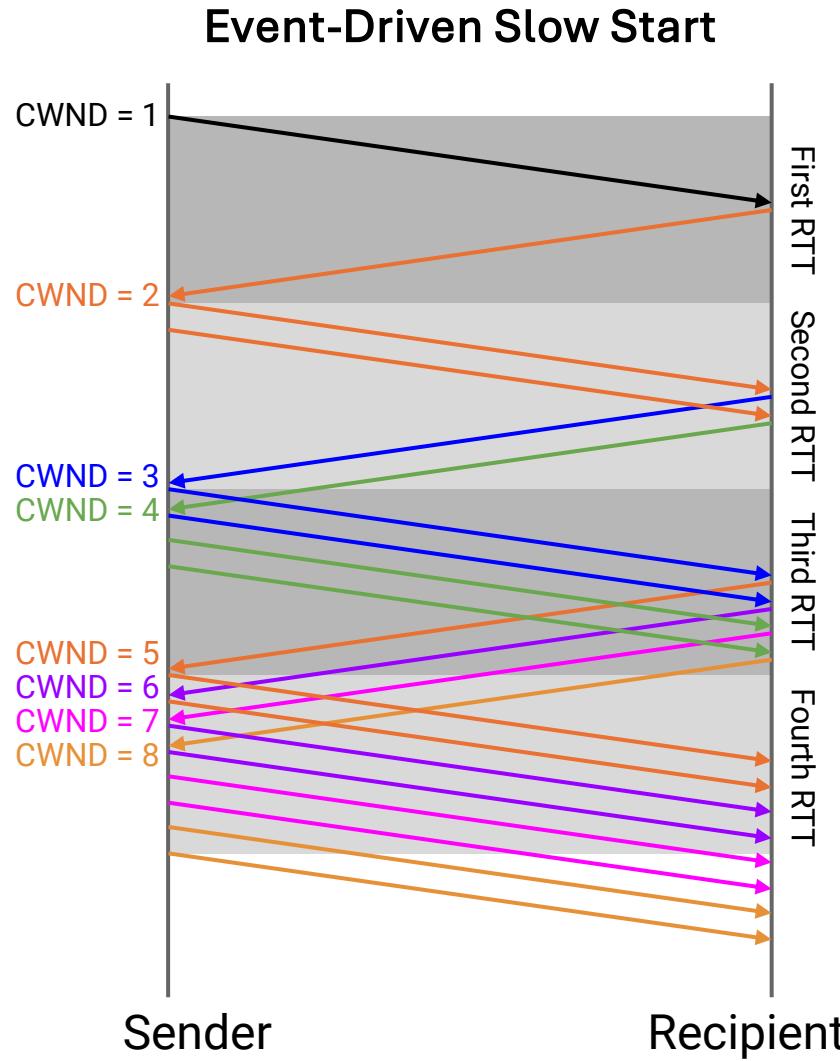
Windows and Rates

Q1: How long is "some period of time"?

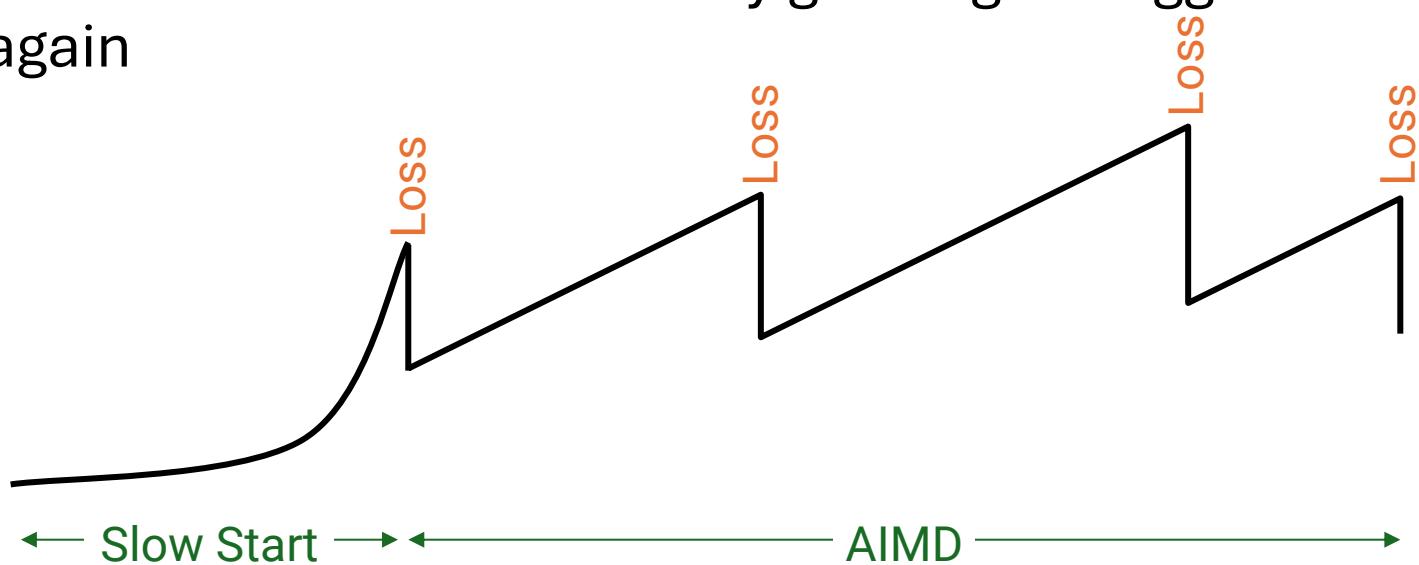
- Ideally, each iteration lasts for one RTT → RTT is hard to measure, changes dynamically
- CWND updates are ***event-driven***: Sender adjusts CWND each time something interesting happens in TCP
 1. We ***get an ack*** for new data:
 - This means there was no congestion
 - **Increase** CWND (using ***slow-start***, or ***additive increase*** in AIMD)
 2. We get ***3 duplicate acks*** and declare a packet lost:
 - Probably an isolated loss, since we're still getting subsequent acks
 - **Decrease** CWND (multiplicative decrease in AIMD)
 3. The ***timer expires*** and we declare a packet lost:
 - We probably lost several packets – didn't get any duplicate acks. **Bad news!**
 - Abandon current rate and start over from ***slow-start***

Setting Window Size (3/3): Don't Overload Network (Congestion Control)

Q2: Event-Driven Slow Start – SSTHRESH (Slow Start Threshold)



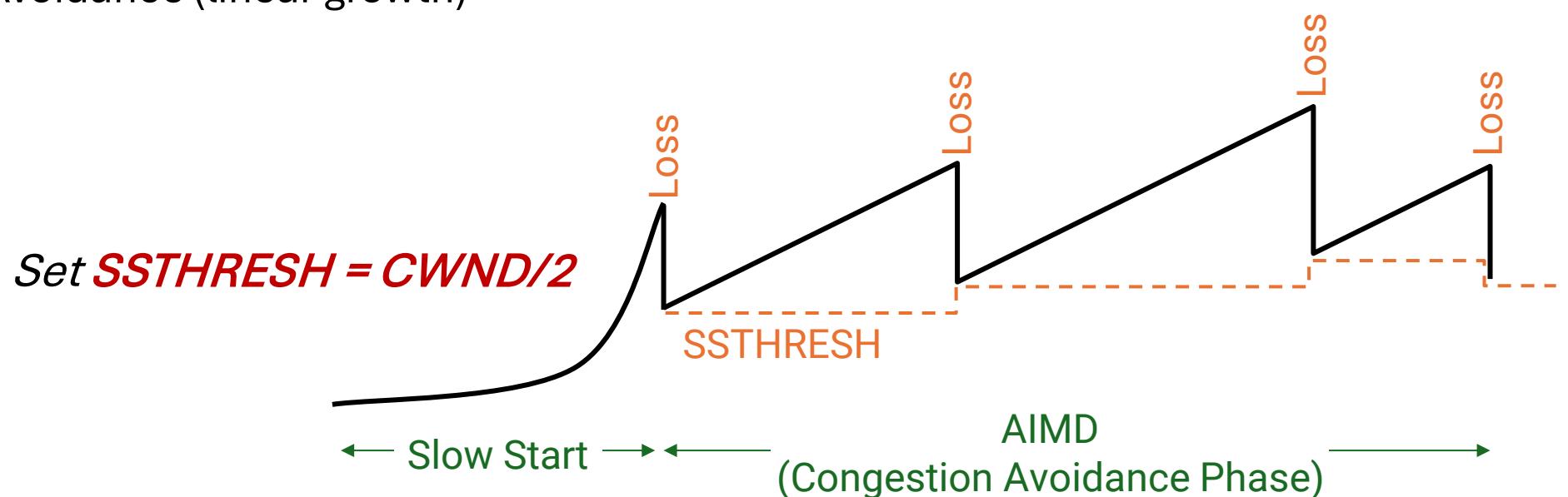
- When connection begins, increase rate/ **cwnd** exponentially until first loss event:
 - Initially **cwnd** = 1 MSS (Maximum Segment Size)
 - Double **cwnd** every RTT
 - Done by incrementing **cwnd** for every ACK received
 - After detecting the first loss, **SSTHRESH** "remembers" approximately where congestion occurred, preventing the sender from immediately growing too aggressively again



Setting Window Size (3/3): Don't Overload Network (Congestion Control)

Q2: Event-Driven Slow Start – STHRESH (Slow Start Threshold)

- When we encounter the packet loss:
 - Enter TCP recovery phase: TCP responds to detected packet loss and attempts to restore normal transmission rates while avoiding further congestion
 - **SSTHRESH** helps us remember the last safe rate
- **SSTHRESH** serves **3** primary functions:
 1. **Phase Transition Control:** Determines when to switch from Slow Start (exponential growth) to Congestion Avoidance (linear growth)



Setting Window Size (3/3): Don't Overload Network (Congestion Control)

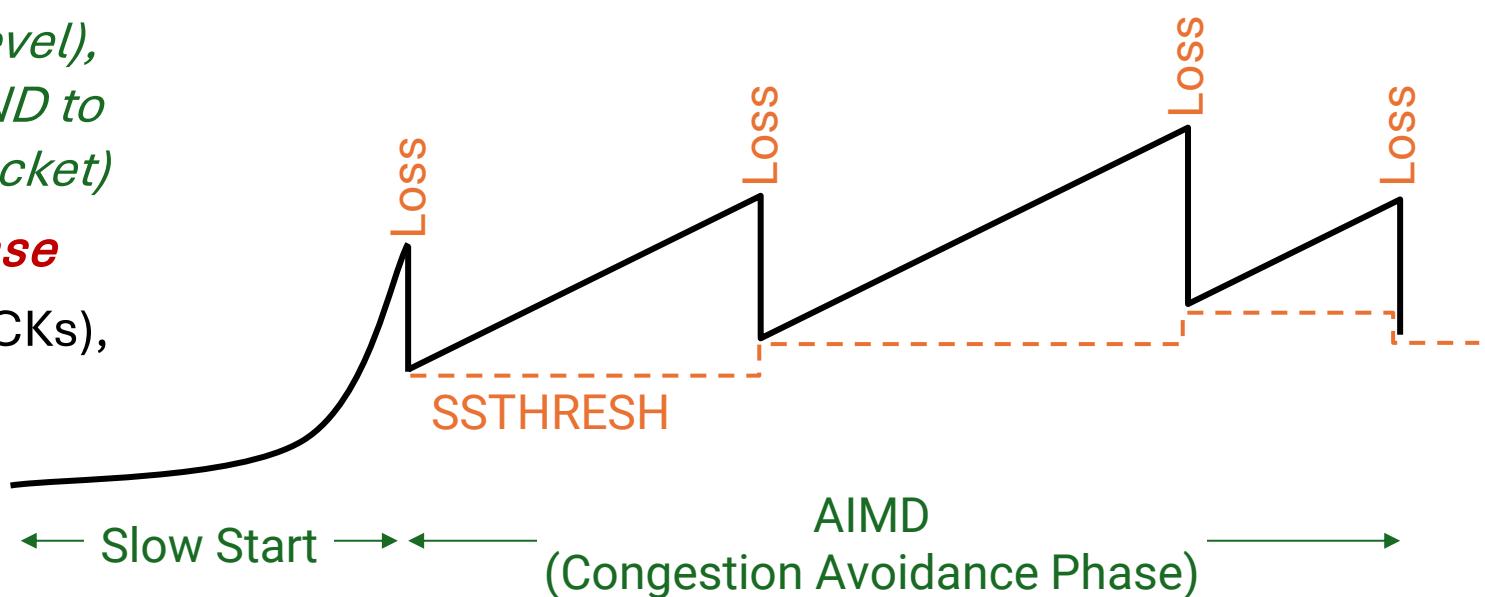
SSTHRESH and TCP Recovery Phases

- **SSTHRESH** serves **3** primary functions:
 - 2. Congestion Memory:** Stores the approximate point where congestion was last detected
 - 3. Recovery Guidance:** Influences behavior during and after congestion recovery

Upon packet loss detection (**3 dup ACKs**):

- Fast Recovery
- **SSTHRESH** = $cwnd / 2 \rightarrow$ Set new threshold
- $cwnd = SSTHRESH$ (Stay at threshold level),
or $cwnd = 1$ (If dividing by 2 causes CWND to fall below 1, it's okay to round up to 1 packet)
- Stay in the **Congestion Avoidance Phase**
 - Network is still working (receiving ACKs), so we assume selective packet loss

If sender receives duplicate ACKs but number is less than 3: The connection **stays** in its current state



Setting Window Size (3/3): Don't Overload Network (Congestion Control)

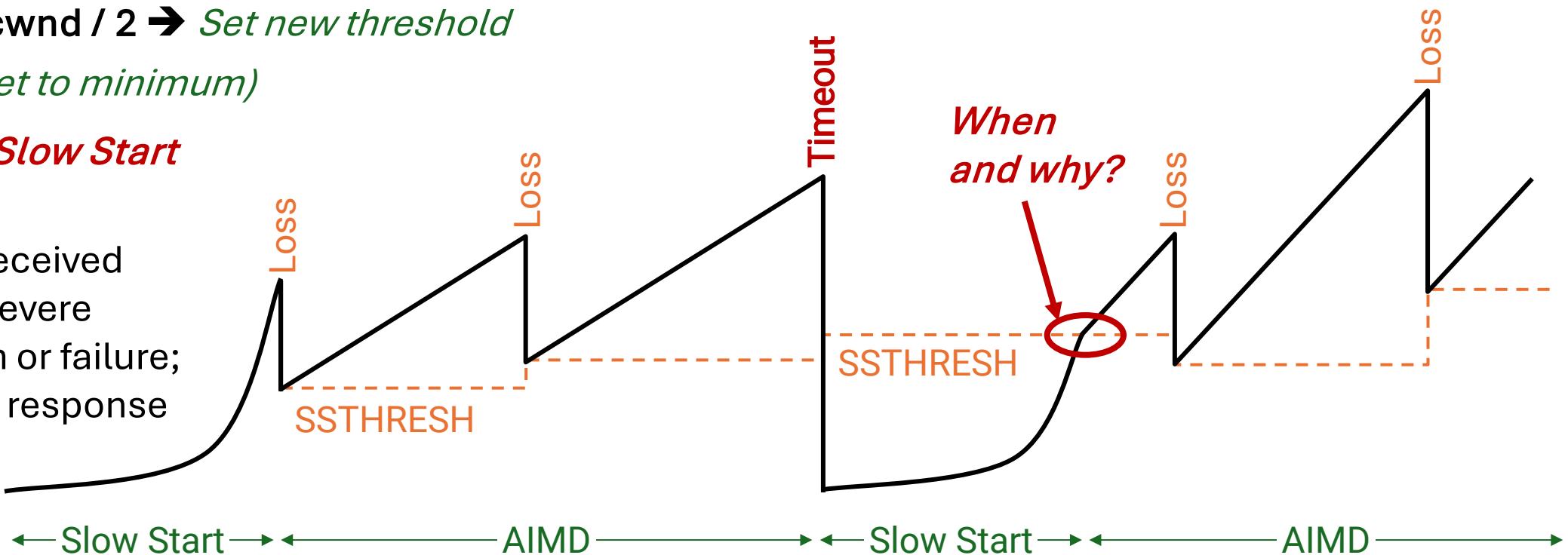
SSTHRESH and TCP Recovery Phases

- **SSTHRESH** serves **3** primary functions:
 - 2. Congestion Memory:** Stores the approximate point where congestion was last detected
 - 3. Recovery Guidance:** Influences behavior during and after congestion recovery

Upon packet loss detection (*timeout*):

- **SSTHRESH**= cwnd / 2 → Set new threshold
- **cwnd = 1** (Reset to minimum)
- **Switch to the Slow Start Phase**

- No ACKs received suggests severe congestion or failure; aggressive response needed



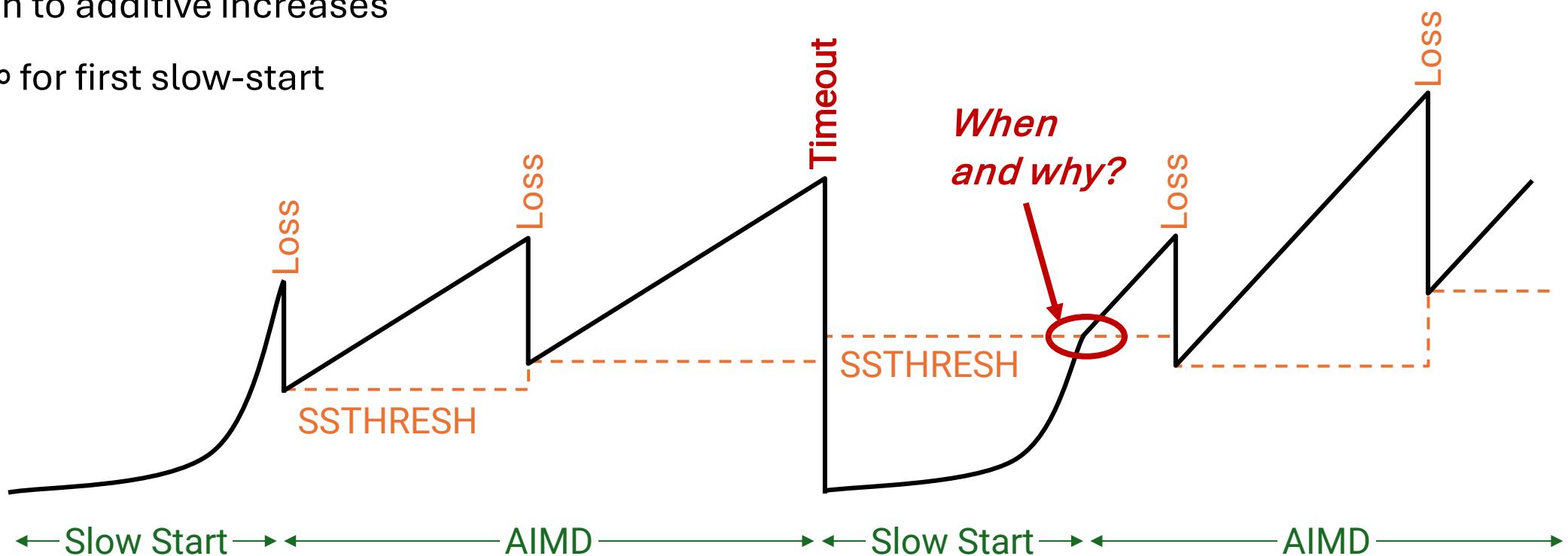
Setting Window Size (3/3): Don't Overload Network (Congestion Control)

SSTHRESH and TCP Recovery Phases

- **SSTHRESH** serves **3** primary functions:
 - 2. Congestion Memory:** Stores the approximate point where congestion was last detected
 - 3. Recovery Guidance:** Influences behavior during and after congestion recovery

During slow start: If **CWND** exceeds **SSTHRESH**, switch to additive increases

- **SSTHRESH** = ∞ for first slow-start



Summary: TCP Congestion Control

Detecting congestion:

- Loss-based

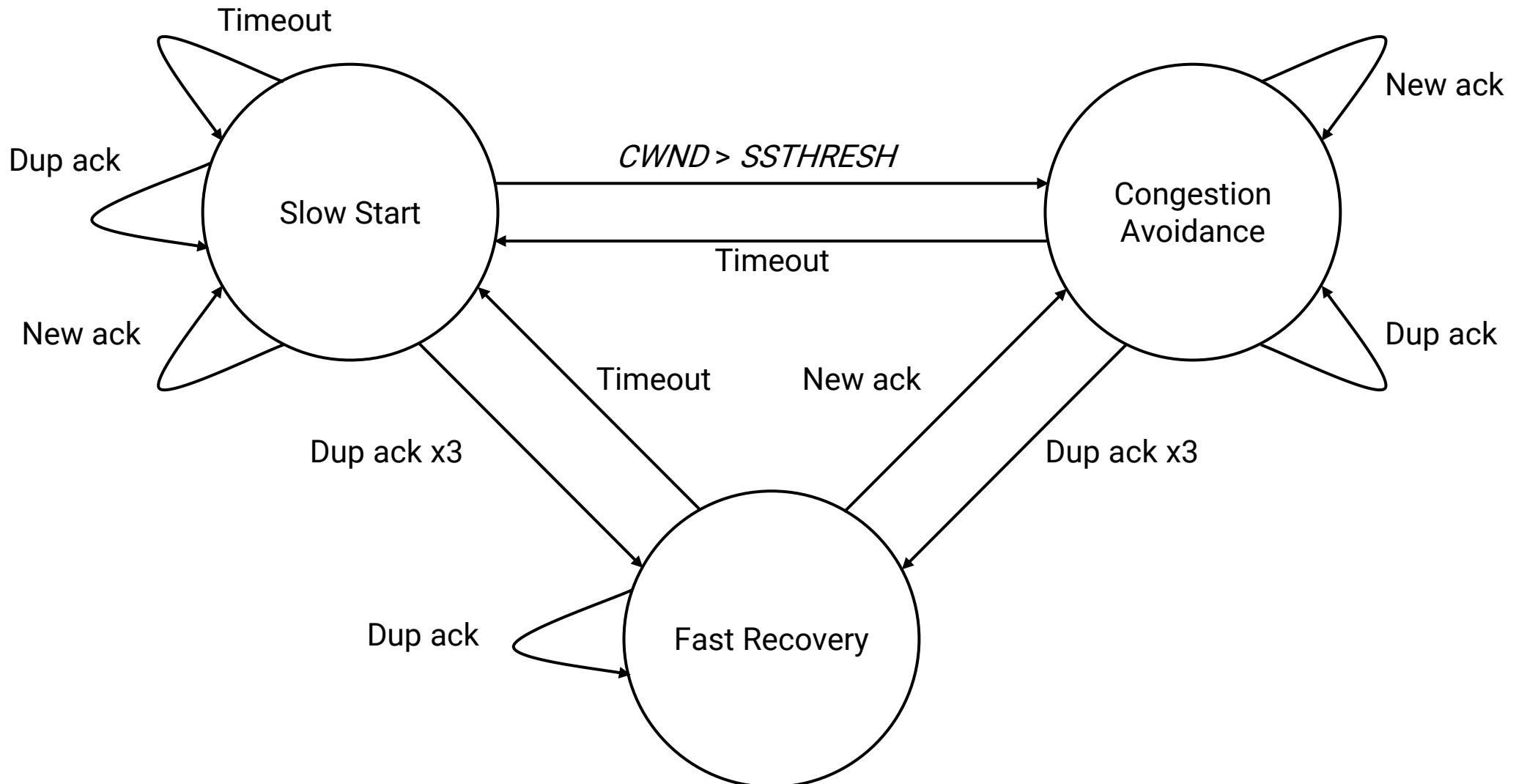
Discovering an initial rate:

- Slow-start. Double rate until loss or safe rate ($SSTHRESH$) exceeded
- Event-driven AIMD: Add 1 to $CWND$ on each ack

Adapting rate to congestion:

- AIMD (additive increase, multiplicative decrease) approaches fairness
- Event-driven:
 - Add $1/CWND$ on each ack. After one RTT, $CWND$ will have increased by 1
 - Divide $CWND$ in half when there are 3 duplicate acks
- When a packet is lost on timeout, restart from slow-start

Summary: TCP Congestion Control State Machine



Setting Window Size: Summary

How big should the window be?

Pick window size W to balance three goals:

1. Take advantage of network capacity ("fill the pipe")... (set $W = \text{RTT} \times \text{bandwidth}$)
2. ...but don't overload the recipient (**flow control**)... (set $W = \text{recipient's advertised window}$)
3. ...and don't overload links (**congestion control**). (set $W = \text{sender's congestion window}$)

Ideally: W is set to the minimum of the 3 values

In practice: W is the minimum of advertised window (2) and congestion window (3)

- It's hard for the sender to discover the bottleneck bandwidth and calculate (1)
- Congestion window \leq pipe-filling window
Equal if we're the only connection, less than if sharing bandwidth with others

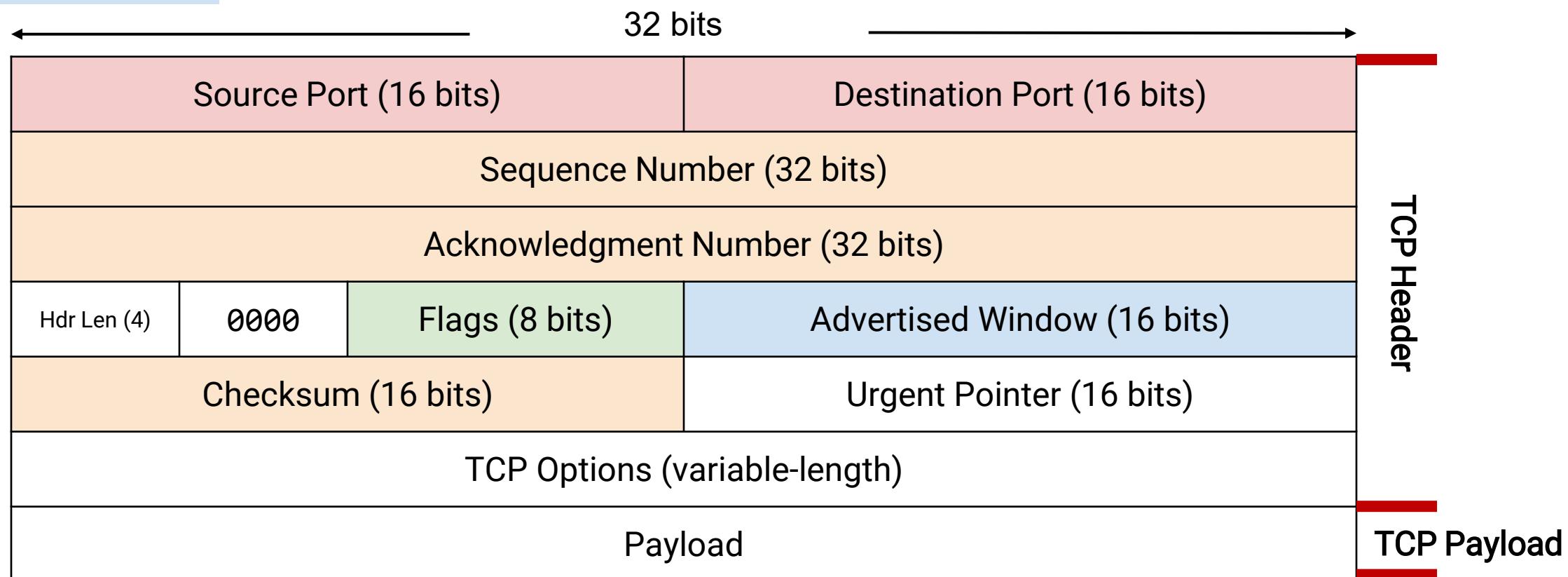
Roadmap

- *TCP implementation: sliding window*
- *Flow control*
- *Congestion control*
- ***TCP header***

TCP Header

What functions does TCP implement?

1. Demultiplexing (*ports*)
2. Reliability (*checksum, sequence and ack numbers*)
3. Connection setup and teardown (*flags*)
4. Flow control (*advertised window/receive window, rwnd*)

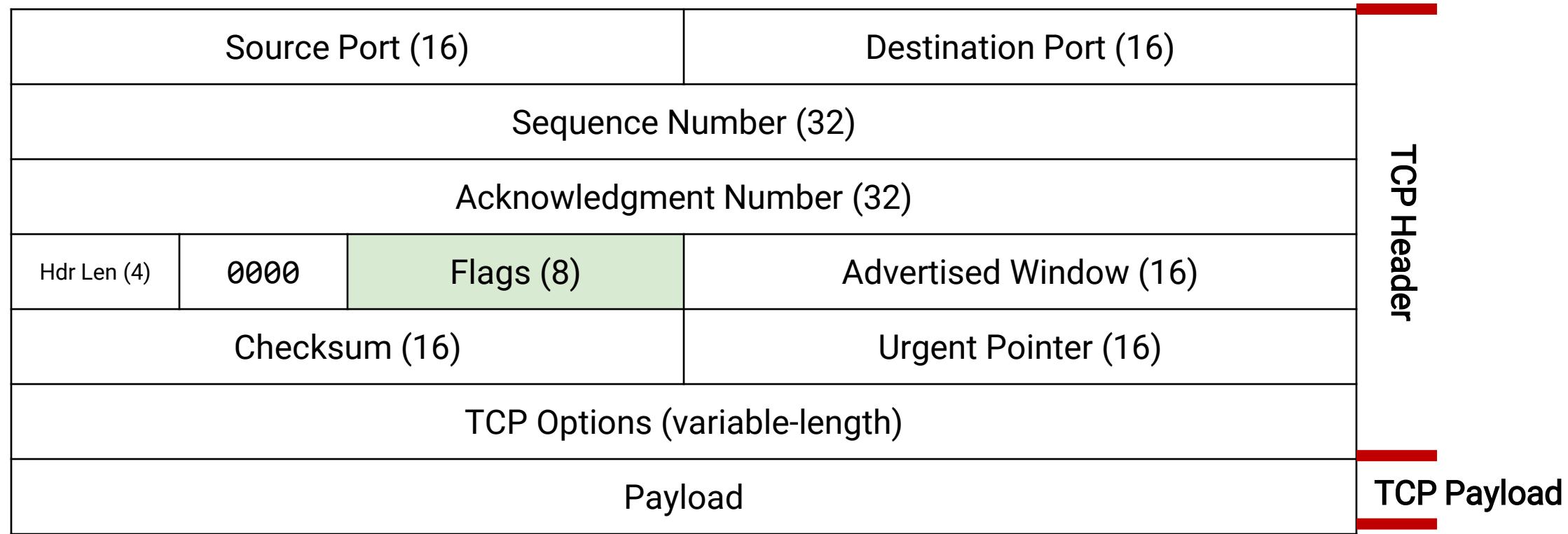


TCP Header

There are 8 flags that can be set in TCP. We care about 4 of them:

- SYN: I'm sending my initial sequence number.
- ACK: I'm acking data (please look at the ack number).
- FIN: I'm done sending data, but will keep receiving data.
- RST: I'm done sending and receiving data.

We won't look at the other four: CWR, ECE, URG, PSH.



CSC 3511 Security and Networking

Week 5, Lecture 1: Network Layer and IP Protocol

Roadmap

- ***Network Layer: Overview***
- *Routers*
- *IP: Internet Protocol*
 - *IP Address and DHCP*
 - *Network Address Translation (NAT)*

Network-layer Services and Protocols

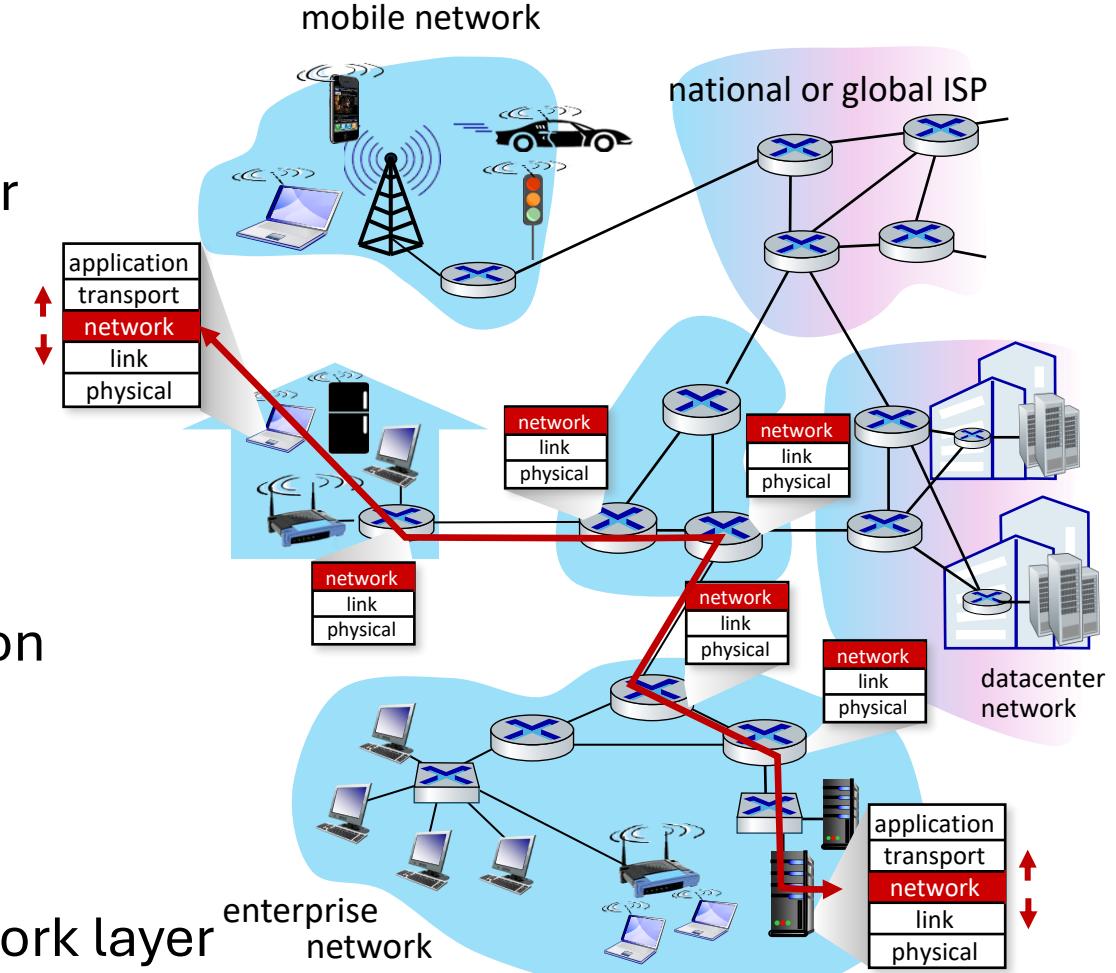
Layer 3 (Network/IP) gave us the ability to transport segment from sending to receiving host

- **Sender:** encapsulates segments into datagrams, passes to link layer
- **Receiver:** delivers segments to transport layer
- Network layer protocols in *every Internet device*: hosts, routers

Internet “**best effort**” service model; **No** guarantees on:

- Successful datagram delivery to destination
- Timing or order of delivery
- Bandwidth available to end-end flow

The **Internet Protocol (IP)** is the dominant network layer protocol that provides this host-to-host delivery service



Two Key Network Layer Functions

Network layer functions:

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

Core component: *routers*

- Examines header fields in all IP datagrams passing through it
- Moves datagrams from input ports to output ports to transfer datagrams along end-end path

IP provides both functions: *IP forwarding* (data plane) + *IP routing* (control plane)

Analogy: taking a trip

- *Forwarding*: process of getting through single interchange
- *Routing*: process of planning trip from source to destination



Forwarding



routing

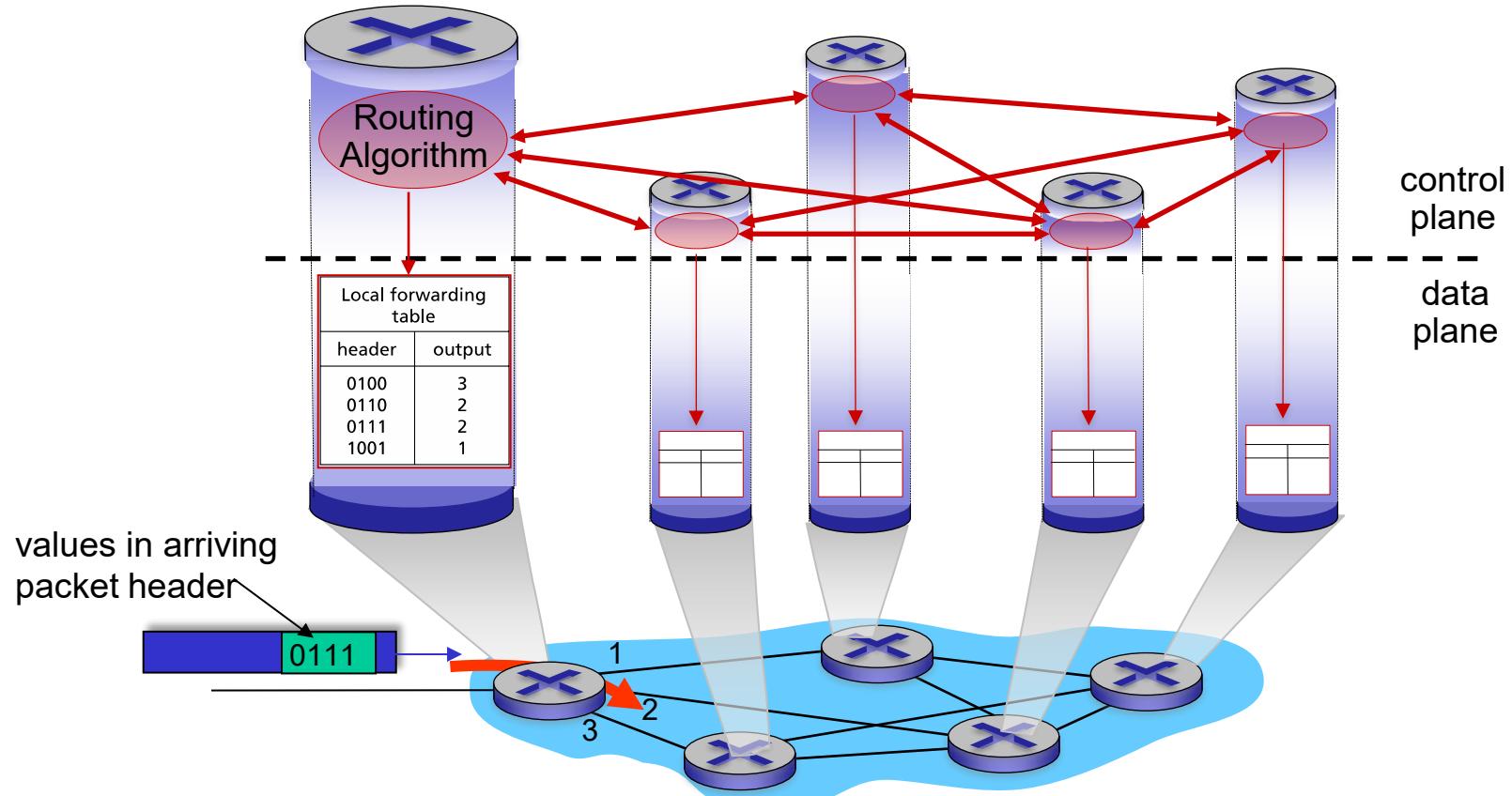
Network Layer: Data Plane and Control Plane

Data plane:

- *Local*, per-router function
- Determines how datagram arriving on router input port is forwarded to router output port

Control plane

- *Network-wide* logic
- Determines how datagram is routed among routers along end-end path from source host to destination host



The Internet Protocol (IP)

What IP does (functions):

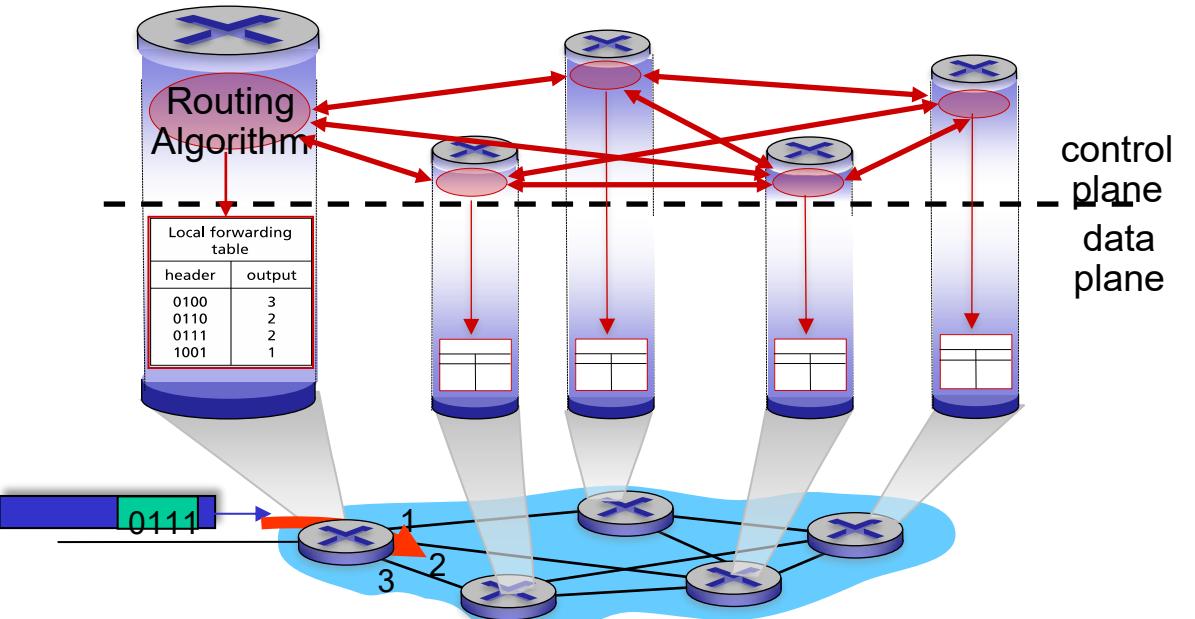
- **Addressing:** Assigns logical IP address
- **Encapsulation:** Wraps transport layer segments in IP datagrams
- **Forwarding:** Moves packets hop-by-hop toward destination
- **Fragmentation:** Breaks/reassembles packets for different link MTUs

Control plane: Separate routing algorithms (RIP, OSPF, BGP, etc.) populate IP forwarding tables

Data plane: Internet Protocol performs forwarding based on destination IP address

What IP Contains/Related Protocols:

- ***IPv4 / IPv6 addressing*** and packet format
- ***Address Configuration:*** DHCP (automatic IP assignment)
- ***Address Translation:*** NAT (private ↔ public addresses)
- ***Support Protocols:*** ICMP (error reporting), ARP (address resolution)

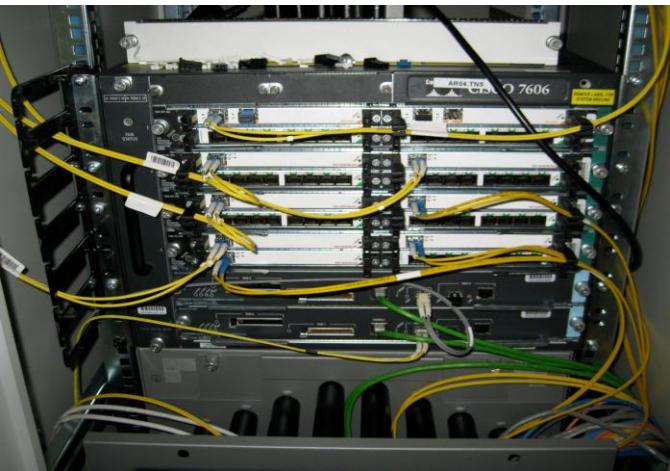


Roadmap

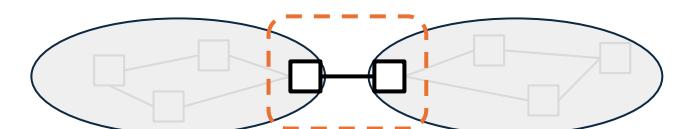
- *Network Layer: Overview*
- ***Routers***
- *IP: Internet Protocol*
 - *IP Address and DHCP*
 - *Network Address Translation (NAT)*

Routers

- A router performs routing protocols to learn about routes
- A router receives packets and forwards them according to the forwarding table
- What do routers look like in real life?
 - A router is just a computer specialized for forwarding packets



Colocation facility (aka carrier hotel): A building where multiple ISPs install routers



Router Planes

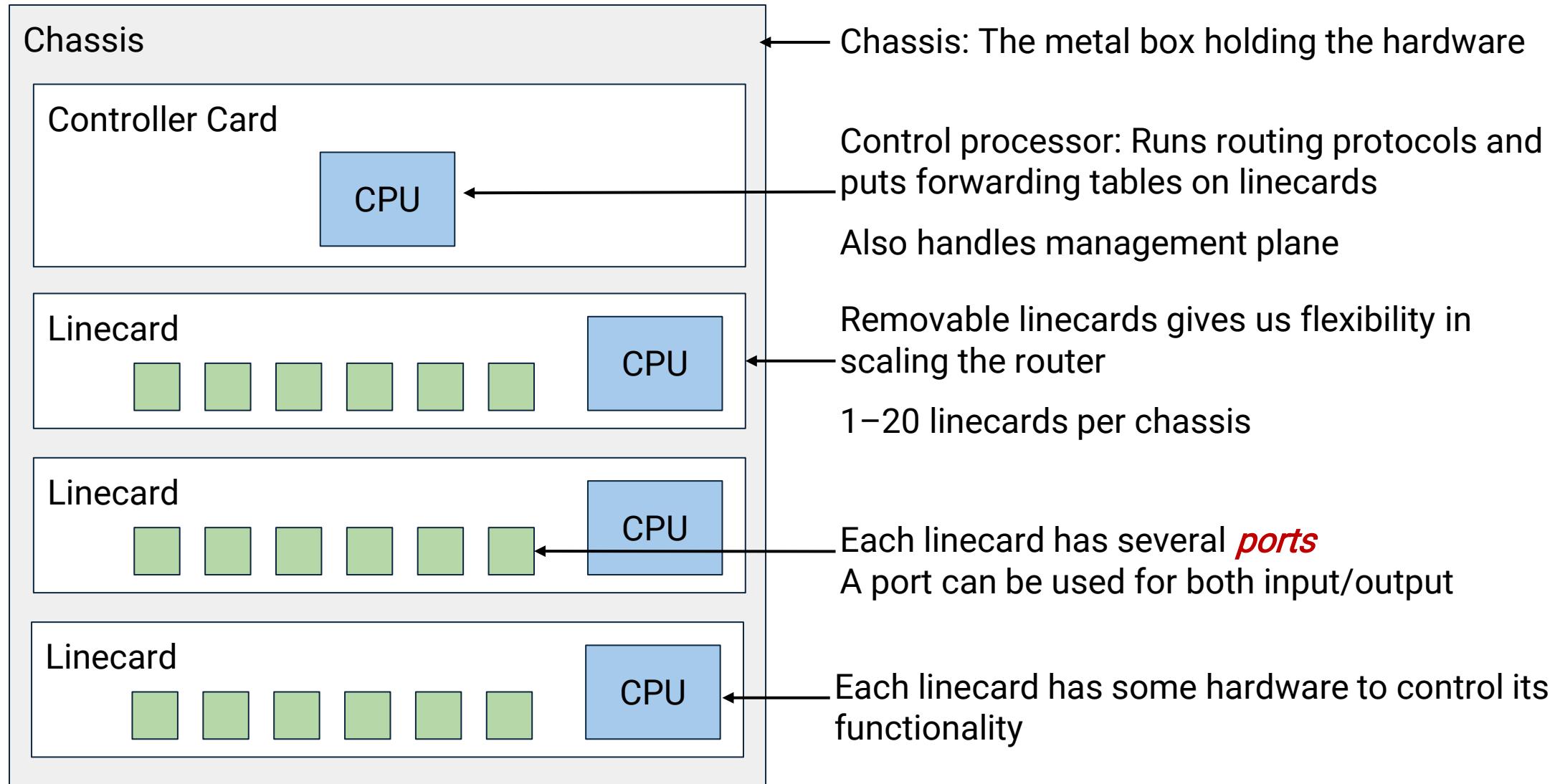
The components of a router can be split into 3 planes:

- The ***data plane*** handles forwarding packets
 - Used every time a packet arrives: Nanosecond time scale
 - Operates locally; No coordinating with other routers
- The ***control plane*** performs routing protocols
 - Used every time the network topology changes; Second time scale
- The **management plane** lets the operator interact with the router.
 - Operator uses **network management system (NMS)** to interact with router
 - Tell the router what to do, and monitor what the router is doing
 - Example: Assigning costs to links
 - Example: How much traffic is being sent on each link?

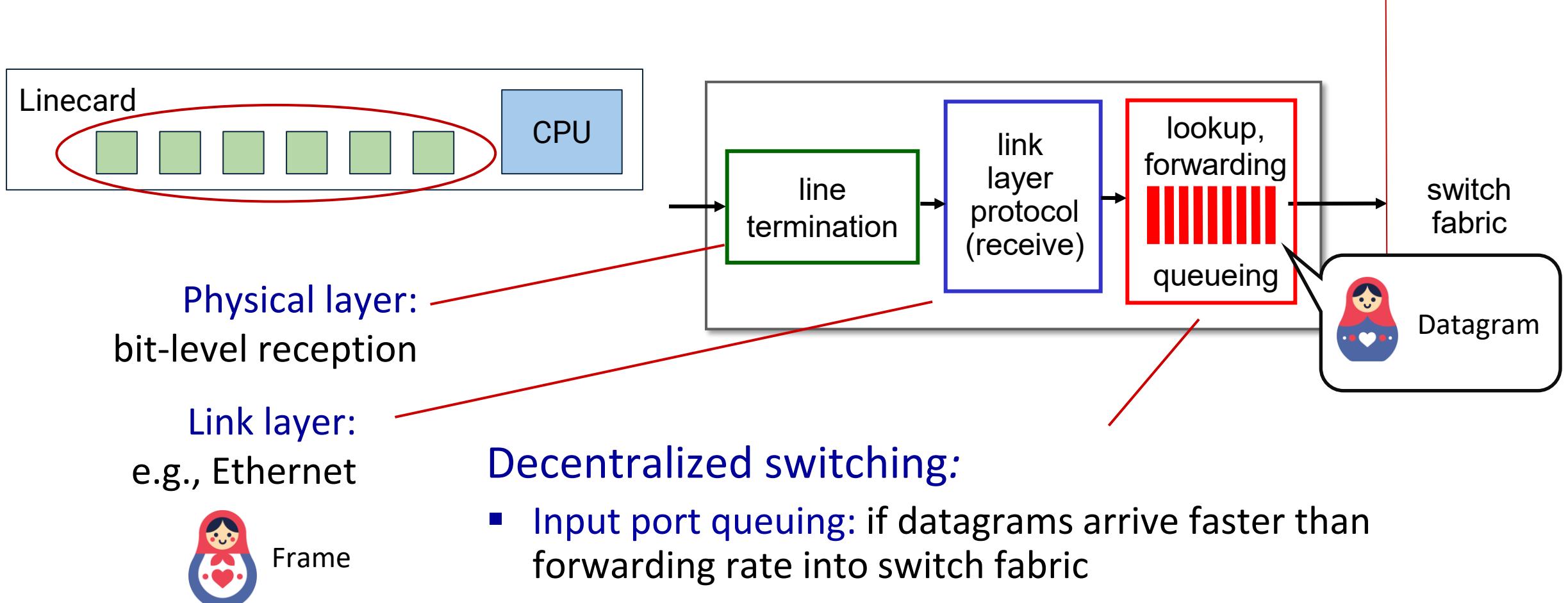
Each plane is optimized for its tasks and its time scale

What's Inside a Router: View of Components

A router is just a computer specialized for forwarding packets



What's Inside a Router: View of Port

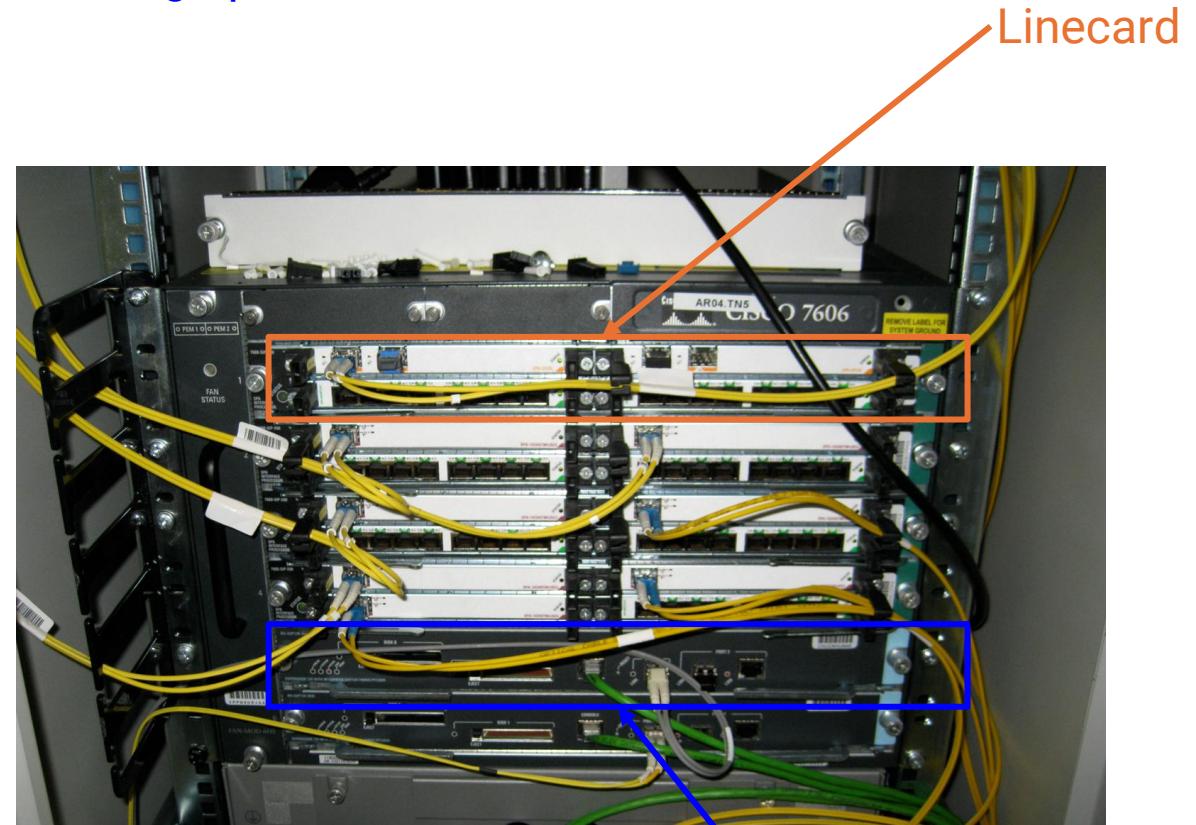
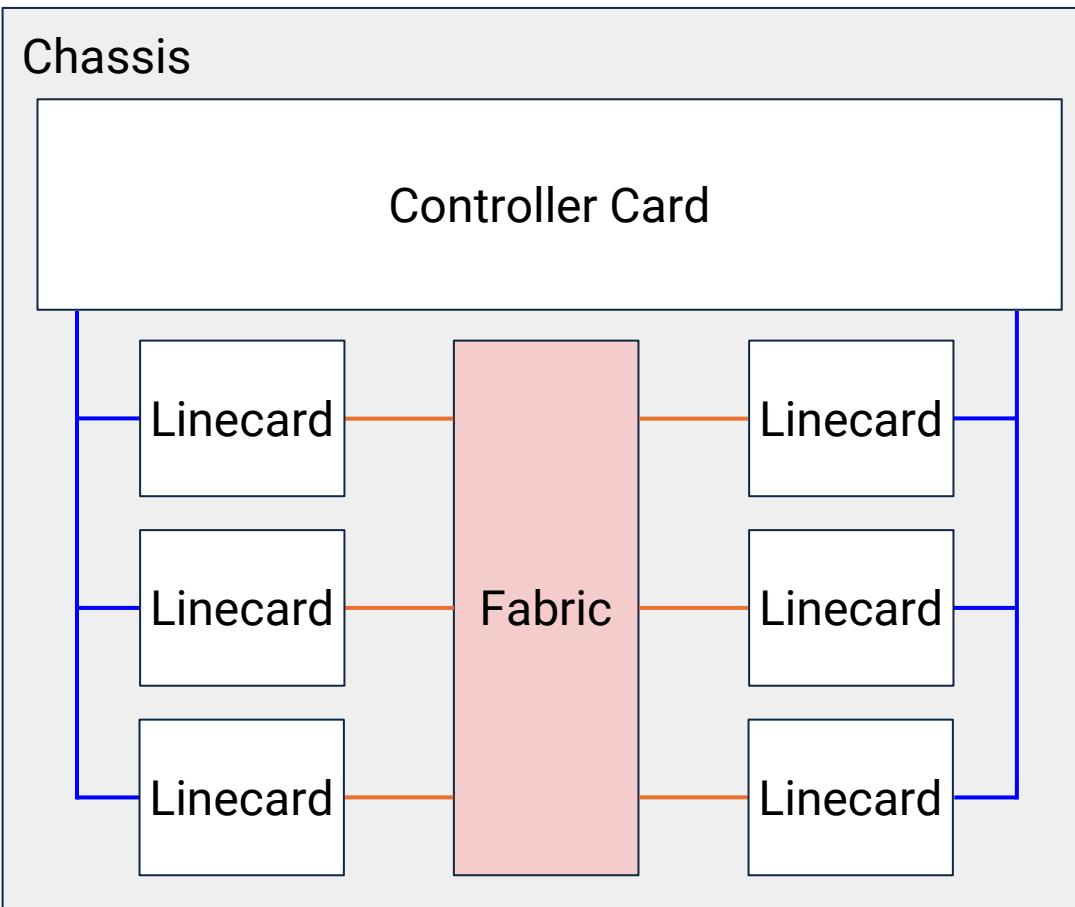


Decentralized switching:

- Input port queuing: if datagrams arrive faster than forwarding rate into switch fabric
- Using header field values, lookup output port using *forwarding table* in input port memory ("match plus action")

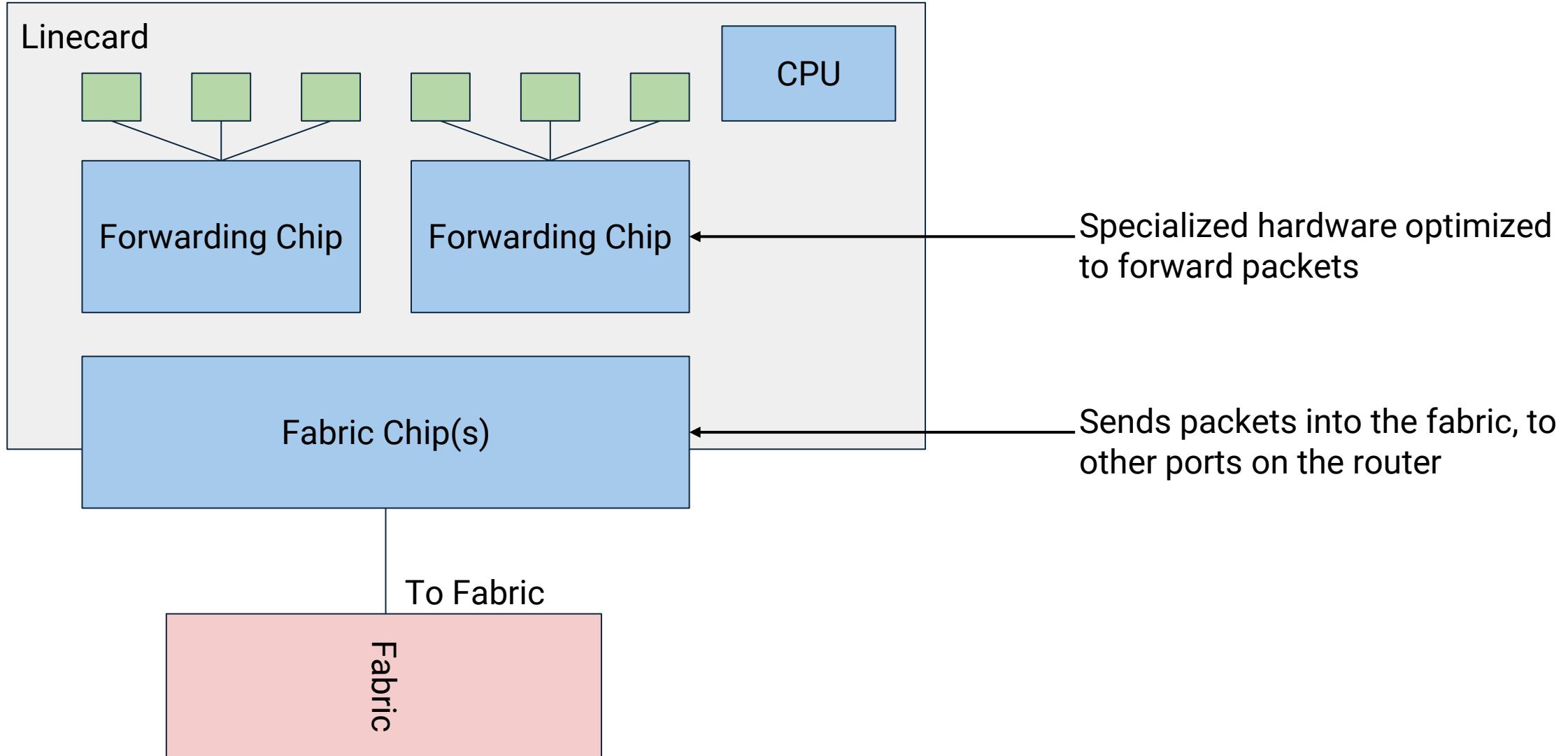
What's Inside a Router: View of Links

Controller card, aka network interface controller (NIC), connects to linecards, provides physical connection to a network, and manages routing operations



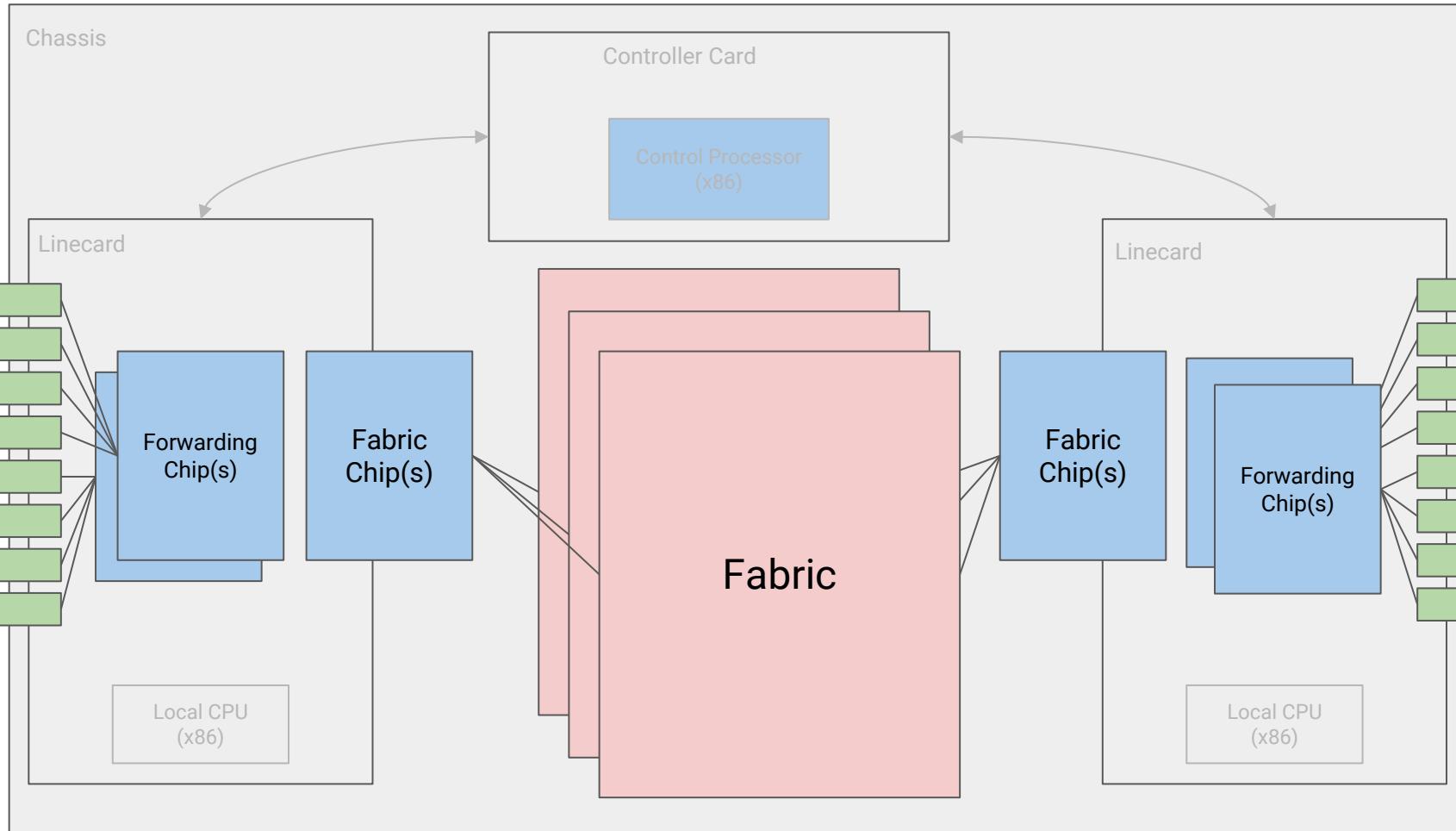
Each linecard is connected to the **fabric**: A bunch of wires providing high-bandwidth interconnection between linecards

What's Inside a Router: View of a Linecard



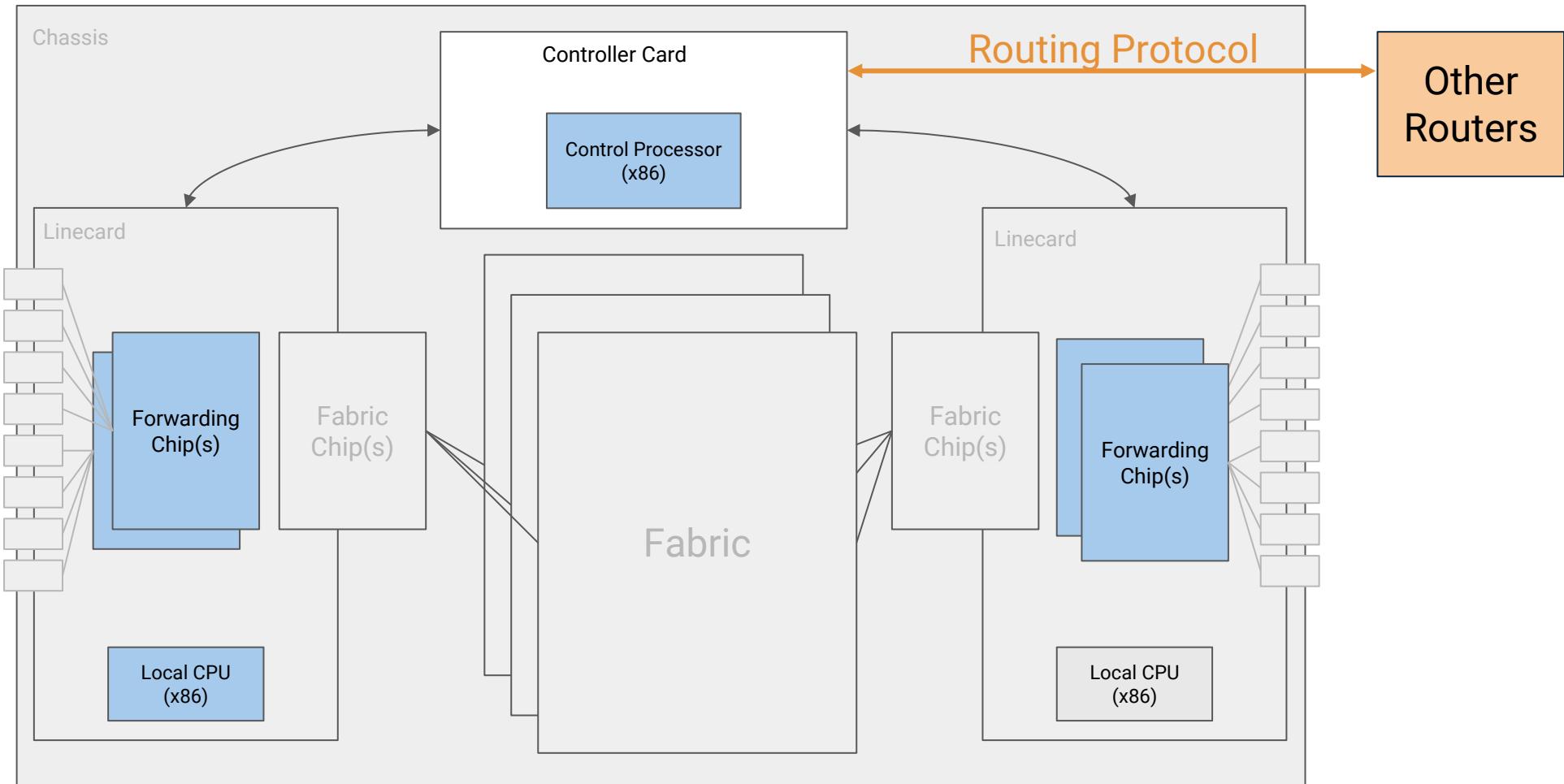
What's Inside a Router: Data Plane

Data plane/Forwarding: Packet travels from port to port, via forwarding chips and fabric (chips)



What's Inside a Router: Data Plane

Control plane/Routing: Controller card talks with other routers, and programs linecards with routes



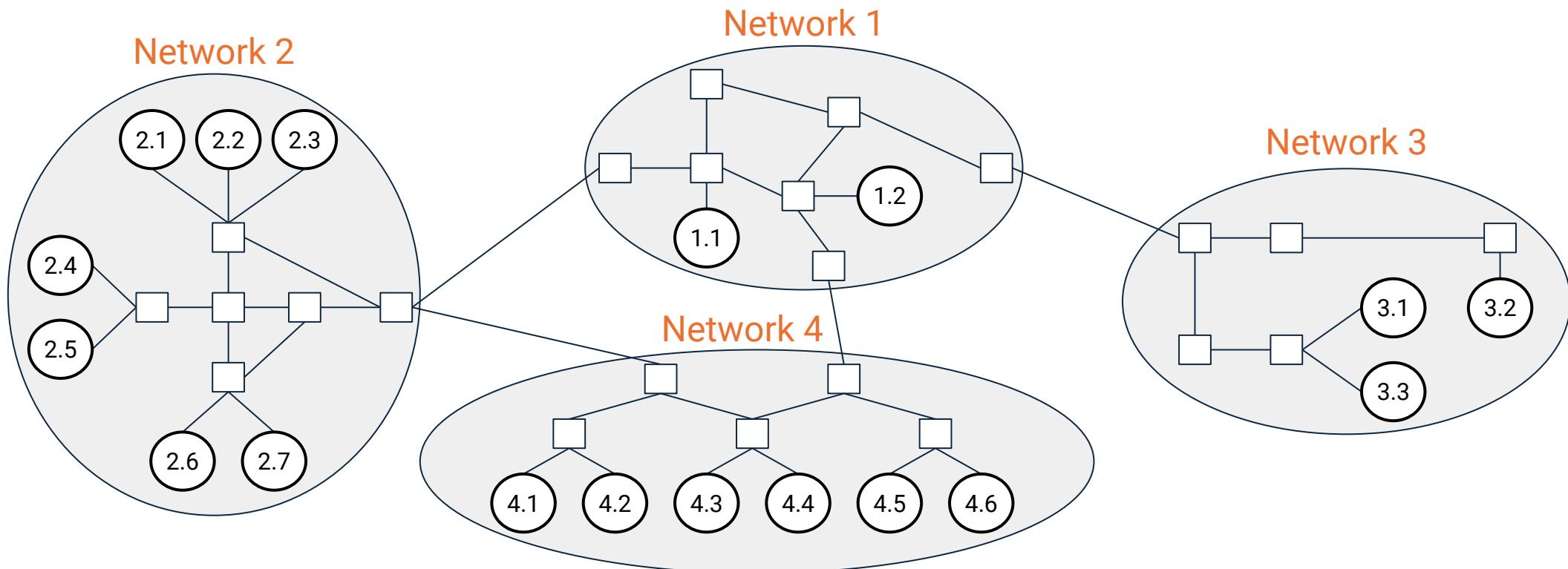
Roadmap

- *Network Layer: Overview*
- *Routers*
- ***IP: Internet Protocol***
 - ***IP Address*** and *DHCP*
 - *Network Address Translation (NAT)*

Hierarchical Network Addressing

Recall: The Internet is a network of networks

- Leads naturally to a hierarchy of addresses
- Each network gets a number; Then, each host gets a number inside the network
- Hosts that are "close to each other" (in the same network) share part of their address
- Analogy: All third-floor room numbers start with the digit 3



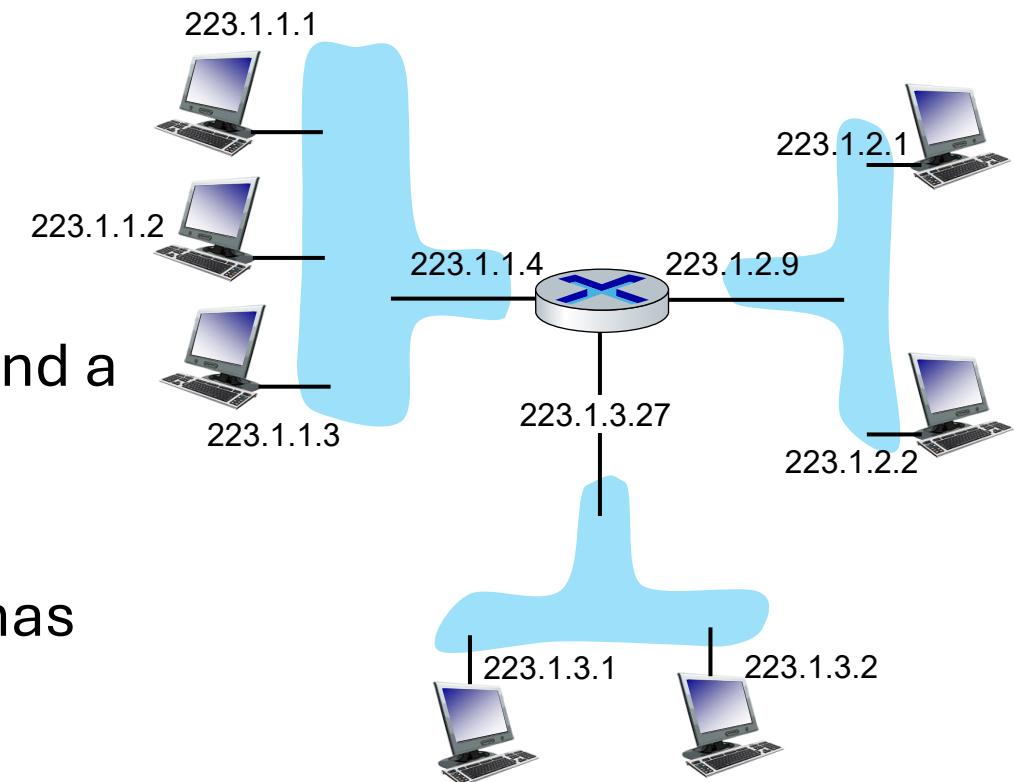
IP (v4) Addresses

An IP address is a 32-bit number that uniquely identifies a *network interface*. We could write an IP address as a 32-bit binary number:

- 11011100111010001001101001011101

What is an interface?

- The physical connection point between a device and a network. E.g., Ethernet port, WiFi adapter, fiber connection, etc.
- Each interface connects to one physical link and has one IP address
- Routers have **multiple interfaces** → multiple IP addresses (one per connected network)
- Hosts typically have **1-2 interfaces** (e.g., Ethernet, WiFi)



IP (v4) Addresses

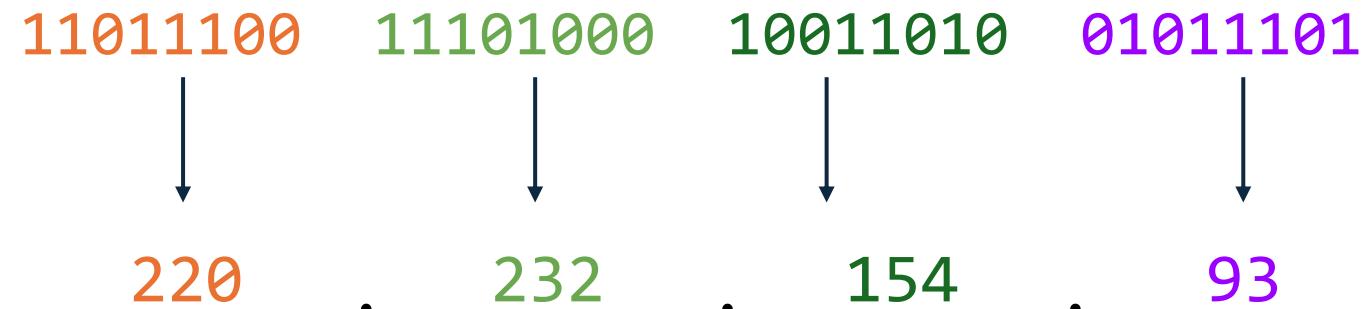
An IP address is a 32-bit number that uniquely identifies a *network interface*. We could write an IP address as a 32-bit binary number:

- 11011100111010001001101001011101 → Nobody wants to read that

Dotted quad (dotted-decimal) notation:

- Split into groups of 8 bits
- Write each **8-bit** number in decimal, **separated by dots**

Q1: How to represent a “network”? How do we know the hosts are “close to each other”



IP Addresses: Network and Host Portions

A1: IP addresses have hierarchical (two-part) structure

- Network prefix (high-order bits): Identify the host belongs to which network
- Host portion (low-order bits): Identify specific host/interface on that network

Why this matters:

- Hosts with same network prefix are on the same subnet (local network)
- They can communicate directly without going through a router
- Routers use *network prefix* for forwarding decisions

Q2: How do we indicate where to split the network and host portions?

220 . 232 . 154 . 93
← Network Prefix → ← Host Portion →

IP (v4) Addresses

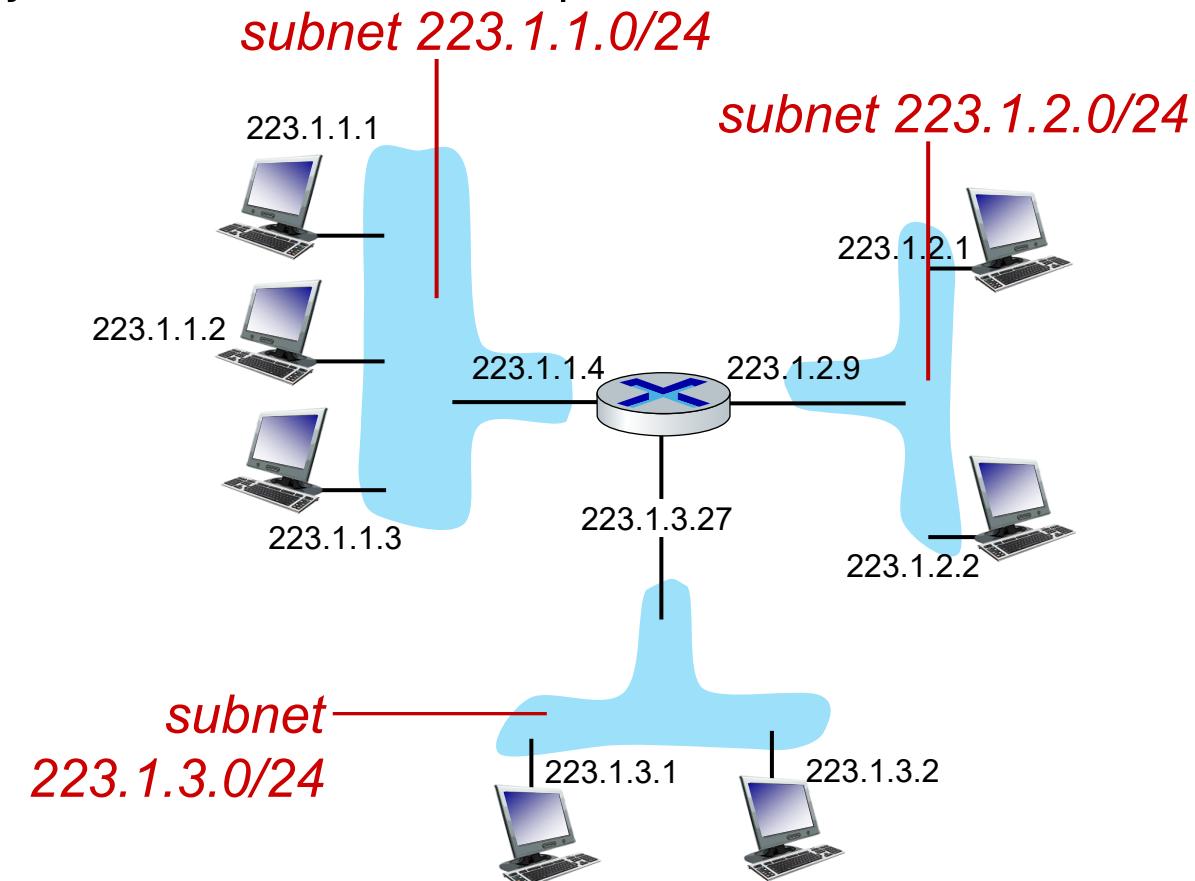
A2: Slash notation (CIDR notation) specifies the network prefix length:

- CIDR: Classless Inter Domain Routing (pronounced “cider”), address format: a.b.c.d/x
- Network prefix: Fixed bits identify the network. Write as dotted quad
- Prefix length: After the slash (/), write how many bits are in the network prefix

Examples:

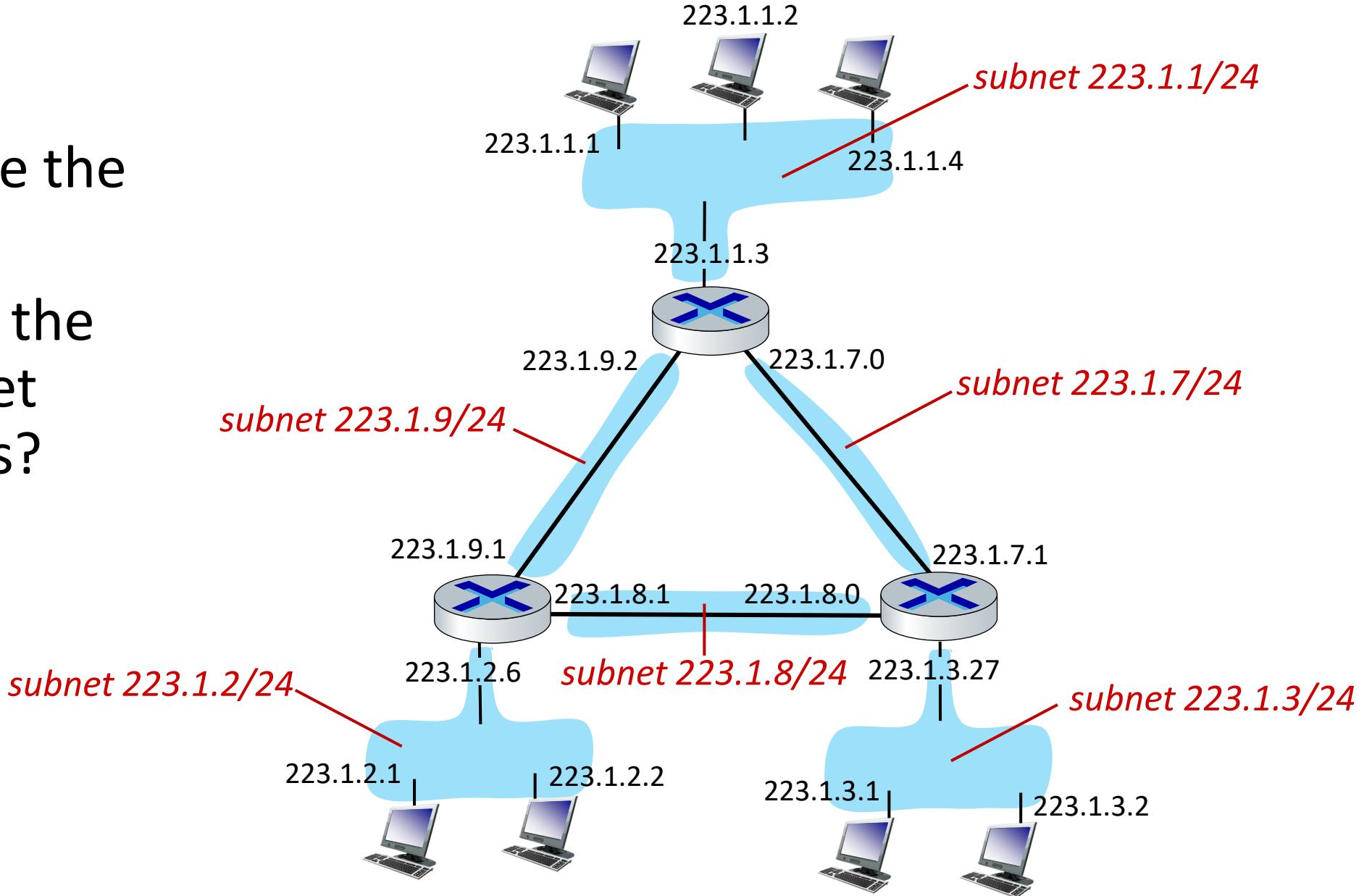
- 220.232.144.0/24 means first 24 bits = (sub)network, last 8 bits = hosts. Host range is 220.232.144.0 – 220.232.144.255 (256 addresses on this subnet)
- 192.168.1.0/29 means 29 bits are fixed. Host range is 192.168.1.0 – 192.168.1.7
- 0.0.0.0/0 means no bits are fixed. All addresses

11011100	11101000	10010001	Set all unfixed bits to 0
11011100	11101000	10010001	00000000	
↓	↓	↓	↓	
220	. 232 .	145 . 0 /24		



Subnets

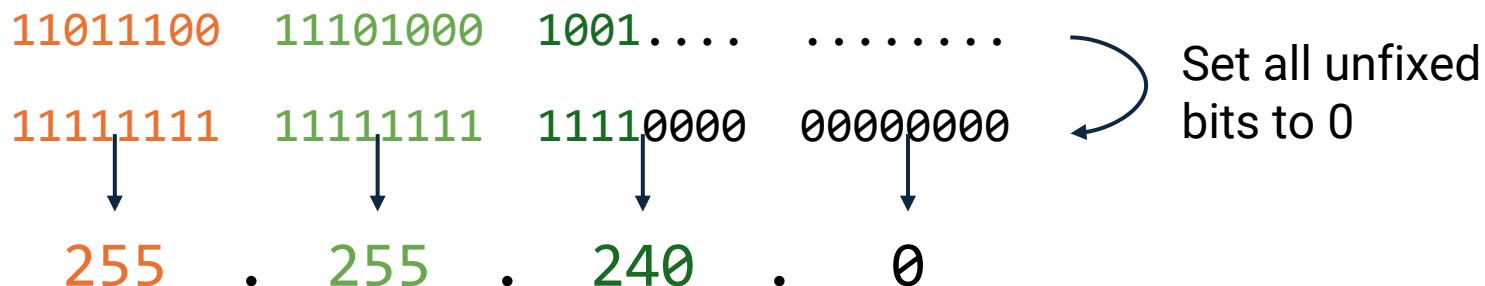
- Where are the subnets?
- What are the /24 subnet addresses?



IP (v4) Addresses

Netmask notation is an older alternative way to specify the network prefix:

- Set network prefix bits to 1, and host bits to 0. Write as dotted quad
- Useful in code: To extract network ID, bitwise AND the address and netmask
- E.g., 220.232.144.93 AND **255.255.255.0** = 220.232.144.0
- Slash notation is more convenient for humans; Netmask still used in some configurations (e.g., older routers, subnet calculators)



Slash notation: 220.232.144.0/20

Netmask notation: 220.232.144.0 (netmask 255.255.240.0)

Roadmap

- *Network Layer: Overview*
- *Routers*
- ***IP: Internet Protocol***
 - *IP Address and **DHCP***
 - *Network Address Translation (NAT)*

IP addresses: how to get one?

That's actually **two** questions:

Q1: How does a *host* get IP address within its network (host part of address)?

A1:

- Hard-coded by sysadmin in config file (e.g., `/etc/rc.config` in UNIX)
- DHCP: Dynamic Host Configuration Protocol: dynamically get address from a network server

Q2: How does *network* get subnet part of IP address?

A2: Gets allocated portion of its provider ISP's address space

DHCP: Dynamic Host Configuration Protocol

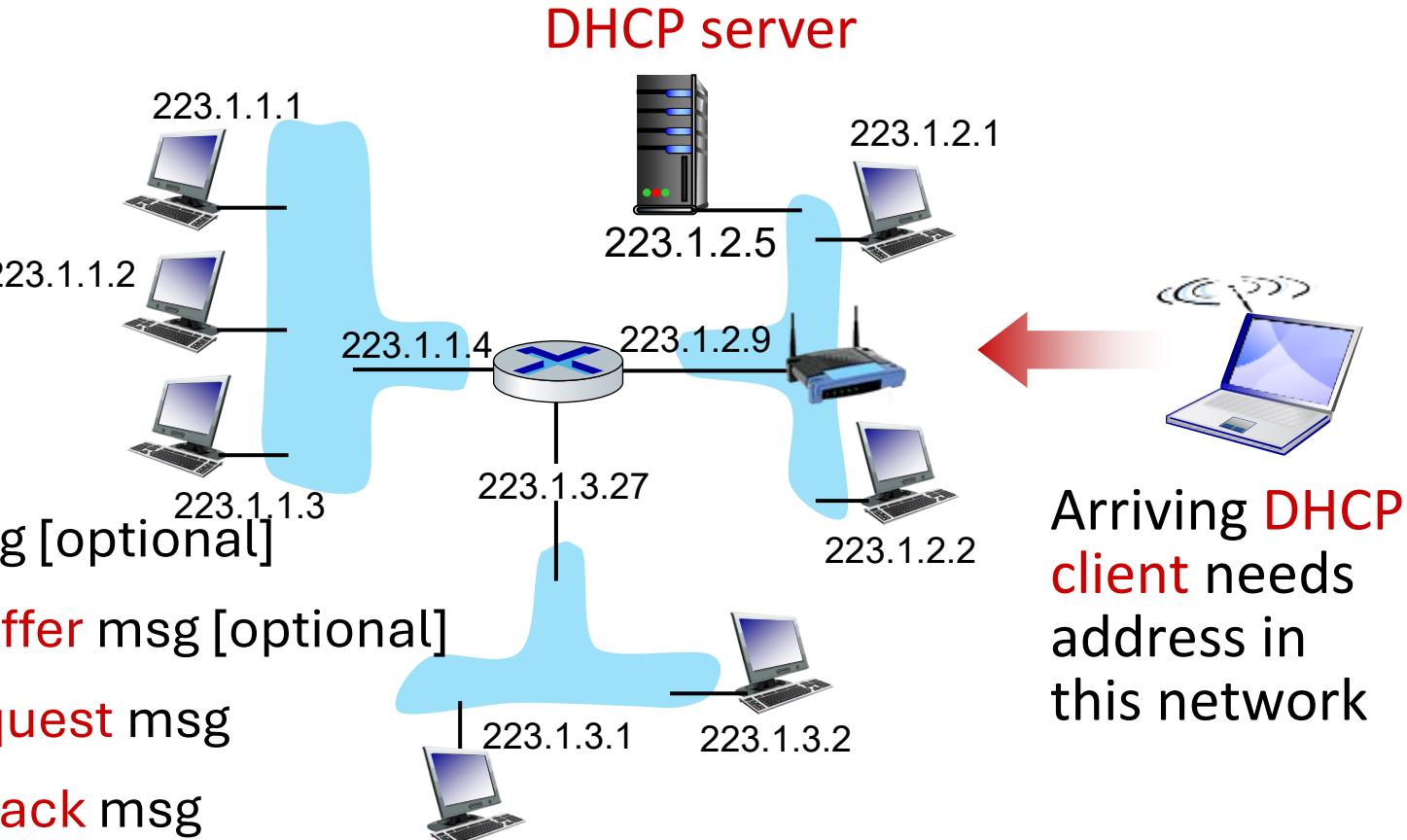
Goal: host *dynamically* obtains IP address from network server when the host “joins” network

- Support for mobile users who join/leave network
- Can renew its lease on address in use
- Allows reuse of addresses (only hold address while connected/on)

DHCP overview:

1. Host broadcasts **DHCP discover** msg [optional]
2. DHCP server responds with **DHCP offer** msg [optional]
3. Host requests IP address: **DHCP request** msg
4. DHCP server sends address: **DHCP ack** msg

Typically, DHCP server will be co-located in router, serving all subnets to which router is attached

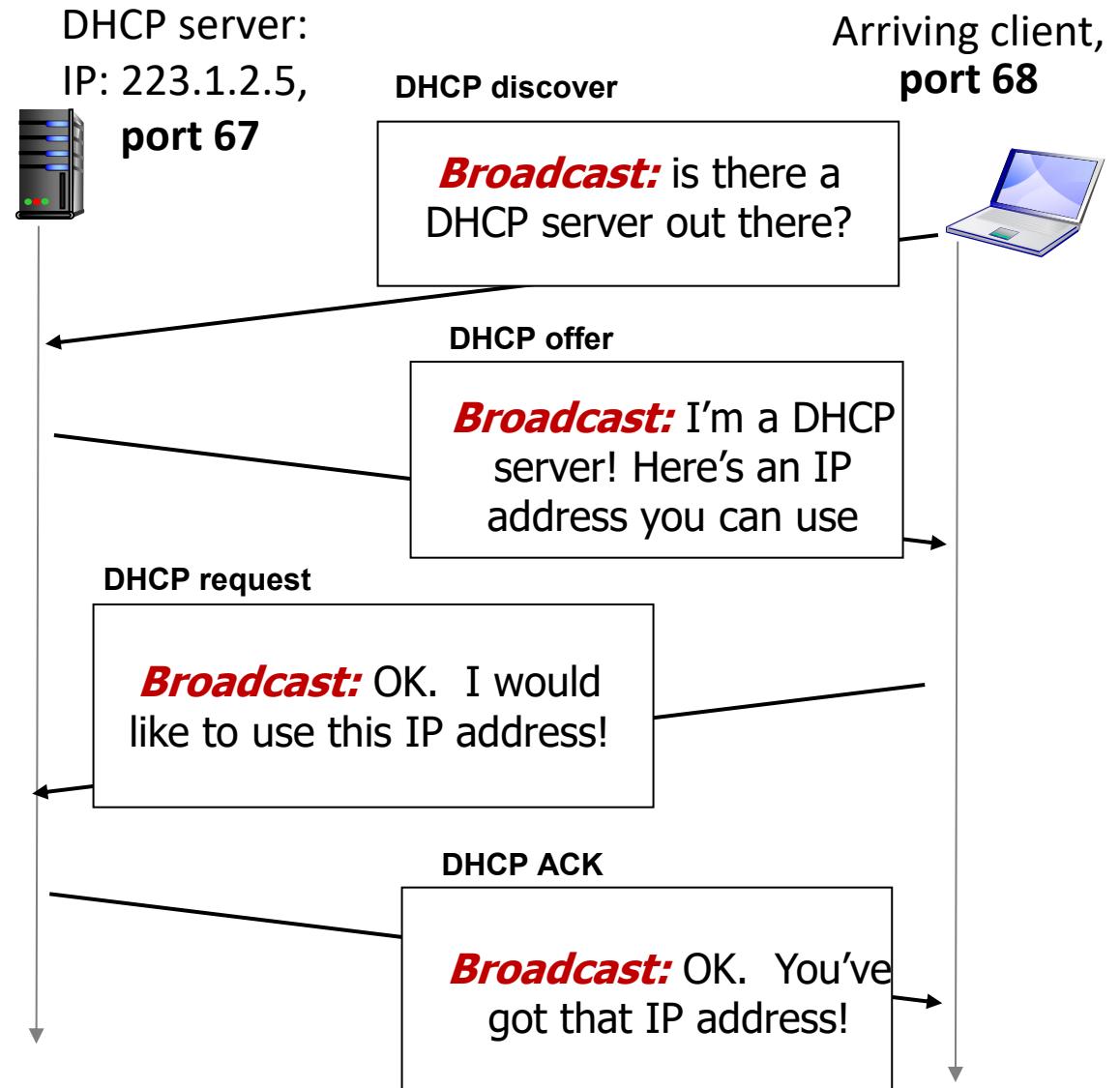


DHCP client-server scenario

The two steps above can be skipped “if a client remembers and wishes to reuse a previously allocated network address” [RFC 2131]

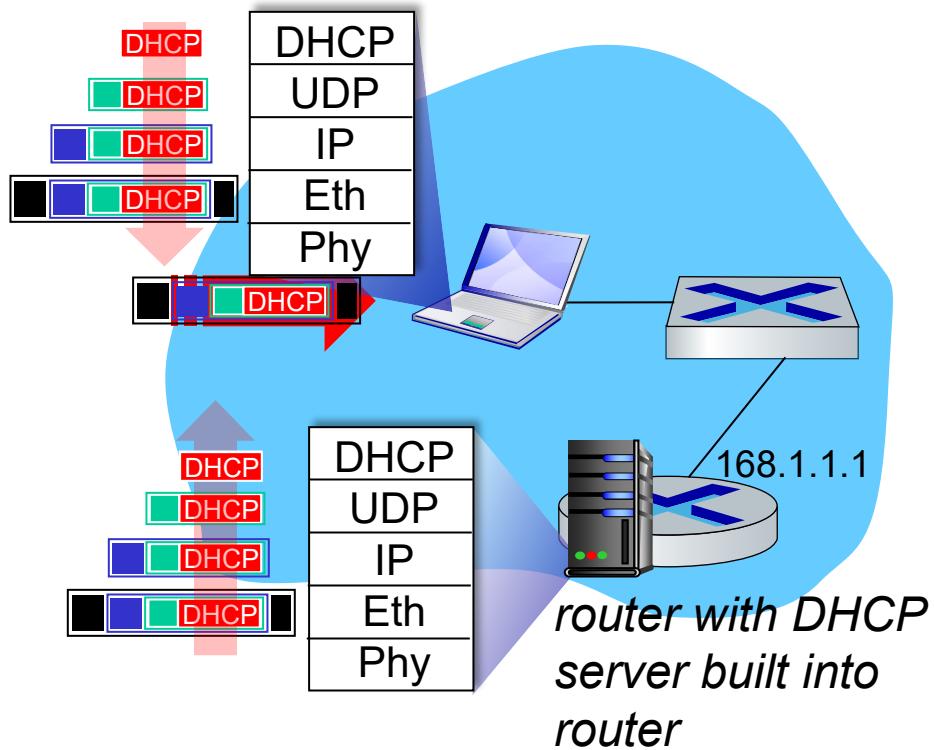
DHCP can return more than just allocated IP address on subnet:

- Name and IP address of DNS sever
- Network mask (indicating network versus host portion of address)
- Address of first-hop router for client



DHCP: Workflow

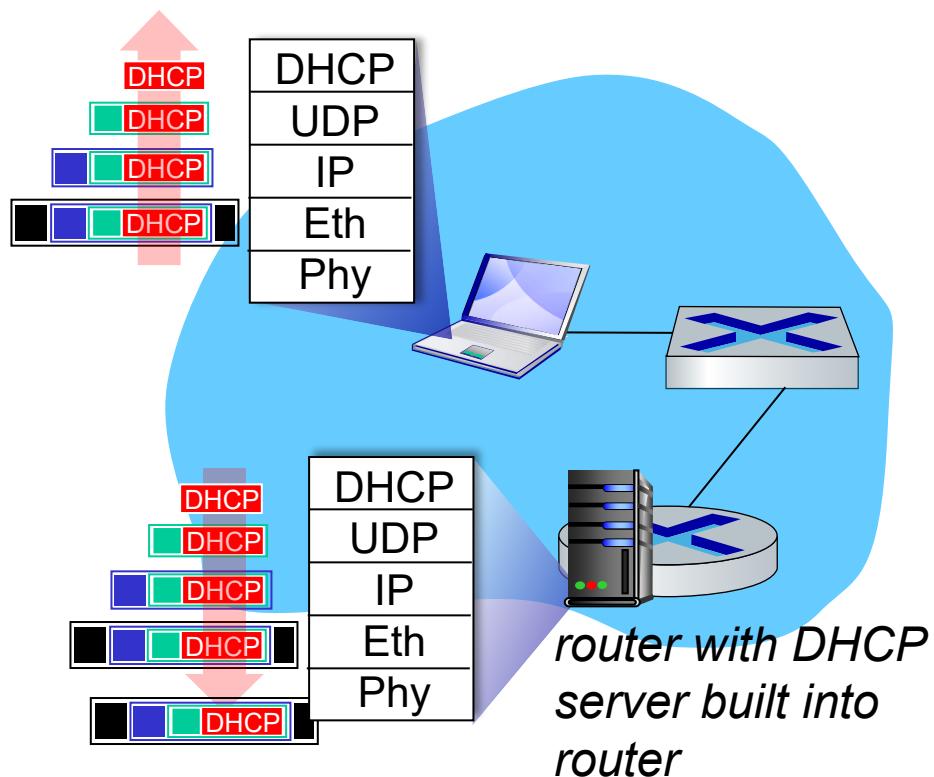
DHCP is an *application layer* protocol, running on top of UDP/IP



- Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server
- DHCP REQUEST message encapsulated in UDP, encapsulated in IP, encapsulated in Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet de-mux'ed to IP de-mux'ed, UDP de-mux'ed to DHCP

DHCP: Workflow

DHCP is an *application layer* protocol, running on top of UDP/IP



- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- Encapsulated DHCP server reply forwarded to client, de-muxing up to DHCP at client
- Client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router

IP Addressing: Last Words ...

IPv4 addresses are 32 bits long. $2^{32} \approx 4$ billion addresses. Is that enough?

ICANN: Internet Corporation for Assigned Names and Numbers
<http://www.icann.org/>

- Allocates IP addresses, through **5 regional registries (RRs)** (who may then allocate to local registries)
- Manages DNS root zone, including delegation of individual TLD (.com, .edu , ...) management
- ICANN allocated last chunk of IPv4 addresses to regional registries in 2011



Introducing IPv6

IPv6 was introduced in 1998 to deal with IPv4 address exhaustion

- Main difference: Addresses are now 128 bits, instead of 32
- Fundamentally, same addressing structure as IPv4
- Some other minor changes (won't discuss here)

Is 128 bits enough?

- $2^{128} \approx 3.4 \times 10^{38}$ possible addresses
- We won't run out again, really?

[IPv5](#) was an experimental protocol from the 1990s. It was never widely implemented

Introducing IPv6

IPv6 uses **hexadecimal** instead of decimal

- 2001:0DB8:CAFE:BEEF:DEAD:1234:5678:9012
- Colon between every 4 hex digits (16 bits)

Shorthand:

- Omit leading zeros per block:
2001:**0**DB8:**0000**:**0000**:**0000**:**0000**:**0001** → 2001:DB8:0:0:0:0:1
- Omit a long string of zeros, once per address:
2001:DB8:**0:0:0:0:1** → 2001:DB8::1

Can still use slash notation for ranges.

- 2001:0DB8::/32 has 32 bits fixed, and 296 addresses.

[IPv5](#) was an experimental protocol from the 1990s. It was never widely implemented

Roadmap

- *Network Layer: Overview*
- *Routers*
- ***IP: Internet Protocol***
 - *IP Address and DHCP*
 - ***Network Address Translation (NAT)***

NAT: Network Address Translation

Problem: IPv4 address is not enough for every host on the Internet.

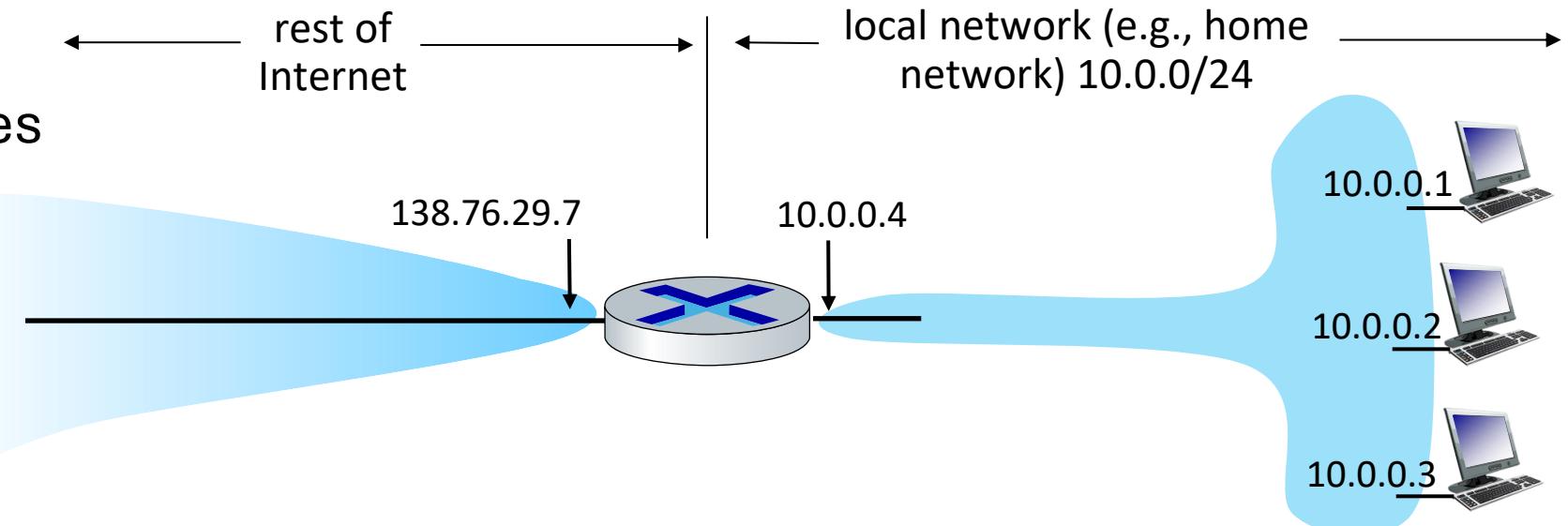
- IPv4 and IPv6 are not compatible with each other
- Requires software and hardware upgrades, from both hosts and routers → IPv6 adoption is slow (and ongoing)

Solution: Use private IP addresses to conserve IP addresses

- Private ranges allocated for networks that don't require Internet access

Weird fact:

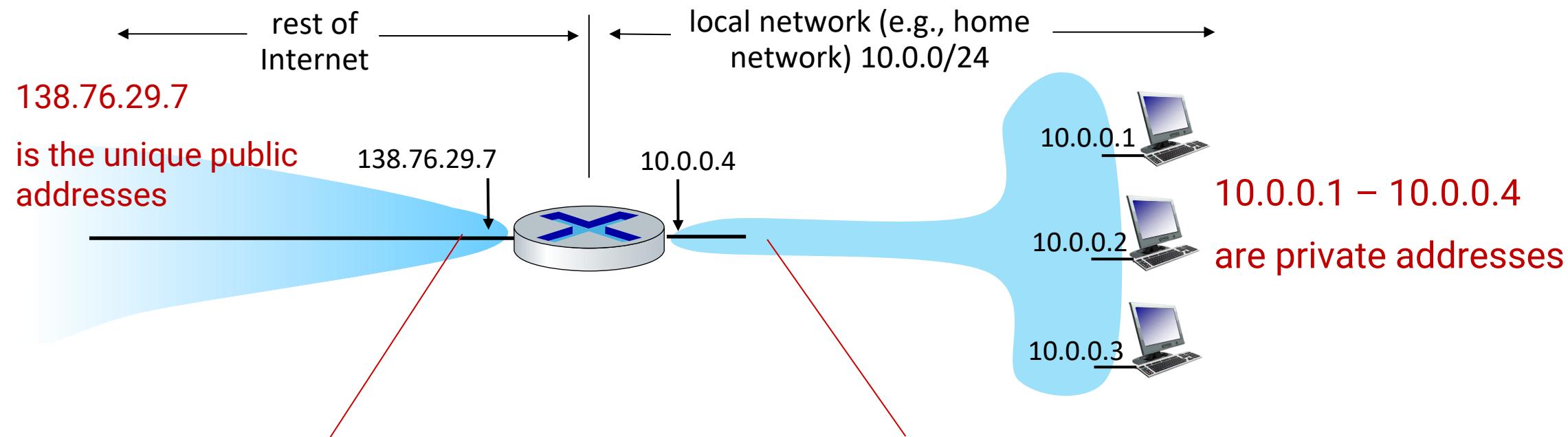
- Your home network uses private IP addresses to conserve addresses
- However, you do need Internet access!



NAT: Network Address Translation

NAT (Network Address Translation): Use a single public IP address to represent many hosts in the local network.

- Outgoing packets: Router changes *private* addresses to the *public* address.
- Incoming packets: Router changes the *public* address back to *private* addresses.



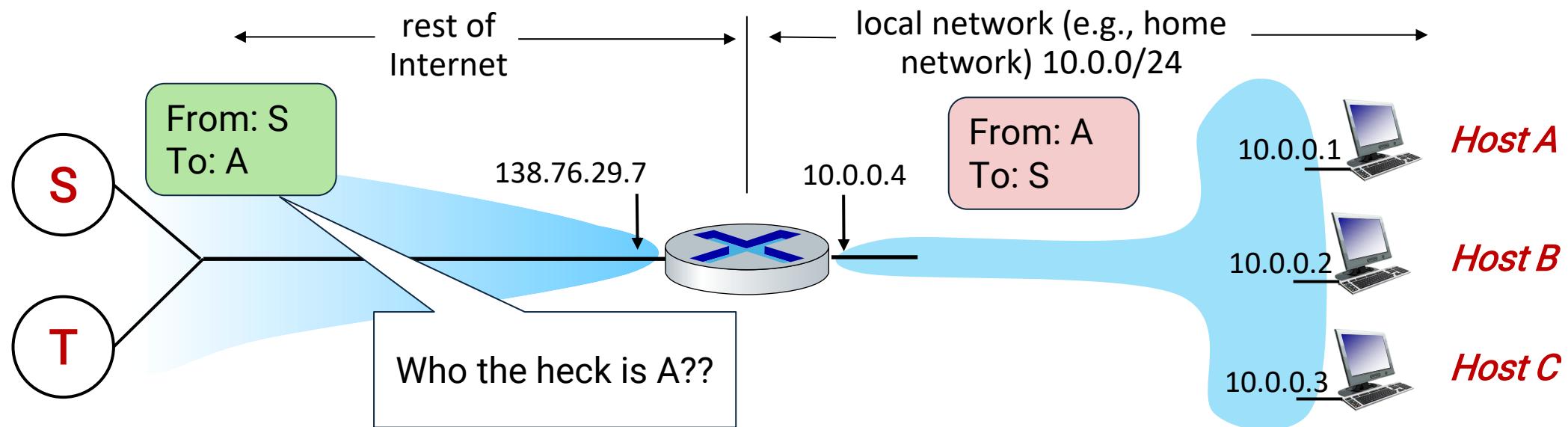
All datagrams *leaving* local network have *same* source NAT IP address: 138.76.29.7, but *different source port numbers*

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

NAT: Network Address Translation

Without NAT, if A sends a packet, replying to A is impossible

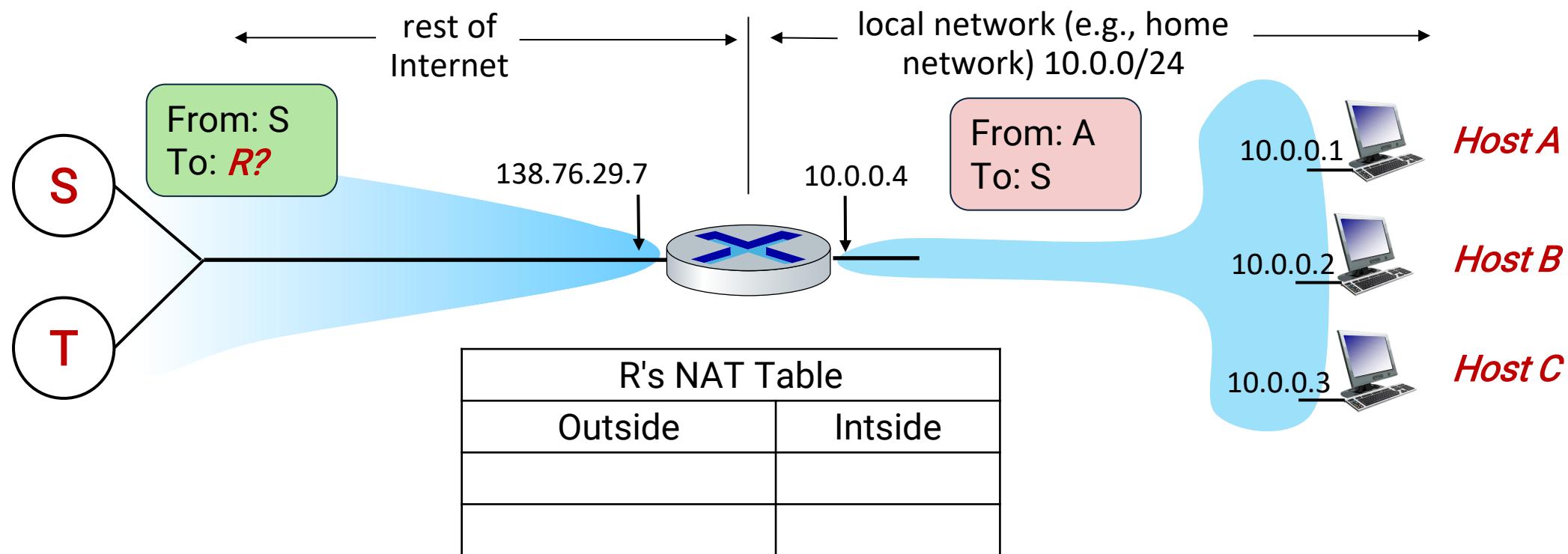
- Because A's IP address is private



NAT: Network Address Translation

With NAT, if A sends an outgoing packet:

- Router ***rewrites the header*** so it's coming from a public IP instead of a private IP
- Router ***keeps a table***, so it remembers where to send any replies
- How to design this table?

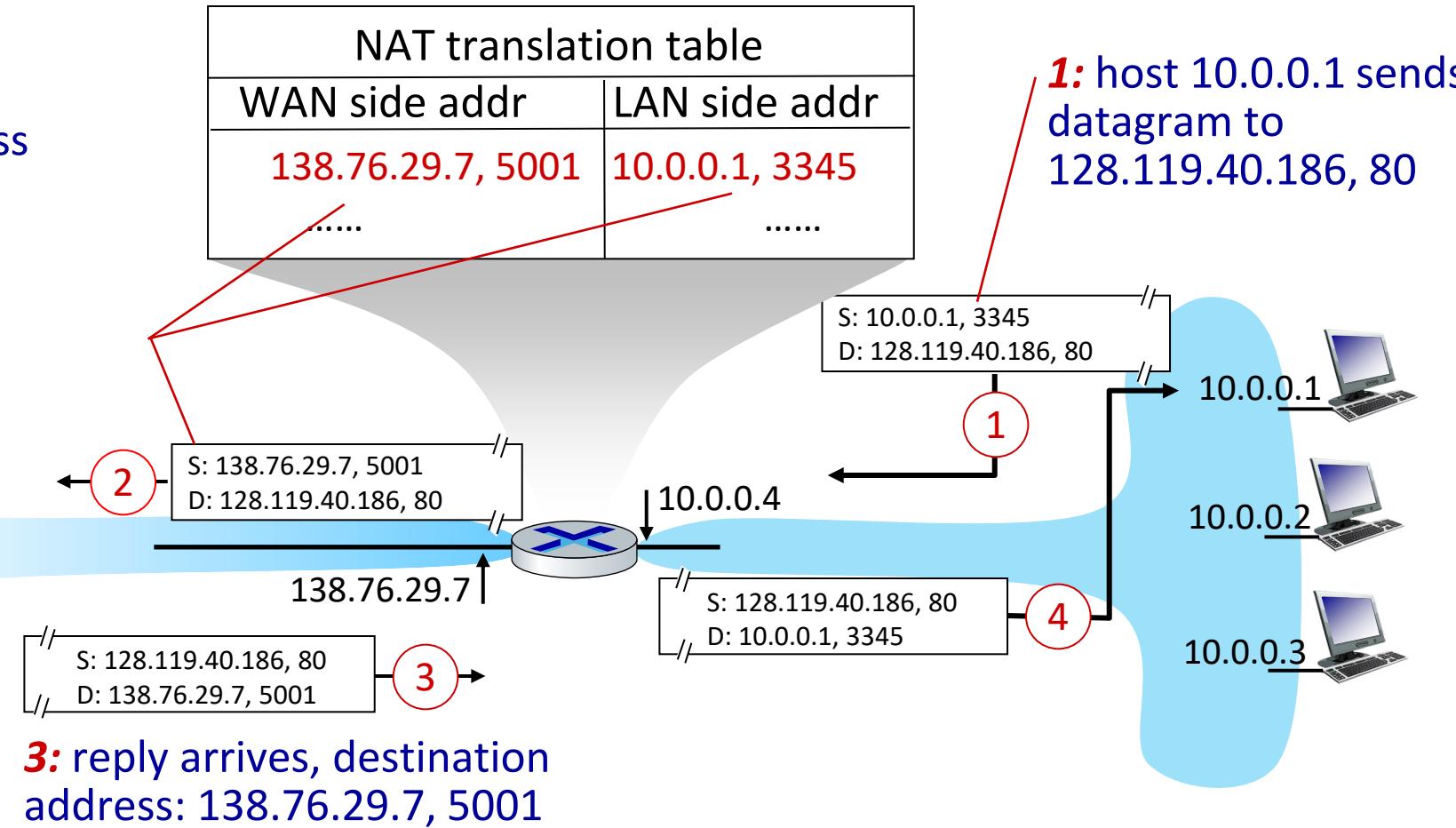


NAT: Network Address Translation

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

Advantages:

- Can change addresses of host in local network without notifying outside world
- Security: devices inside local net not directly addressable, **visible** by outside world



NAT: Network Address Translation

Implementation: NAT router must (transparently):

- Outgoing datagrams: replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
 - Remote clients/servers will respond using (NAT IP address, new port #) as destination address
- Remember (in NAT translation table) every (source IP address, port #) to (NAT IP address, new port #) translation pair
- Incoming datagrams: replace (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

CSC 3511 Security and Networking

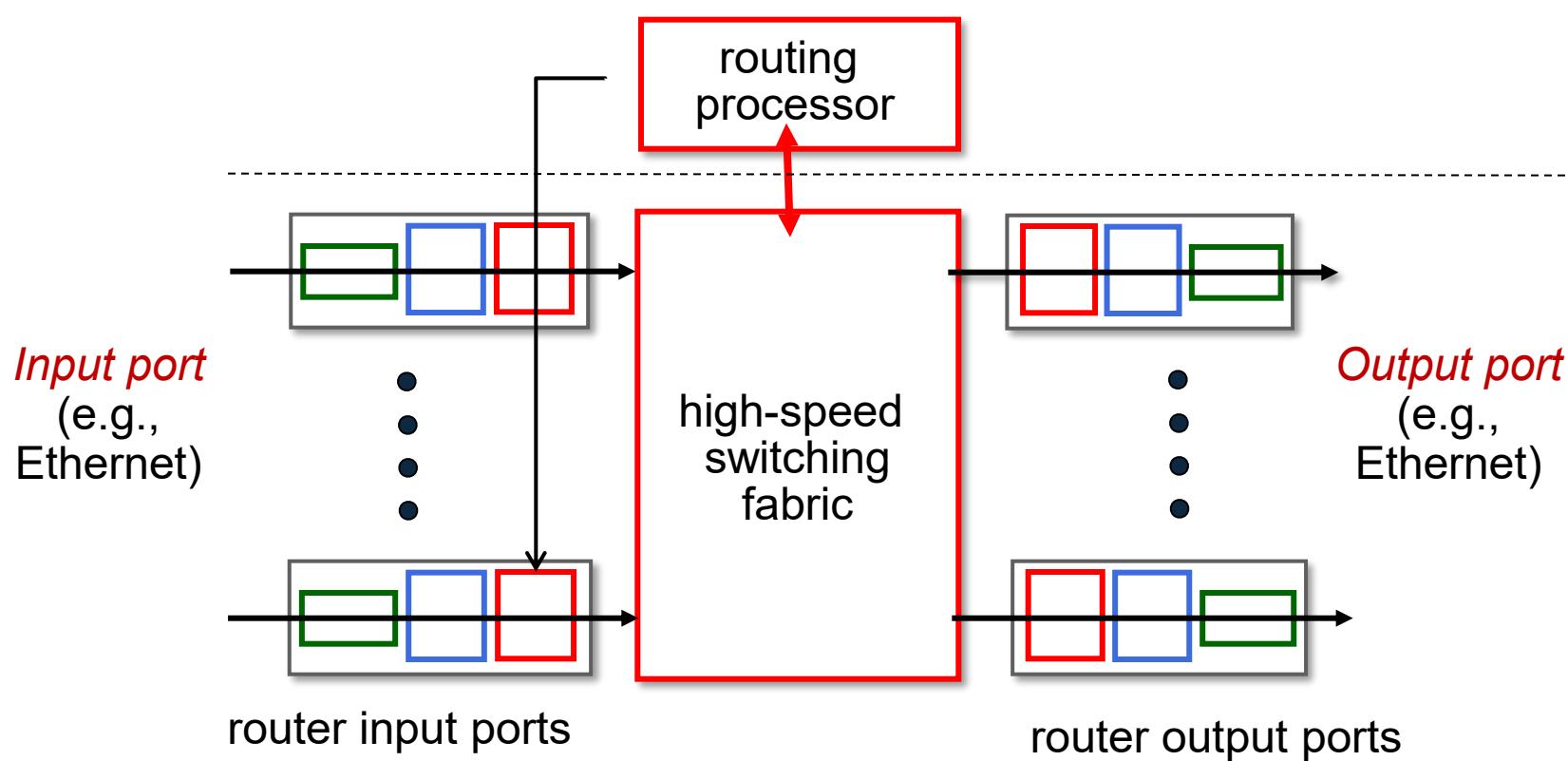
Week 5, Lecture 2: Forwarding and Routing

Roadmap

- ***Packet Forwarding***
 - *Define the Routing Problem*
 - *Routing Protocol: Link State*

Review: Router Architecture

Generic router architecture:



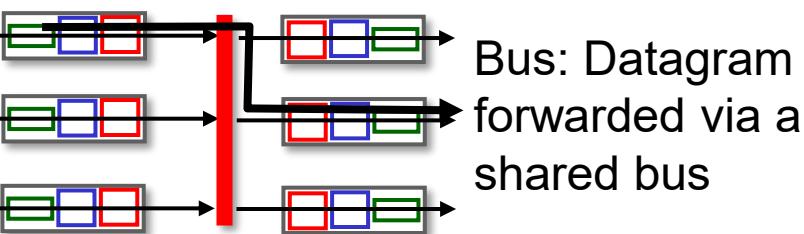
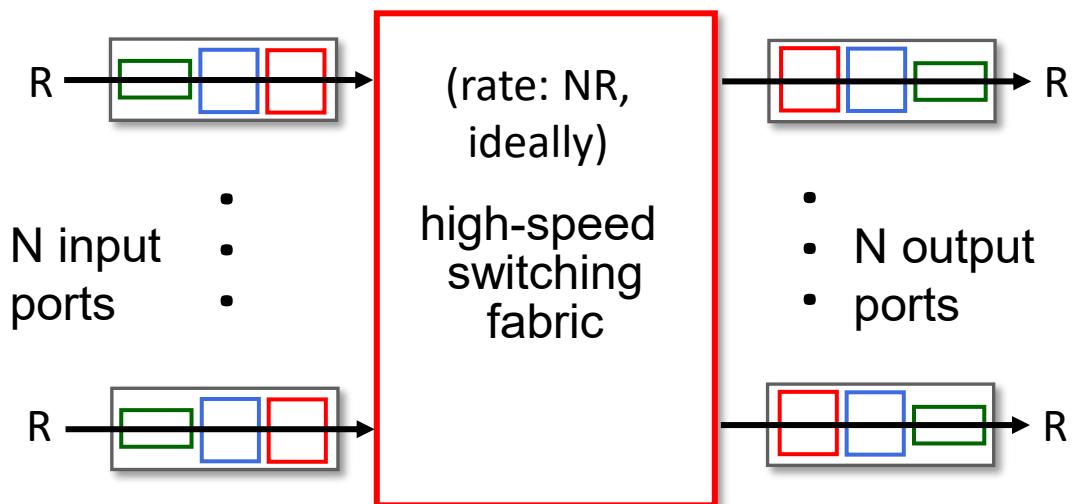
Routing, control plane
(software): operates in
millisecond time frame

Forwarding, data plane
(hardware): operates in nanosecond

Input ports and output ports are connected via switching fabrics

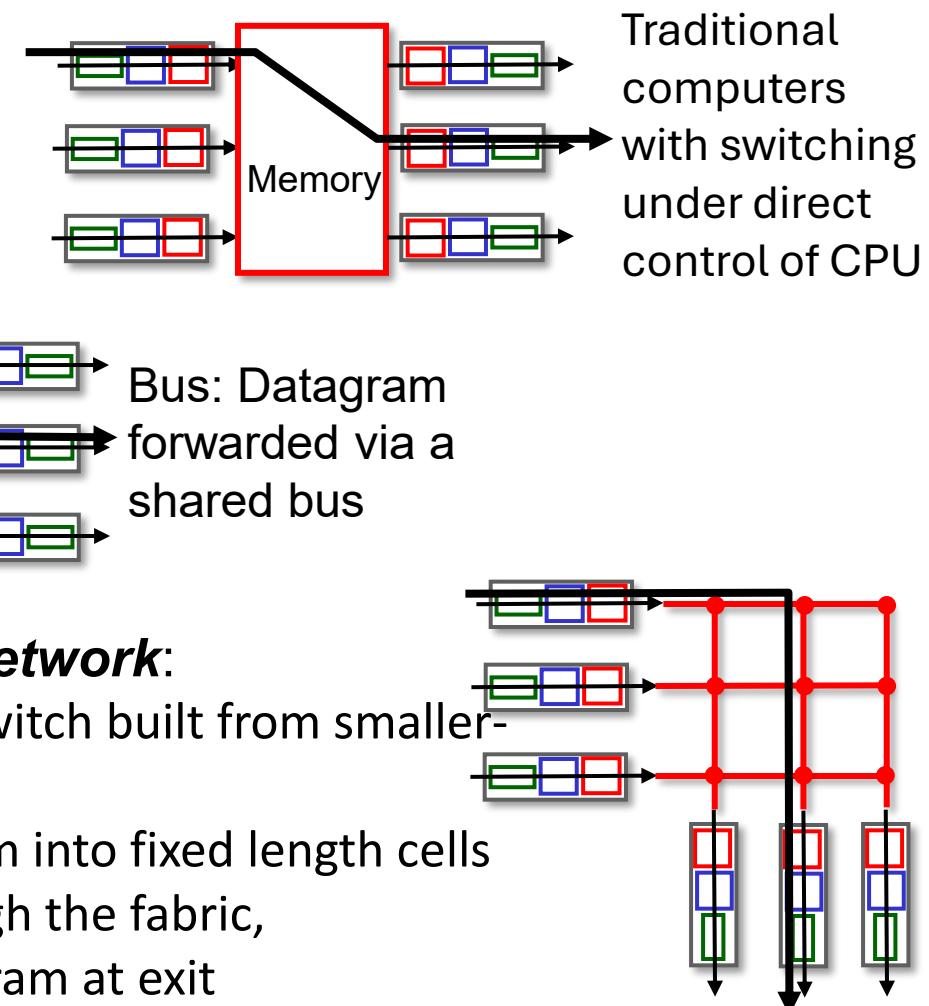
Switching fabrics

- Transfer packet from input link to appropriate output link
- Switching rate: rate at which packets can be transferred from inputs to outputs
 - Often measured as multiple of input/output line rate
 - N inputs: switching rate N times line rate R
- Three major types of switching fabrics:

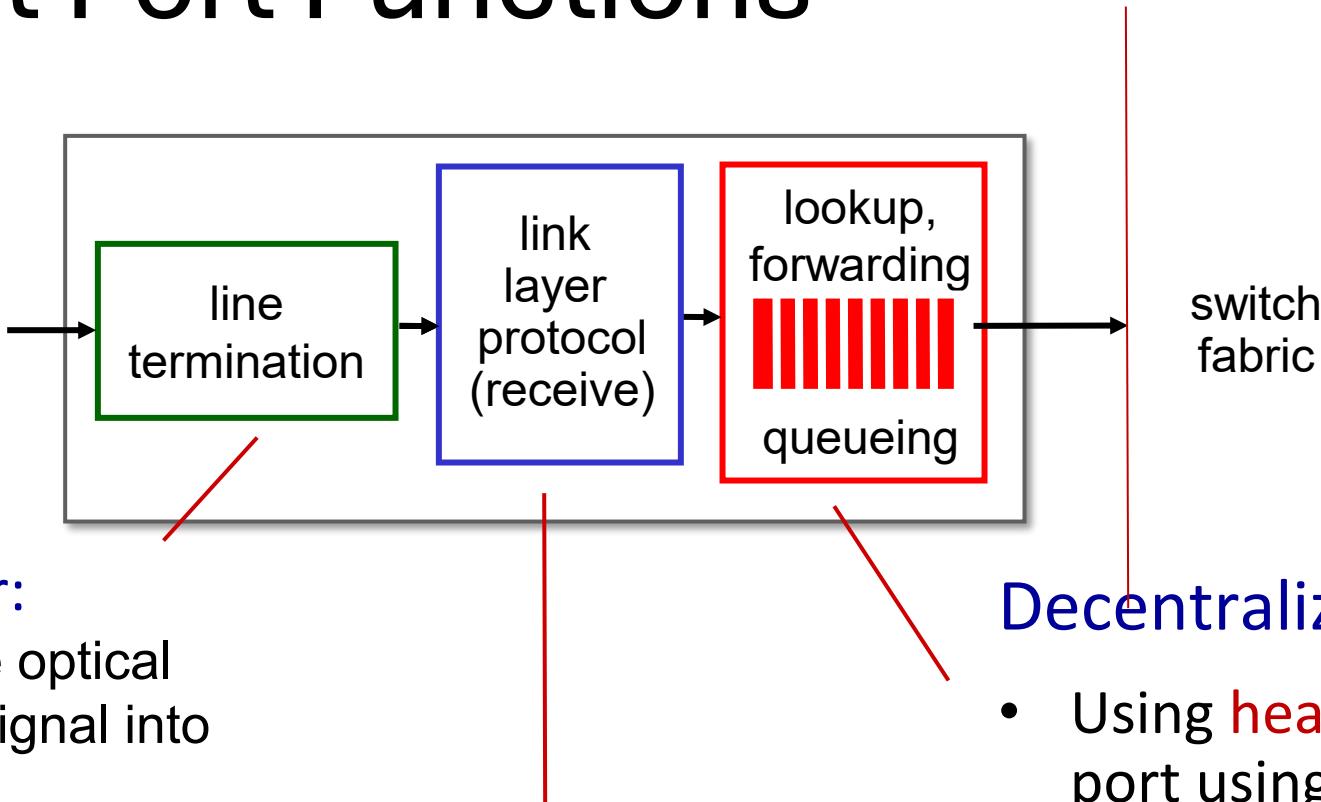


Interconnection network:

- $n \times n$ multistage switch built from smaller-sized switches
- fragment datagram into fixed length cells
- switch cells through the fabric, reassemble datagram at exit



Input Port Functions



Physical layer:

- Decode the optical/electrical signal into 1s and 0s

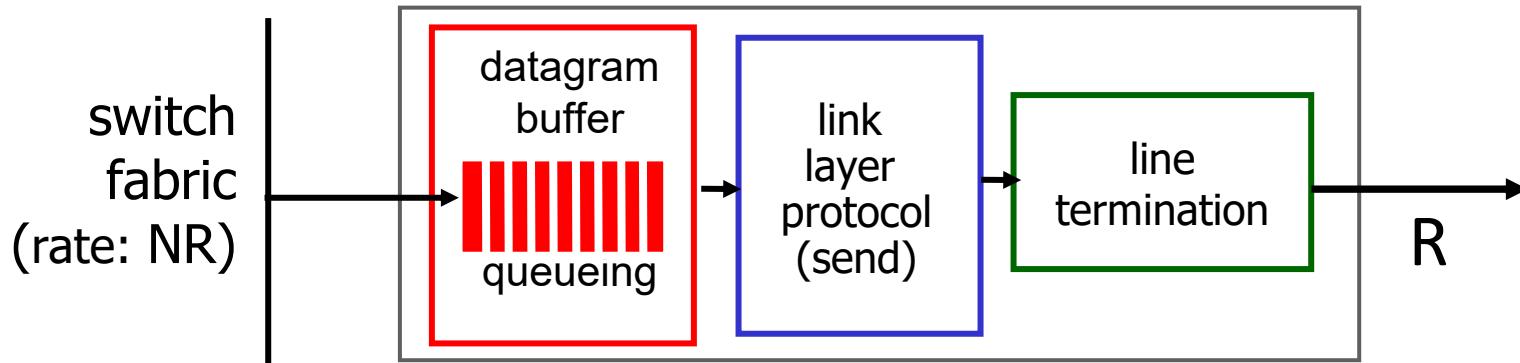
Link layer:

- E.g., Ethernet
- Perform link-layer operations (next week)

Decentralized switching:

- Using **header field** values, lookup output port using forwarding table in input port memory ("*match plus action*")
- ***Destination-based forwarding***: forward based only on destination IP address (traditional)
- ***Generalized forwarding***: forward based on any set of header field values

Output Port Queuing



- *Buffering* required when datagrams arrive from fabric faster than link transmission rate
- *Drop policy*: drop datagrams if no free buffers
- *Scheduling discipline* chooses among queued datagrams for transmission

Datagrams can be lost due to congestion, **lack of buffers**

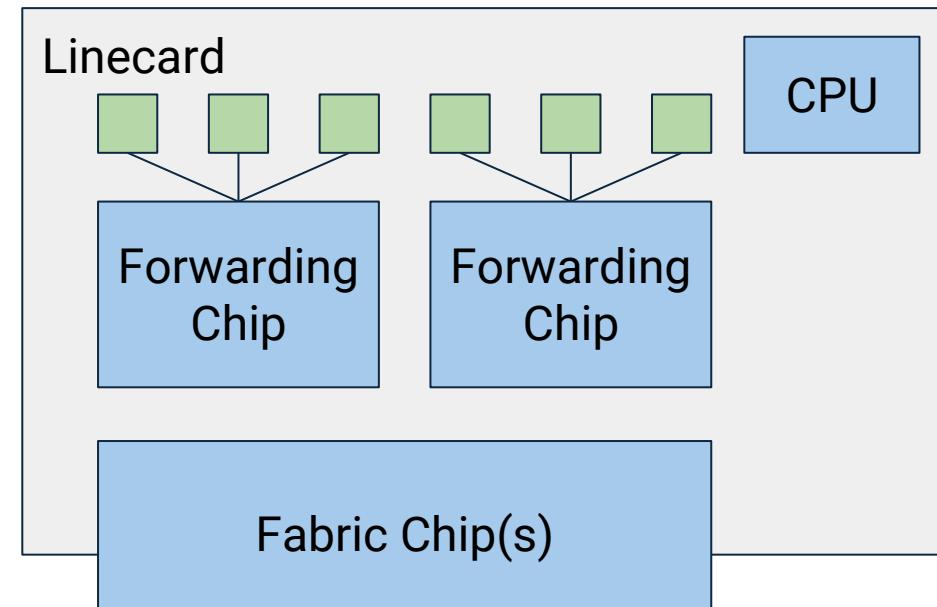
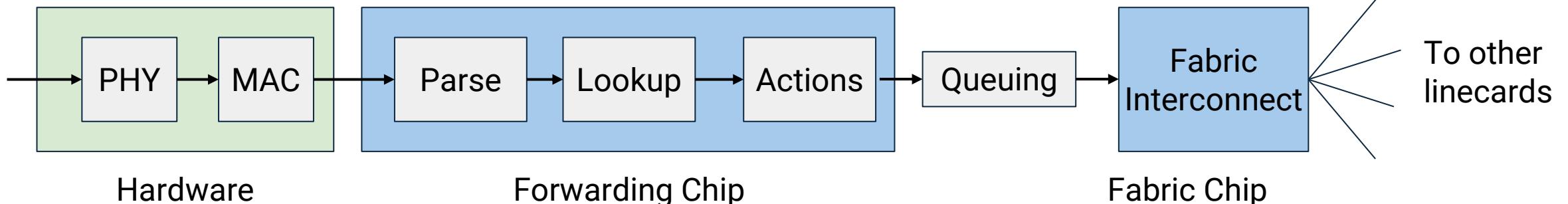
Scheduling policies: performance vs. network neutrality

- First come, first served
- Priority
- Round Robin
- Weighted fair queueing, etc.

Forwarding Pipeline

When a packet arrives, what does the input linecard need to do?

1. Input port: Receive signals and assemble data packets
2. Process the packet:
 1. Parse the packet to understand its headers, e.g. IPv4 or IPv6
 2. Look up the next hop in the ***forwarding table***
 3. ***Update the packet***: Decrement TTL, update checksum, fragment packet if it's too big, etc.
3. Send the packet onwards:
 1. Fabric interconnect chip sends packets to other linecards via inter-chassis links



Efficient Forwarding

The forwarding table is a map (key-value pairs)

How do we do fast lookups?

Challenges:

- Entries can contain a range of addresses, not just one
- Ranges might overlap; A destination can match multiple entries. E.g., output port 5 connects to hosts in 2.1.1.0/24

Naive solution: Write out the whole range:

- Table gets really big
- If a route changes, we have to update tons of entries. E.g., 2.1.1.0/24 connects to port 6?
- We need something smarter

R2's Table	
Destination	Port
2.1.1.0/24	5
↓	
R2's Table	
Destination	Port
2.1.1.0	5
2.1.1.1	5
2.1.1.2	5
2.1.1.3	5
2.1.1.4	5
...	...
2.1.1.252	5
2.1.1.253	5
2.1.1.254	5
2.1.1.255	5

Destination-based Forwarding

- Maps destination IP address ranges to specific output link interfaces in a router
- Router checks the ***destination address*** of each incoming packet and uses the table to determine which link interface to forward the packet to

forwarding table	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through 11001000 00010111 00010000 11111111	0
11001000 00010111 00010000 00000100 through 11001000 00010111 00010000 00000111	3
through 11001000 00010111 00011000 11111111	1
11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111	2
otherwise	3

Q: What happens if ranges don't divide up so nicely?
E.g., packets with a destination address within a subset in this first range should go to interface 3?

A: Split the first address range into multiple pieces and add new entries. Or a more elegant way: ***longest prefix matching***

Longest Prefix Matching

We want a fast implementation of **longest prefix matching**

- Instead of using explicit ranges, use **longest address prefix** that matches destination address
- If the address matches multiple prefixes, take the most specific (longest) match
- If the address matches no prefixes, take the default route
- If there's no default route, drop the packet.

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

examples:

11001000 00010111 00010110 10100001	which interface?
11001000 00010111 00011000 10101010	which interface?

Longest Prefix Matching

- Instead of using explicit ranges, use **longest address prefix** that matches destination address
- If the address matches multiple prefixes, take the most specific (longest) match
- If the address matches no prefixes, take the default route
- If there's no default route, drop the packet.

Destination Address Range	Link interface
11001000 00010111 00010*****	0
11001000 00010111 00011000 *****	1
11001000 1 00011*** *****	2
otherwise	3

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

Longest Prefix Matching

- Instead of using explicit ranges, use **longest address prefix** that matches destination address
- If the address matches multiple prefixes, take the most specific (longest) match
- If the address matches no prefixes, take the default route
- If there's no default route, drop the packet.

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

match!

examples:

11001000 00010111 00010110 10100001	which interface?
11001000 00010111 00011000 10101010	which interface?

Longest Prefix Matching

- Instead of using explicit ranges, use **longest address prefix** that matches destination address
- If the address matches multiple prefixes, take the most specific (longest) match
- If the address matches no prefixes, take the default route
- If there's no default route, drop the packet.

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

match!

examples:

11001000 00010111 00010110 10100001	which interface?
11001000 00010111 00011000 10101010	which interface?

Efficient Longest Prefix Matching with Tries

Is there a map data structure that supports efficient longest prefix match?

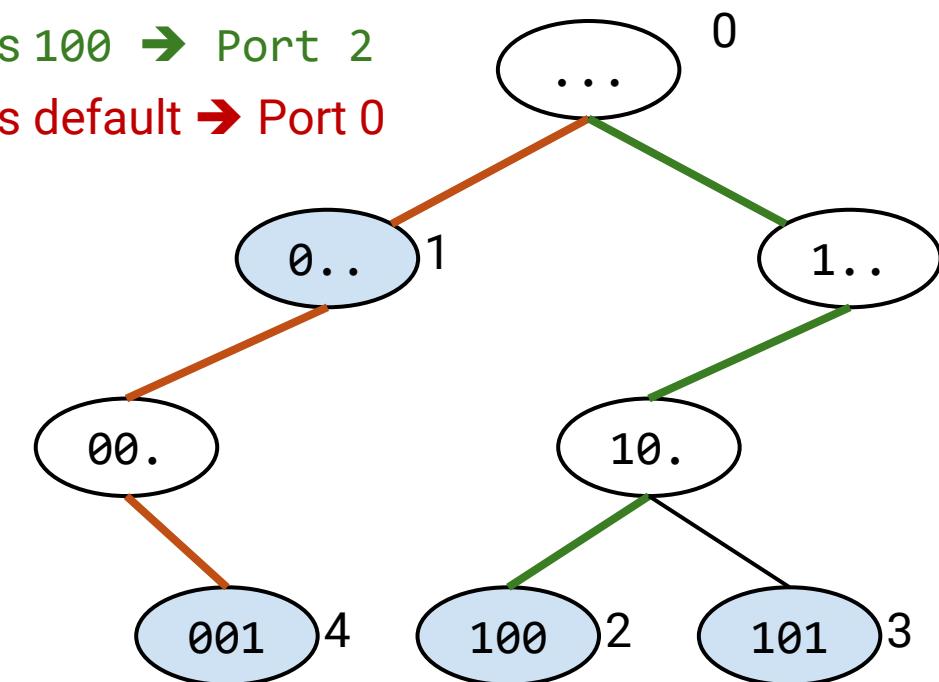
- We can use a more obscure data structure: **Tries**
- Idea: Spell out each key, one letter/digit at a time
- A node is marked blue if walking from root to that node forms a valid key

Longest prefix matching on a trie:

- Start at the root, and spell out the word
- Remember *most recent* (longest) key you see along the way
- Stop when you're done, or fall off the tree. Return the longest key you saw
- If you finish spelling or fall off the tree without seeing any other prefix, the default is returned

Longest prefix of 00100 is 001 → Port 4
Longest prefix of 10000 is 100 → Port 2
Longest prefix of 11100 is default → Port 0

Key	Value
...	0
0..	1
100	2
101	3
001	4



Roadmap

- *Packet Forwarding*
- ***Define the Routing Problem***
- *Routing Protocol: Link State*

Modeling the Network

Two types of machines on the network (*simplified*):

End hosts send and receive packets, to and from other end hosts.

- Example: Your personal computer
- End hosts don't forward intermediate packets.

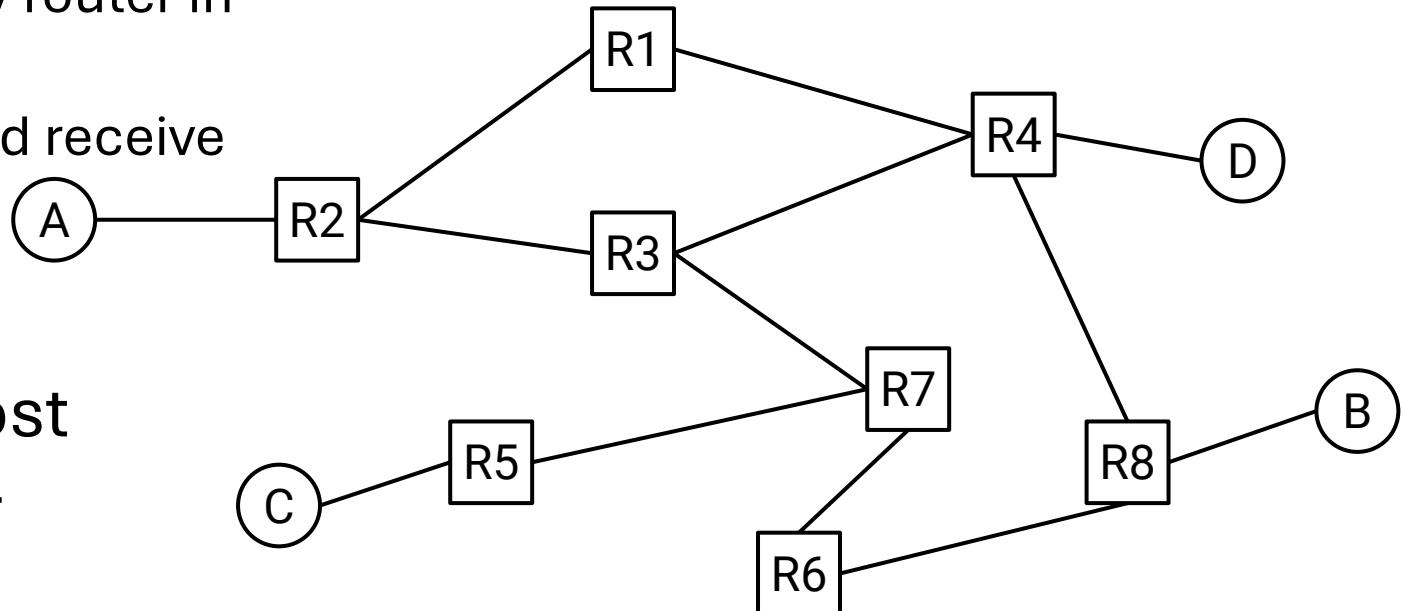
Routers forward intermediate packets

- Example: Your home router, heavy-duty router in a datacenter
- For now, assume routers don't send and receive packets of their own

(A) = End host
R1 = Router

We'll draw the network as a graph:

- Each node represents a machine
- Each edge represents a link. For now, assume every link is undirected and connects exactly 2 machines
- Assume each machine is identified by a unique label (e.g., IP addr)



Modeling the Network

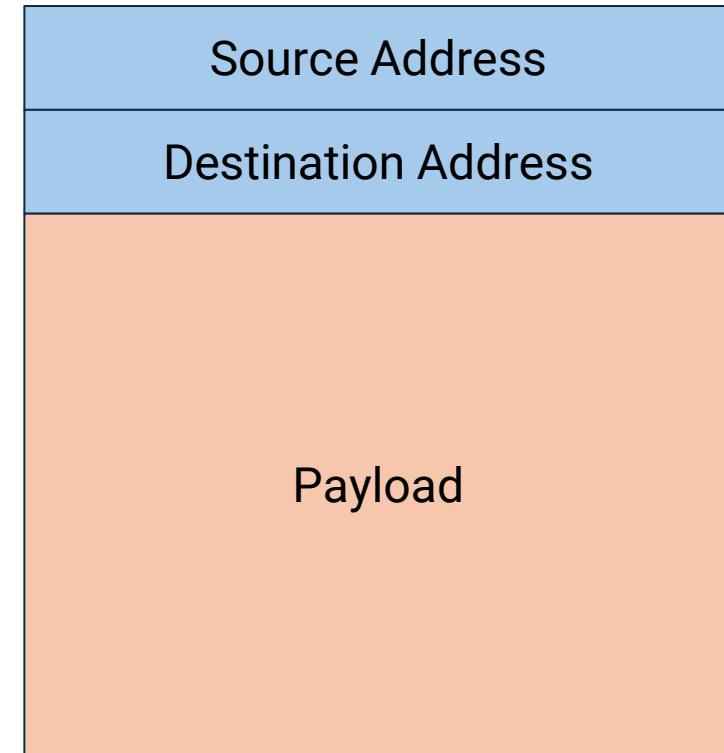
Packets are the basic unit of data sent across the network.

Packets have a header with metadata

- For now, we only care about the *source address* and *destination address* fields

Packets have a payload with the application data

- Example: Website contents, image, etc.
- Routing isn't concerned about the payload. It's just some sequence of 1s and 0s we have to forward



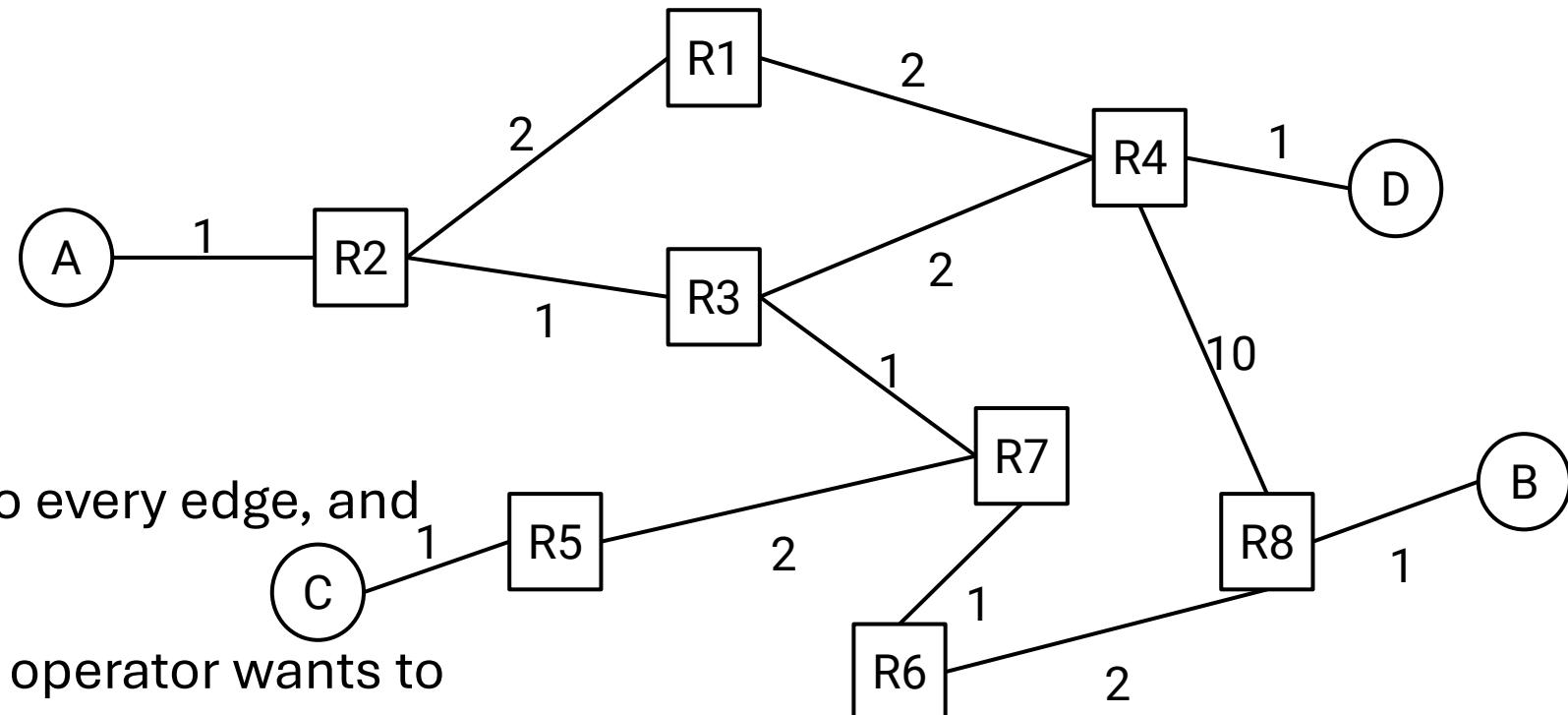
Least-Cost Routing

What makes a routing solution *good*? Many different ways to define good, including:

- Transmission delay
- Price cost
- Unreliability
- Bandwidth constraints

Least-cost routing: Assign costs to every edge, and find paths with lowest cost

- Cost depends on the metric the operator wants to minimize.
- Costs can be arbitrary. Routing protocols don't care where the costs come from.

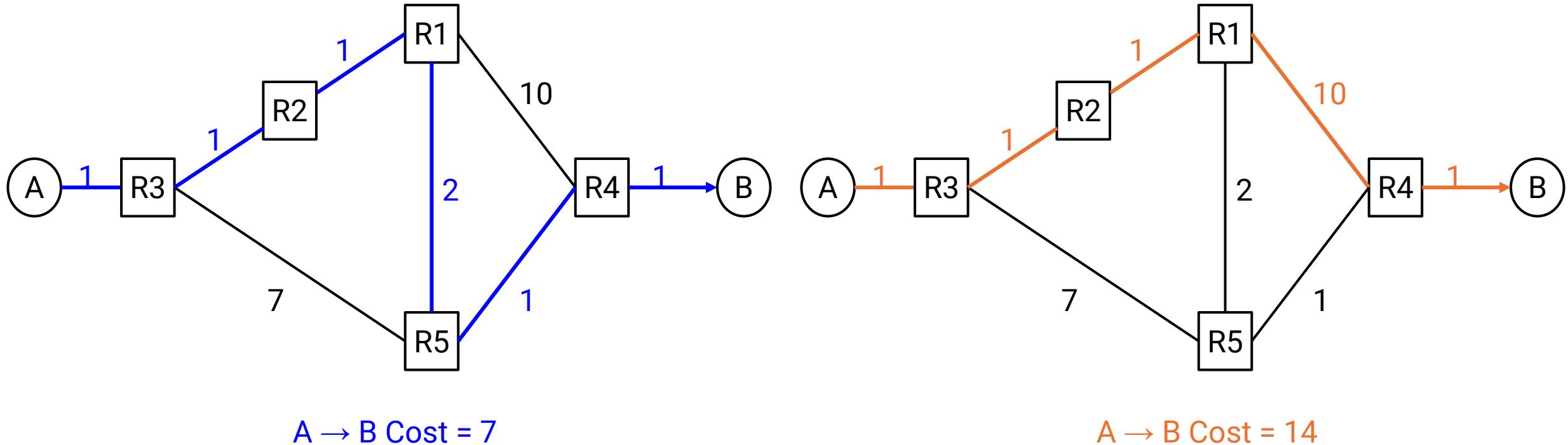


Least-Cost Routing

In least-cost routing, routers should forward packets such that they take the *lowest-cost path* to the destination

Example: If all costs are 1, the protocol finds paths with the fewest hops

- Usually, if edges are unlabeled, assume they have cost 1



Least-Cost Routing

Where do costs come from?

- Costs are local to a router. Each router knows the cost of its own links
- In practice, generally configured by an operator

Properties of costs:

- Costs are always positive integers
 - Consistent with almost any practical metric you'd use: bandwidth, delay, etc.
- Usually, costs are symmetrical
 - $A \rightarrow B$ costs the same as $B \rightarrow A$
 - Exceptions possible in theory (e.g. different upload/download bandwidth)
- These two assumptions will simplify our routing protocols

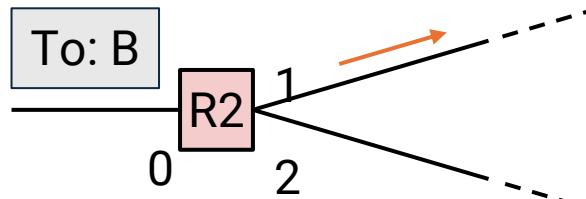
Routing vs. Forwarding

Forwarding:

- Look up packet's destination in table, and send packet to neighbor.
- Inherently *local*. Depends only on arriving packet and local table.
- Occurs every time a packet arrives (nanoseconds)

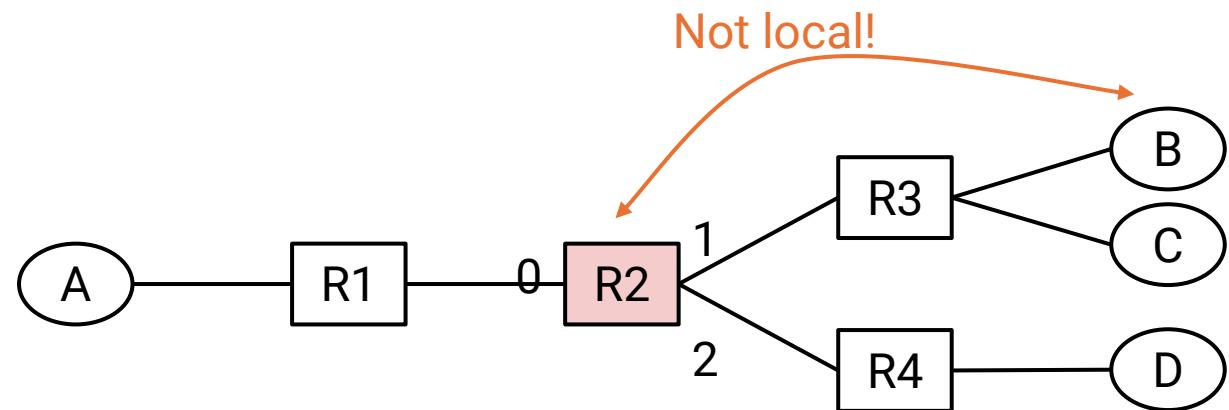
Routing:

- *Communicates* with other routers to determine how to populate tables
- *Inherently global*: Must know about all destinations, not just local ones
- Occurs every time the network changes (e.g. a link fails)



Forwarding is local

R2's Table	
Destination	Port
A	0
B	1
C	1
D	2



Routing is global

Roadmap

- *Packet Forwarding*
- *Define the Routing Problem*
- ***Routing Protocol: Link State***

Routing Protocols: Roadmap

Routing protocols can be classified by:

- *Where* they operate:
 - Intra-domain: routing packets within a local network
 - Inter-domain: routing packets across different networks
- *How* they operate (Distance-vector, link-state, or path-vector)

Today, we'll look at **link-state** protocols

- Major examples:
 - IS-IS (Intermediate System to Intermediate System)
 - OSPF (Open Shortest Path First)

	Intra-domain (Interior Gateway Protocol)	Inter-domain (Exterior Gateway Protocol)
Distance-vector	Routing Information Protocol (RIP)	–
Link-state	IS-IS, OSPF	–
Path-vector	–	BGP

Link-State Protocols: Definition

Link-state protocols:

1. Every router learns the full network graph.
2. Then, each router runs a shortest-path algorithm on the graph to populate the forwarding table.

Link-state:

- ***Global data:*** Each node knows about the full network graph.
- ***Local computation:*** Each node computes the full solution by itself.

How do routers learn this?

What algorithm do we run?

Distance-vector:

- ***Local data:*** Each node only knows about part of the network.
- ***Global (distributed) computation:*** Each node computes part of the solution, working with other nodes.

Link-State Protocols: Definition

Suppose R3 has learned the full network graph

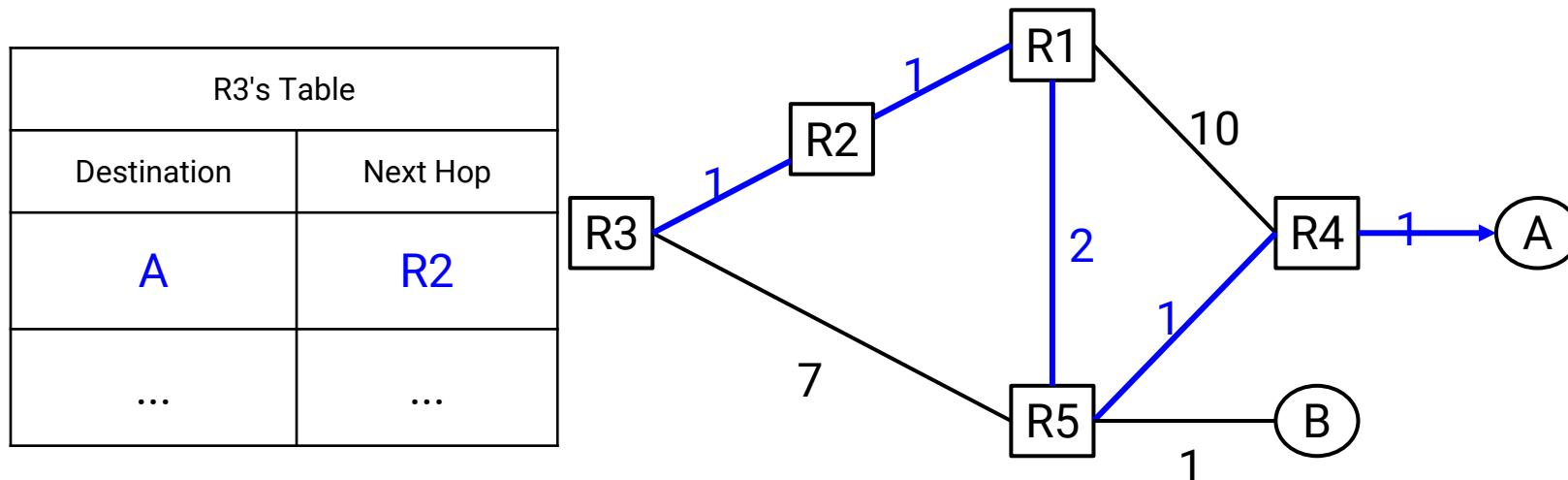
How do routers learn this?

- We know all the links, their status (up or down?), and their costs
- We know about all the destinations

R3 can run a graph algorithm to compute paths

What algorithm do we run?

- We can populate the forwarding table with the next-hop (in the computed path)



Link-State Algorithms

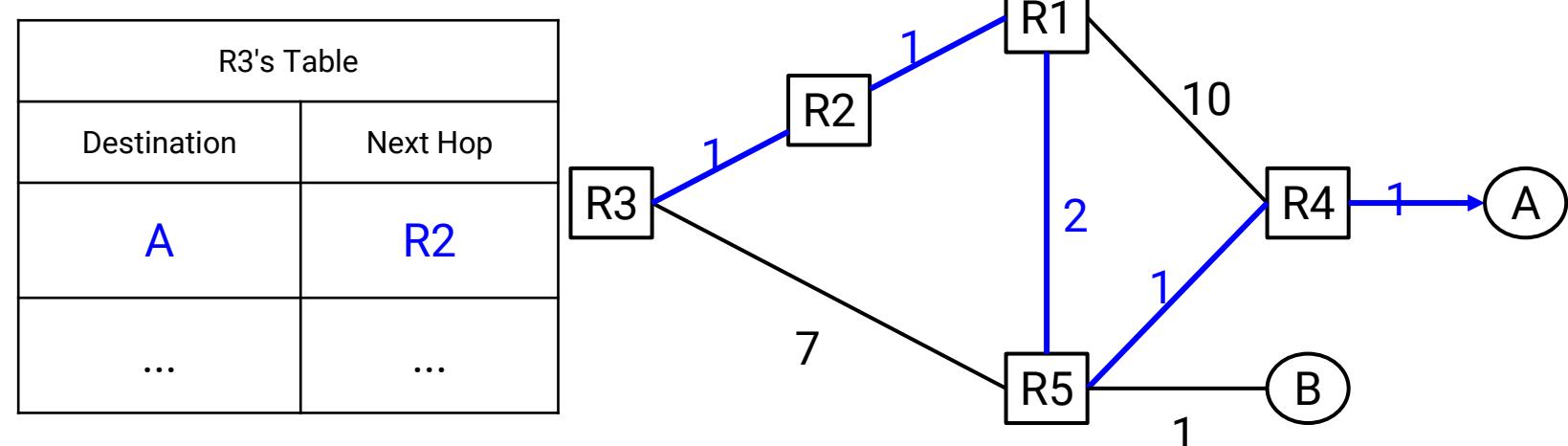
What graph algorithm do we run to compute paths?

Any *single source shortest-path algorithm* will work

- Need to find shortest paths from a single source (the router) to every host

Some possible choices:

- Bellman-Ford (the original serial version)
- **Dijkstra's algorithm**
- Breadth-first search
- Parallel single-source shortest path
- Dynamic shortest path
- Approximate shortest path
- etc.



Steps of Learning Network Graph

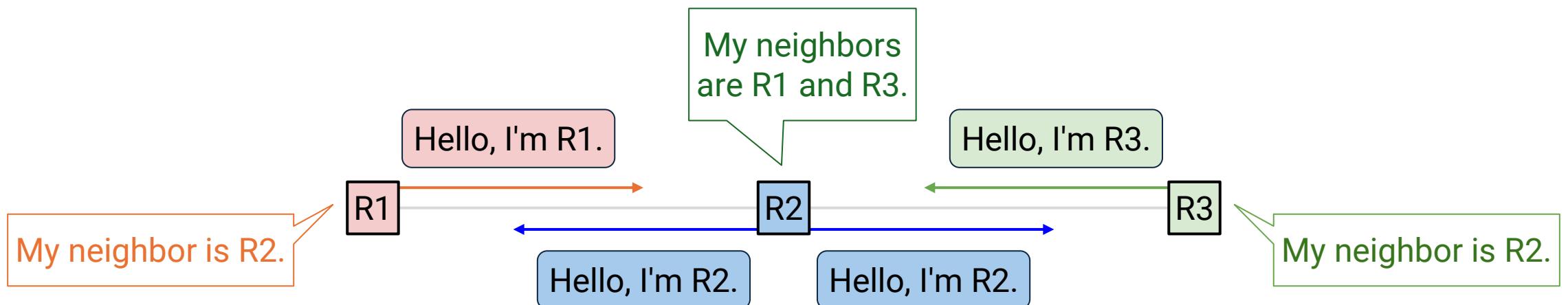
Link-state protocols:

1. Every router learns the full network graph
 - Step 1A: Discover my neighbors
 - Step 1B: Tell everybody about my neighbors
 2. Then, each router runs a shortest-path algorithm on the graph to populate the forwarding table
- How do routers learn this?

Steps of Learning Network Graph (1/2): Learn Neighbors

How do we discover who is adjacent to us and their identity? Say hello!

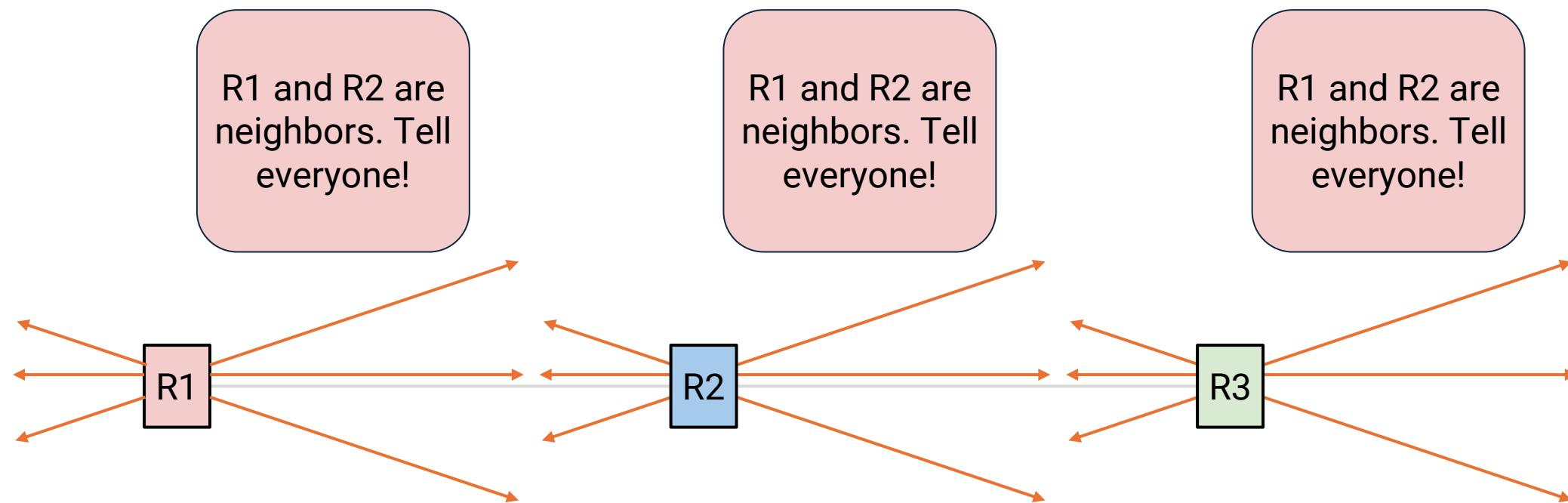
- Routers periodically send *hello* messages to their neighbors.
- If they stop saying hello, assume that they disappeared.
- This helps us learn about our direct neighbors, but not the whole network.
 - R1 does not know about the rest of the network (e.g. R3).



Steps of Learning Network Graph (2/2): Propagate Neighbor Information

How do we learn about the rest of the network, beyond our neighbors?

- Solution: **Flood** information across the network.
- When local information changes, send it to everyone.
- When you receive information from your neighbor, send it to everyone.



Steps of Learning Network Graph

Link-state protocols:

1. Every router learns the full network graph
 - Step 1A: Discover my neighbors
 - Step 1B: Tell everybody about my neighbors
2. Then, each router runs a shortest-path algorithm on the graph to populate the forwarding table

Dijkstra's Shortest Path Algorithm

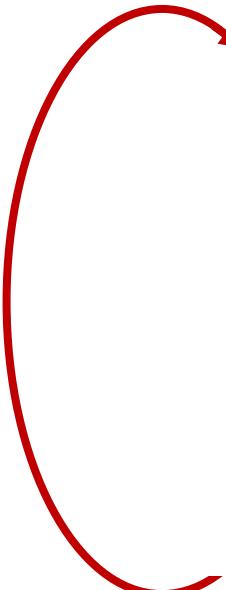
- **Centralized:** network topology, link costs known to *all* nodes
 - Accomplished via “link state broadcast”
 - All nodes have same info
- Computes least cost paths from one node (“source”) to all other nodes
 - Gives *forwarding table* for that node
- **Iterative:** after k iterations, know least cost path to k destinations

notation

- $c_{x,y}$: direct link cost from node x to y ; $= \infty$ if not direct neighbors
- $D(v)$: current estimate of cost of least-cost-path from source to destination v
- $p(v)$: predecessor node along path from source to v
- N' : set of nodes whose least-cost-path *definitively* known

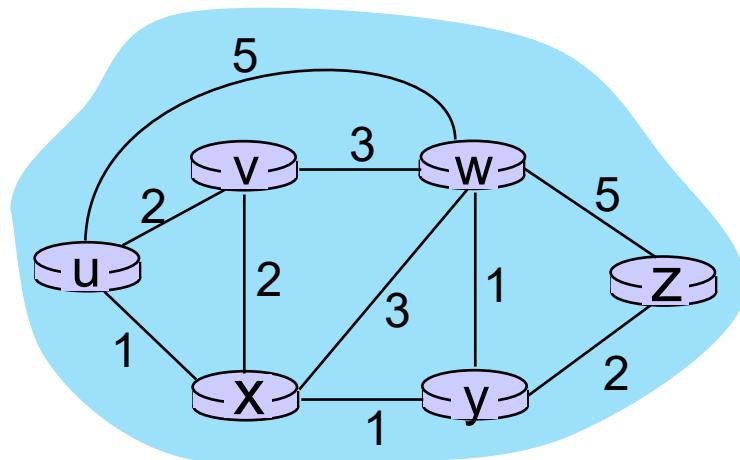
Dijkstra's Shortest Path Algorithm

```
1 Initialization:
2  $N' = \{u\}$                                 /* compute least cost path from u to all other nodes */
3 for all nodes  $v$ 
4   if  $v$  adjacent to  $u$                       /*  $u$  initially knows direct-path-cost only to direct neighbors */
5     then  $D(v) = c_{u,v}$                       /* but may not be minimum cost! */
6   else  $D(v) = \infty$ 
7
8 Loop
9   find a node  $w$  not in  $N'$  such that  $D(w)$  is a minimum
10  add  $w$  to  $N'$ 
11  update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $N'$ :
12     $D(v) = \min(D(v), D(w) + c_{w,v})$ 
13  /* new least-path-cost to  $v$  is either old least-cost-path to  $v$  or known
14  least-cost-path to  $w$  plus direct-cost from  $w$  to  $v$  */
15 until all nodes in  $N'$ 
```



Dijkstra's Algorithm: an Example

Step	N'	V D(v),p(v)	W D(w),p(w)	X D(x),p(x)	Y D(y),p(y)	Z D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1						
2						
3						
4						
5						

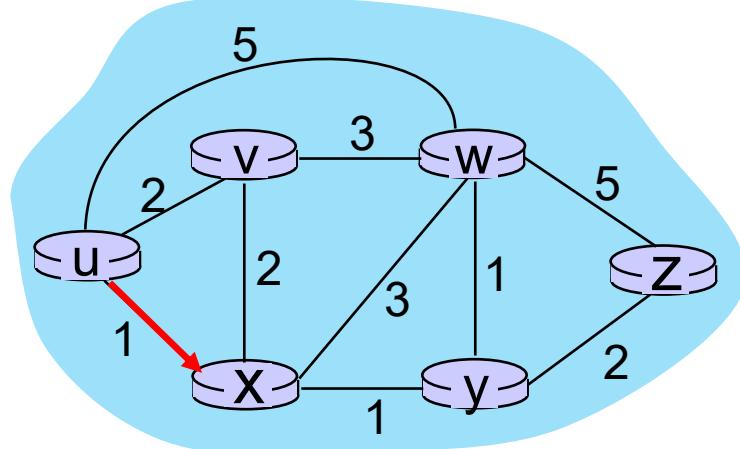


Initialization (step 0):

For all a : if a adjacent to u then $D(a) = c_{u,a}$

Dijkstra's Algorithm: an Example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux					
2						
3						
4						
5						



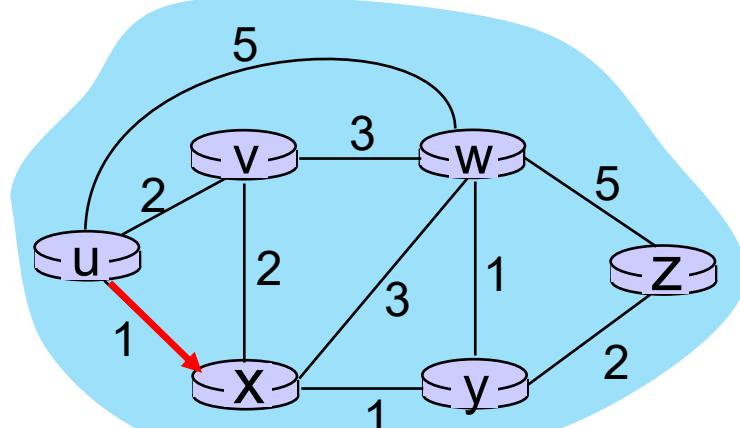
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's Algorithm: an Example

Step	N'	V $D(v), p(v)$	W $D(w), p(w)$	X $D(x), p(x)$	Y $D(y), p(y)$	Z $D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2						
3						
4						
5						

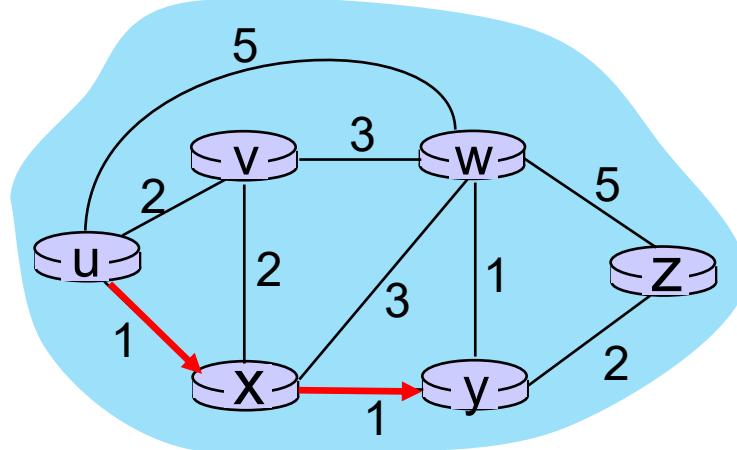


- 8 *Loop*
- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'
- 11 update $D(b)$ for all b adjacent to a and not in N' :
- $$D(b) = \min (D(b), D(a) + c_{a,b})$$
- $D(v) = \min (D(v), D(x) + c_{x,v}) = \min(2, 1+2) = 2$
- $D(w) = \min (D(w), D(x) + c_{x,w}) = \min (5, 1+3) = 4$
- $D(y) = \min (D(y), D(x) + c_{x,y}) = \min(\infty, 1+1) = 2$

NEW!

Dijkstra's Algorithm: an Example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy					
3						
4						
5						



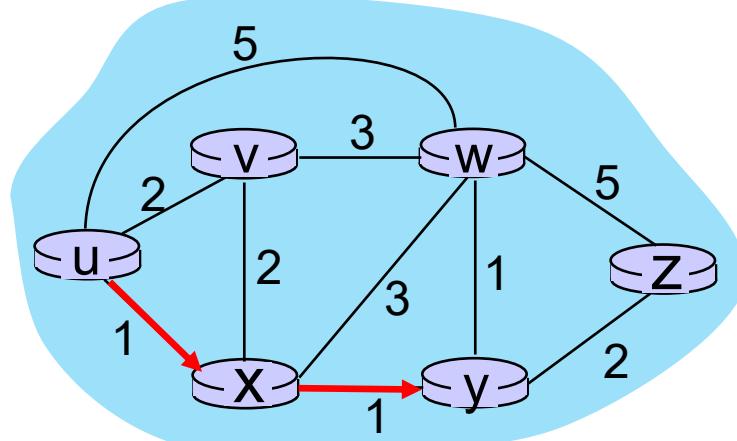
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's Algorithm: an Example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3						
4						
5						

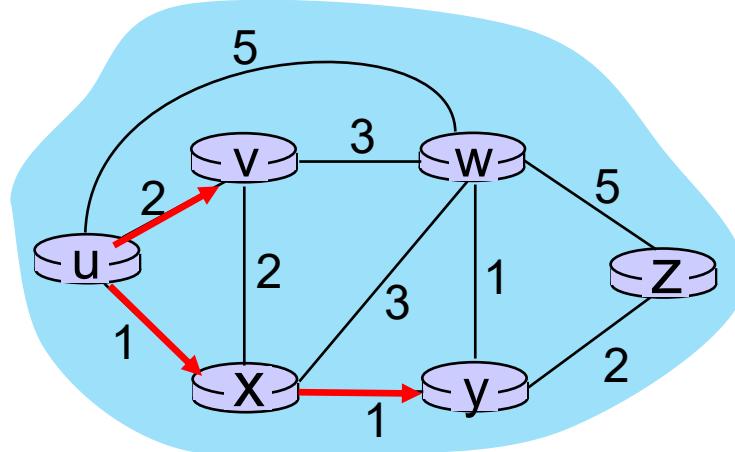


- 8 *Loop*
- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'
- 11 update $D(b)$ for all b adjacent to a and not in N' :
- $$D(b) = \min (D(b), D(a) + c_{a,b})$$
- $D(w) = \min (D(w), D(y) + c_{y,w}) = \min (4, 2+1) = 3$
- $D(z) = \min (D(z), D(y) + c_{y,z}) = \min(\inf, 2+2) = 4$

NEW!

Dijkstra's Algorithm: an Example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv					
4						
5						



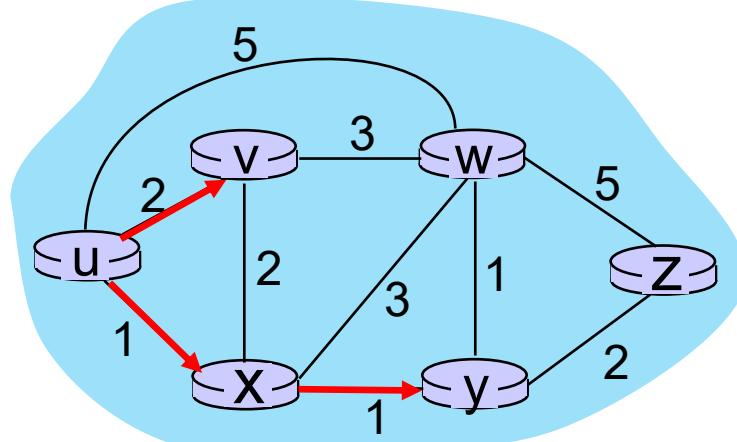
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's Algorithm: an Example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyyv		3,y			4,y
4						
5						



8 Loop

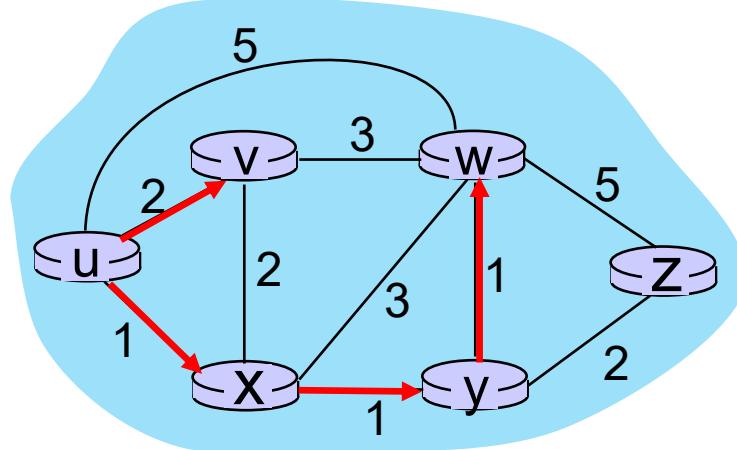
- 9 find a not in N' such that $D(a)$ is a minimum
 10 add a to N'
 11 update $D(b)$ for all b adjacent to a and not in N' :

$$D(b) = \min (D(b), D(a) + c_{a,b})$$

$$D(w) = \min (D(w), D(v) + c_{v,w}) = \min (3, 2+3) = 3$$

Dijkstra's Algorithm: an Example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyy		3,y			4,y
4	uxyvw					
5						



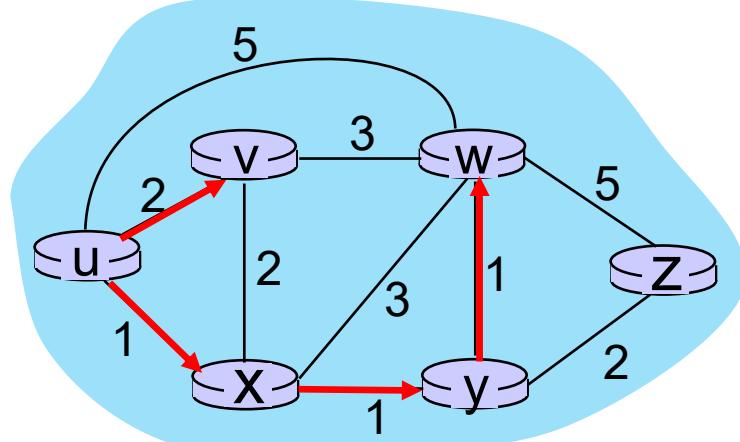
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's Algorithm: an Example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyyv		3,y			4,y
4	uxyvw					4,y
5						



8 Loop

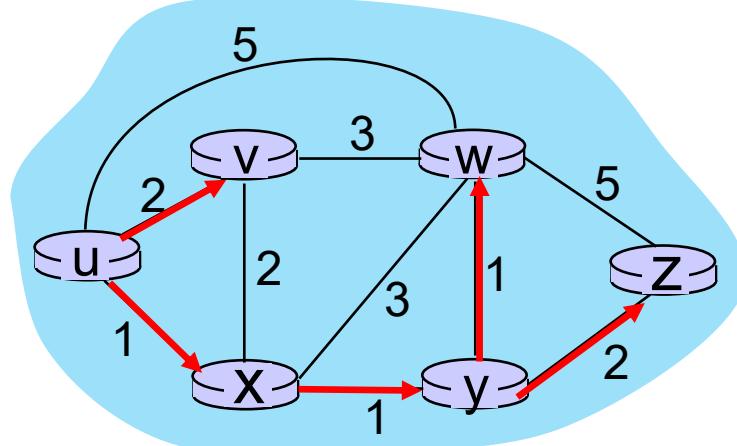
- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'
- 11 update $D(b)$ for all b adjacent to a and not in N' :

$$D(b) = \min (D(b), D(a) + c_{a,b})$$

$$D(z) = \min (D(z), D(w) + c_{w,z}) = \min (4, 3+5) = 4$$

Dijkstra's Algorithm: an Example

Step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					4,y



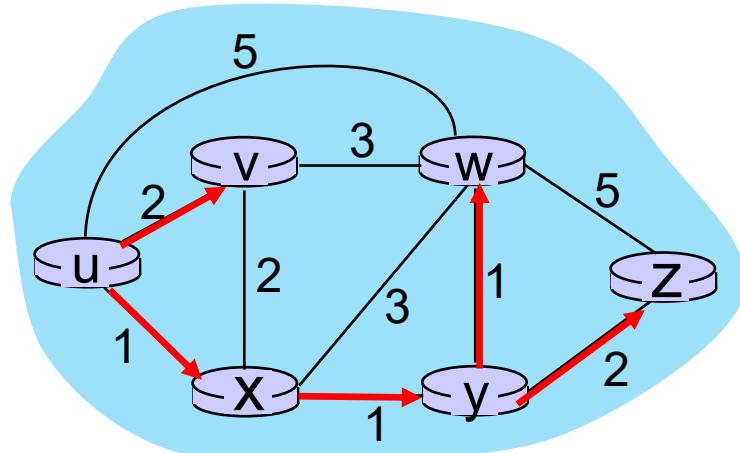
8 Loop

9 find a not in N' such that $D(a)$ is a minimum

10 add a to N'

Dijkstra's Algorithm: an Example

Step	N'	V $D(v), p(v)$	W $D(w), p(w)$	X $D(x), p(x)$	Y $D(y), p(y)$	Z $D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyy		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

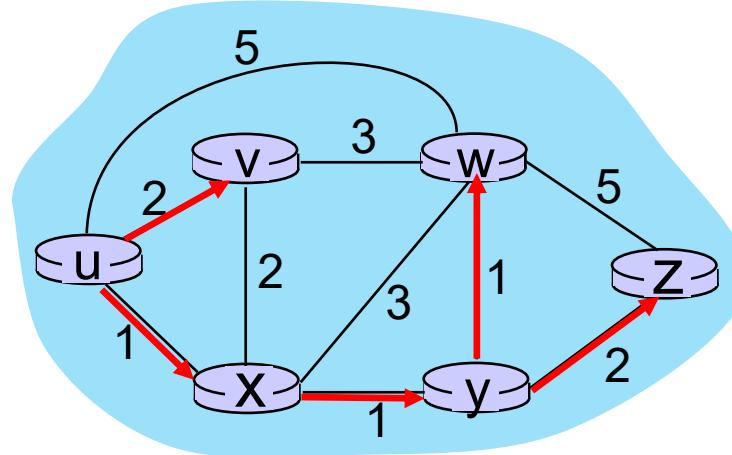


8 Loop

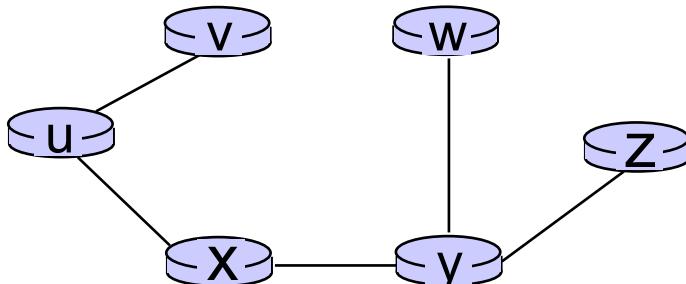
- 9 find a not in N' such that $D(a)$ is a minimum
- 10 add a to N'
- 11 update $D(b)$ for all b adjacent to a and not in N' :

$$D(b) = \min (D(b), D(a) + c_{a,b})$$

Dijkstra's Algorithm: an Example



resulting least-cost-path tree from u:



resulting forwarding table in u:

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

route from u to v directly

route from u to all other destinations via x

Dijkstra's Algorithm: Discussion

Algorithm complexity: n nodes

- Each of n iteration: need to check all nodes, w , not in N
- $n(n+1)/2$ comparisons: $O(n^2)$ complexity
- More efficient implementations possible: $O(n \log n)$

Message complexity:

- Each router must *broadcast* its link state information to other n routers
- Efficient (and interesting!) broadcast algorithms: $O(n)$ link crossings to disseminate a broadcast message from one source
- Each router's message crosses $O(n)$ links: overall message complexity: $O(n^2)$

CSC 3511 Security and Networking

Week 6, Lecture 1: IP Header and IP Security

Roadmap

- *IP(v4) Header*
- *IP Security*

IP Header Design Goals

Think of the IP header as an interface:

- Forwarding and routing: allows the source and intermediate routers to exchange information
- Data exchange: allows the source and destination to exchange information

Design goals:

- Small: A larger IP header makes every single IP packet (IP datagram) larger
- Simple: Routers have to process packets quickly

What fields are in the IP header?

- It depends on what tasks IP needs to perform

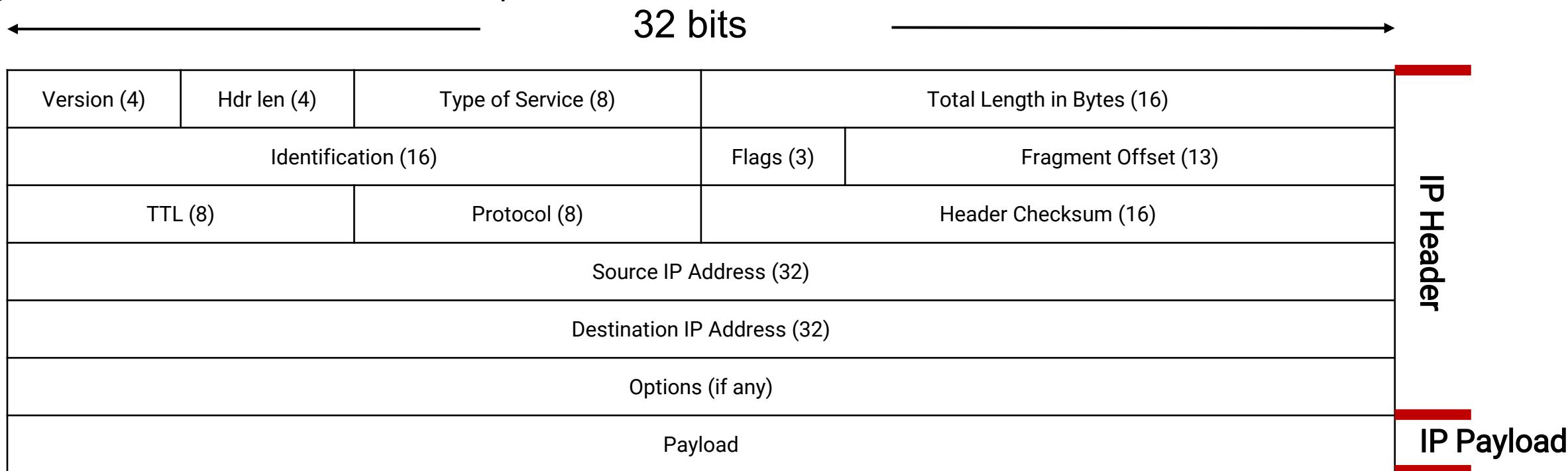
Introducing the IP Header

What IP does (functions):

- **Addressing:** IP addr, addr configuration & translation
- **Encapsulation:** Wraps segments in IP datagrams
- **Forwarding:** Moves packets hop-by-hop
- **Fragmentation:** Breaks/reassembles packets

Our goal:

- Understand the fields in IP header
- Understand the design choices in making this header



IP Tasks

IP functionality can be classified into 6 tasks. Some tasks are done by only routers or the destination host; Others are done by both:

1. Parse the packet (both router and destination)
2. Forward packet to the next hop (router only)
3. Tell the destination what to do next (destination only)
4. Send responses back to the source (both router and destination)
5. Handle errors (both router and destination)
6. Specify any special packet handling (both router and destination)

Task 1: Parse the Packet

Version: What version of IP?

Header length (Internet Header Length (IHL)):

- A **4-bit** field called IHL that specifies the header length
- Each value represents **4 bytes** instead of 1 byte

Version (4)	Hdr len (4)	Type of Service (8)	Total Length in Bytes (16)
-------------	-------------	---------------------	----------------------------

```
Internet Protocol Version 4
 0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (0000) CS0, ECN: Not-ECT
  0000 00.. = Differentiated Services Codepoint: Default (0)
  .... ..00 = Explicit Congestion Notification: Not ECN-capable Transport (0)
-
  Total Length: 20
  - [Expert Info (Error/Protocol): IPv4 total length exceeds packet length (4 bytes)]
    [IPv4 total length exceeds packet length (4 bytes)]
    [Severity level: Error]
    [Group: Protocol]
-
  [Malformed Packet: IPv4]
  - [Expert Info (Error/Malformed): Malformed Packet (Exception occurred)]
    [Malformed Packet (Exception occurred)]
    [Severity level: Error]
    [Group: Malformed]
```

Total length (measured in bytes):

- Total Length = IP Header Length + IP Payload Length
- Where does the packet end?

“Malformed Packet: IPv4”:

Wireshark detected something wrong

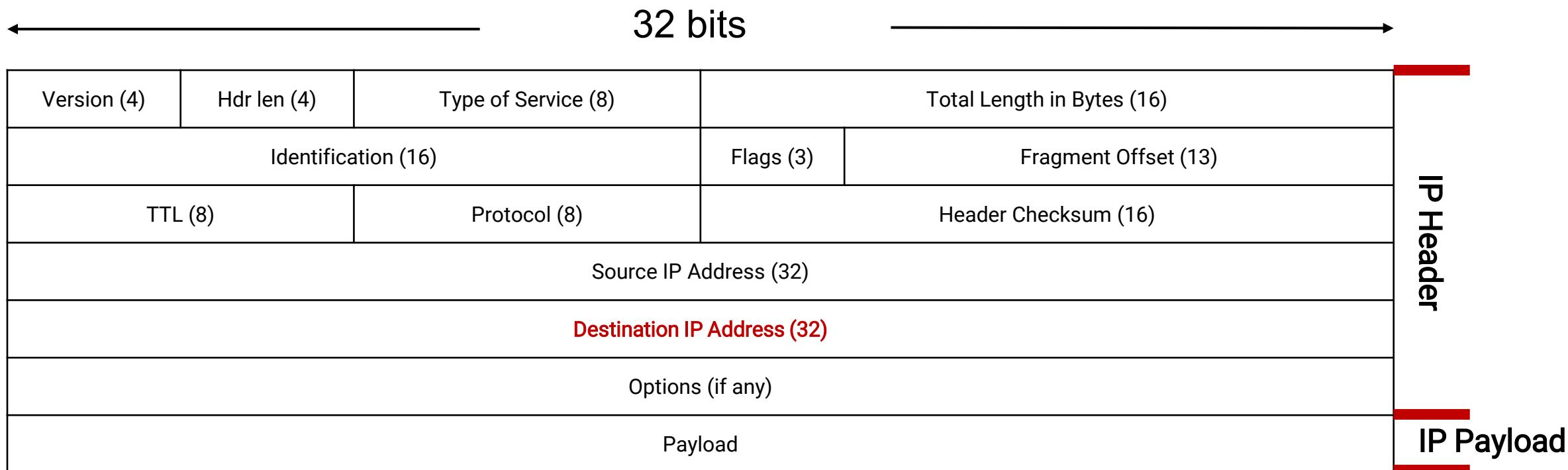
- Packet capture error: Only part of the packet was captured
- Network corruption: Packet was damaged in transit
- Malicious packet: Deliberately crafted to be malformed, etc.

In a **healthy IP packet**, you'd see something like:

IHL: (0101) 20 bytes Total Length: 1500 bytes → IP Payload: $1500 - 20 = 1480$ bytes (TCP/UDP/ICMP data)

Task 2: Forward Packet to Next Hop

Destination IP address



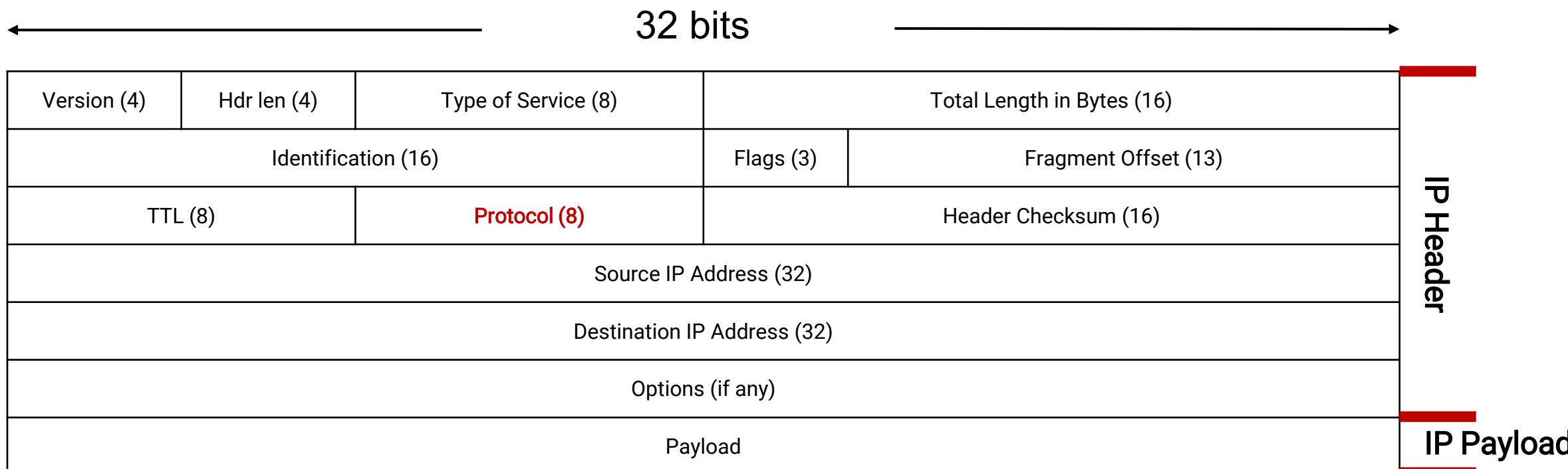
Task 3: Tell Destination What to Do Next

Protocol field:

- Identify the Layer 4 (Transport Layer) protocol to pass the payload to
- List of IP protocol numbers:
https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers

Protocol numbers are maintained and published by the [Internet Assigned Numbers Authority \(IANA\)](#)

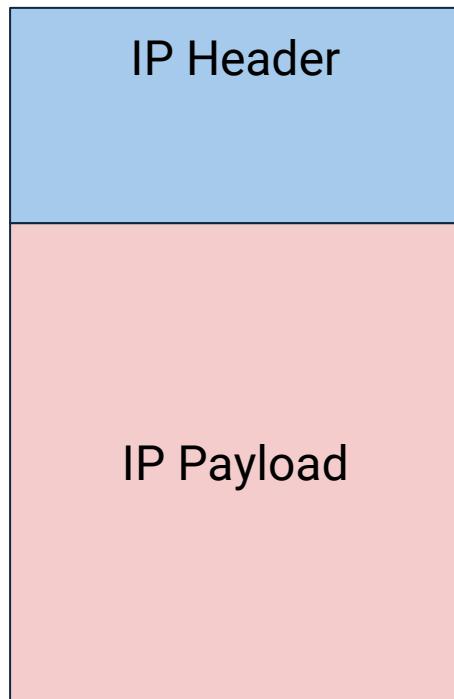
Hex	Protocol Number	Keyword	Protocol
0x00	0	HOPOPT	IPv6 Hop-by-Hop Option
0x01	1	ICMP	Internet Control Message Protocol
0x02	2	IGMP	Internet Group Management Protocol
0x03	3	GGP	Gateway-to-Gateway Protocol
0x04	4	IP-in-IP	IP in IP (encapsulation)
0x05	5	ST	Internet Stream Protocol



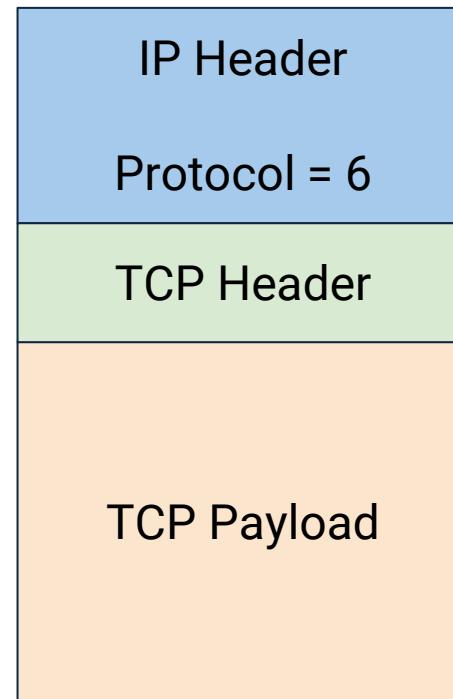
Task 3: Tell Destination What to Do Next

We need to indicate which protocol should handle the packet next

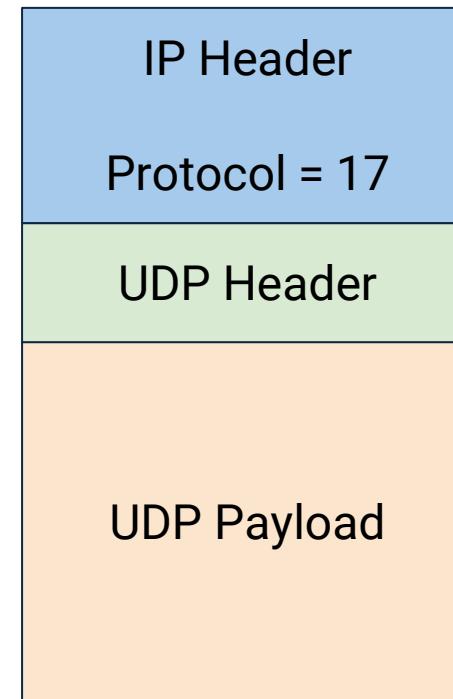
De-multiplexing: Picking one of several possible Layer 4 protocols



Without the protocol field,
we have no idea what to do
with the bits in red



If we see protocol = 6:
Pass the red bits to TCP code

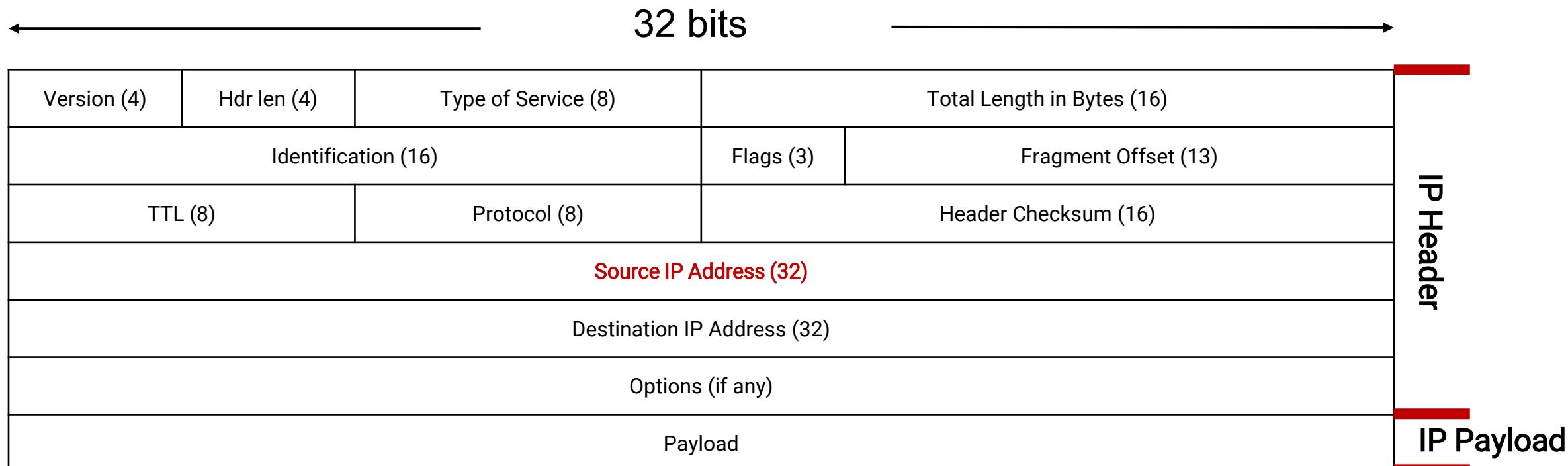


If we see protocol = 17:
Pass the red bits to UDP code

Task 4: Send Response Back to Source

Source IP address

- Note: We said both the router and destination might do this
- The router might want to send a response back to the source too (e.g. Ping 10.9.0.7 in Lab 5, an error occurs)



Task 5: Handle Errors - TTL

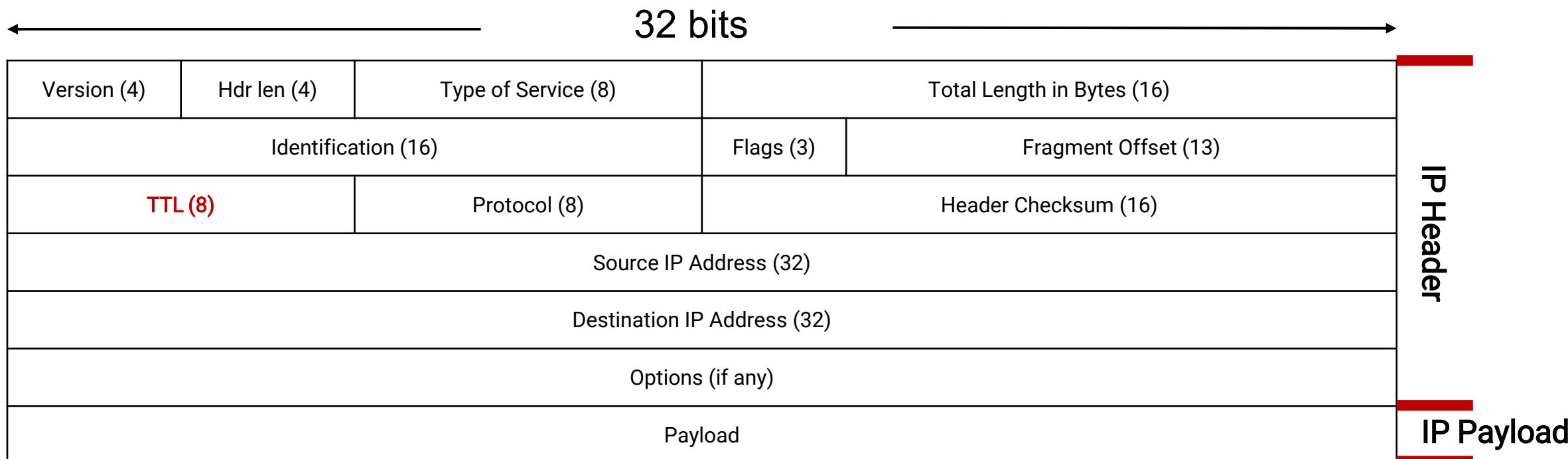
Problem: Forwarding errors (e.g., loops) cause packets to cycle indefinitely; Wastes bandwidth

Time-to-live (TTL):

- Specifies maximum number of hops a packet can take
- TTL is decremented at every hop by the router.

If router receives packet with TTL 1:

- Router decrements TTL: $1 - 1 = 0$
- Router sees TTL = 0 after decrementing
- Router **drops** the packet and **sends** ICMP “Time Exceeded” packet back to the source



Task 5: Handle Errors - Checksum

Problem: The packet could get corrupted in transit

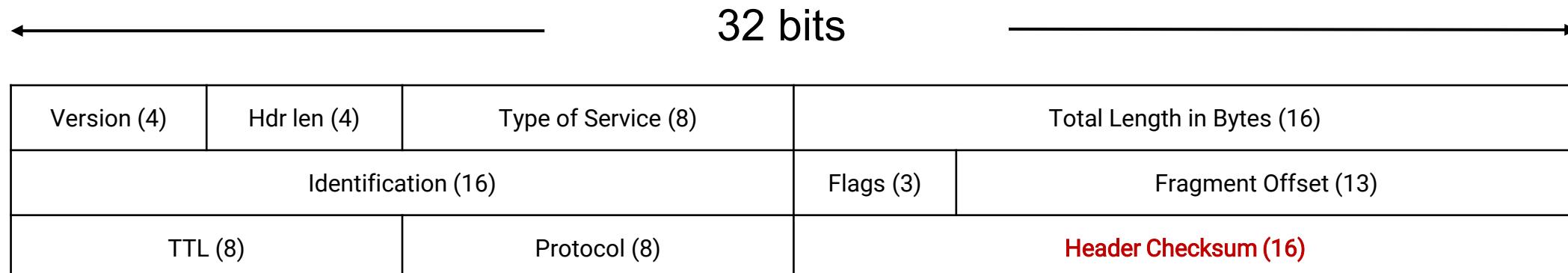
Checksum: Small number of bits used to verify that data hasn't been corrupted

The checksum is only computed on the **IP header**, not the payload (How does TCP/UDP calculate the checksum? → including the pseudo-header, header, and *payload data*):

- IP header can't detect the payload being corrupted
- Why this design? **End-to-end principle** → The payload should be checked by the end hosts, not intermediate routers

The checksum has to be updated at every router:

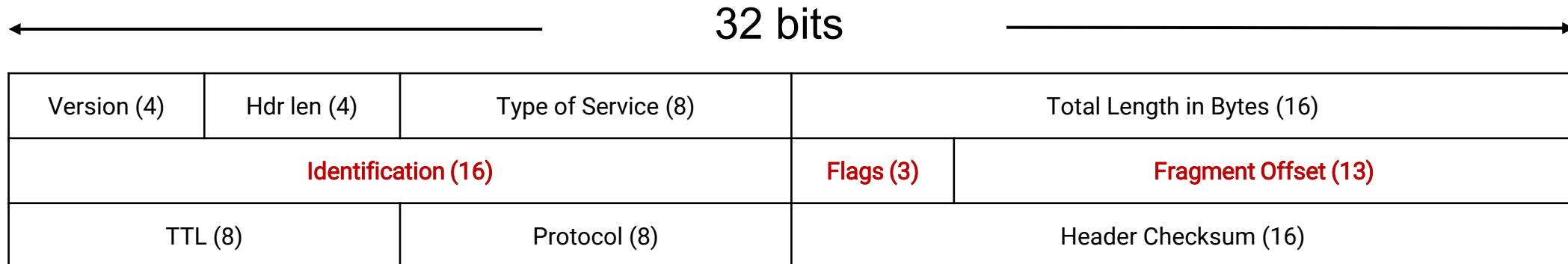
- Why? Because the TTL changes



Task 5: Handle Errors - Fragmentation

Problem:

- The packet might be too large for a link: IPv4 total length field is 16 bits → Maximum IP packet has 65,535 bytes
- Every link has a Maximum Transmission Unit (**MTU**): Largest number of bits the link can carry as one unit. E.g., Ethernet has an MTU of 1500 bytes, some wide area networks (WANs), such as X.25, have an MTU of 576 bytes
- Sender doesn't know the path (and MTU). If a packet size exceeds a link's MTU, the **router** must split the packet into multiple **fragments**
- Then, the receiver has to reassemble the fragments to recover the original packet



Task 5: Handle Errors - Fragmentation

ID: Each fragment of the same packet has the same ID. The destination needs to know which fragments belong together

Flags (3 bits, 1 bit unused):

- DF: Don't Fragment. If this packet is too large, just **drop** it
- MF: More Fragments. There's more data after this fragment

Offset:

- Identifies which bytes of the original packet are in this fragment
- Offset = (Position of fragment data in original packet) \div 8

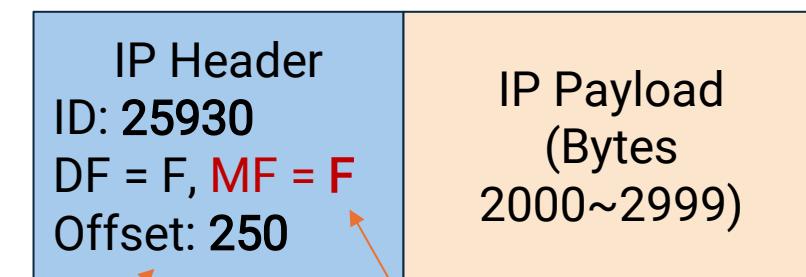
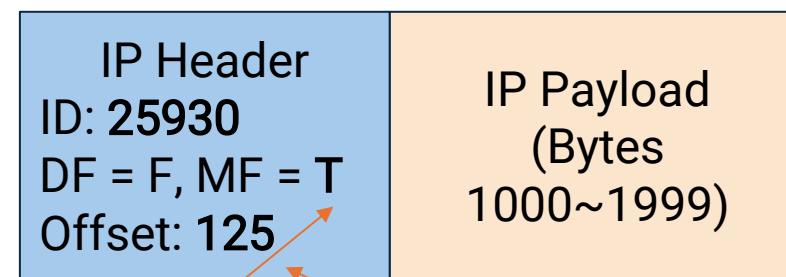
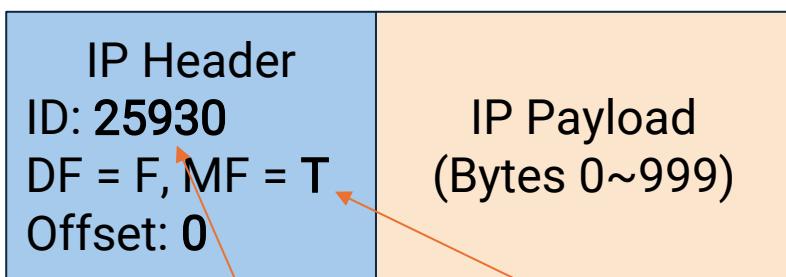
The diagram illustrates the structure of an IP header. A horizontal double-headed arrow spans the entire width of the header fields, labeled "32 bits". Below the header, a red double-headed arrow highlights the "Identification (16)" field, which is part of the 32-bit offset. The header fields are as follows:

Version (4)	Hdr len (4)	Type of Service (8)	Total Length in Bytes (16)		
Identification (16)			Flags (3)	Fragment Offset (13)	
TTL (8)	Protocol (8)		Header Checksum (16)		

Task 5: Handle Errors - Fragmentation

Source host assigns the ID, mandatory

Don't Fragment = False allows us to fragment



Router assigns the same ID on each fragment of this packet

IP header size = 20 bytes,
payload size = 1000 bytes

More Fragments = True on all but the last fragment

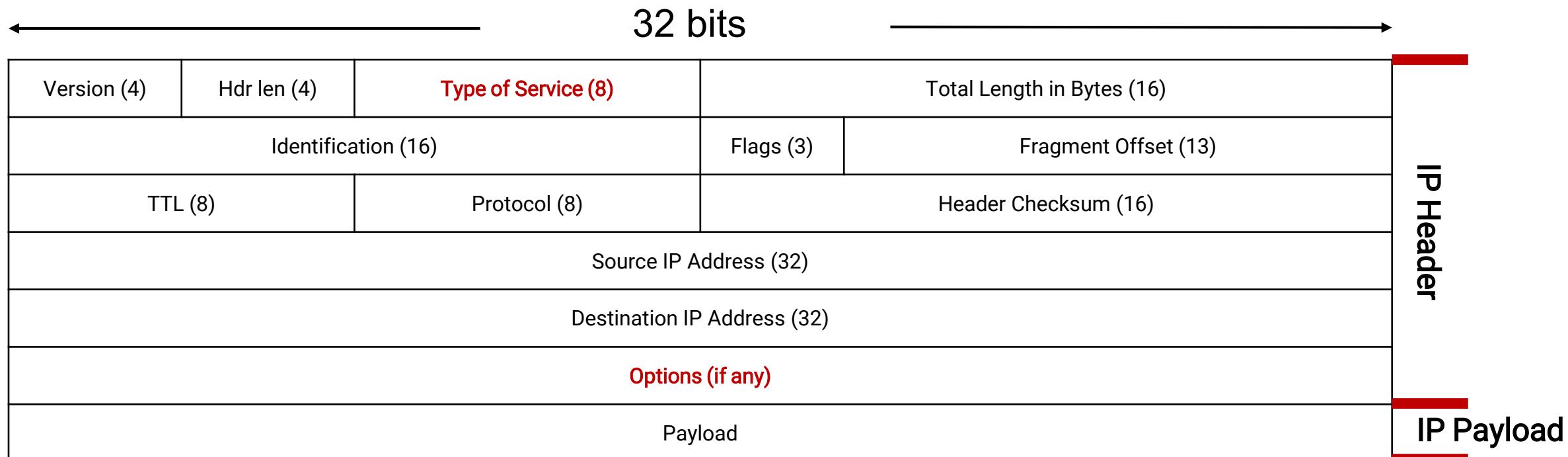
Offset measured in multiples of 8 bytes
 $1000 / 8 = 125$
 $2000 / 8 = 250$

More Fragments = False on the last fragment

Task 6: Specify Special Packet Handling

Type of Service: Treat packets differently depending on application / customer needs

Options: Request advanced functionality for this packet



Task 6: Specify Special Packet Handling

Original idea:

- Request different forms of delivery depending on these bits
- Based on priority, delay, throughput, reliability, cost, etc.
- These bits were frequently re-defined and never fully deployed
- Only the notion of **priorities** remained

Modern usage:

- Request advanced functionality for this packet. For example:
- Record Route: Tell me the path that this packet took (useful for diagnostics)
- Source Route: Send the packet along the route I provide
- Timestamp: Record when each router processes this packet

Version (4)	Hdr len (4)	Type of Service (8)	Total Length in Bytes (16)
.....			
Options (if any)			
Payload			

IP Header: Summary

1. Parse the packet (both router and destination)
2. Forward packet to the next hop (router only)
3. Tell the destination what to do next (destination only)
4. Send responses back to the source (both router and destination)
5. Handle errors (both router and destination)
6. Specify any special packet handling (both router and destination)

Version (4)	Hdr len (4)	Type of Service (8)	Total Length in Bytes (16)						
Identification (16)		Flags (3)	Fragment Offset (13)						
TTL (8)	Protocol (8)	Header Checksum (16)							
Source IP Address (32)									
Destination IP Address (32)									
Options (if any)									
Payload									

Roadmap

- *IP(v4) Header*
- ***IP Security***

Exploiting Source IP: Spoofing

Spoofing: Lie about the source address

- Attacker pretends to be somebody else and sends the packet.

Using spoofing to attack the impersonated user (Lab 5):

- Attacker pretends to be Bob and sends a virus
- Now Bob is wrongly blamed for sending the virus
- Return traffic (e.g. angry messages) goes to Bob instead

Using spoofing to attack a destination:

- Denial-of-service (DoS) attack: Overwhelm a service by flooding it with packets
- Without spoofing: The service can block your source address
- Spoofing makes DoS more effective: Server doesn't know which packets to block

Exploiting Type of Service

Recall type of service (ToS): Bits indicating priority of the packet

Attack: Prioritize your own traffic

- If anybody can set ToS bits, attackers can make their own packets high-priority → Network prefers attacker traffic
- Today, ToS bits are mostly set/used by operators, not end hosts

Attack: Use spoofing to make someone else pay for priority traffic

- Suppose the network charges for high-priority traffic
- Attacker sends a spoofed packet: "I am Bob and this packet is high-priority."
- Network makes Bob pay for that packet

Exploiting Fragmentation and Options

Recall: Fragmentation and options are more work for the router

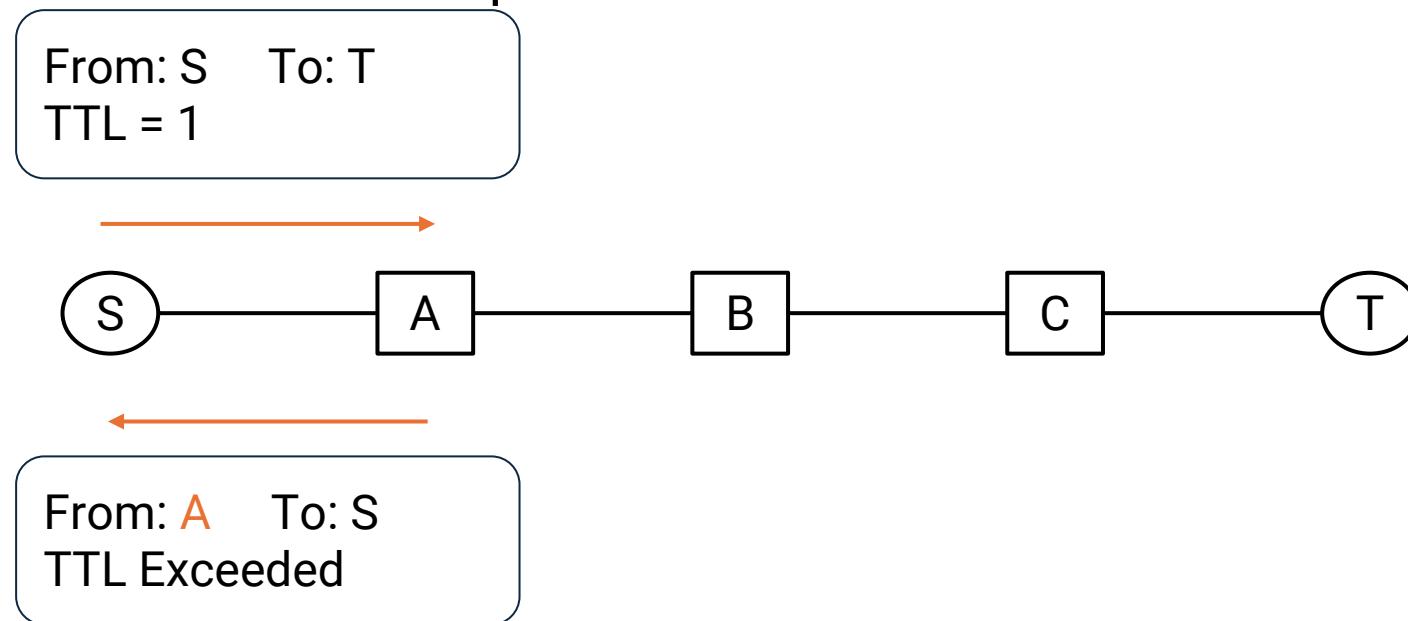
Denial-of-service (DoS) attack: Overwhelm routers by making them do more work. For example:

- Fragmentation: Send large packets on purpose
- Options: Send packets with complicated options on purpose
 - Defense: Routers often ignore options, or drop packets with options

Exploiting TTL: Traceroute

Traceroute: Exploiting the TTL to discover the path a packet travels along

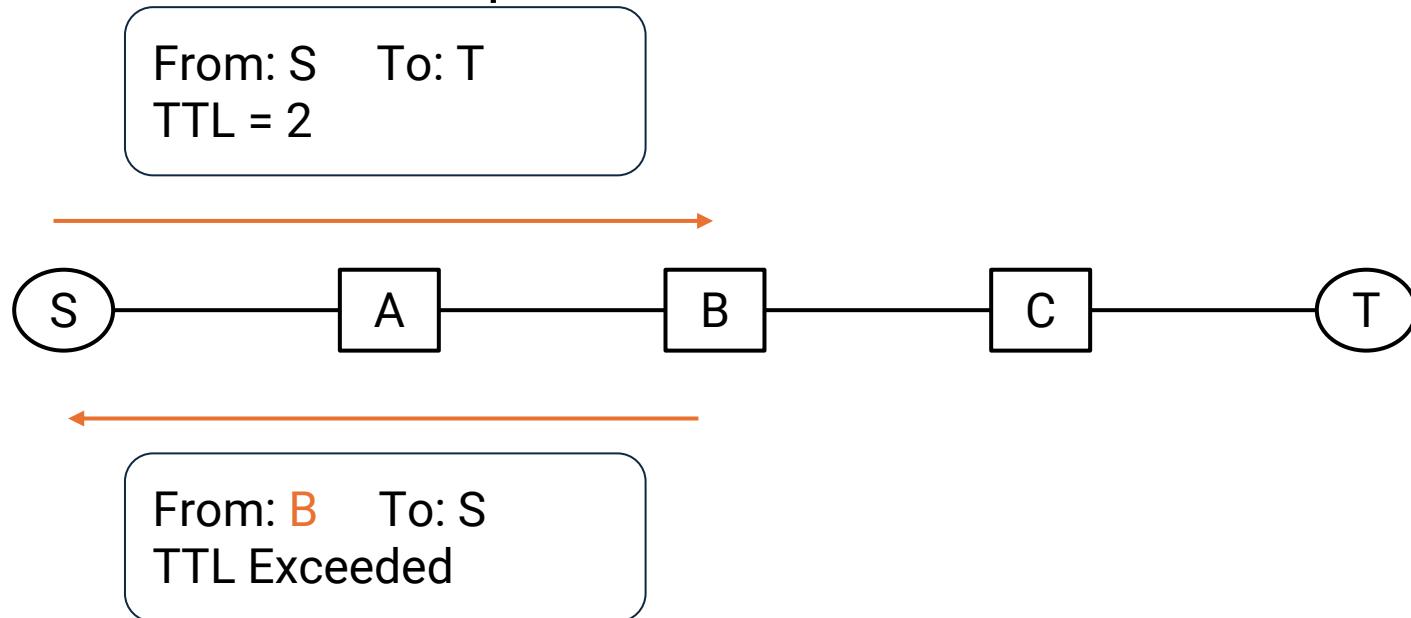
- You have done this in Lab 5
- Key idea: When TTL expires, the router sends back a "TTL Exceeded" error. This allows you to learn the identity one of the routers on the path
- Basic approach: Send packets with TTL=1, TTL=2, TTL=3, etc.
- In practice, not all routers respond with the "TTL Exceeded" message



Exploiting TTL: Traceroute

Traceroute: Exploiting the TTL to discover the path a packet travels along

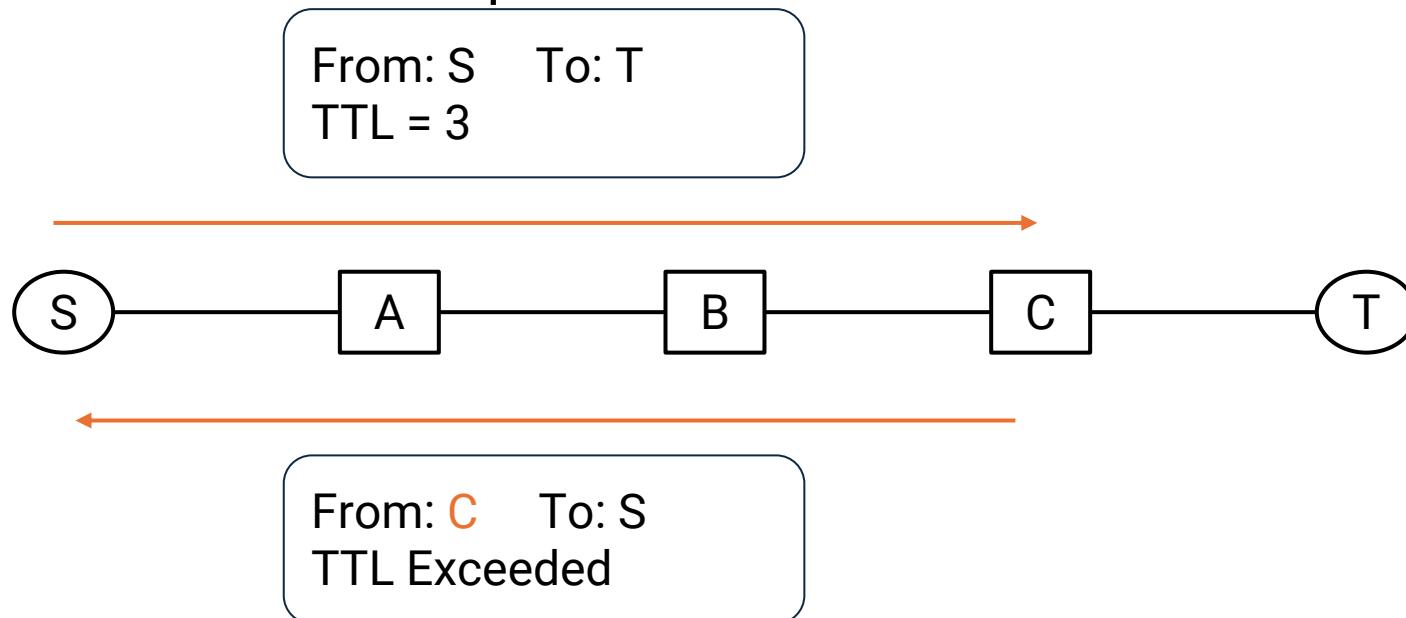
- You have done this in Lab 5
- Key idea: When TTL expires, the router sends back a "TTL Exceeded" error. This allows you to learn the identity one of the routers on the path
- Basic approach: Send packets with TTL=1, TTL=2, TTL=3, etc.
- In practice, not all routers respond with the "TTL Exceeded" message



Exploiting TTL: Traceroute

Traceroute: Exploiting the TTL to discover the path a packet travels along

- You have done this in Lab 5
- Key idea: When TTL expires, the router sends back a "TTL Exceeded" error. This allows you to learn the identity one of the routers on the path
- Basic approach: Send packets with TTL=1, TTL=2, TTL=3, etc.
- In practice, not all routers respond with the "TTL Exceeded" message



Exploiting Other Fields?

Exploiting protocol field?

- If protocol is set incorrectly, the next layer will find the packet malformed.

Exploiting checksum?

- A bad checksum causes the network to drop the packet.

These are not effective attacks.

Summary of IP Attacks

Source IP address: Spoofing

Type of service: Prioritize attacker traffic

Fragmentation, Options: Denial-of-service

TTL: Traceroute

Protocol, Checksum: No apparent problems

Version	Hdr len	Type of Service	Total Length in Bytes				
Identification			Flags	Fragment Offset			
TTL	Protocol	Header Checksum					
Source IP Address (32 bits)							
Destination IP Address (32 bits)							
Options (if any)							
Payload							

CSC 3511 Security and Networking

Week 6, Lecture 2: Link Layer

Roadmap

- *Link Layer and LAN*
- *Multiple Access Protocol*
- *Sending Ethernet Packets*

Link Layer: Introduction

Link layer has responsibility of transferring data between *adjacent nodes* over a single link

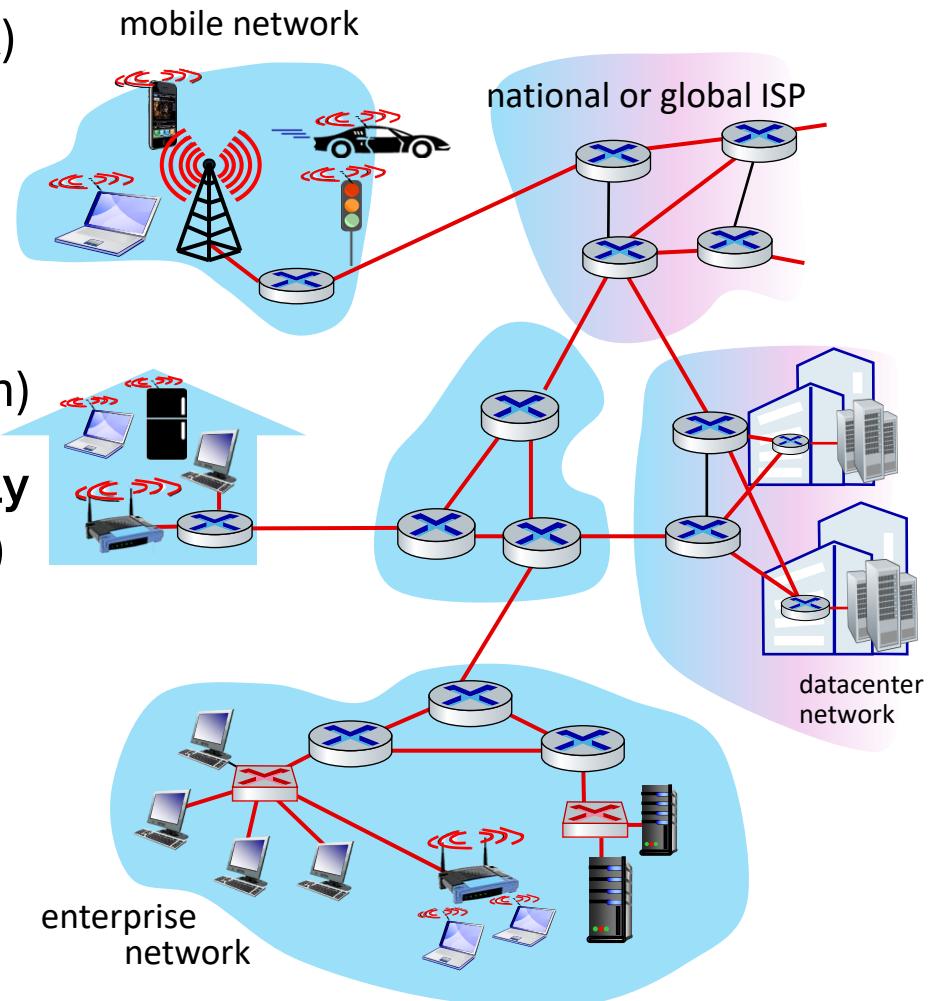
- Scope: One "hop" at a time (node-to-node, **same local network**)
- Different from Network Layer: Link layer → one hop, Network layer → end-to-end path

Link Layer Terminology

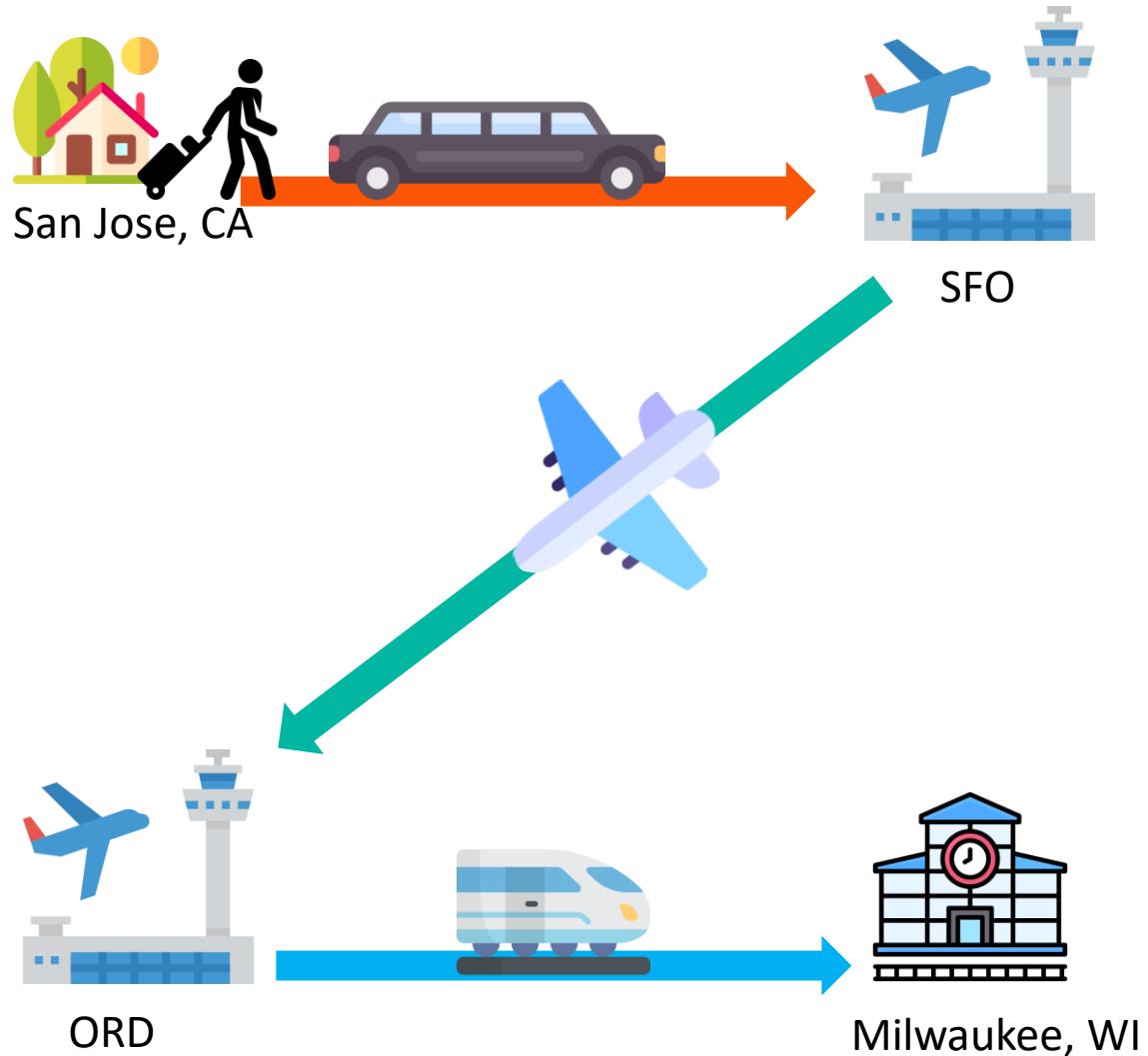
- *Frame*: Link layer packet (encapsulates network layer datagram)
- *Links*: Physical communication channels connecting **physically adjacent nodes**. E.g., Wired (fiber optical cable, copper cable) or wireless (WiFi channel, cellular)
- *Nodes*: Host, routers

Different links use different protocols:

- WiFi on first hop → **Ethernet** on next hop → ...
- Same datagram, different **link-layer protocols** at each hop



Transportation analogy



Transportation analogy:

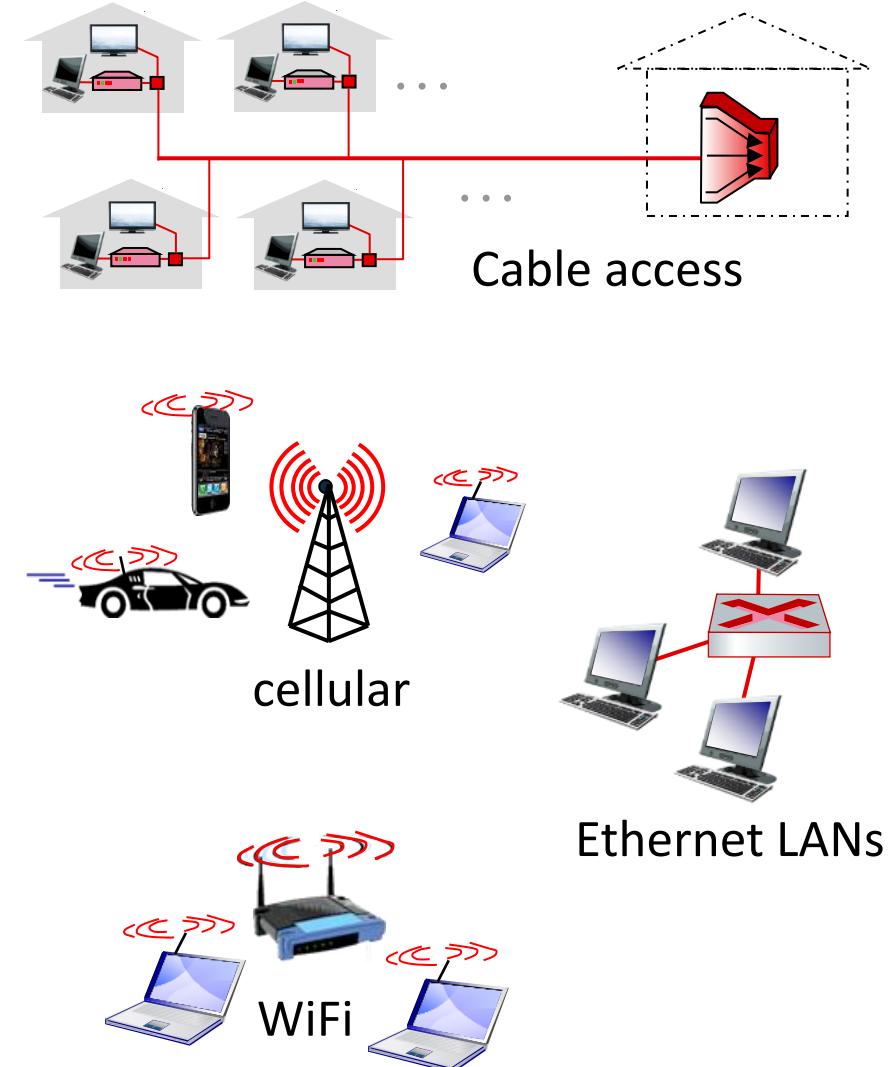
- Travel agent = **routing algorithm**
 - Determines the overall path and which modes to use
- Trip from San Jose, CA to Milwaukee, WI
 - Uber: San Jose to SFO
 - Plane: SFO to ORD
 - Train: ORD to Milwaukee
- Passenger = **frame**
 - The "packet" being transported
- Transport segment = **communication link**
 - The physical path/medium
- Transportation mode = **link-layer protocol**
 - Different rules/methods for each segment
 - E.g., Uber – road protocol; Airline – air protocol; Amtrak – rail protocol

Link layer: Services

- **Framing, link access:**
 - Encapsulate datagram into frame, adding header, trailer
 - “MAC” addresses in frame headers identify source, destination (different from IP address!)
 - Channel access if shared medium
- **Reliable delivery between adjacent nodes**
 - Unnecessary, mostly used on high bit-error links (wireless) where data corruption or loss is more frequent
 - *Correct* errors locally

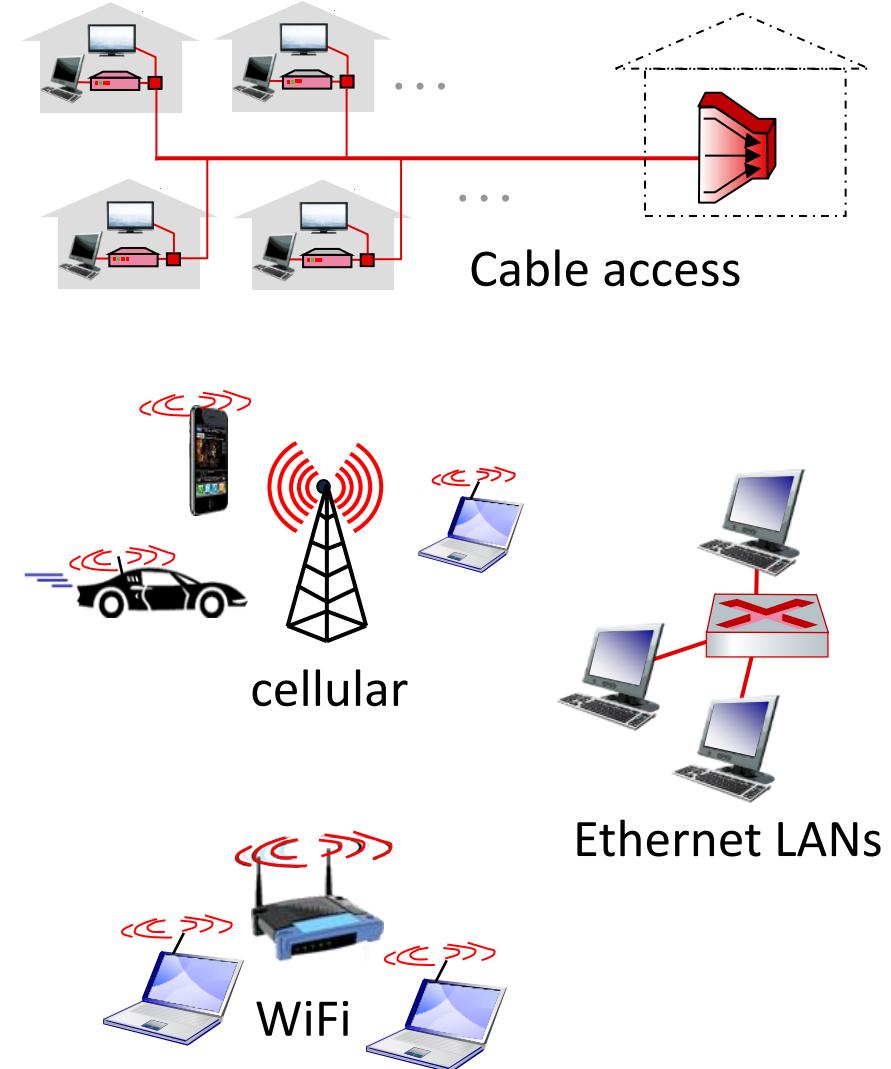
Q: why both link-level and end-end reliability?

1. Link layer errors do not propagate further, more efficient than end-to-end retransmission
2. An additional layer of error checking and **correction**



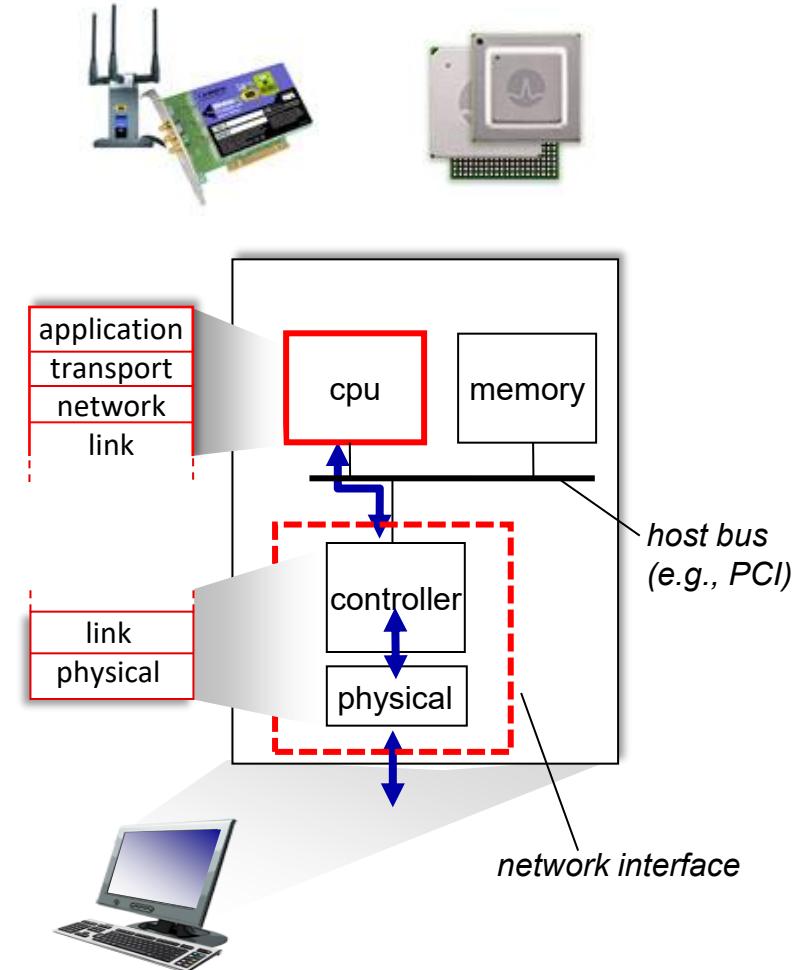
Link Layer: Services

- **Flow control:**
 - Pacing between adjacent sending and receiving nodes
- **Error detection:**
 - Errors caused by signal attenuation, noise.
 - Receiver detects errors, signals retransmission, or drops frame
- **Error correction:**
 - Receiver identifies *and corrects* bit error(s) without retransmission
- **Half-duplex and full-duplex:**
 - With half duplex, nodes at both ends of link can transmit (two-way transmission), but not at same time

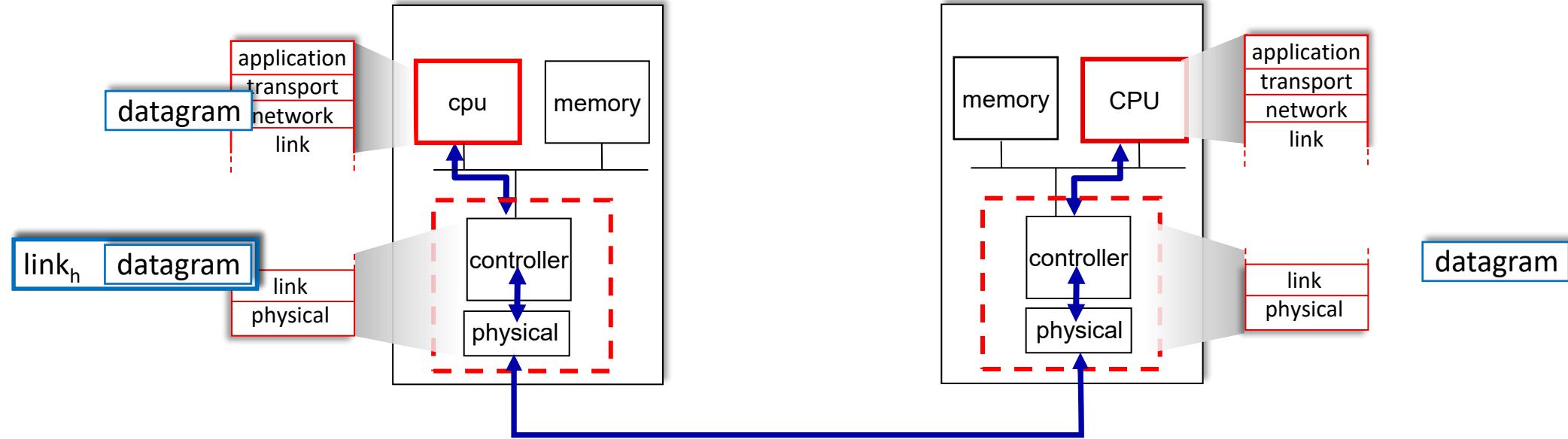


Host Link Layer Implementation

- In each-and-every host
- Link layer implemented on-chip or in network interface card (NIC) (aka, network adapter)
 - Implements link and physical layer
- Attaches into host's system buses
- Combination of hardware, software, firmware



Interfaces Communicating



Sending side:

- Encapsulates datagram in frame
- Adds error checking bits, reliable data transfer, flow control, etc.

Receiving side:

- Looks for errors, reliable data transfer, flow control, etc.
- Extracts datagram, passes to upper layer at receiving side

Roadmap

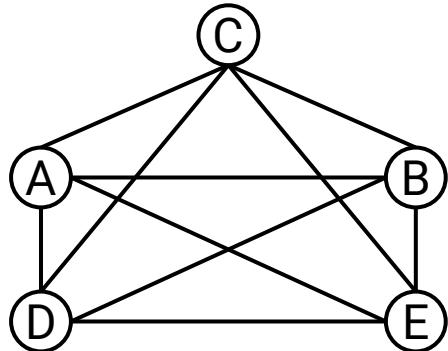
- *Link Layer and LAN*
- ***Multiple Access Protocol***
- *Sending Ethernet Packets*

Connecting Local Hosts

Challenge: Building a Local Network

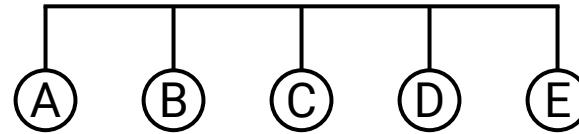
Multiple hosts need to communicate in the same local area (e.g., office, home, campus building)

Q: How do we physically connect them?



Option 1: Point-to-point links

- Every pair of hosts has a dedicated link
- Expensive cabling and ports, does not scale:
 - For N hosts, need $N(N-1)/2$ links
 - For every new host, we have to add new links to every other host
 - Need a lot of (physical) ports per host



Option 2: Shared Medium (Broadcast link, bus Topology)

- A single wire can connect multiple computers. From the wire's perspective, the router is just a machine like any other.
- All hosts connect to a single shared wire:
 - Just one cable - easy to add new hosts
 - Each host needs only 1 interface
- Introduces a new problem - **shared media**:
 - All hosts share the same communication channel
 - What if A and B transmit at the same time? → **COLLISION!**
 - Signals interfere, data corrupted

Shared Media and Multiple Access Protocols

Shared media:

- If multiple machines transmit at the same time, signals will interfere or collide
- Analogy: People talking simultaneously on a group call



humans at a cocktail party
(shared air, acoustical)



shared radio: 4G/5G

Note: Shared media is not necessarily a wire

- Could be light signals on a shared optical fiber
- Or radio waves on a shared wireless link



shared radio: WiFi



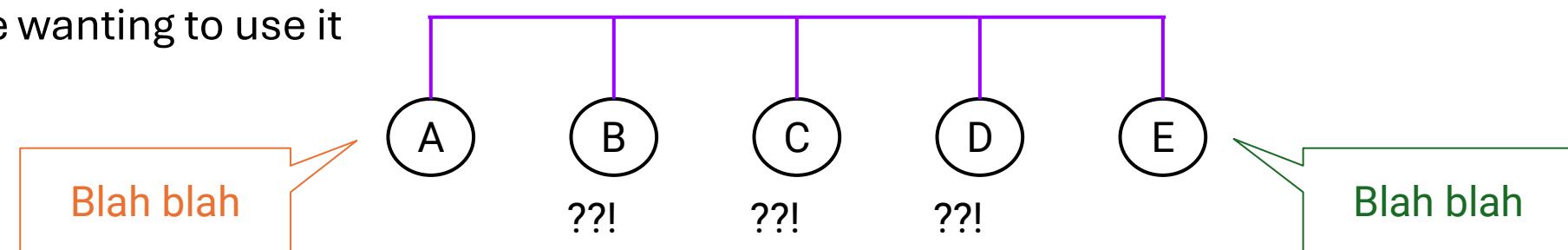
shared radio: satellite



shared wire (e.g.,
cabled Ethernet)

Q: How can multiple nodes share one channel without collisions?

A: Multiple Access Protocol → allocates the shared media to everyone wanting to use it



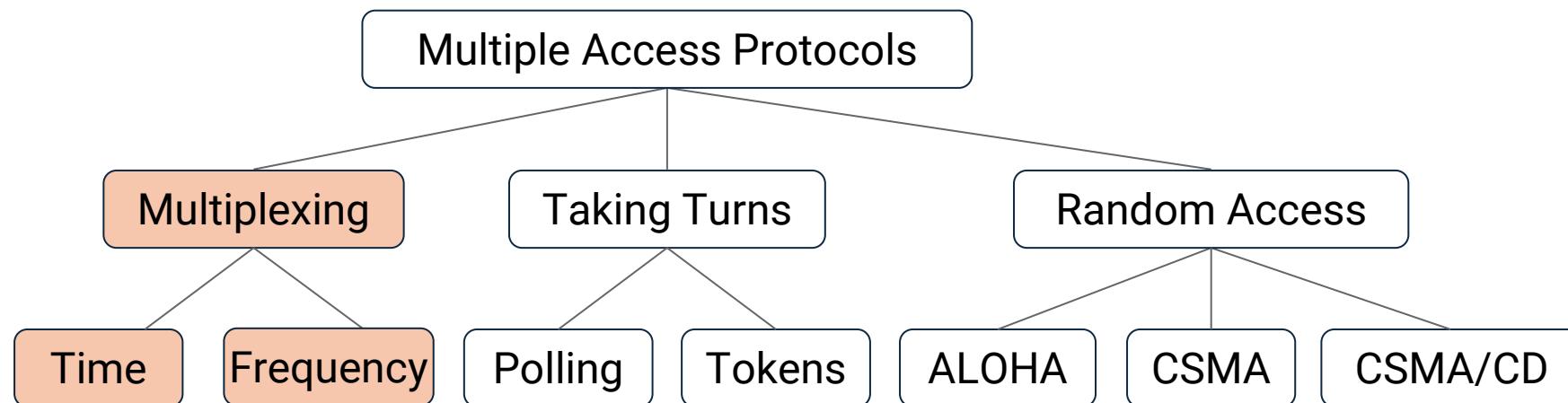
Multiple Access Protocols (1/3) - Multiplexing

A **multiple access protocol** allocates the shared media to everyone wanting to use it

- 3 types of approaches
- Several protocols of each type

Multiplexing(aka Channel partitioning):

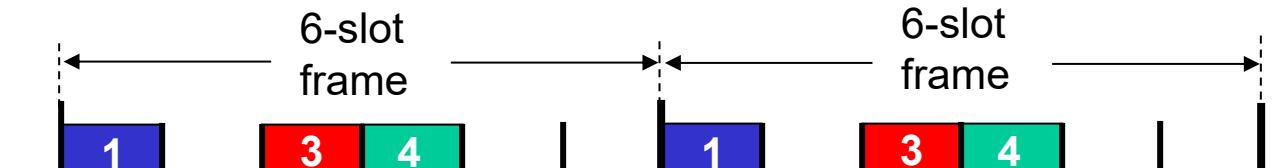
- Divide channel into smaller “pieces” (time slots, frequency, code)
- Allocate piece to node to each node for exclusive use
- **Time-based multiplexing:** Give each node some fixed time slots
- **Frequency-based multiplexing:** Divide medium into frequency channels



Multiple Access Protocols (1/3) - Multiplexing

TDMA: time division multiple access

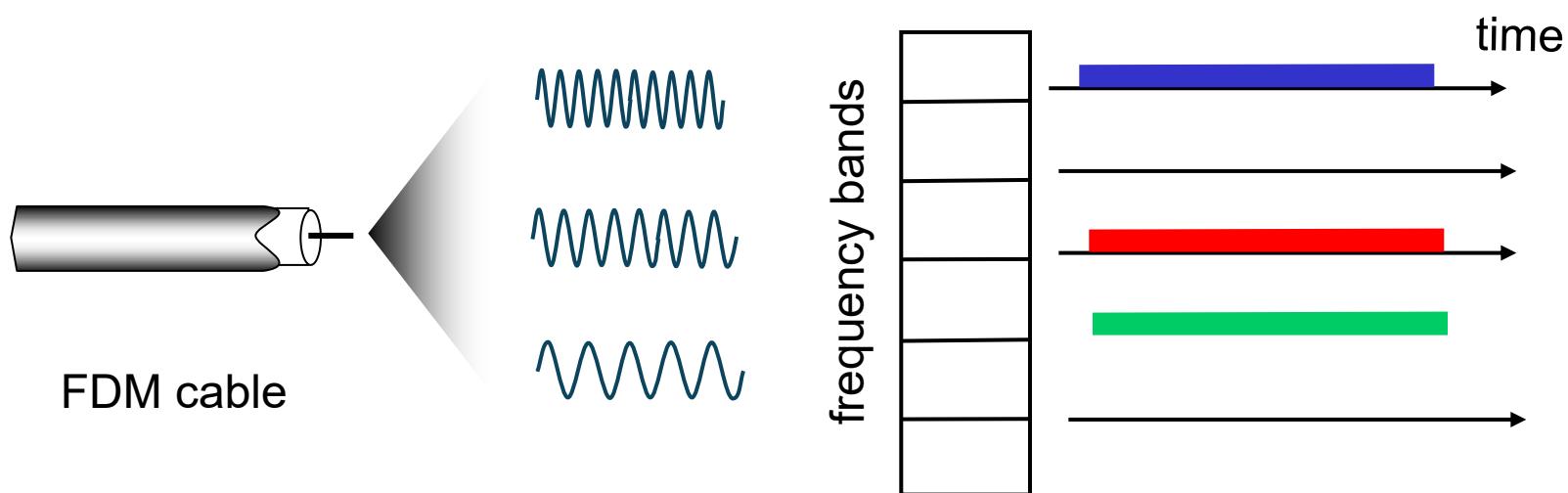
- Access to channel in “rounds”
- Each node gets fixed length slot (length = packet transmission time) in each round
- Unused slots go idle
- Example: 6-node LAN, 1,3,4 have packets to send, slots 2,5,6 idle



Multiple Access Protocols (1/3) - Multiplexing

FDMA: frequency division multiple access

- Channel spectrum divided into frequency bands
- Each node assigned fixed frequency band
- Unused transmission time in frequency bands go idle
- Example: 6-node LAN, 1,3,4 have packet to send, frequency bands 2,5,6 idle

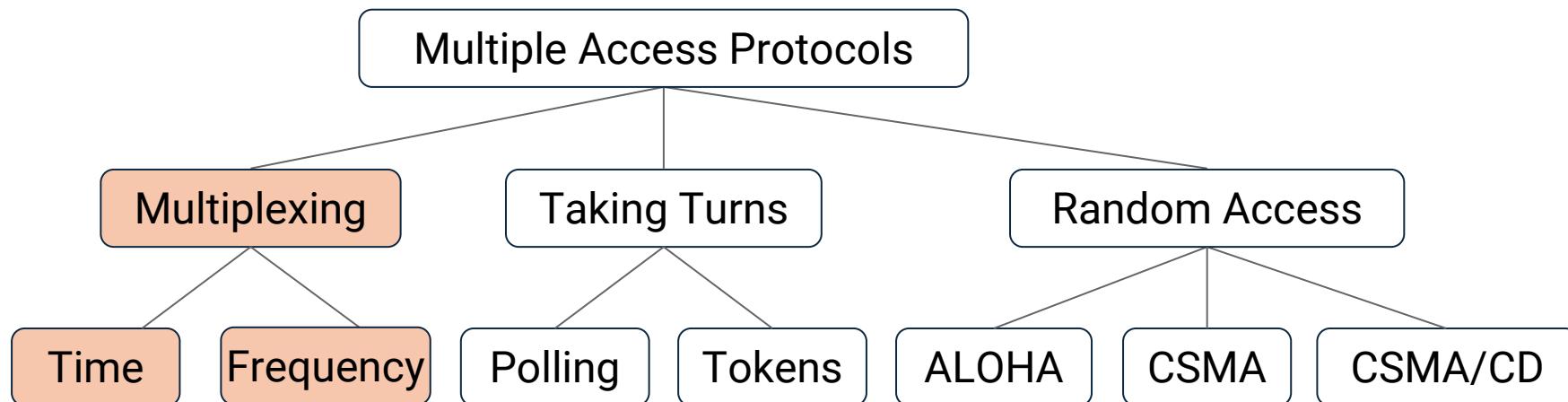


Multiple Access Protocols (1/3) - Multiplexing

Idea: Allocate a fixed slice of resources to each node

Problem: Can be wasteful.

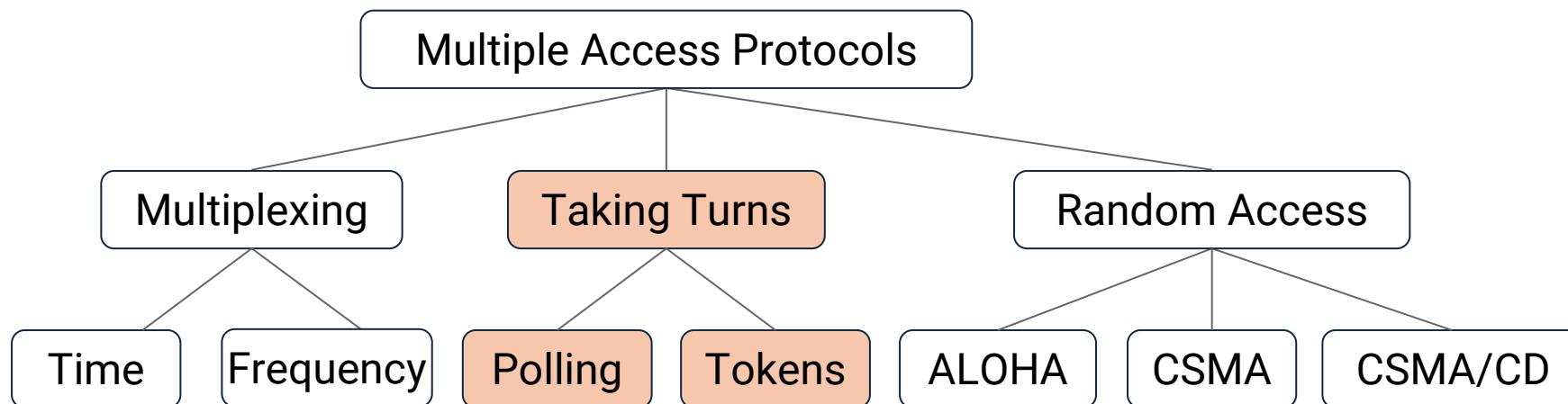
- There's only so much frequency/time available to allocate
- If a node has nothing to say, some frequency/time goes unused



Multiple Access Protocols (2/3) – Taking Turns

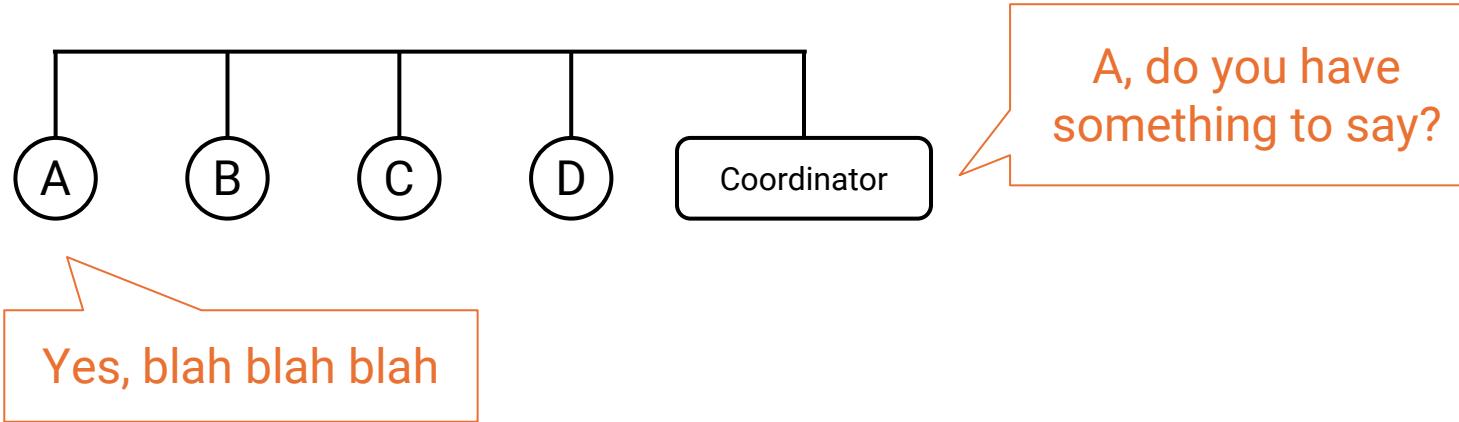
Taking turns:

- Nodes take turns speaking
- **Polling protocols:** A coordinator decides when each node can speak
 - Example: Bluetooth
- **Token passing:** Pass a virtual token around. Only the node with the token can speak
 - Example: IBM Token Ring, FDDI

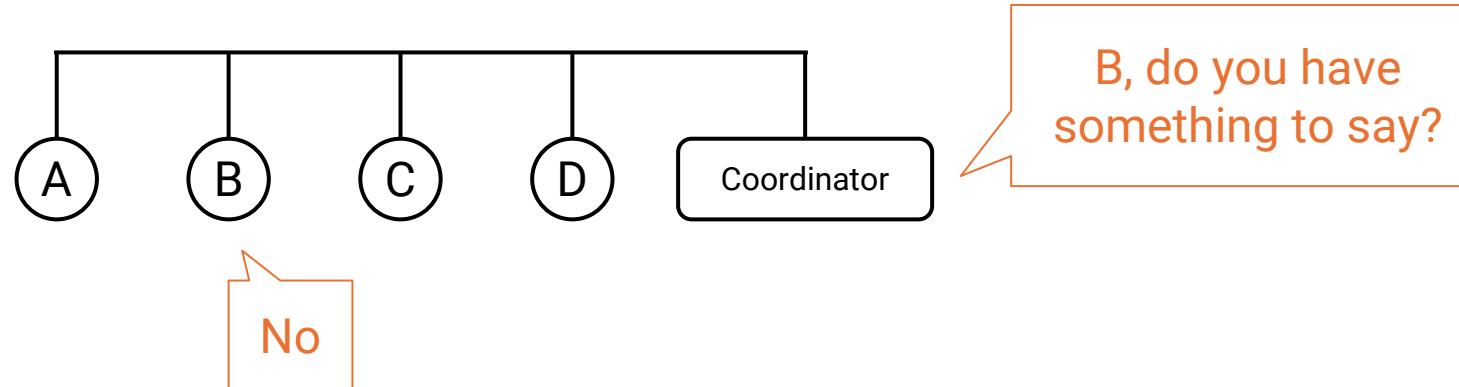


Multiple Access Protocols (2/3) – Taking Turns

Polling protocols: A coordinator decides when each node can speak



A can speak for as long as it needs

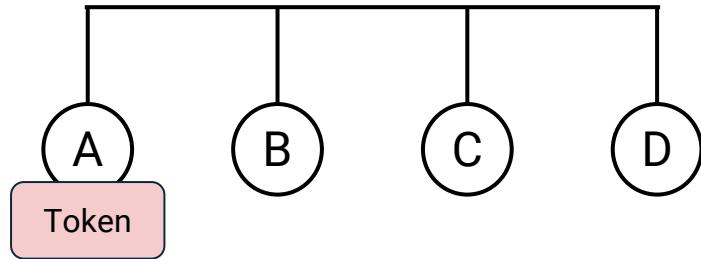


B has nothing to say

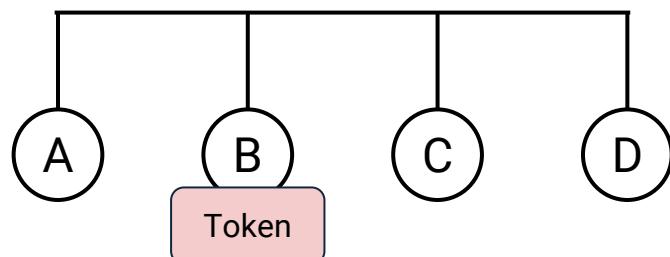
Coordinator can immediately move on to C

Multiple Access Protocols (2/3) – Taking Turns

Token passing: Pass a virtual token around. Only the node with the token can speak



A holds the token.
A can speak for as long as it needs.
When A is done, it passes the token to B.

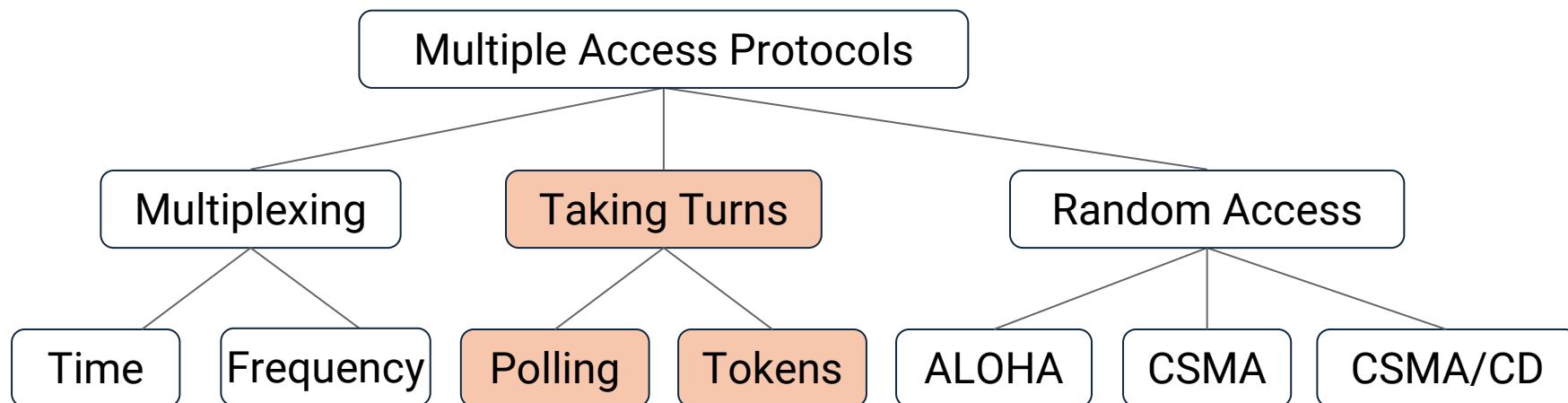


B has nothing to say.
It can immediately pass the token to C.

Multiple Access Protocols (2/3) – Taking Turns

Idea: Nodes take turns speaking

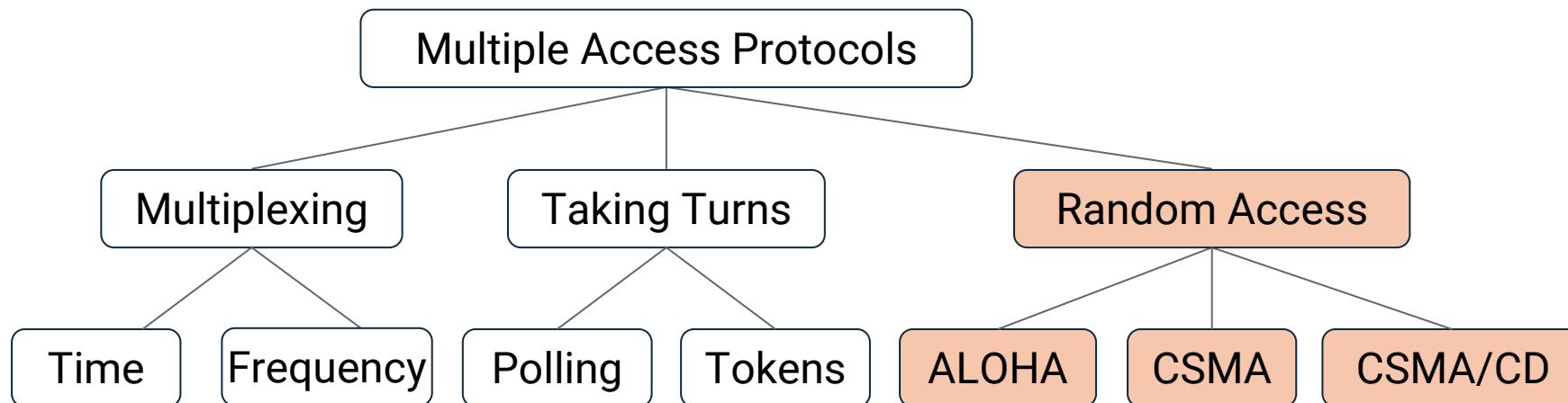
- Benefit: No more time wasted on idling
 - If someone has nothing to say, immediately move on
- Problem: Complexity
 - Need to implement inter-node communication
 - How do we elect the central coordinator?
 - What if two nodes both think they have the token?



Multiple Access Protocols (3/3) – Random Access

ALOHA random access scheme: "rude" version

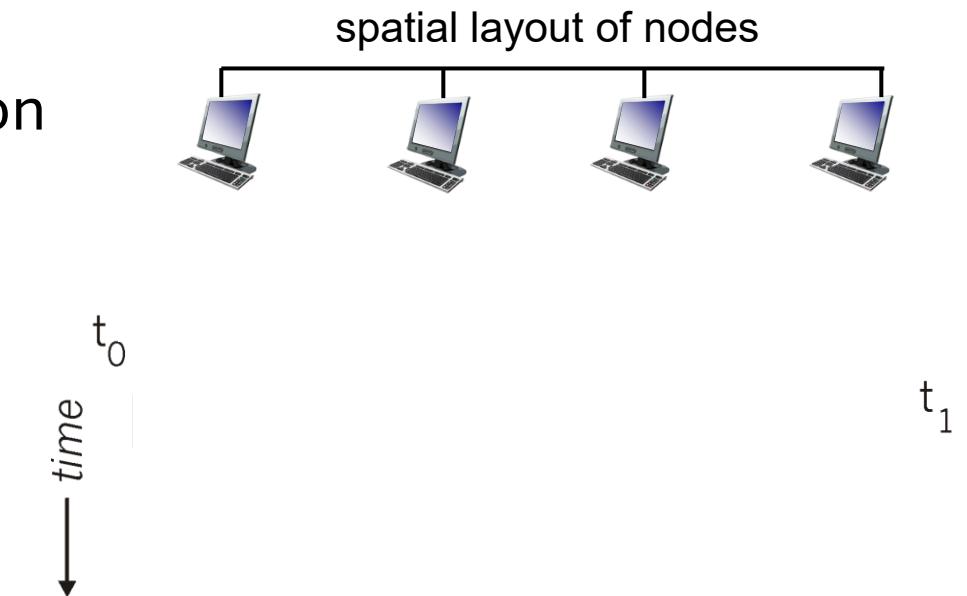
- If you have a packet, just send it
 - Recipient replies with an **ack**
- If two nodes send simultaneously, collision corrupts the packets.
 - **No ack!**
- If you don't get an ack: Wait some random amount of time, then resend.
 - Randomness helps avoid another collision.



Multiple Access Protocols (3/3) – Random Access

CSMA (Carrier Sense Multiple Access): "polite" version

- Listen before transmit:
 - If channel sensed idle: transmit entire frame
 - If channel sensed busy: defer transmission
- Human analogy: don't interrupt others!



Collision can *still* occur with carrier sensing:

- **Propagation delay** means two nodes may not hear each other's just-started transmission
- Collision: entire packet transmission time wasted
 - Distance & propagation delay play role in determining collision probability

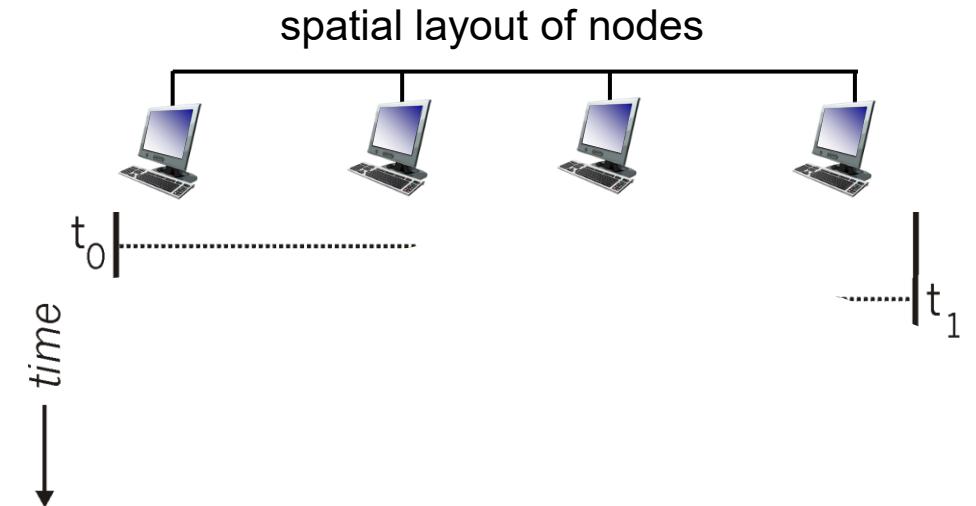
Multiple Access Protocols (3/3) – Random Access

CSMA/CD (Carrier Sense Multiple Access with Collision Detection):

- Listen before sending, but also while you're sending
- If you hear someone else sending, **stop!** Collision detected!

CSMA/CD uses binary exponential backoff:

- After every collision, wait up to twice as long before resending.
 - After first collision: Wait between 0–4 seconds.
 - If resend collides again: Wait between 0–8 seconds.
- Resends fast when possible, slowing down when necessary (e.g. many senders)



Ethernet CSMA/CD Algorithm

1. Ethernet receives datagram from network layer, creates frame
2. If Ethernet senses channel:
 - if **idle**: start frame transmission.
 - if **busy**: wait until channel idle, then transmit
3. If entire frame transmitted without collision - done!
4. If another transmission detected while sending: abort, send jam signal
5. After aborting, enter *binary (exponential) backoff*:
 - after m th collision, chooses K at random from $\{0,1,2, \dots, 2^m-1\}$.
Ethernet waits $K \cdot 512$ bit times, returns to Step 2
 - *more collisions: longer backoff interval*

Roadmap

- *Link Layer and LAN*
- *Multiple Access Protocol*
- ***Sending Ethernet Packets***

Ethernet as LAN Network Protocol

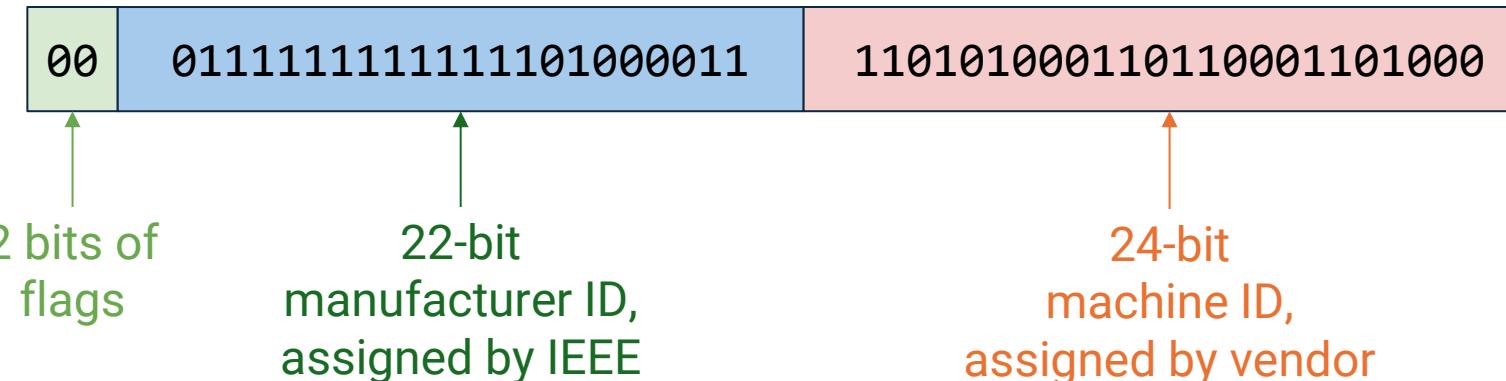
Local Area Networks (LANs) are typically Ethernet

- Machines in the same LAN can exchange messages directly at Layer 2!
 - No need for IPs, routers, forwarding, etc.
 - Use ***MAC addresses*** instead
- Analogy: If we're in the same office building
 - Same floor (LAN): Walk directly to colleague's desk → Layer 2 (Ethernet)
 - Different buildings (different networks): Need address & postal system
→ Layer 3 (IP routing)

Ethernet Addressing

At layer 2, each machine has a **MAC address** (*Media Access Control*)

- Function: used “locally” to get frame from one interface to another *physically-connected interface* (same subnet)
- 48 bits, written in hex, e.g. f8:ff:c2:2b:36:16 [Beware of endianness](#)
- Stored **permanently** on the machine ("burned in")
 - Often can be overridden by software
- Allocated according to organization, e.g. manufacturer of the machine
- Globally unique. You might plug your computer in anywhere



Vendors Get OUIs (Organizationally Unique Identifier, the first 24 bits) with first bit = 0, e.g., 00:1A:2B:xx:xx:xx

Manufacturer fills in the last 24 bits (xx:xx:xx) for each device

Types of LAN Communication

Ethernet supports three types of communication:

- **Unicast:** Send a packet to a single recipient
- **Broadcast:** Send a packet to everyone on the local network
- **Multicast:** Send a packet to everyone in a specific group
 - Machines in the local network can join groups

Unicast: Send a packet to a single recipient

- Destination = the recipient's MAC address

Recall: On a shared medium, everybody gets the signal

- When you get a packet, check the destination to see if it's meant for you
- If not, ignore the packet

Broadcast: Send a packet to **everyone** on the local network

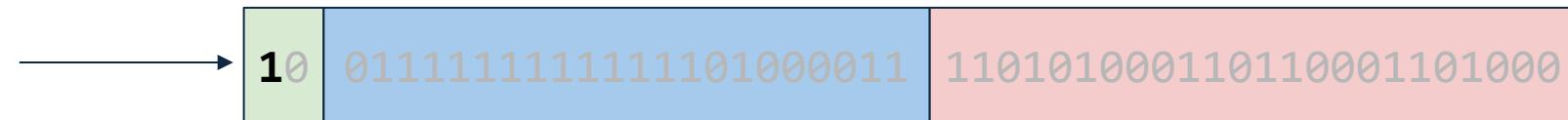
- Packet already reaches everybody on the shared medium
 - We just need to make sure everyone knows it's meant for them
- Destination = the **broadcast address**, FF:FF:FF:FF:FF:FF

Types of LAN Communication

Multicast: Send a packet to everyone in a specific group

- How group addresses work:
 - These are used by *software/protocols*, not burned into hardware
 - First bit = 1 → reserved for multicast/group address, e.g., 01:00:5E:xx:xx:xx = IPv4 multicast
 - When a device wants to join a multicast group, it configures its NIC to listen for that group address
- A device can listen to multiple multicast addresses, but its burned-in address is always unicast
- **Broadcast** is just a special case of multicast, where everyone is in a group

If this bit is 1, it's
a group address



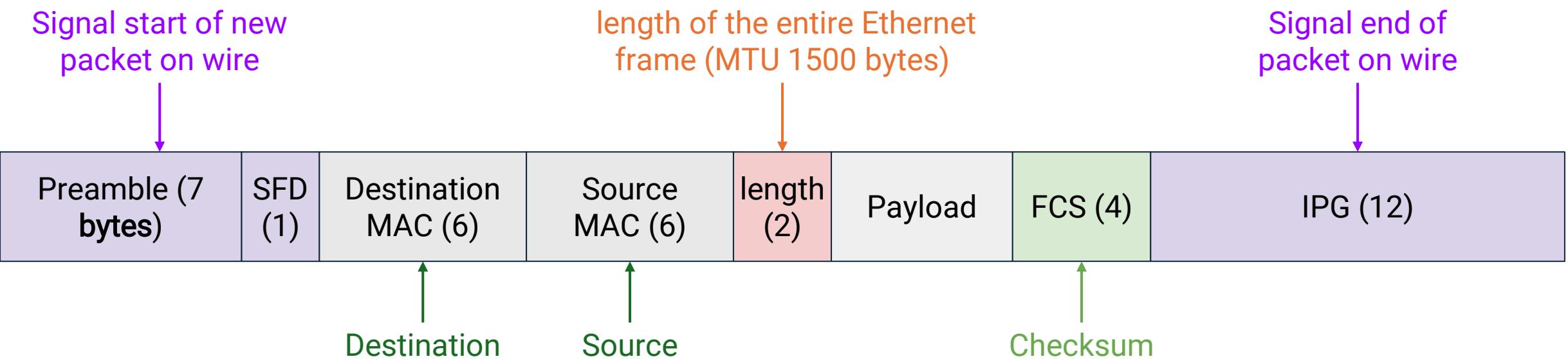
Example of using multicast: Bonjour/mDNS

- Invented by Apple.
- All Apple products join a multicast group.
- Apple devices can multicast to the group.
 - iPhone says: "Hey, local Apple products, can anyone play music?"
 - Apple TV says: "Hey, local Apple products, I'm an Apple TV."

Ethernet Packet Structure

A data packet in Ethernet ([IEEE 802.3](#)) is often called a *frame*.

- Many fields (destination, source, type, checksum) similar to IP header
- Need additional fields to separate packets on the wire



CSC 3511 Security and Networking

Week 7, Lecture 1: Link Layer, Switches, and ARP

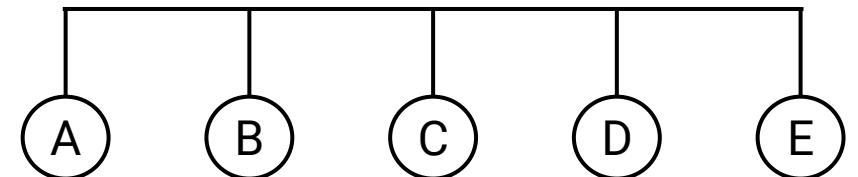
Roadmap

- ***Layer 2 Networks***
 - *Switches and Self-learning*
 - *ARP*

Layer 2 Networks

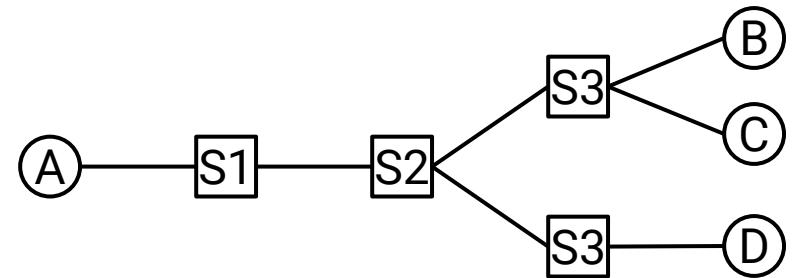
We could use more than one wire in a local network:

- Previously: Layer 2 operating on a single shared link with multiple computers
- Now: We introduce **multiple links** and build a network entirely using Layer 2
- **Switches** interconnect these links:
 - Create a larger local network by connecting multiple links
 - Switches forward frames between different network segments



Packet forwarding and routing (e.g., Spanning Tree Protocol) in Layer 2:

- Forwarding uses MAC addresses as destinations
- Switches maintain MAC address tables to forward frames



Q1: If switches can forward packets like routers, why do we need routers?

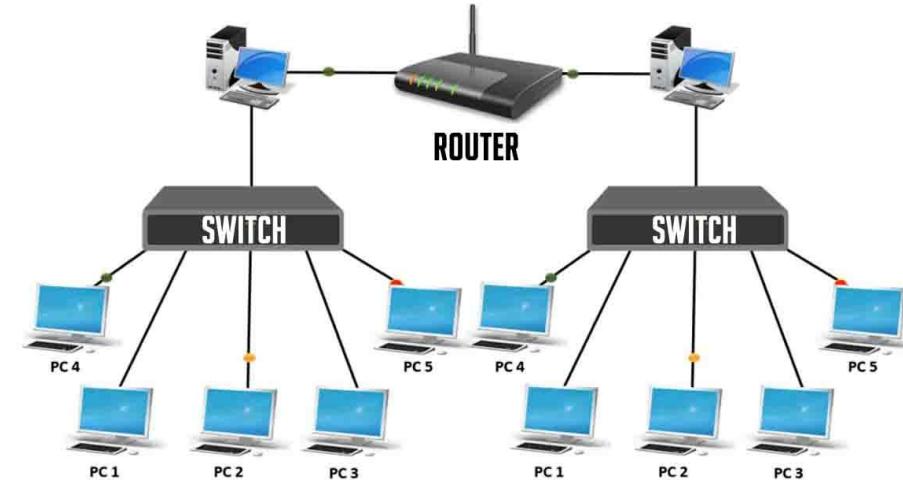
Q2: Could we scale Layer 2 switching to the entire Internet?

Switches vs. Routers

A switch connects devices within a single network using MAC addresses; while a router connects multiple networks

Switches (Layer 2):

- Connect devices within a ***local network (LAN)***
- Fast, simple forwarding based on MAC addresses
- ***Transparent***: hosts unaware of presence of switches
- Random and unique MAC addresses means no scalability beyond local networks
 - Switches must remember every single device; Can't group addresses together (E.g., *Can't say "all devices starting with XX go to output port 1"*)



Routers (Layer 3):

- Designed to interconnect ***different networks***
- Intelligent path selection between networks using logical addresses (IP address)
- Hierarchical IP addressing enables route aggregation and Internet-scale routing → More complex, requires configuration, but **scales** globally

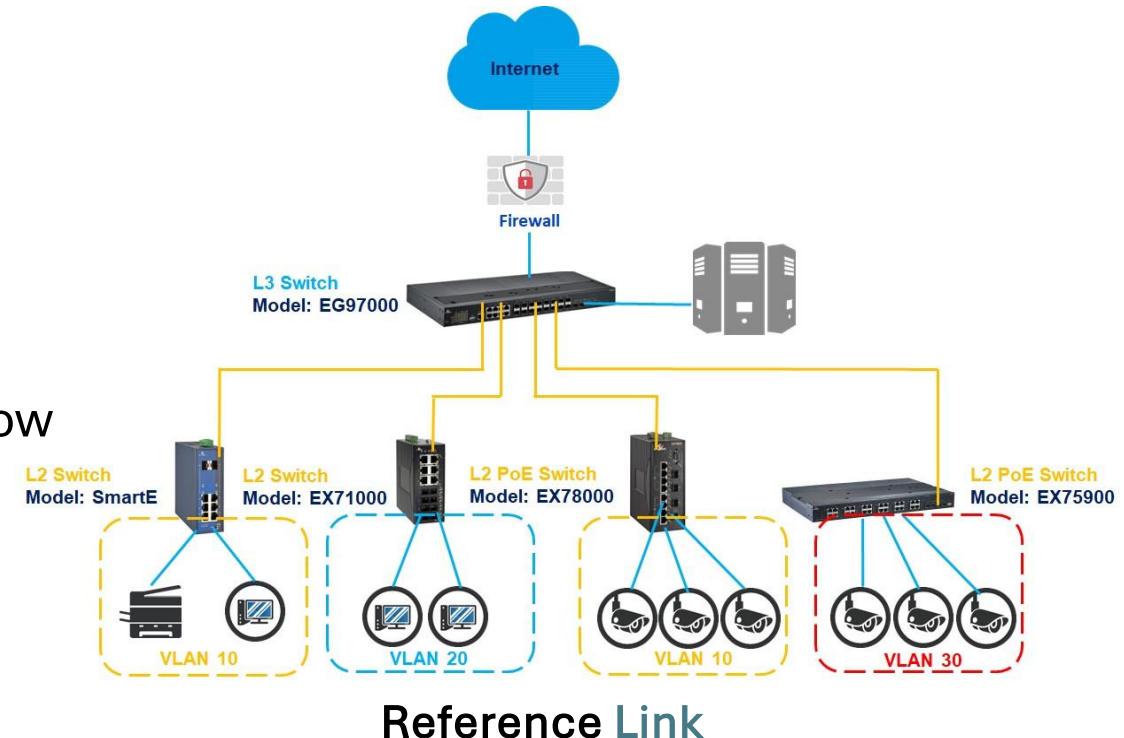
Modern convergence of switches and routers: Layer 3 switches blur the lines, but fundamental scalability differences remain

Why Layer 2 Cannot Scale to Internet

Layer 2 cannot scale to Internet-size networks →

Problem: MAC addresses have no structure

- No route aggregation: Every MAC needs separate forwarding table entry; *Internet: 5 billion devices*
- No hierarchy: Can't build efficient routing
 - Can't say “all US traffic → this way” → Must know exact path to every MAC
- Broadcast storms:
 - DHCP/ARP flood entire network → millions of broadcast messages/second



Layer 3 (IP) solves these problems:

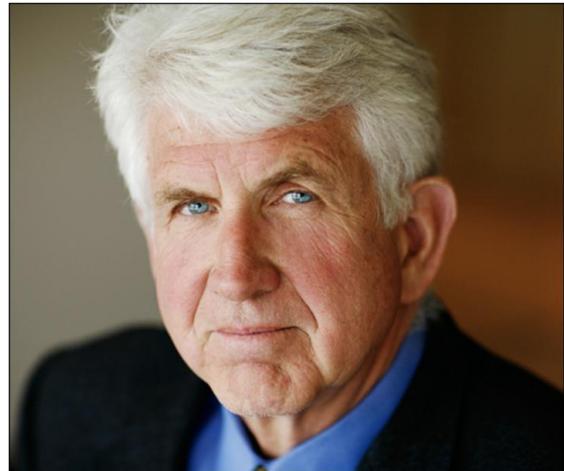
- Hierarchical addressing: IPs assigned by geographic location
- Route aggregation (Longest Prefix Matching): $10.0.0.0/8 = 16M$ addresses in **ONE** entry
- Result: BGP handles 900K+ routes efficiently

Ethernet and Ethernet Physical Topology

“Dominant” wired LAN technology:

- First widely used LAN technology (1980s)
- Simpler, cheap
- Kept up with speed race: 10 Mbps – 400 Gbps
- Single chip, multiple speeds (e.g., Broadcom BCM5761)

2022 ACM A.M. Turing Award
Laureate

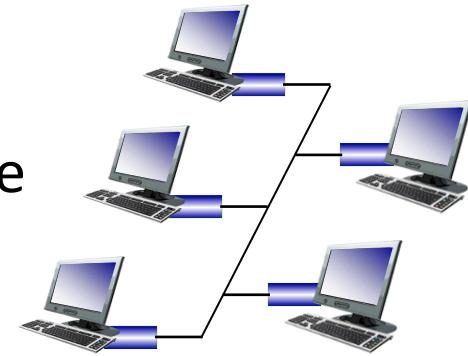


Bob Metcalfe: Ethernet co-inventor,
2022 ACM Turing Award recipient

Bus: popular through mid 90s

- All nodes in same collision domain (can collide with each other)

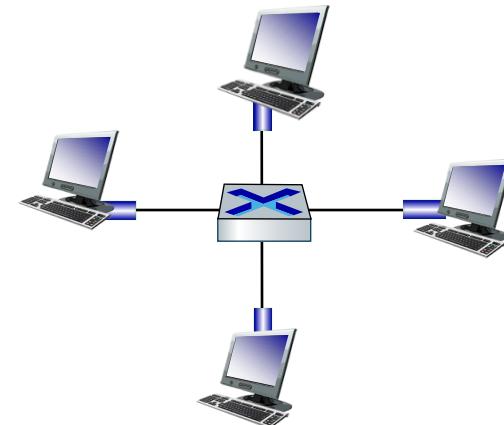
bus: coaxial cable



Switched: prevails today

- Active link-layer 2 switch in center
- Each “spoke” runs a (separate) Ethernet protocol (nodes do not collide with each other)

switched



Roadmap

- *Layer 2 Networks*
- ***Switches and Self-learning***
- *ARP*

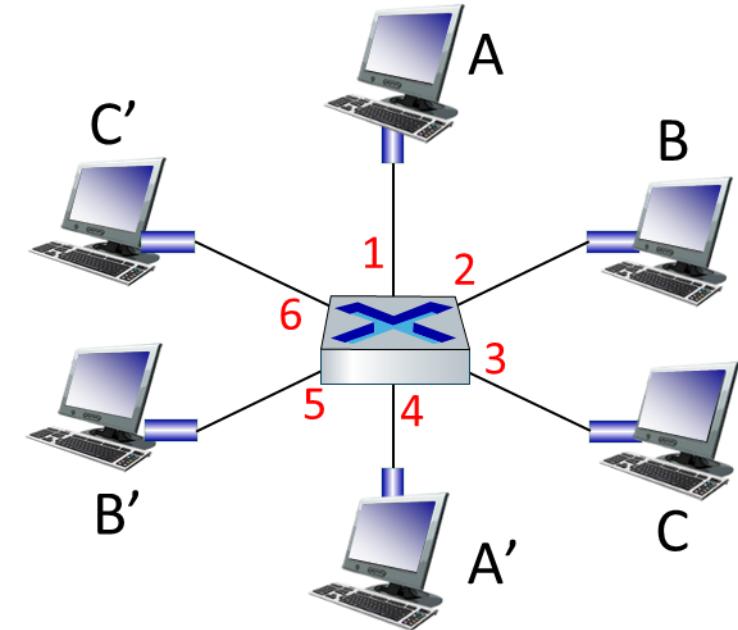
Ethernet Switch

Switch is a **link-layer** device:

- Hosts have dedicated, direct connection to switch
- Store, buffer, and forward Ethernet frames

Switches vs. Routers:

- Router forwarding tables: Must be ***manually*** configured
 - Network admin configures IP routing tables
 - Requires knowledge of network topology
- Switch MAC Address Table (aka switch table or forwarding table for switches):
 - Plug-and-play: No configuration needed
 - Learns MAC addresses by observing traffic
 - ***Automatically*** builds forwarding table → ***Self-learning***



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

- Dedicated wires for each connection → Each link is point-to-point
- No shared medium = impossible to collide!
- Each port has its own buffer
- Switching: A-to-A' and B-to-B' can transmit simultaneously, ***without collisions***

Self-learning and Forwarding

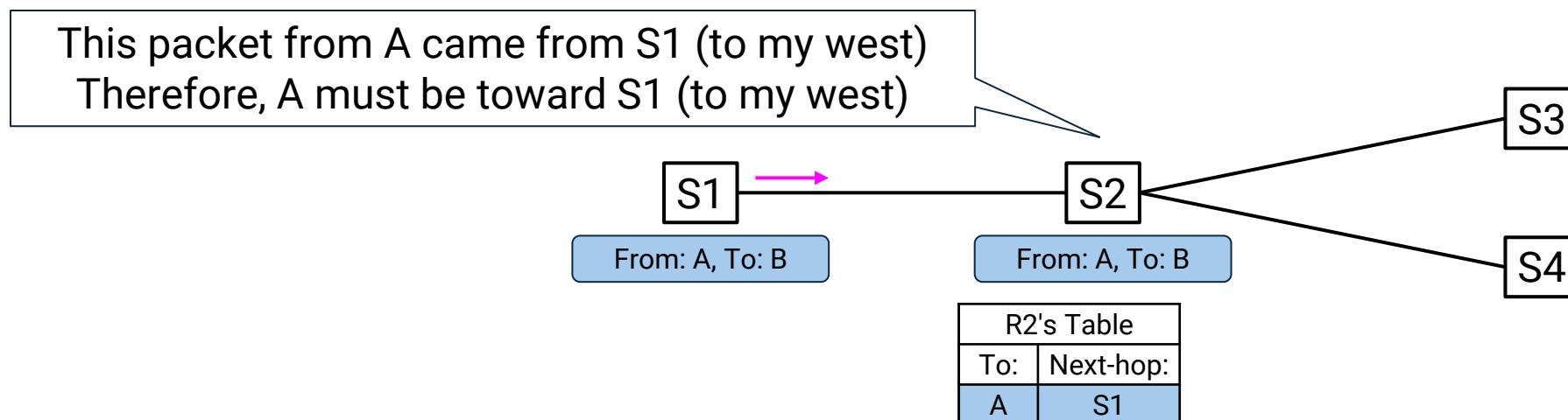
Q: How to “**learn**” and build a forwarding table?

Key idea: When you receive a data packet, you get a clue about where the **sender** is

- If a packet from host A comes from the west...
- ...then host A must be to my west!

Use this idea to add a table entry for the sender

- As more packets get sent, more table entries get added
- Note: The packet only helps you learn about the sender, not the receiver

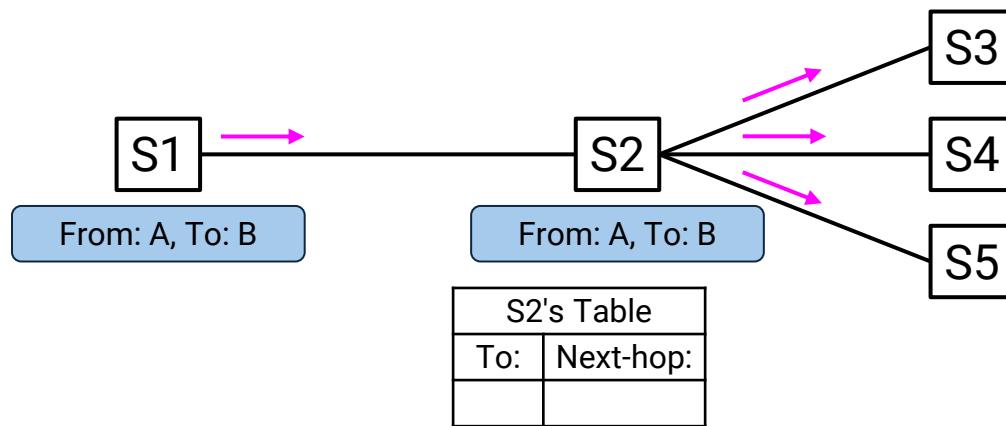


Self-learning and Forwarding

Q: How to forward a frame?

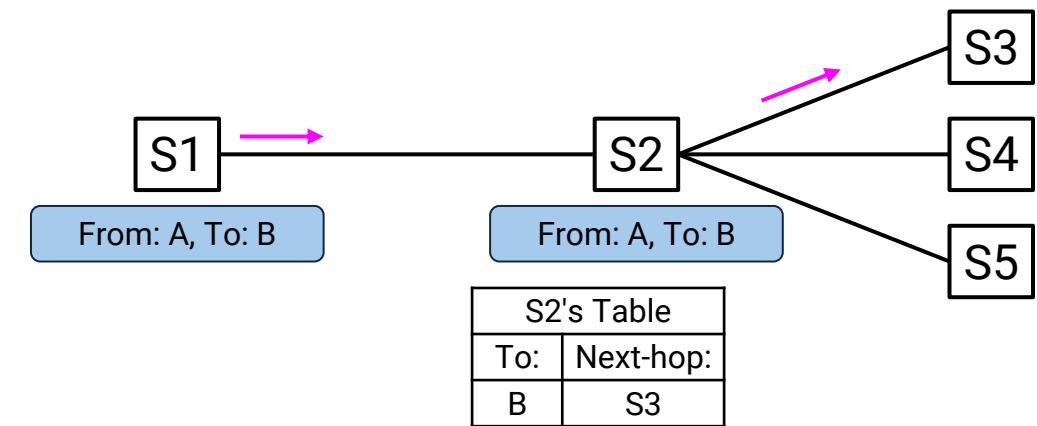
Key idea: Use the destination to decide the next-hop

- If the destination exists in the table: Forward to corresponding next-hop
- If the destination is not in the table: *Flood* of all ports (except incoming port)



Case 1: No entry for B (destination) in table

Flood the packet to all ports
(except incoming port)



Case 2: Entry for B (destination) is in table

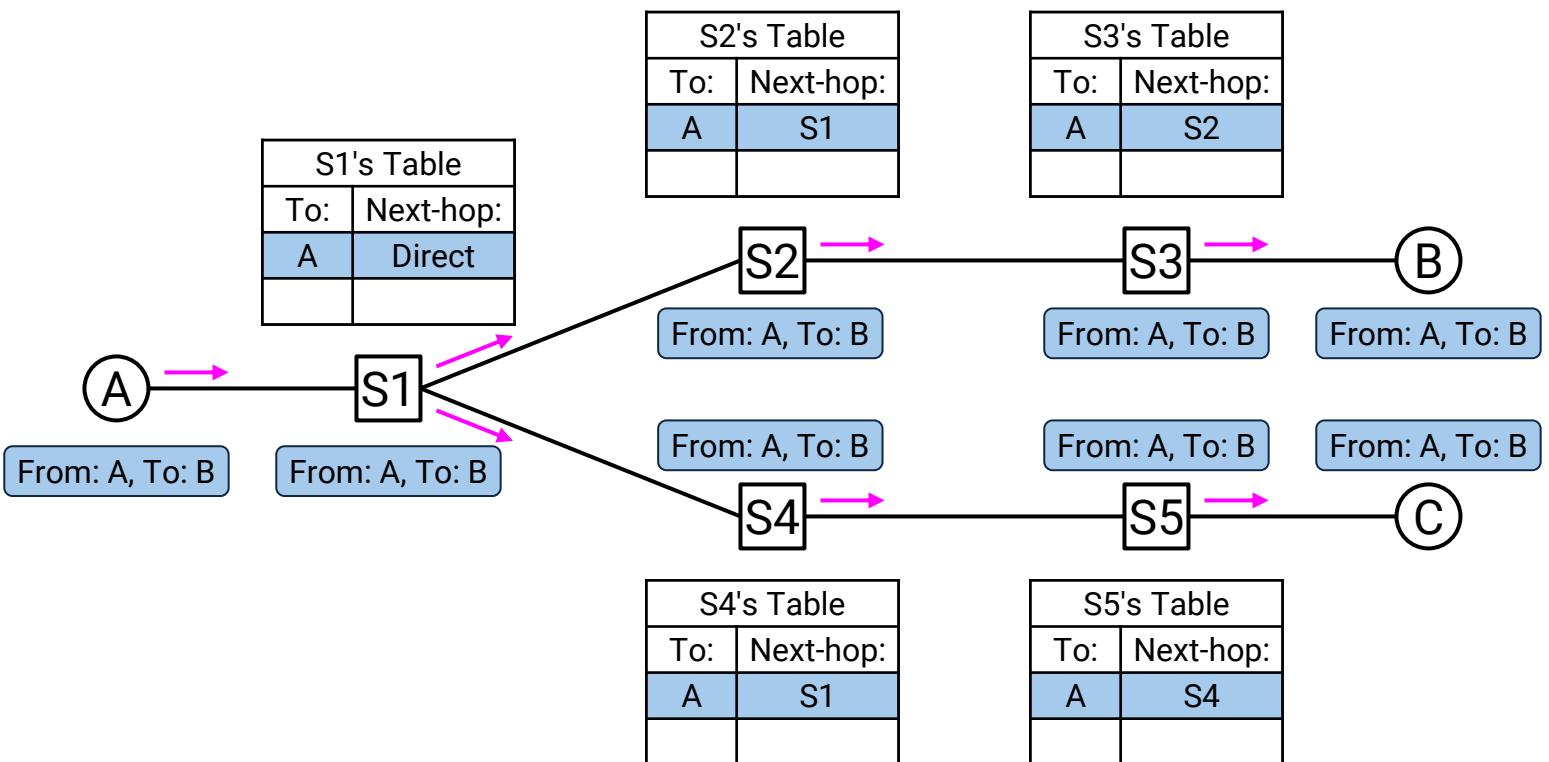
Use table entry to forward to next-hop

Self-learning and Forwarding

Let's build the forwarding table using “self-learning”:

- Host A wants to send a frame to host B
- All tables start empty. Nobody has an entry for B, so the packet gets flooded
- Along the way, everyone gets a clue about A's location, and adds a table entry for A

1. A forwards the packet to S1 (Port 1)
2. S1 *learns A must be toward Port 1*
3. S1 floods this packet out of all ports (except the incoming port)
4. S2 and S4: Both receive the “From A, to B” packet from S1; *Add a mapping for A* to their forwarding tables; Flood...
5. S3 and S5: *Update* forwarding tables; Flood the packet... (except...)
6. C: Receives packet; Check the header; Drop the packet
7. B: Receives this packet

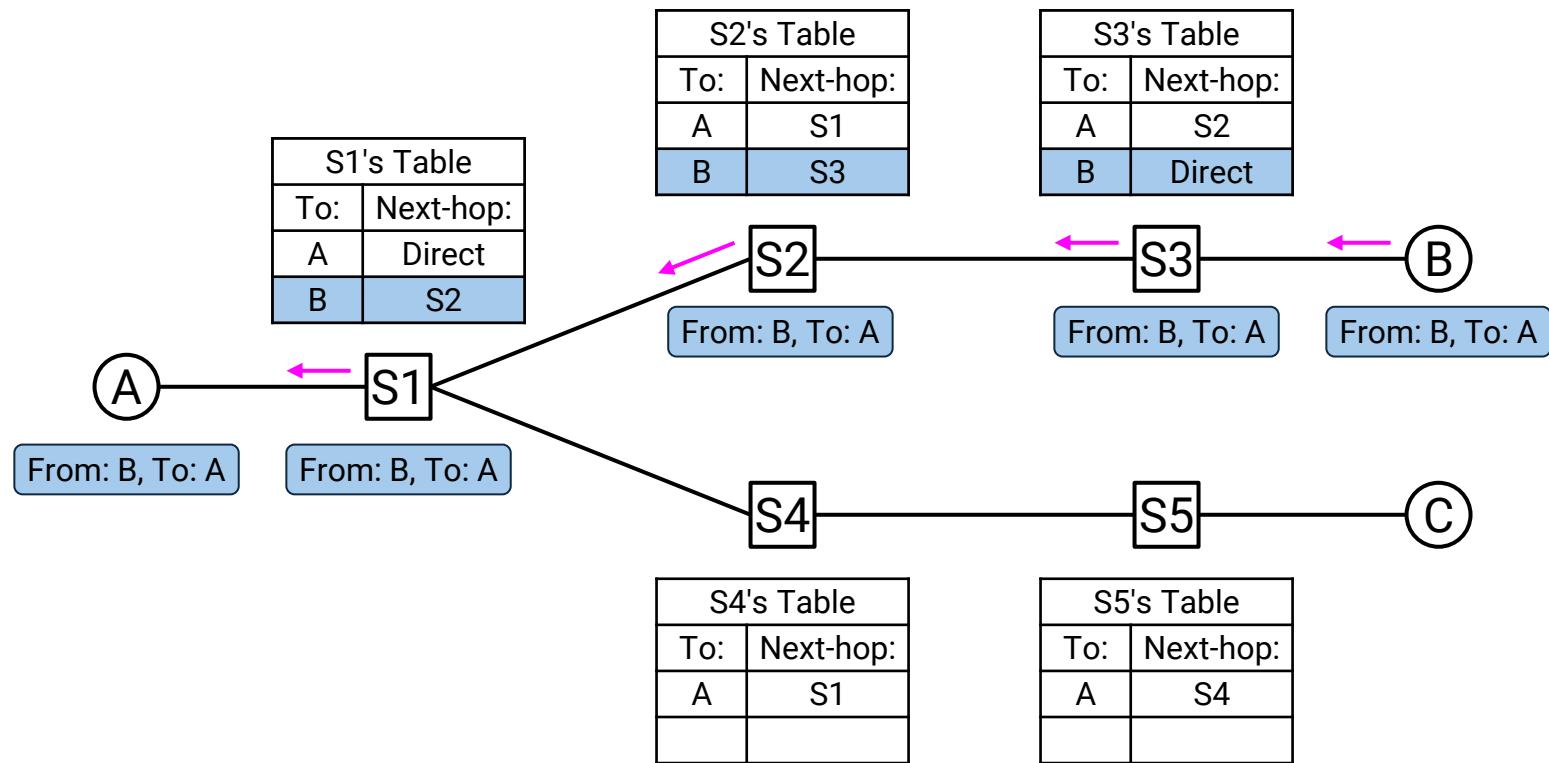


Self-learning and Forwarding

When host B wants to reply to host A:

- This packet can get directly forwarded to A (no flooding) using the forwarding tables
- Switches along the B-to-A path get a clue about B's location, and add a table entry for B

1. B forwards the packet to S3 (Port 2)
2. S3 *learns B must be toward Port 2*
3. S3 **forwards** the packet along the next-hop to A (instead of flooding the packet)
4. S2: Adds a mapping for B to its forwarding table; **Forwards** packet to S1
5. S1: Adds a mapping for B to its forwarding table; **Forwards** packet to A



Self-Learning and Forwarding Algorithm

When frame received at switch:

Step 1: Learn (from source MAC)

- Record incoming port and source MAC address in table
 - “MAC_source_addr is reachable via port X”

Step 2: Forward (using destination MAC)

- Look up destination MAC address in table

Step 3: Forwarding decision

- *if* entry found for destination:
 - forward frame out the port indicated in table entry
- *else* (destination unknown):
 - flood → forward on all ports except arriving port

Learning:

- Uses *source MAC* to build table

Forwarding:

- Uses *destination MAC* to make decisions

Plug-and-play:

- No configuration needed, learns automatically

Roadmap

- *Layer 2 Networks*
- *Switches and Self-learning*
- **ARP**

ARP: Connect Layer 2 and Layer 3

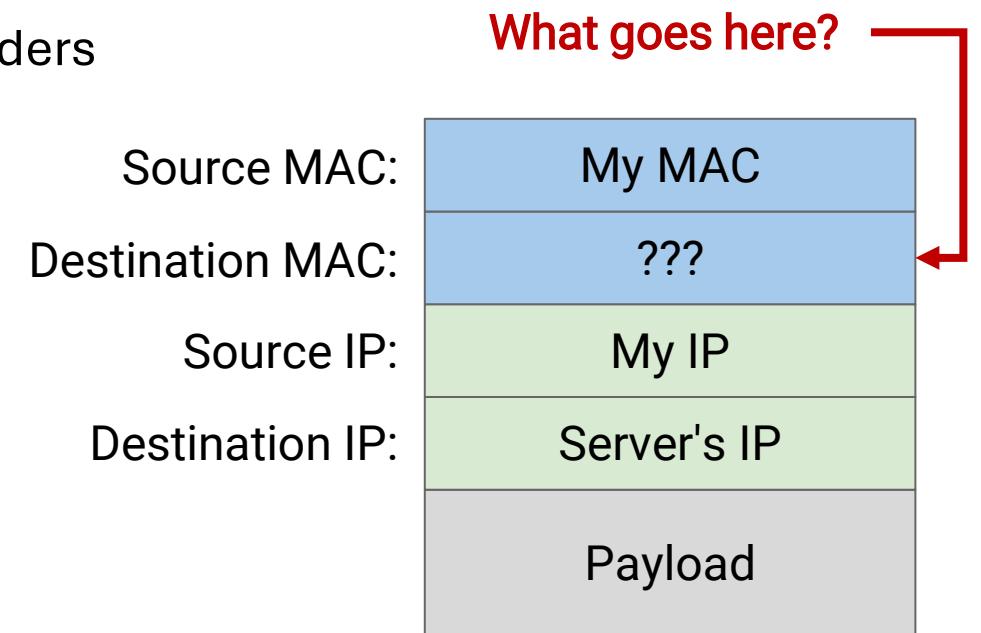
Recall: Packet gets passed down the stack, picking up more headers

Layer 3 fills in the IP addresses

Then, Layer 2 needs to fill in the MAC addresses

Q: How to fill in the destination MAC address?

1. If the destination IP is in our local network:
 - Find the **destination's** MAC address, and send to destination on Layer 2
2. If the destination IP is *not* in our local network:
 - Find the **router's** MAC address, and **send to the router** on Layer 2
 - Router can forward our packet toward the destination



ARP: Connect Layer 2 and Layer 3

Q: How do we send packets to the destination (local) or the router (non-local)?

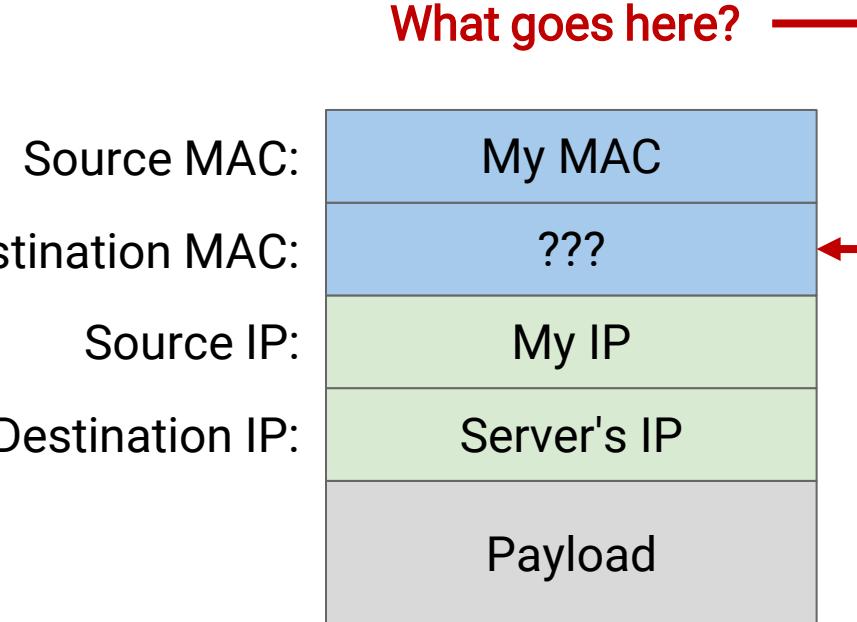
A: We need some way to translate IP addresses to MAC addresses.

ARP translates Layer 3 IP addresses to Layer 2 MAC addresses

- Example: Alice knows Bob's IP address is 1.2.3.4
She wants to know Bob's MAC address

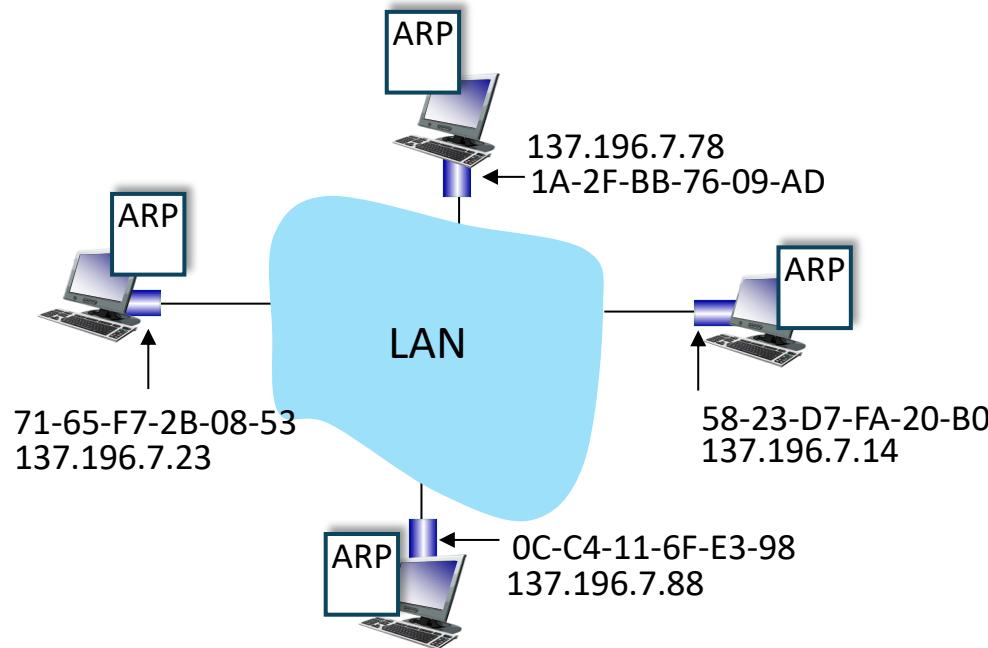
Steps of the protocol:

1. Alice checks her **cache** (ARP table) to see if she already knows Bob's MAC address
2. If Bob's MAC address is not in the cache, Alice **broadcasts**: "What is the MAC address of 1.2.3.4?"
3. Bob responds by unicasting to Alice: "My IP is 1.2.3.4 and my MAC address is ca:fe:f0:0d:be:ef." Everyone else does nothing.
4. Alice caches the result



ARP: Address Resolution Protocol

Q: How to determine interface's MAC address, knowing its IP address?



ARP table: each node (host, router) on LAN has table
<IP address; MAC address; TTL>

ARP table		
IP addr	MAC addr	TTL
137.196.7.14	58-23-D7-FA-20-B0	500

- IP/MAC address mappings for some LAN nodes
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

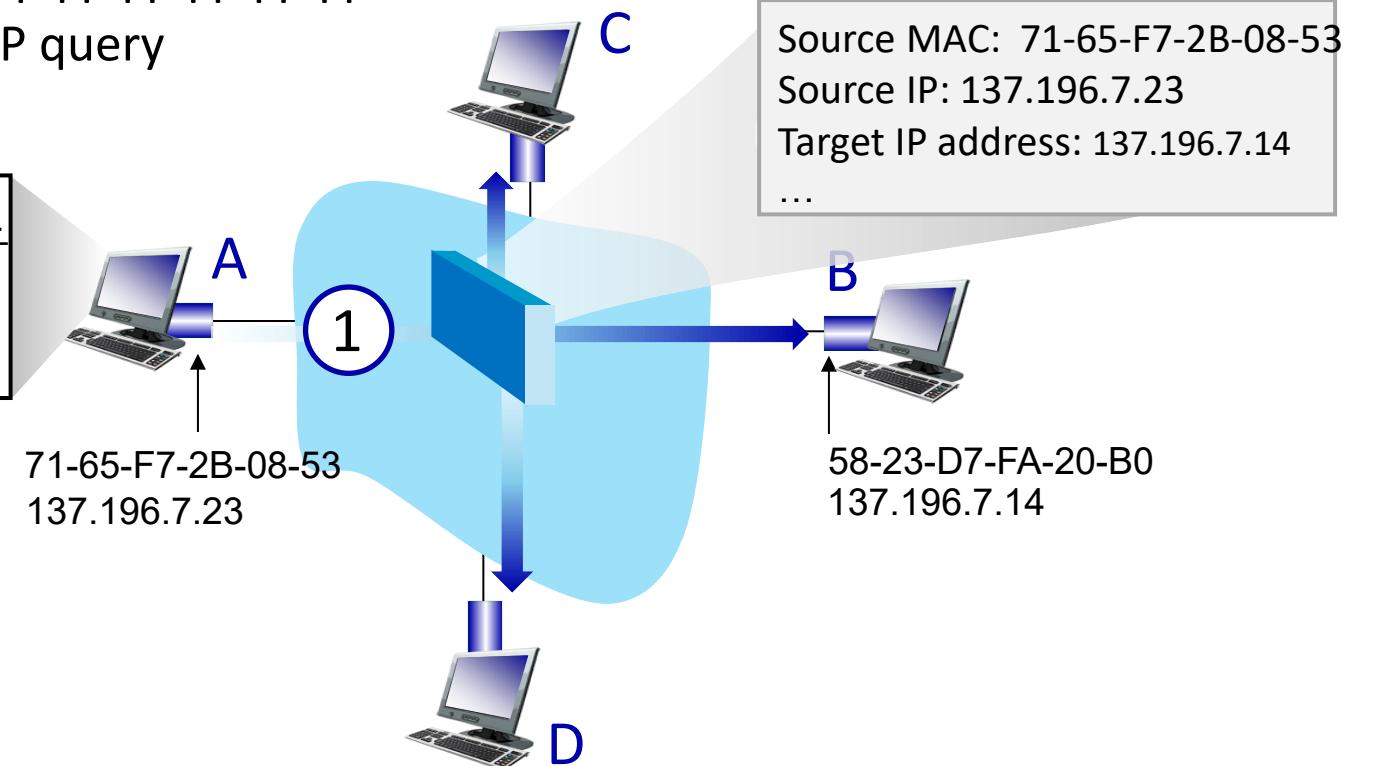
ARP: Address Resolution Protocol

Example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address

- 1 A broadcasts ARP query, containing B's IP addr
- destination MAC address = FF-FF-FF-FF-FF-FF
 - all nodes on LAN receive ARP query

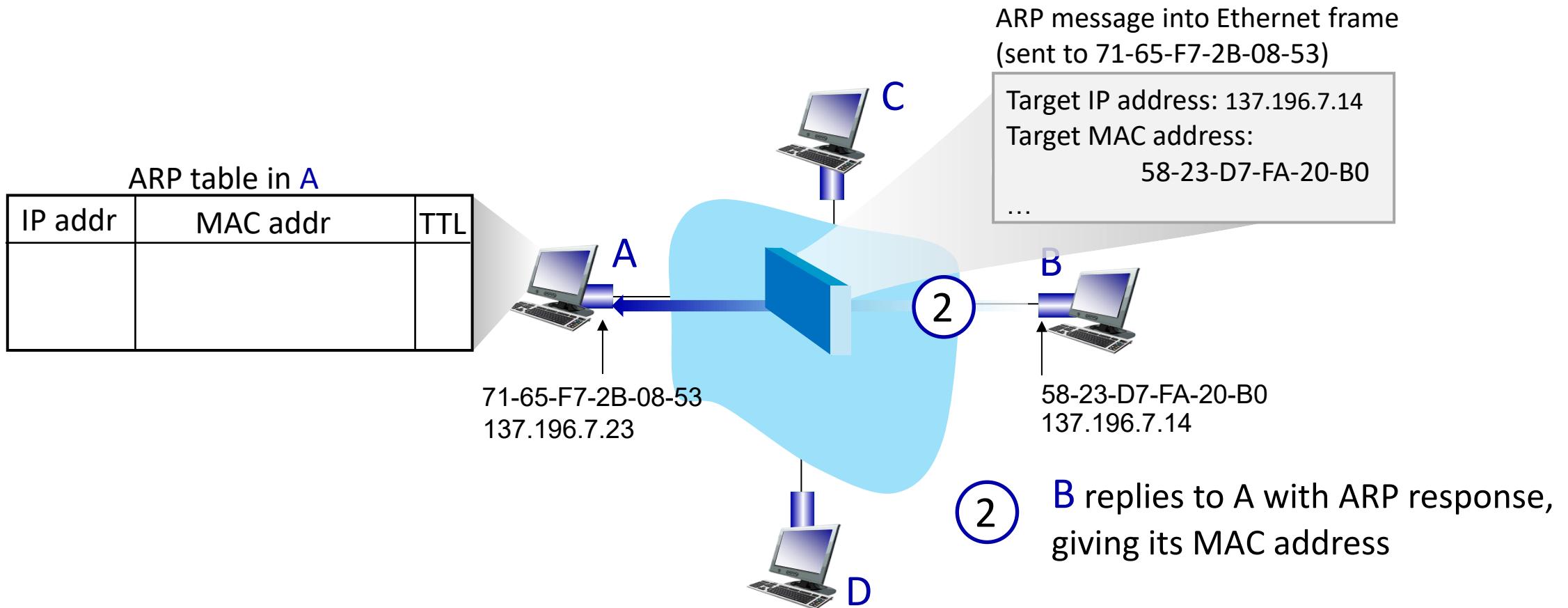
ARP table in A		
IP addr	MAC addr	TTL



ARP: Address Resolution Protocol

Example: A wants to send datagram to B

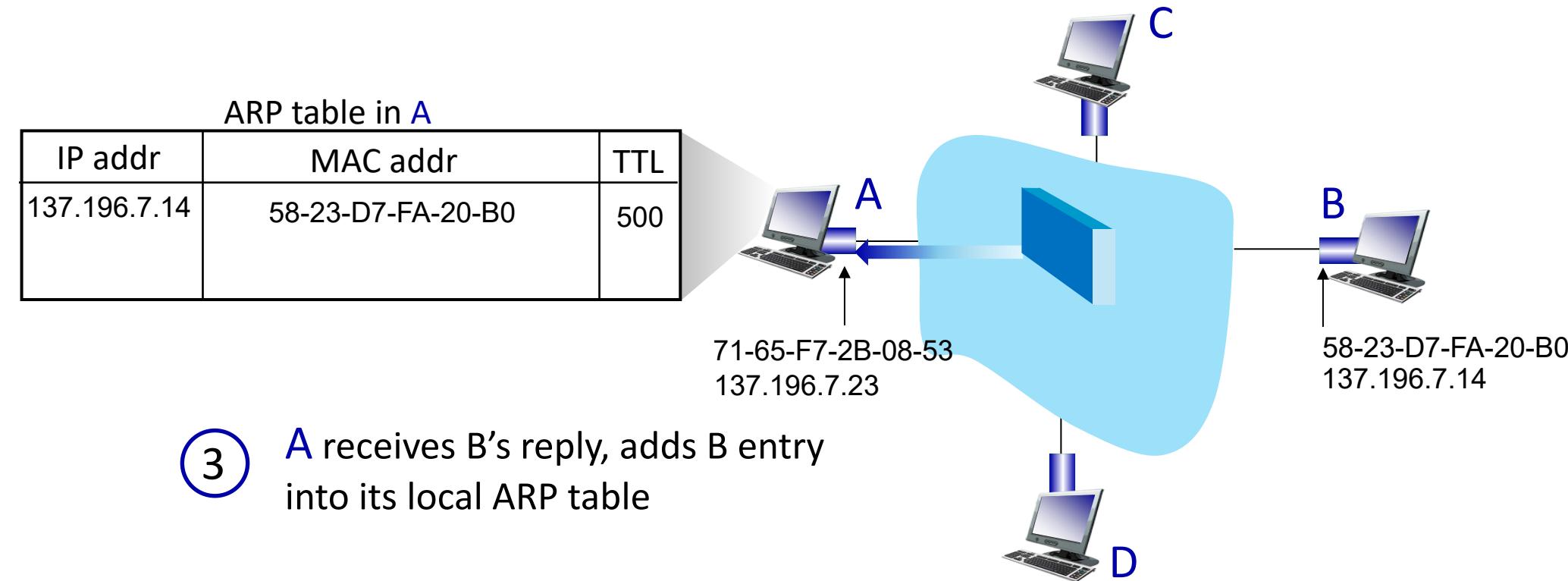
- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



ARP Protocol in Action

Example: A wants to send datagram to B

- B's MAC address not in A's ARP table, so A uses ARP to find B's MAC address



Using ARP in Routers and Hosts

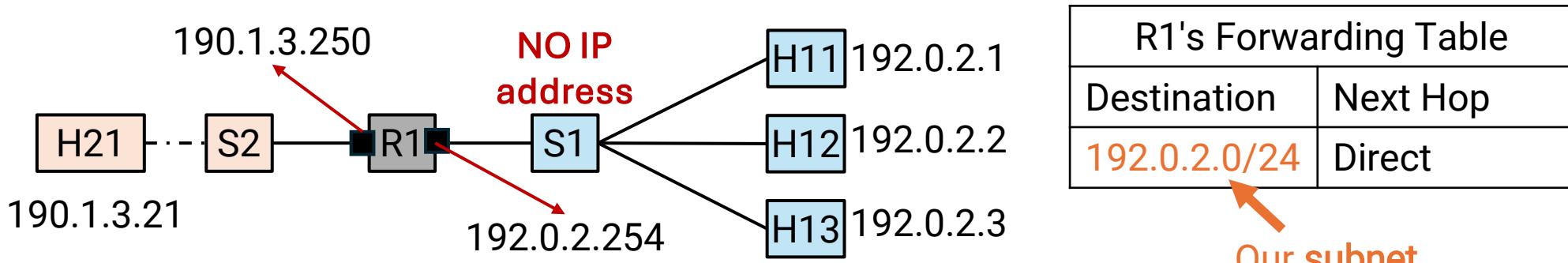
ARP runs directly on Layer 2 (not Layer 3)



Note: Any router/host can broadcast an unsolicited response ("Gratuitous ARP")

- Device broadcasts its own IP-to-MAC mapping without being asked
- Common when device boots up or changes IP/MAC
- Purpose: Announce presence, detect IP conflicts, update neighbors' ARP caches
- Example: "My IP is 1.2.3.4, and my MAC is ca:fe:f0:0d:be:ef"

Connecting ARP and Forwarding Tables



The IP forwarding table contains IP addresses...
...and addresses can be **aggregated!**

The router's forwarding table contains a single entry, mapping the entire subnet's range of IP addresses to be direct

If the router (R1) receives a packet whose destination (e.g., H1) is in this local range:

- The router runs ARP (or look in our cached ARP table) to find the corresponding MAC address
 - H1 replies through S1 with its MAC address
- Uses Layer 2 to send the unicast Ethernet frames to the correct host on the link
 - R1 sends packet in **frame** (dest MAC = H1's MAC) to S1
 - S1 looks up H1's MAC in its table && S1 forwards to H1
- Router **doesn't "skip"** the switch → the switch is transparent but still in the data path!

Using ARP in Hosts

How do hosts forward packets?

1. Determine if destination is local (same LAN) or non-local:
 - Uses subnet mask to check. (See note at bottom of slide for example)
2. Use ARP based on destination:
 - If destination IP is *local*:
Use ARP to find MAC of destination. Send packet to destination
 - If destination IP is *non-local*:
Use ARP to find MAC of router. Send packet to router

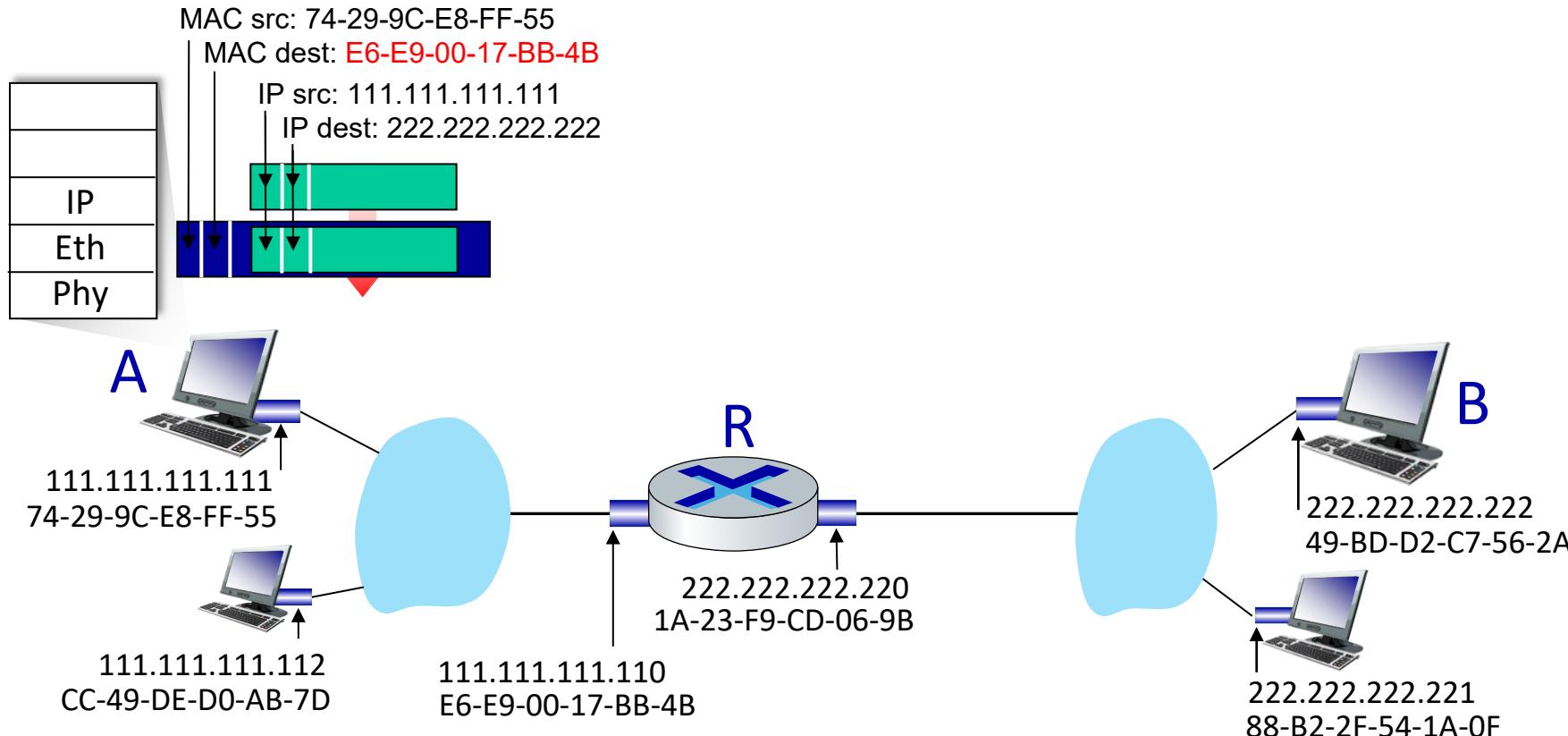
Q: How does host know router's IP and subnet mask?

A: DHCP provides all network configuration:

- IP address for the host (e.g., **192.168.1.5**)
- Subnet mask (e.g., **255.255.255.0**)
- Default gateway (router's IP, e.g., **192.168.1.1**)
- DNS server(s) (e.g., **8.8.8.8**)
- Lease time (how long IP is valid)

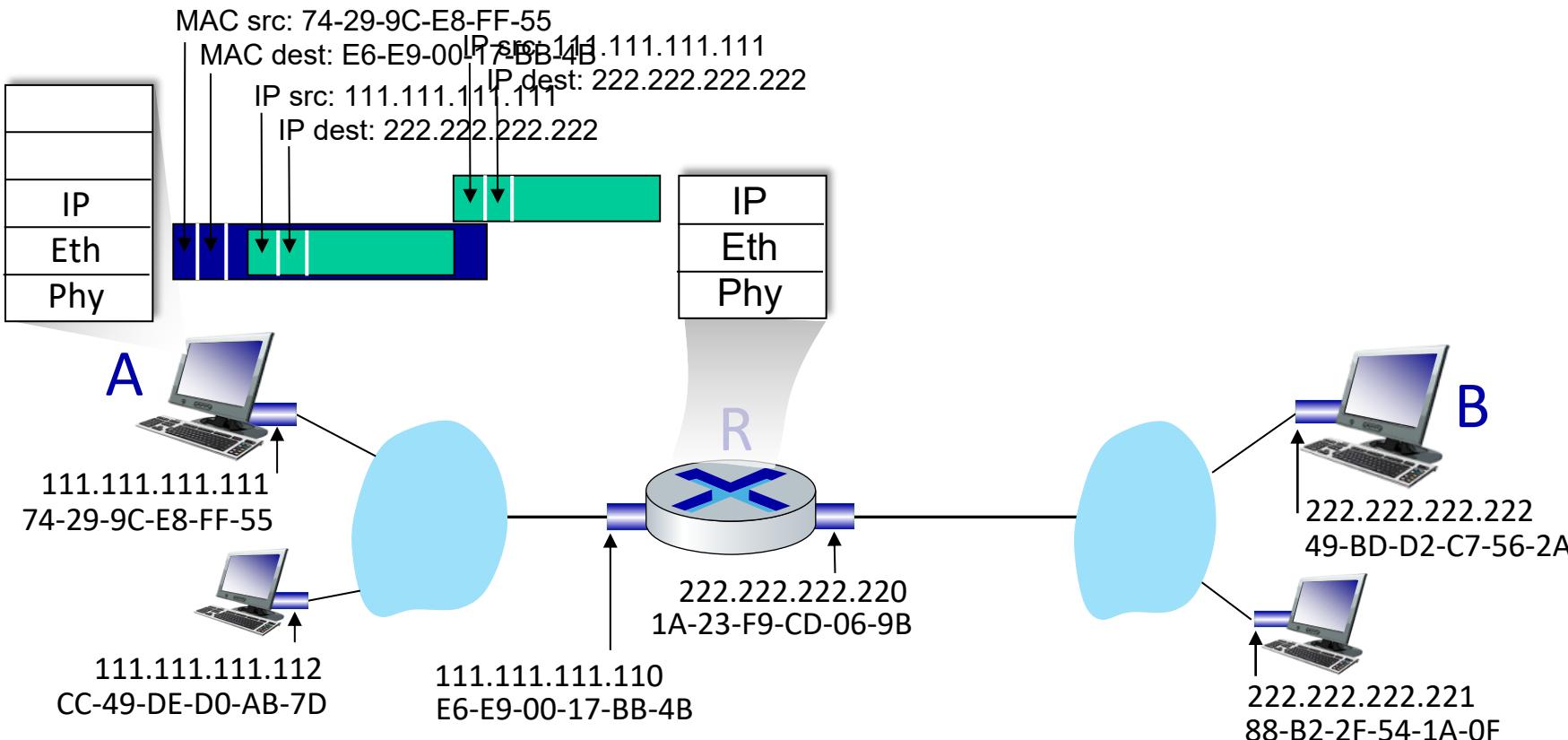
Sending to Another Network: MAC = Next Hop, IP = Final Destination

- A creates IP datagram with IP source A, destination B
- A creates link-layer frame containing A-to-B IP datagram
 - R's MAC address is frame's destination



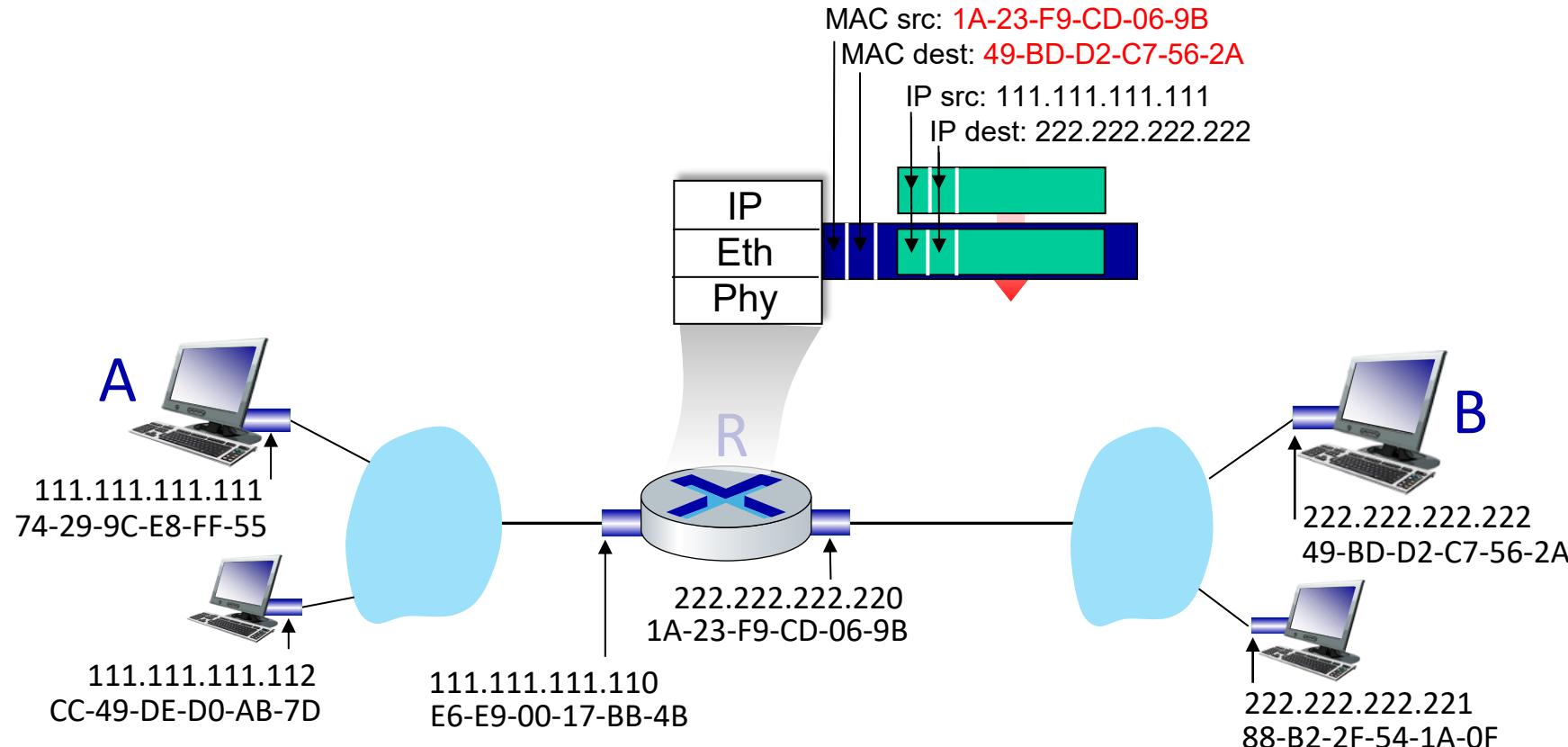
Sending to Another Network: MAC = Next Hop, IP = Final Destination

- Frame sent from A to R
- Frame received at R, datagram removed, passed up to IP



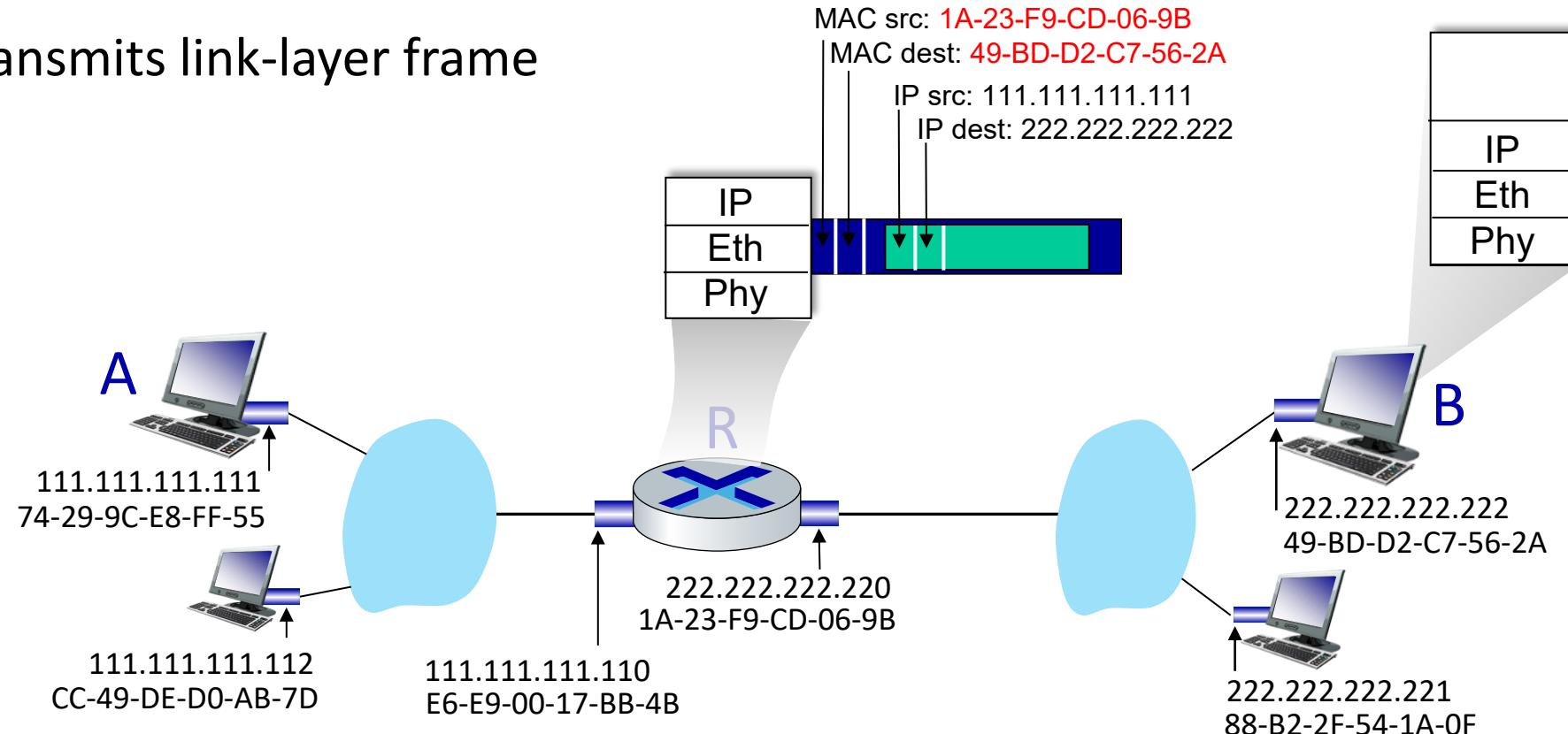
Sending to Another Network: MAC = Next Hop, IP = Final Destination

- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address



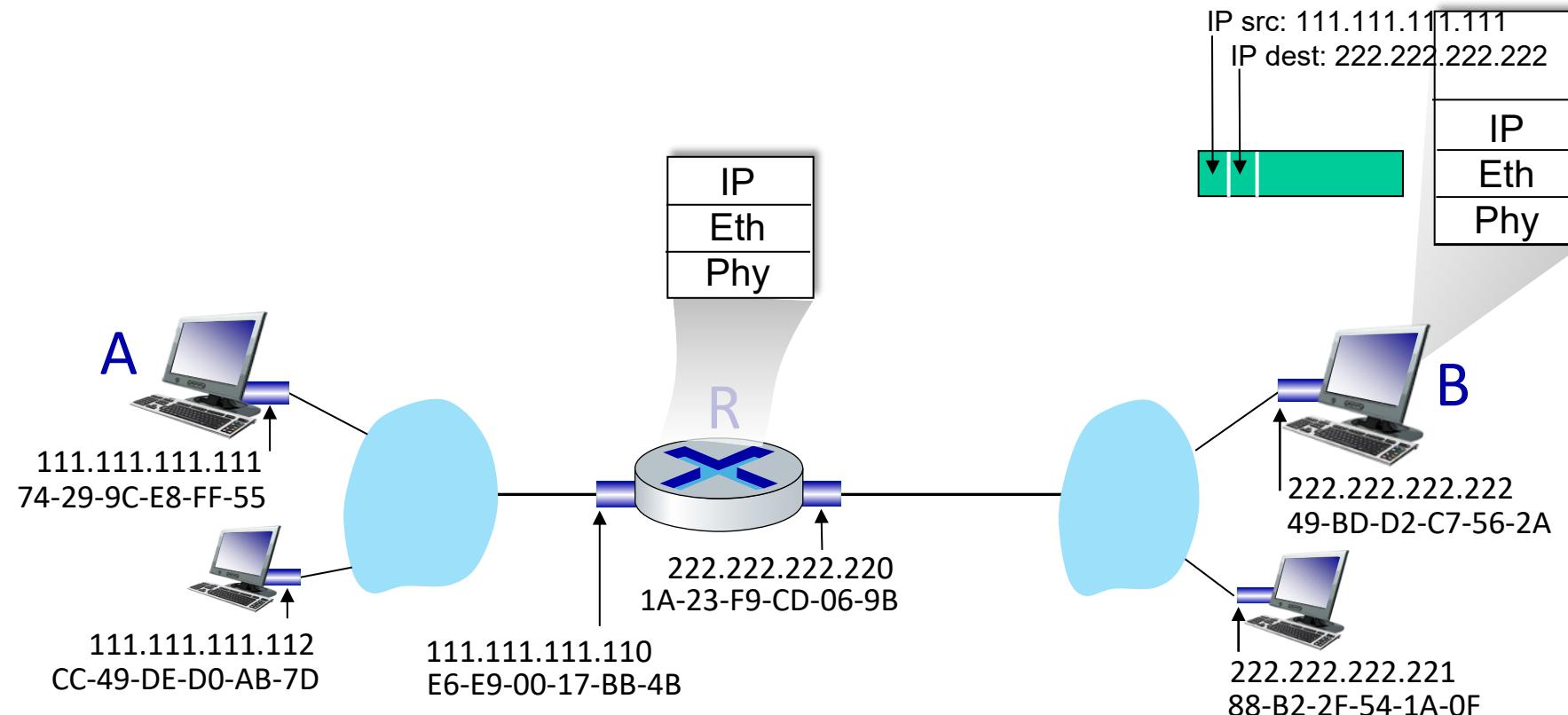
Sending to Another Network: MAC = Next Hop, IP = Final Destination

- R determines outgoing interface, passes datagram with IP source A, destination B to link layer
- R creates link-layer frame containing A-to-B IP datagram. Frame destination address: B's MAC address
- Transmits link-layer frame



Sending to Another Network: MAC = Next Hop, IP = Final Destination

- B receives frame, extracts IP datagram destination B
- B passes datagram up protocol stack to IP



CSC 3511 Security and Networking

Week 7, Lecture 2: Link Layer

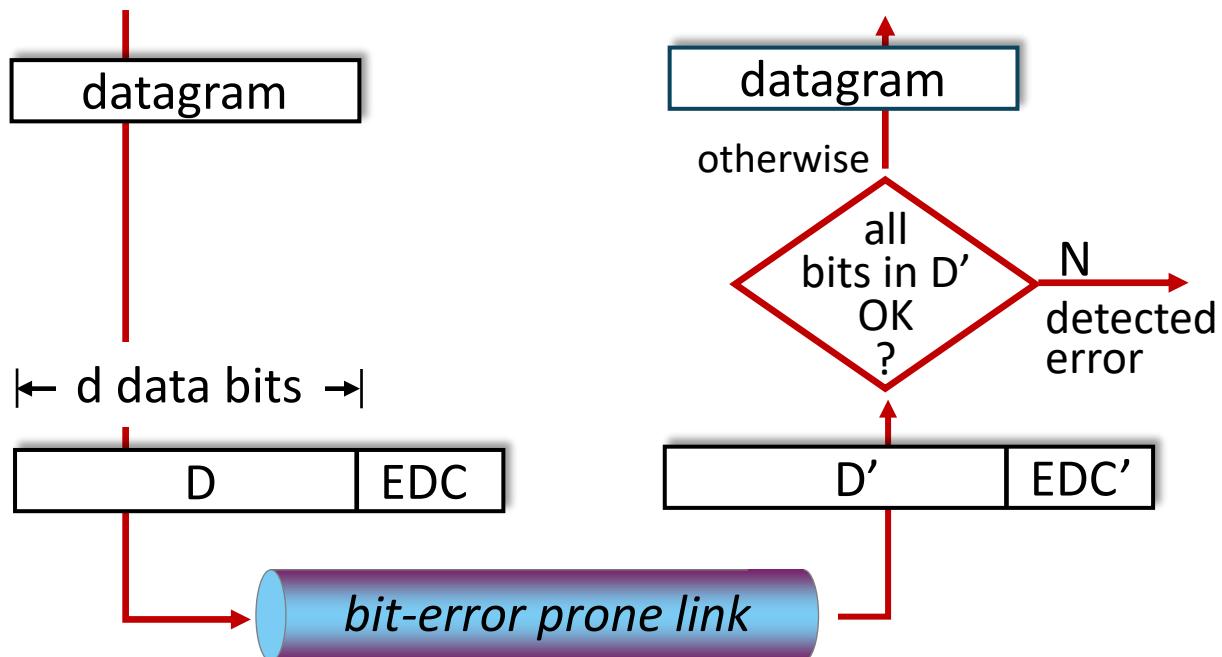
Roadmap

- ***Error Detection and CRC***
- ***Synthesis: A Day in the Life of a Web Request***

Error Detection

EDC: error detection and correction bits (e.g., redundancy)

D: data protected by error checking, may include header fields



Error detection not 100% reliable!

- Protocol may miss some errors, but rarely
- Larger EDC field yields better detection and correction

Checksum (Review, see W3D2 UDP)

Goal: Detect errors (*i.e.*, flipped bits) in transmitted datagrams/segments

Sender (UDP):

- Prepare the UDP packet
 - Header: source & destination port, checksum (initialized as 0)
 - Data: The actual payload
- Create the pseudo-header
 - Source IP, destination IP, protocol number (17 for UDP), total length (header & data)
- Compute & insert checksum
 - A special case: if the computed checksum is zero; it's transmitted as **0xFFFF**

Receiver (UDP):

- Receive the UDP packet
- Rebuild the pseudo-header
- Verify checksum of received segment:
 - Add everything together, verify if the result is all 1s (1111 1111 1111 1111)
 - Not all 1s → **Error detected**
 - All 1s → No error detected. *But maybe errors nonetheless?* More later

Parity Checking

Single bit parity:

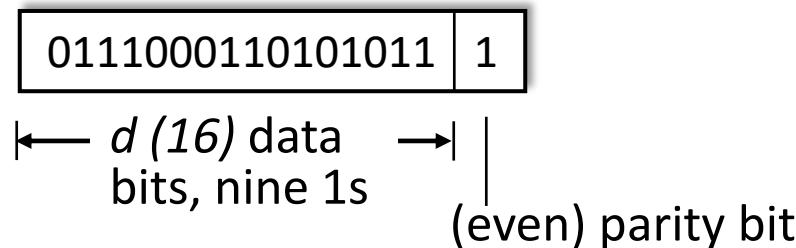
- Add one extra bit to make the total number of 1s either even or odd
- **Detect single bit errors**

Even parity:

Total number of 1s (including the parity bit) should be **even**

Odd parity:

Total number of 1s (including the parity bit) should be **odd**

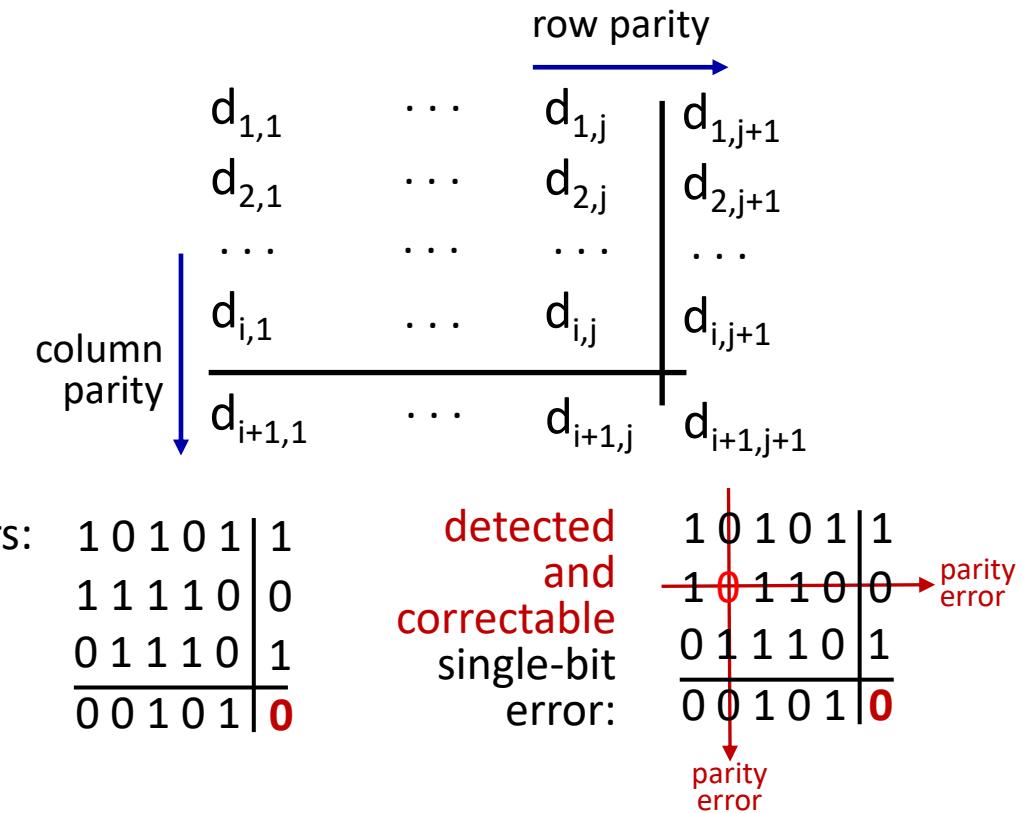


At receiver:

- Compute parity of d received bits
- Compare with received parity bit – if different than error detected

Two-dimensional parity:

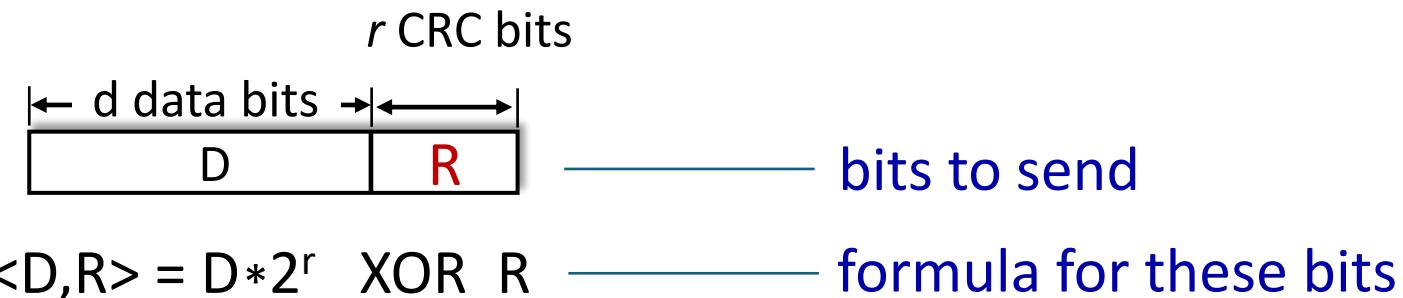
- Detect **and correct** single bit errors (without retransmission!)
- See the even parity example below



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/2d_parity.php

Cyclic Redundancy Check (CRC)

- *More powerful* error-detection coding
- Treat data as a **polynomial**, divide by a **generator polynomial (G)**, use **remainder** as checksum
- D: data bits (given, think of these as a binary number)
- G: bit pattern (generator), of $r+1$ bits (given, specified in CRC standard)



Sender: compute r CRC bits, **R**, such that $\langle D, R \rangle$ exactly divisible by G ($\text{mod } 2$)

- Receiver knows G , divides $\langle D, R \rangle$ by G . If non-zero remainder: error detected!
- Can detect all burst errors less than $r+1$ bits
- Widely used in practice (Ethernet, 802.11 WiFi)

Cyclic Redundancy Check (CRC): Example

Sender wants to compute R , such that:

$$D \cdot 2^r \text{ XOR } R = nG$$

... or equivalently (XOR R both sides):

$$D \cdot 2^r = nG \text{ XOR } R$$

... which says:

if we divide $D \cdot 2^r$ by G , we want remainder R to satisfy:

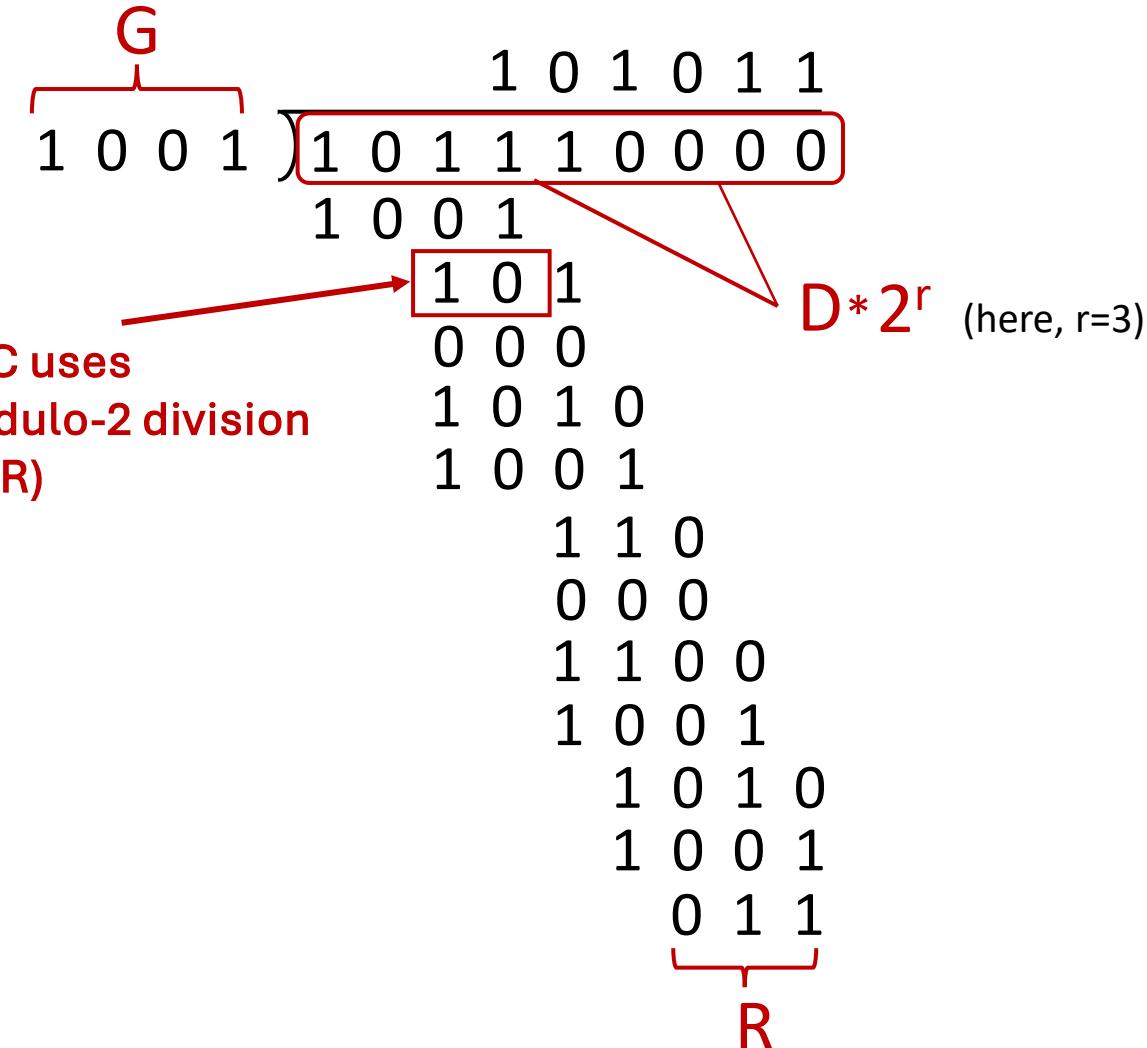
$$R = \text{remainder} \left[\frac{D \cdot 2^r}{G} \right]$$

algorithm for computing R

Generator polynomial: $x^3 + 1 \rightarrow$ Binary: 1001 (4 bits)

Data to send: 101110 (6 bits)

Step 1: Append 3 zeros to data (degree of generator = 3)



- Check out the online interactive exercises for more examples:
http://gaia.cs.umass.edu/kurose_ross/interactive/CRC.php

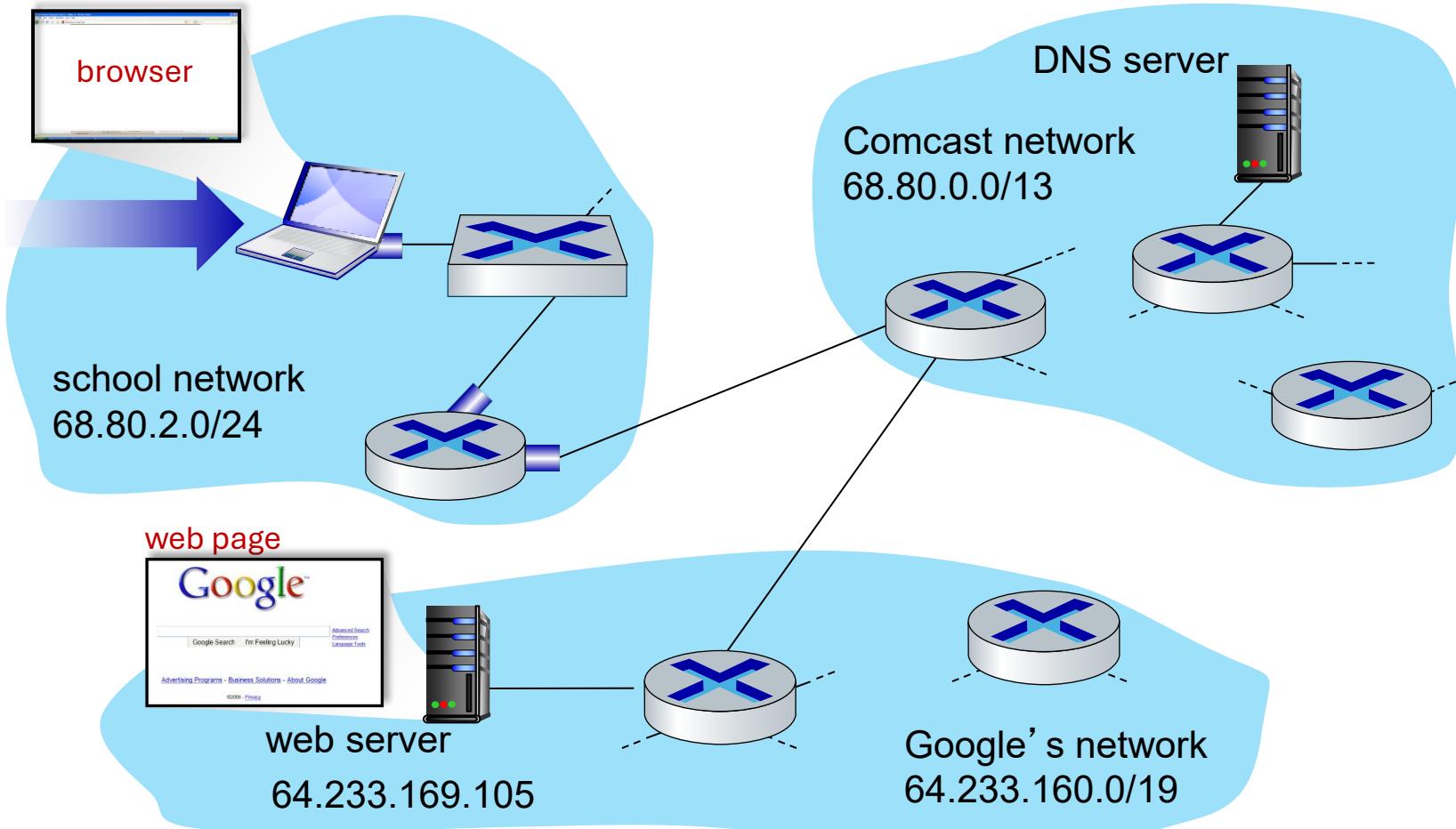
Roadmap

- *Error Detection and CRC*
- ***Synthesis: A Day in the Life of a Web Request***

Synthesis: A Day in the Life of a Web Request

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

A Day in the Life: Scenario

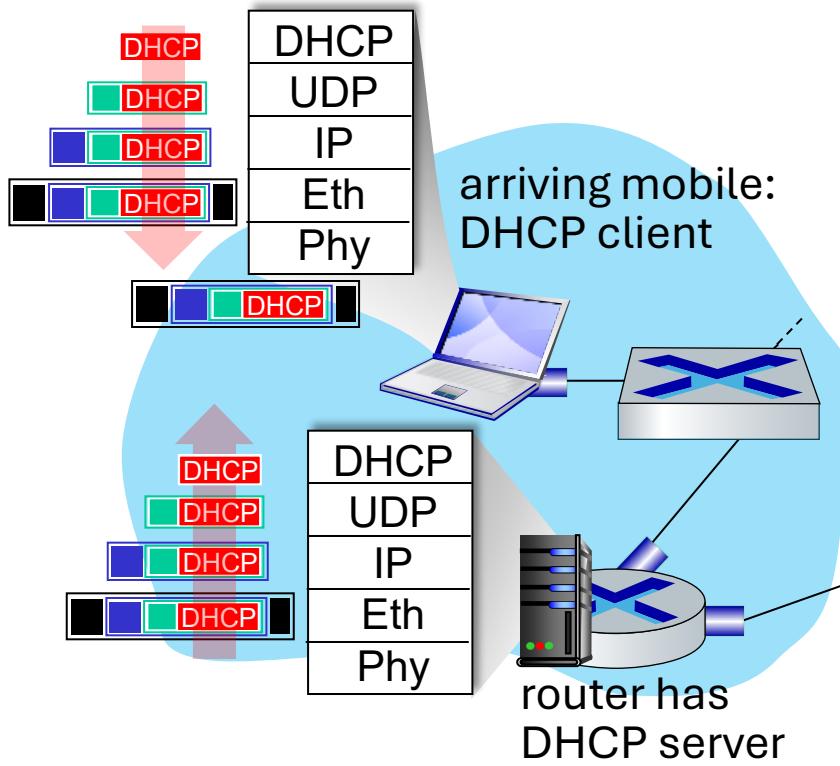


scenario:

- Arriving mobile client attaches to network ...
- Requests web page:
www.google.com

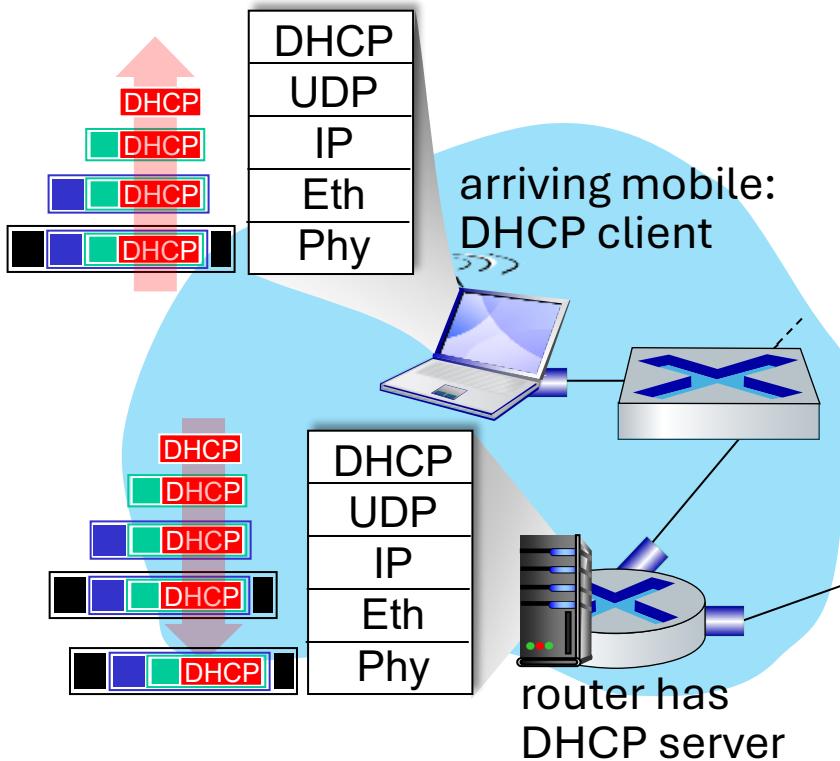
Sounds simple!

A Day in the Life: Connecting to the Internet



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

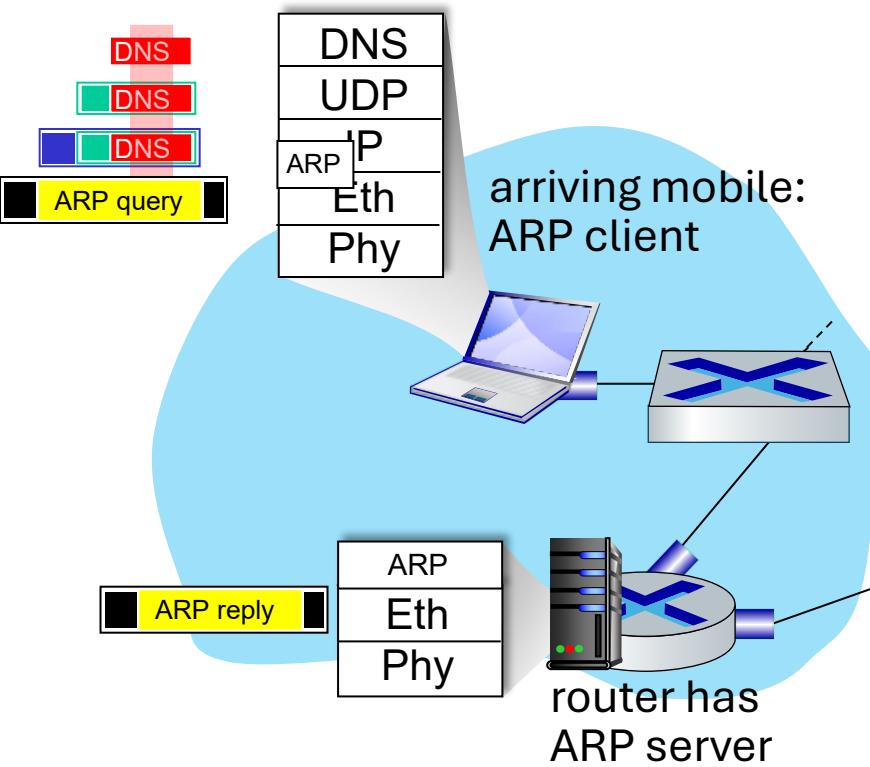
A Day in the Life: Connecting to the Internet



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

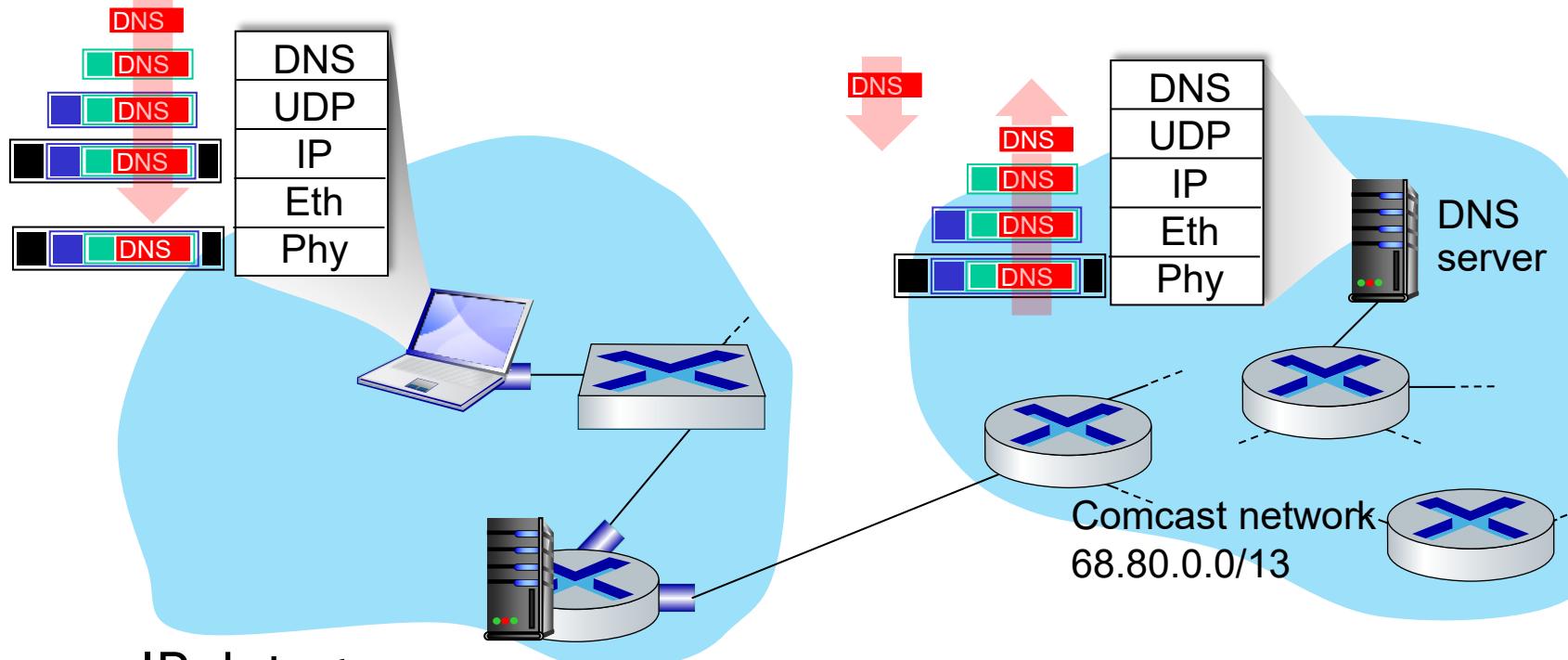
Client now has IP address, subnet mask, knows name & addr of DNS server, IP address of its first-hop router

A Day in the Life... ARP (before DNS, before HTTP)



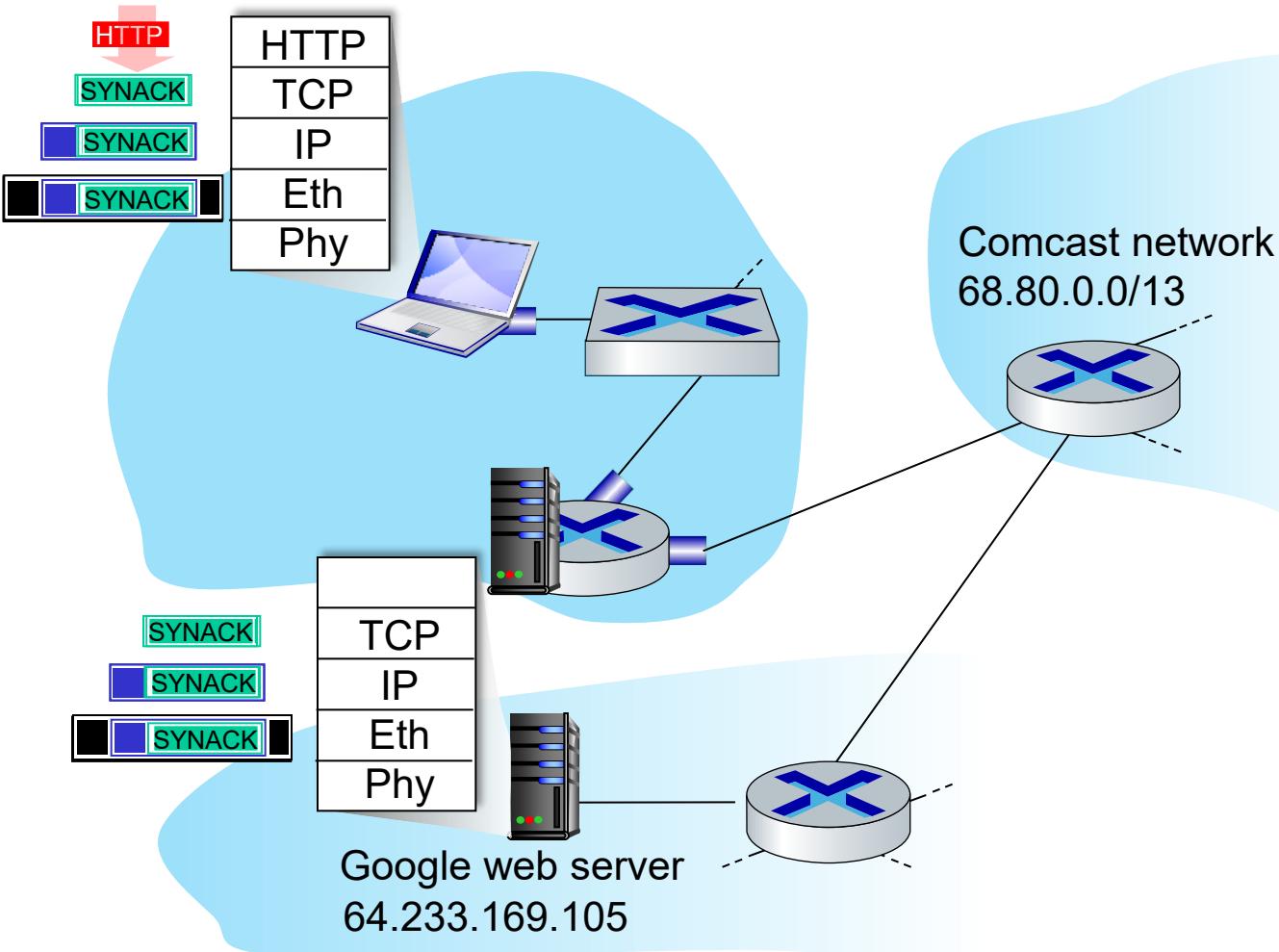
- Before sending **HTTP** request, need IP address of `www.google.com`: **DNS**
- DNS query created, encapsulated in UDP, encapsulated in IP, encapsulated in Eth. To send frame to router, need MAC address of router interface: **ARP**
- **ARP query** broadcast, received by router, which replies with **ARP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

A Day in the Life... Using DNS



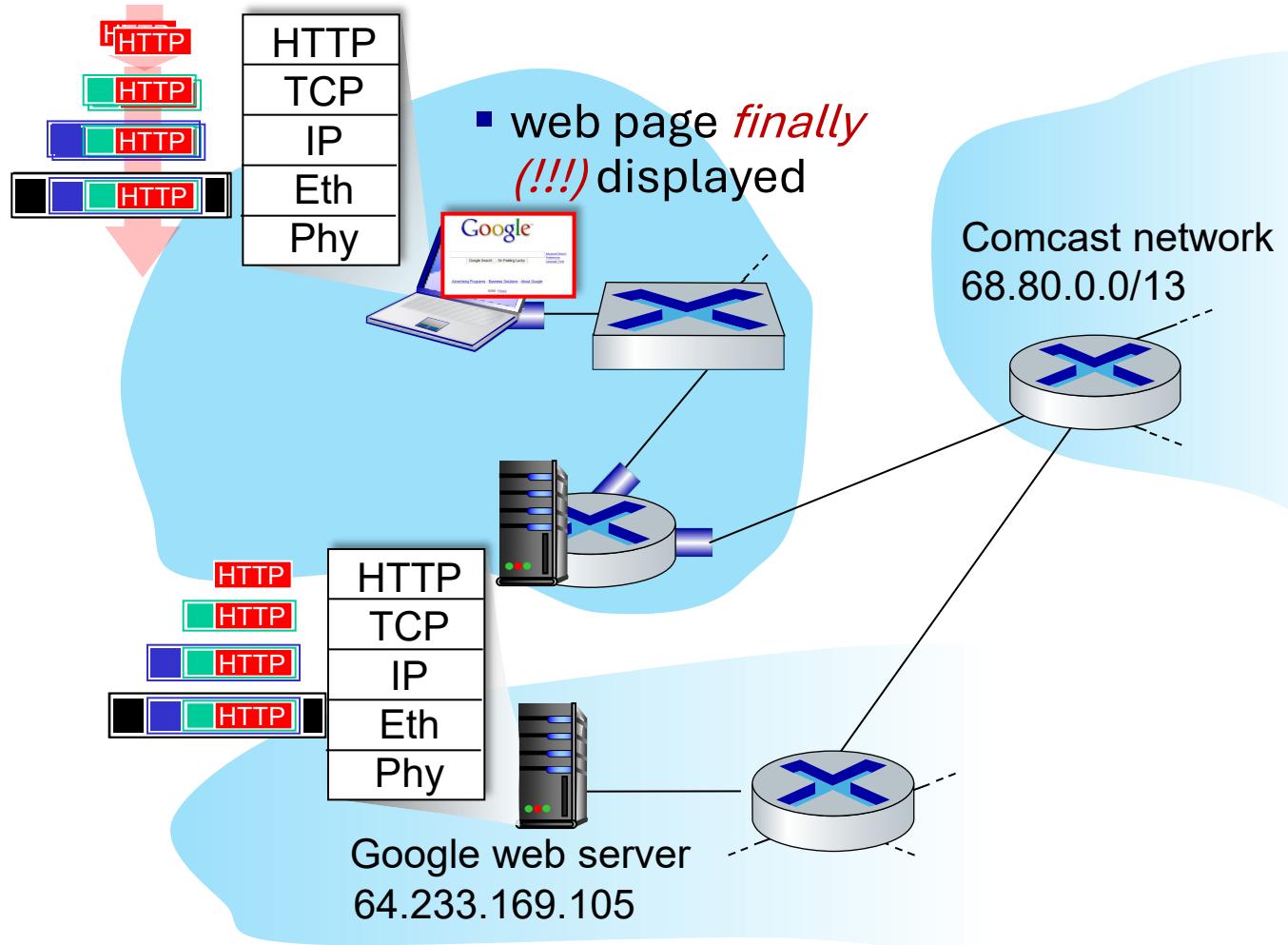
- IP datagram containing DNS query forwarded via LAN switch from client to 1st hop router
- IP datagram forwarded from campus network into Comcast network, routed (tables created by **RIP, OSPF, IS-IS** and/or **BGP** routing protocols) to DNS server
- de-muxed to DNS
- DNS replies to client with IP address of www.google.com

A Day in the Life...TCP Connection Carrying HTTP



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

A Day in the Life... HTTP Request/Reply



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

CSC 3511 Security and Networking

Week 9, Lecture 1: DNS Vulnerabilities and Attacks

Roadmap

- **DNS: Review**
- *DNS Security and DNS Cache Poisoning Attacks*
- *DNS Kaminsky Attack*

Week 3 Lecture 1: DNS

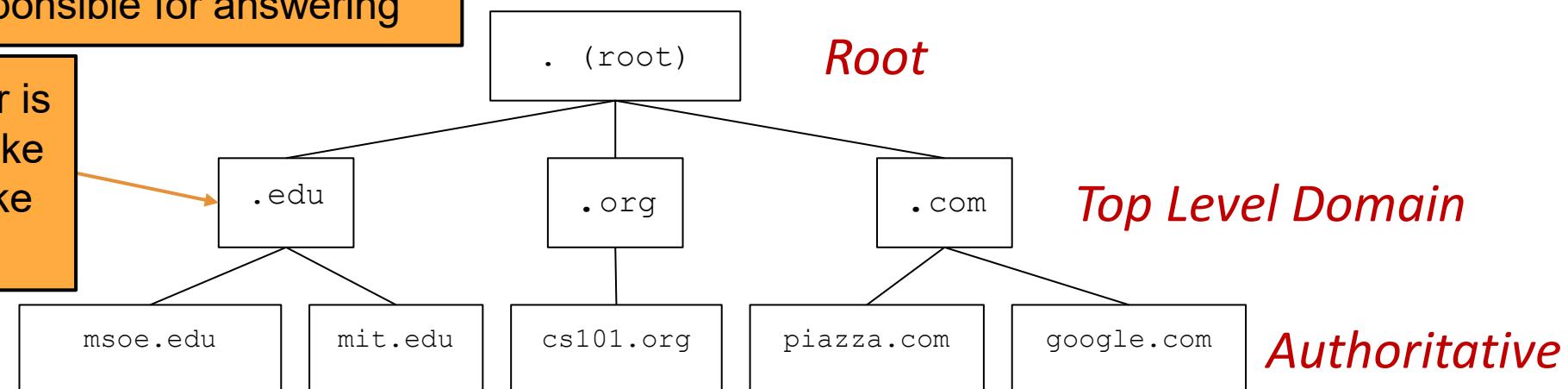
DNS (Domain Name System):



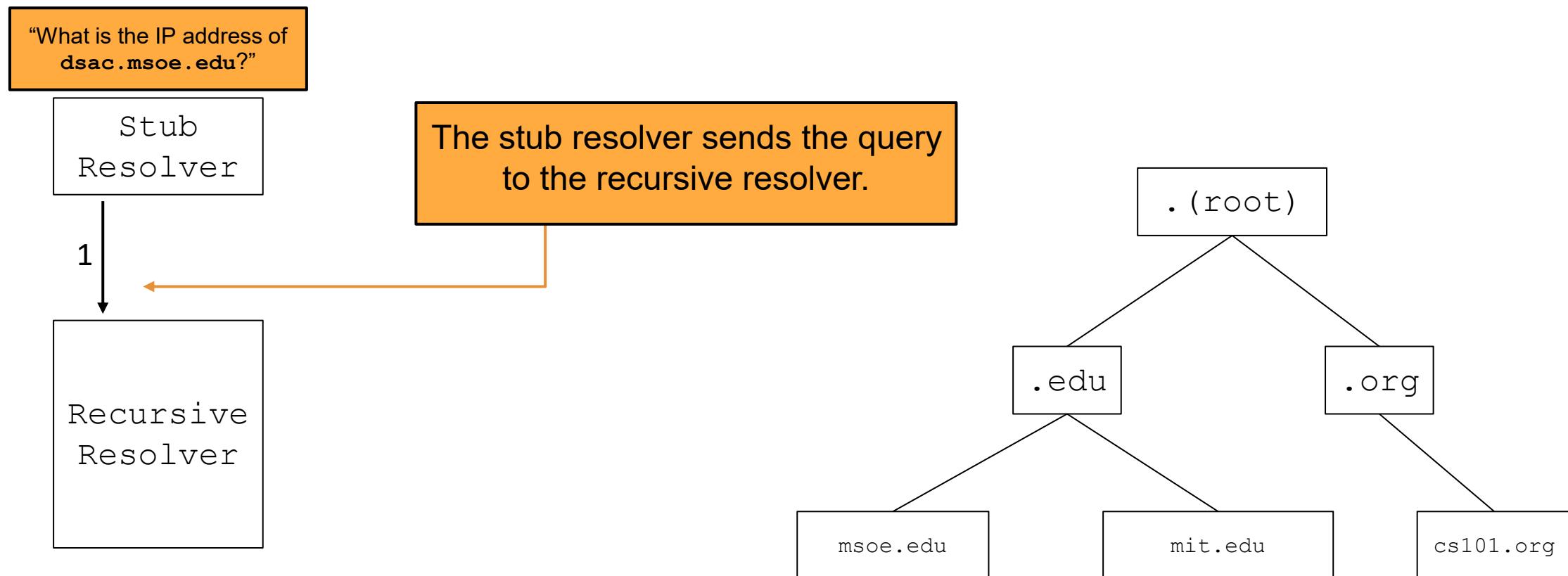
- An application layer protocol that maps between hostname and IP address, and vice versa (e.g., `dig -x 8.8.8.8`)
- Name servers are arranged in a domain hierarchy tree
- Lookups proceed down the domain tree: name servers will direct you down the tree until you receive an answer
- The stub resolver (a software/library on the host machine) tells the recursive resolver (DNS server) to perform the lookup

Each box is a name server. The label represents which queries the name server is responsible for answering

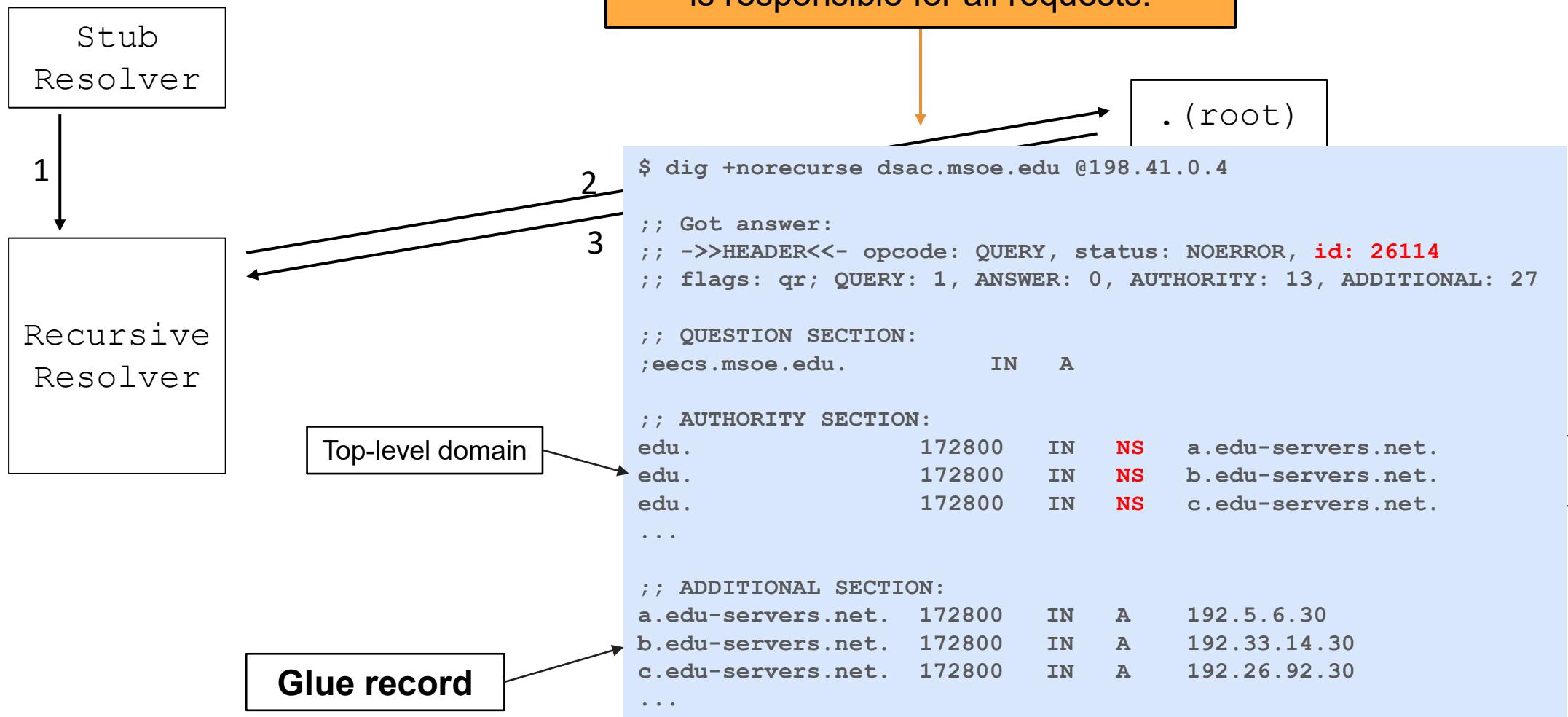
For example, this name server is responsible for **.edu** queries like **msoe.edu**, but not a query like **google.com**



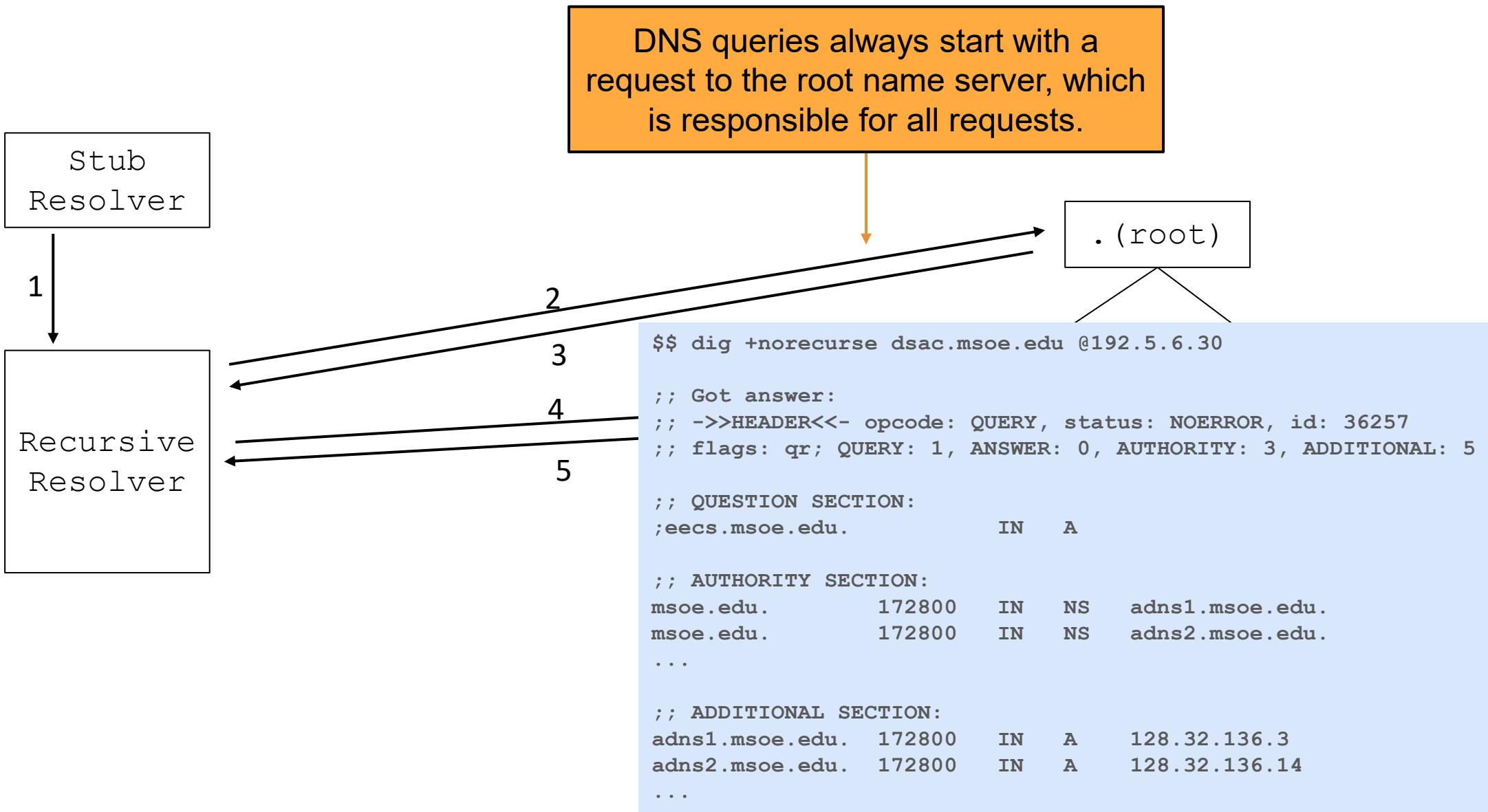
Steps of a DNS Lookup



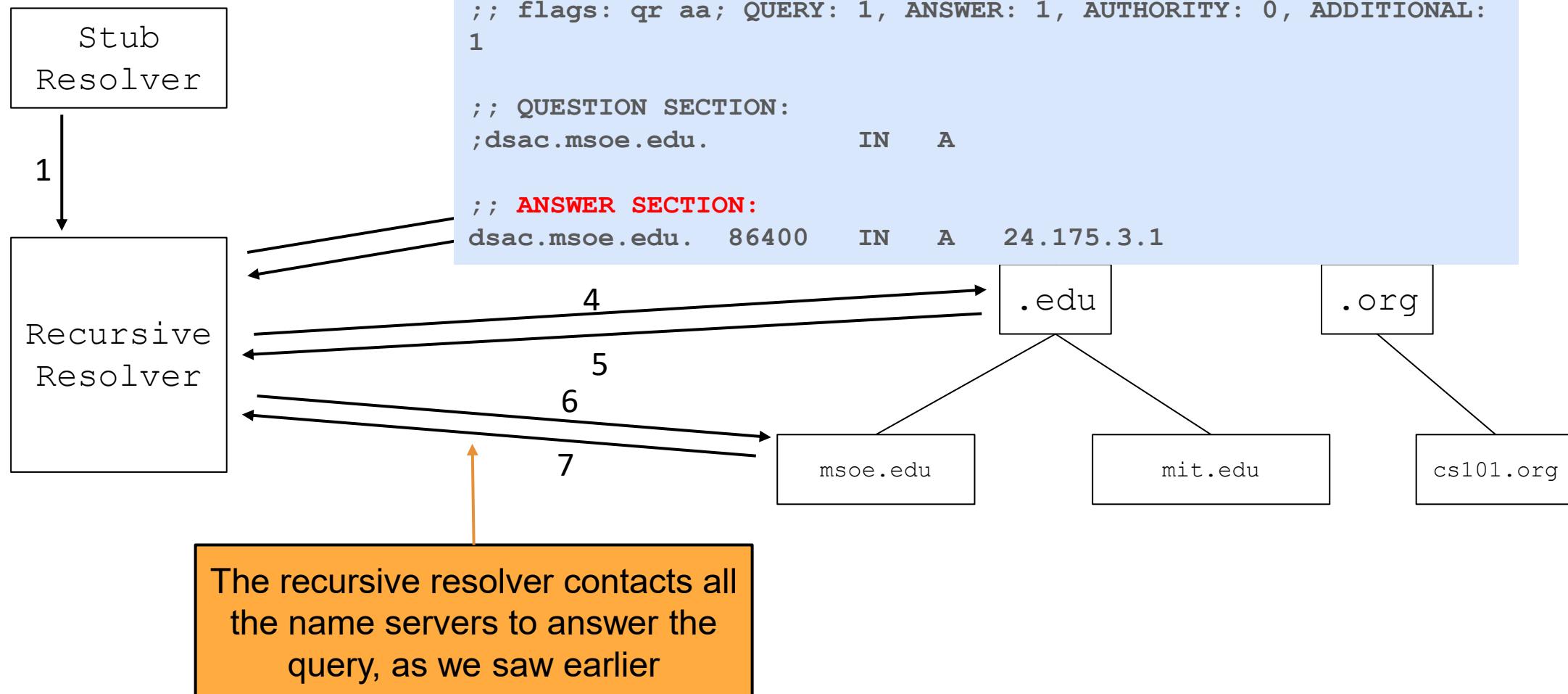
Steps of a DNS Lookup



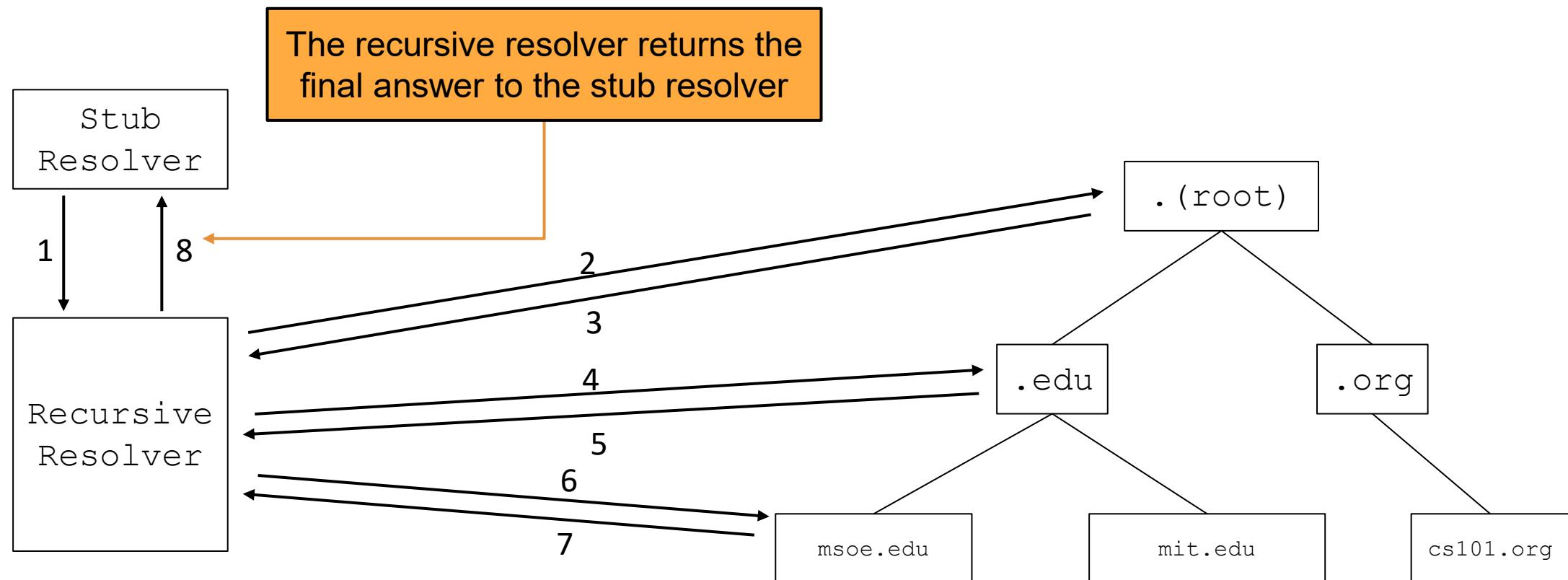
Steps of a DNS Lookup



Steps of a DNS Lookup

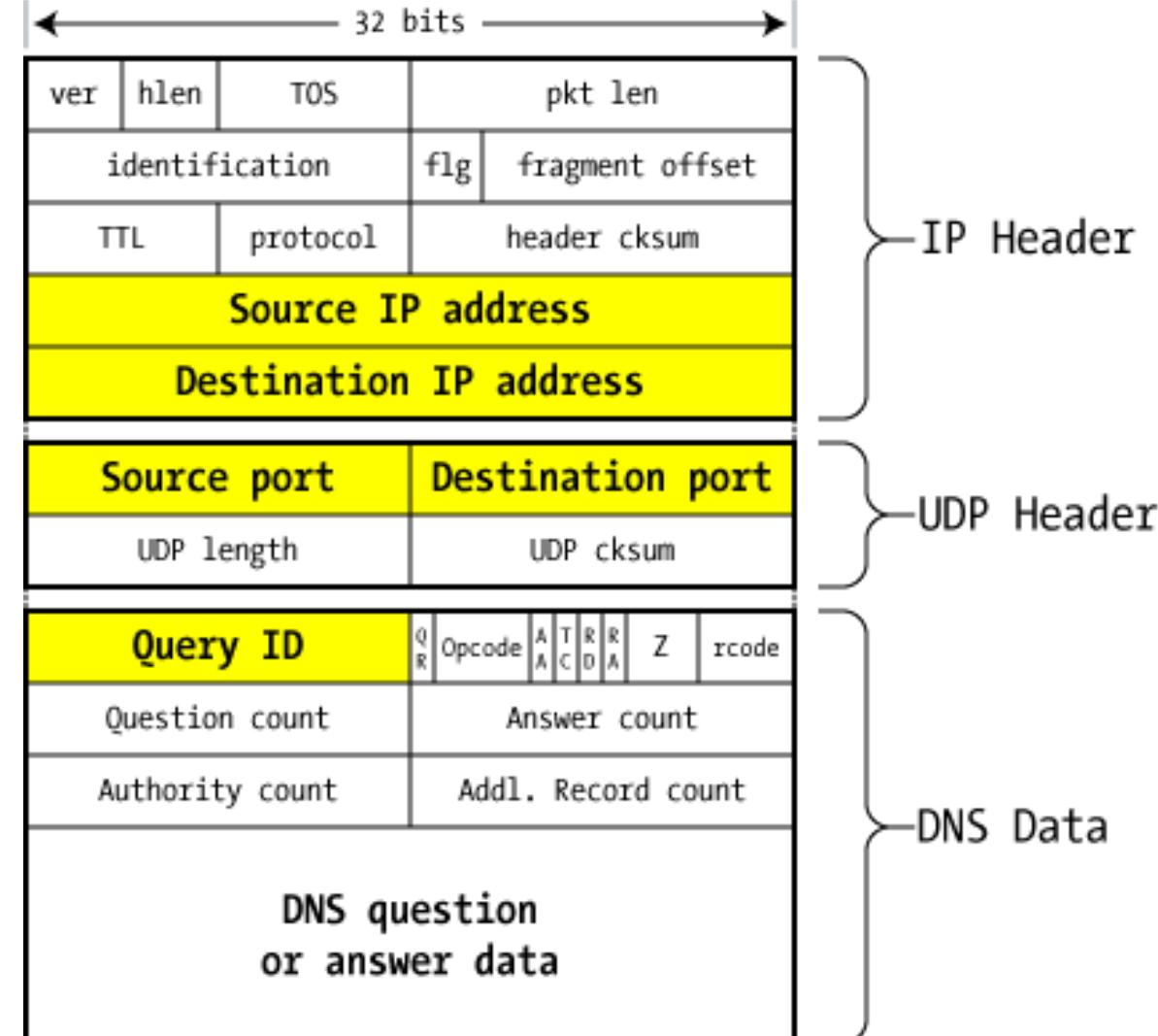


Steps of a DNS Lookup



What's in a DNS packet?

- Source / Destination IP address
- Source / Destination port numbers:
 - DNS servers listen on port **53/UDP**
 - The source port varies considerably
 - Sometimes it's also port **53/UDP**
 - Sometimes it's chosen at **random** by the operating system
- Query ID:
 - A unique identifier
 - This is also sometimes called the **Transaction ID** (TXID)
- Counts: The number of records of each type in the DNS payload

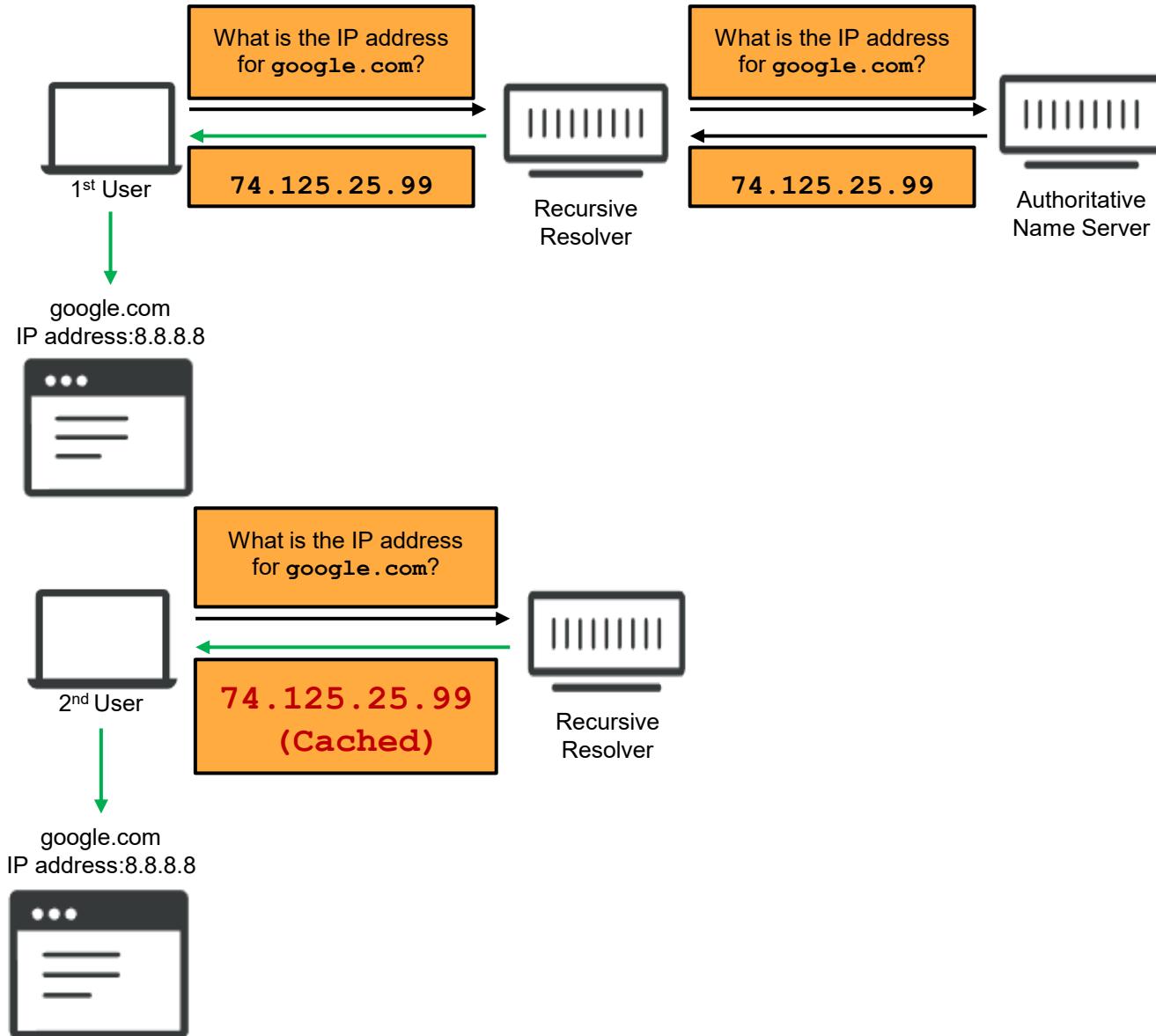


DNS packet on the wire

<http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html>

DNS Caching

- DNS *responses* are cached
 - Quick response for repeated translations
- DNS *negative queries* are cached
 - Save time for nonexistent sites, e.g., misspelling
- Cached data periodically times out
 - Lifetime (TTL) of data controlled by owner of data
 - TTL passed with every record



Roadmap

- *DNS: Review*
- ***DNS Security and DNS Cache Poisoning Attacks***
- *DNS Kaminsky Attack*

Why Should We Care About DNS Security?

- Suppose you take over a lot of home routers... How do you make money from your attack?
- Change the DNS server settings **DNS Hijacking Attacks Statistics by Sea Turtle**
 - Each router is programmed to...
 - Replace the IP address...
 - **Cache poisoning** and...
- Redirect DNS requests from the victim
 - DNS: what's the IP address?
 - The attacker can now...
- Are attacks on DNS systems...



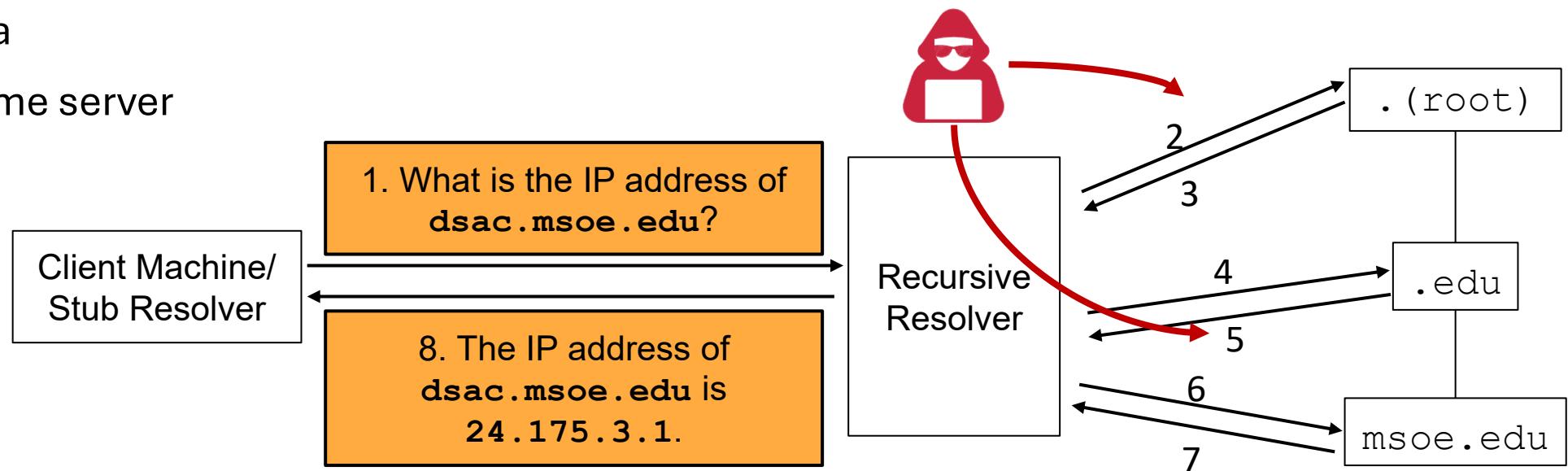
<https://blog.talosintelligence.com/seaturtle/>

DNS Security: Know Your Threat Model

Threat Model:

- **Who your attacker is?**
 - Malicious name server
 - External attacker
- **What capabilities do they have? ➔**
- **What attacks can they launch?**
 - Forge DNS data
 - Hijack DNS name server
 - DNS Flood
 - ...

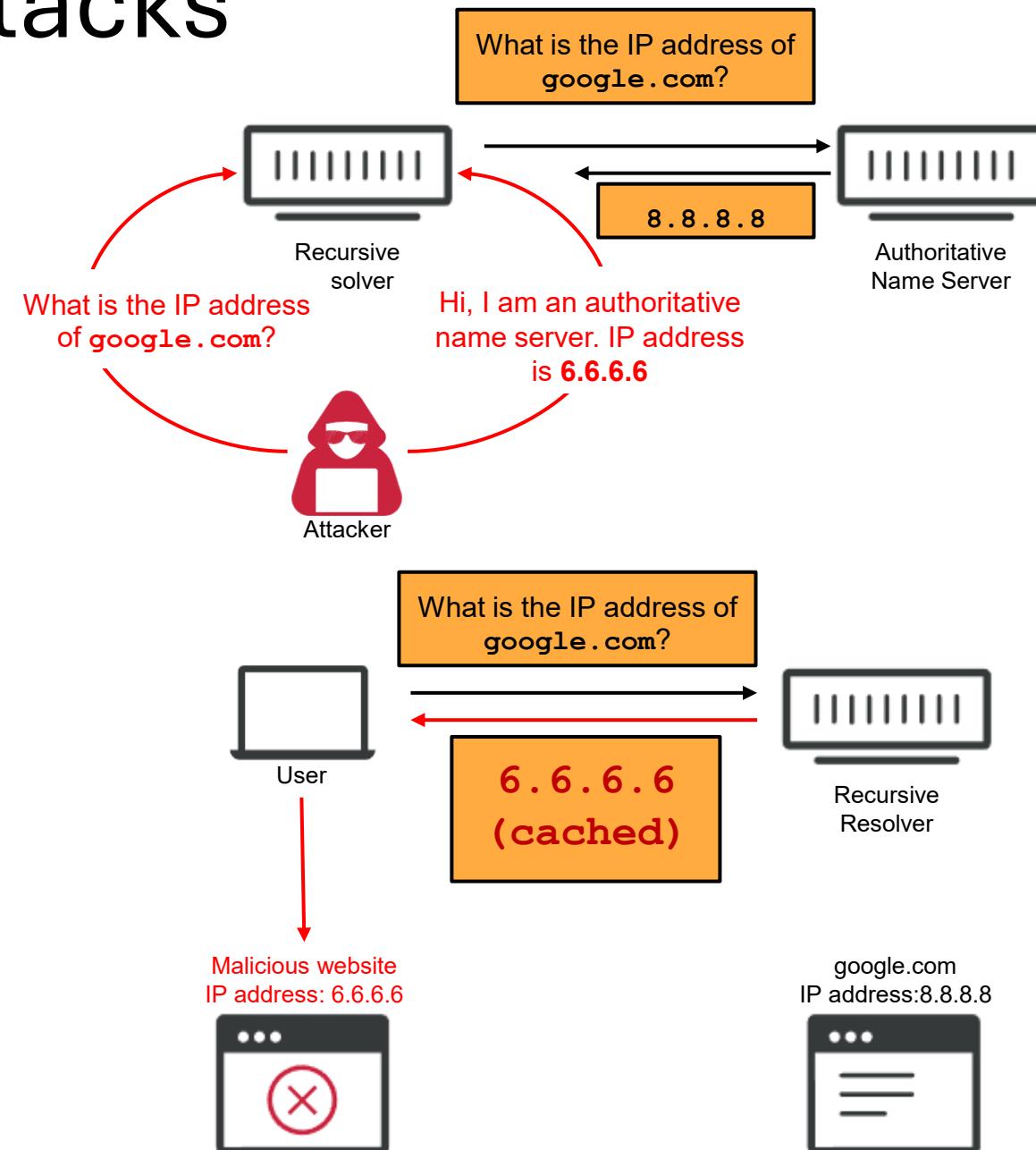
	Can modify or delete packets	Can read packets
In-path (MITM) attacker	✓	✓
On-path attacker		✓
Off-path attacker		



DNS Cache Poisoning Attacks

Cache poisoning attack:

- Returning a false information into the DNS cache
- The victim will cache the malicious records, “poisoning” it
- Later, DNS queries return an incorrect response, and users are directed to the wrong websites
- Example: Supply a malicious record mapping the attacker’s IP address to a legitimate domain
 - Now when the victim visits google.com, they’ll actually be sending packets to the attacker (**6.6.6.6**), who can launch the **MITM attack!**



Security Risk: Malicious Name Servers

- Malicious name servers can lie and supply a malicious answer
- Malicious records could also poison the cache with **other records**
- DNS query results include Additional Records section:
 - Provide records for anticipated next resolution step
- Early servers *accepted and cached all additional records* provided in query response

```
$ dig +norecurse dsac.msoe.edu @128.42.156.7  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52788  
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1  
  
;; QUESTION SECTION:  
;dsac.msoe.edu.          IN      A  
  
;; ANSWER SECTION:  
dsac.msoe.edu.    86400    IN      A      24.175.3.1  
  
;; ADDITIONAL SECTION:  
www.google.com.   172800   IN      A      6.6.6.6
```

We made a query to a malicious **msoe.edu** name server...

...and it returned a **malicious record for www.google.com!**

Defense: Bailiwick Checking

- Idea: Limit the amount of damage a malicious name server can do
- **Bailiwick checking**: the resolver only accepts records if they are in the name server's zone
 - Bailiwick: “one’s sphere of operations or particular area of interest”
 - Example: The **.edu** name server can provide a record for **msoe.edu** and **mit.edu**, but not **google.com**
 - Example: The **msoe.edu** name server can provide a record for **dsac.msoe.edu**, but not **cs.uchicago.edu**
 - Exception: The root name server can provide a record for any domain (everything is in bailiwick for the root)

```
$ dig +norecurse dsac.msoe.edu @128.42.156.7  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52788  
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1  
  
;; QUESTION SECTION:  
;dsac.msoe.edu.           IN      A  
  
;; ANSWER SECTION:  
dsac.msoe.edu.    86400    IN      A      24.175.3.1  
  
;; ADDITIONAL SECTION:  
www.google.com.    172800   IN      A      6.6.6.6
```

We made a query to a malicious **msoe.edu** name server...

...and it returned a malicious record for **www.google.com!**

Security Risk: In-Path (MITM) Attackers

- DNS is not secure against MITM attackers
- MITM attackers can poison the cache by **adding, removing, or changing** any record in the DNS response

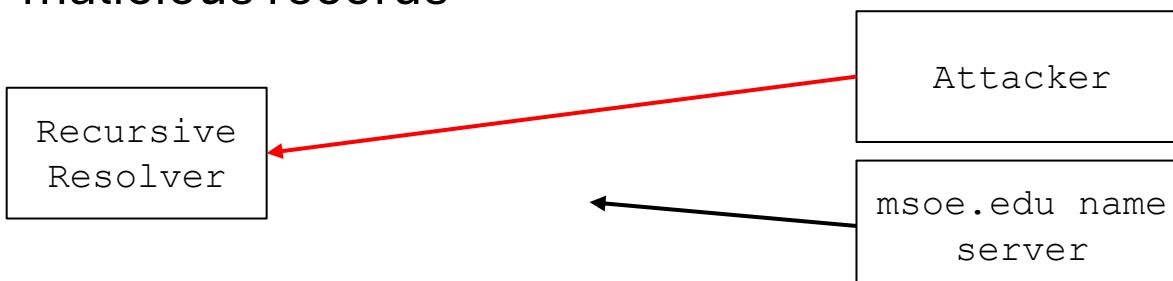
```
$ dig +norecurse dsac.msoe.edu @128.42.156.7  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 52788  
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1  
  
;; QUESTION SECTION:  
;dsac.msoe.edu.           IN      A  
  
;; ANSWER SECTION:  
dsac.msoe.edu.    86400    IN      A      24.175.3.1-7.7.7.7
```

We made a query to msoe.edu name server...

MITM attacker modifies the answer

Security Risk: On-Path Attackers

- DNS is not secure against on-path attackers
- How does client **authenticate** response?
 - **Port numbers** must match
 - **Query ID** number must match
- On-path attackers can poison the cache by *sending a spoofed response*
 - The on-path attacker can see every field in the unencrypted DNS request. Nothing to guess!
 - If the spoofed response arrives **before** the legitimate response, the victim will cache the attacker's malicious records

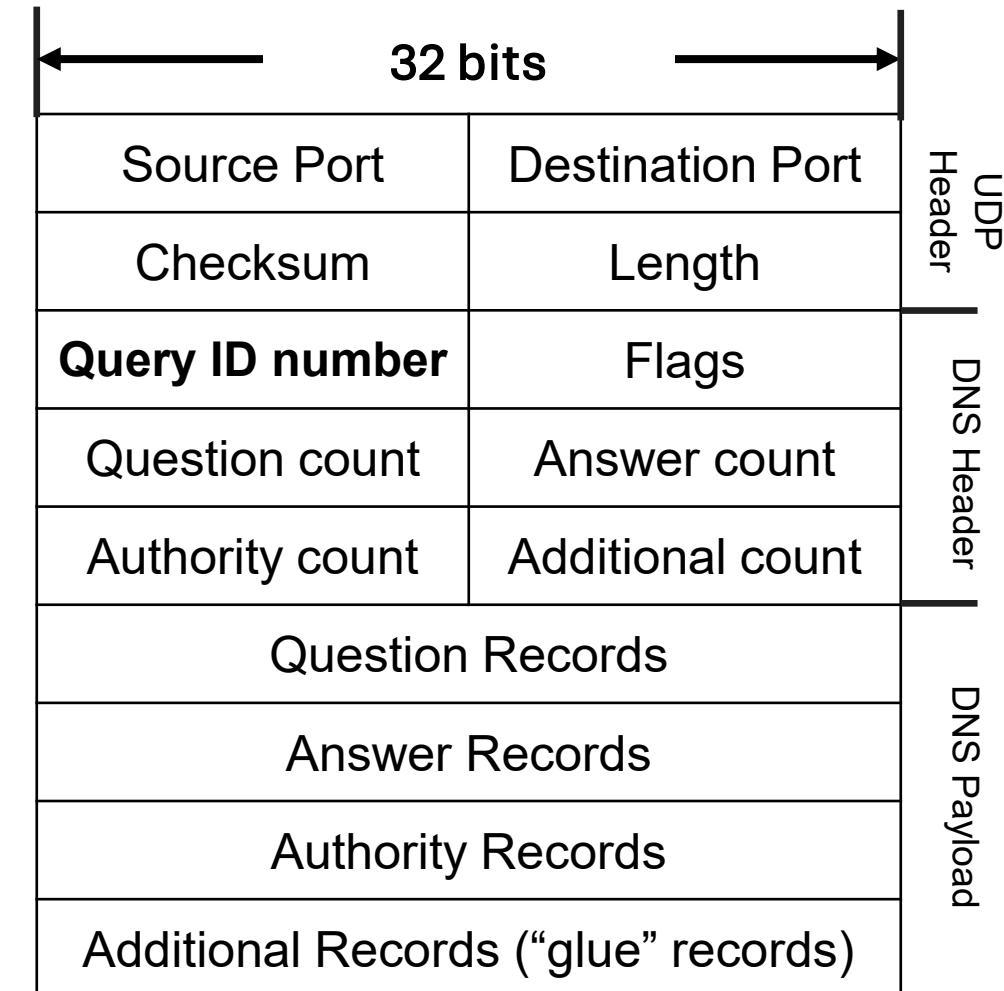


Source Port	Destination Port
Checksum	Length
Query ID number	Flags
Question count	Answer count
Authority count	Additional count
Question Records	
Answer Records	
Authority Records	
Additional Records ("glue" records)	

Header
DNS Header
DNS Payload

Security Risk: Off-Path Attackers

- The off-path attacker needs to guess the *query ID* field to spoof a response
 - If the ID in the response doesn't match the ID in the request, the resolver won't accept the response
- If the ID number is *randomly* generated:
 - Recall: The Query ID number is 16 bits long
 - Probability of guessing correctly = $1/2^{16}$
 - Requires approximately 65,000 tries to successfully send a spoofed packet
 - This is too small!



Security Risk: Off-Path Attackers

- What if the Query ID field is incremented by 1 for every request?
- Off-path attacker can spoof a packet as follows:
 - Trick the victim into visiting the attacker's website
 - Include this HTML on the attacker's website:
 - The victim's browser will make a DNS query for www.attacker.com
 - If the attacker controls the **attacker.com** DNS name server, they can see the request and learn the ID field
 - Include this HTML on the attacker's website:
 - The victim's browser will make a DNS query for www.google.com
 - The attacker knows the ID is 1 more than the previous ID, so they can spoof a response!
- Query ID numbers need to be *random* in DNS requests

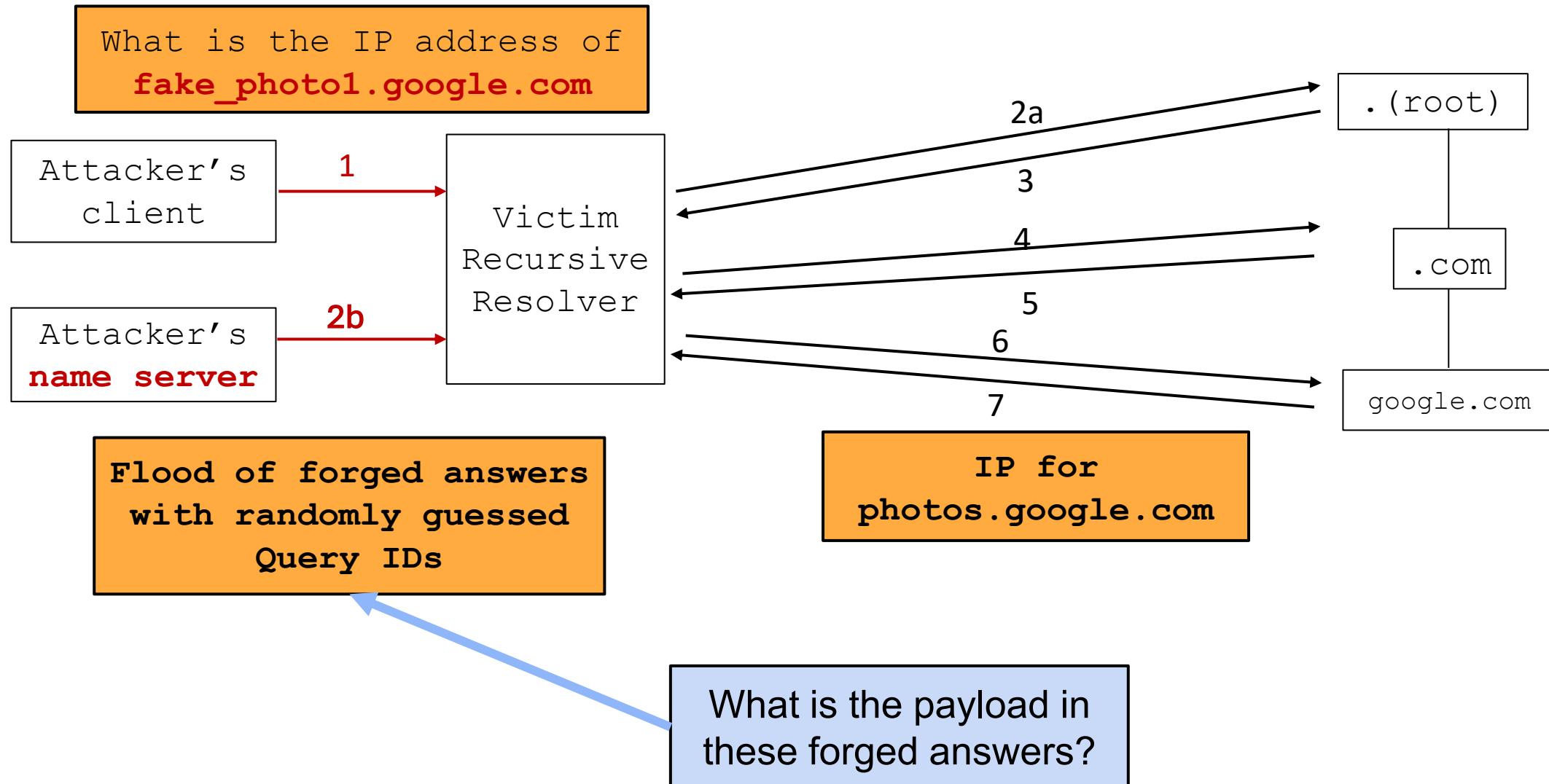
Roadmap

- *DNS: Review*
- *DNS Security and DNS Cache Poisoning Attacks*
- ***DNS Kaminsky Attack***

Kaminsky Attack

- Dan Kaminsky, security researcher, noticed that DNS clients would cache additional glue records as if they were authoritative answers, **even though they aren't**
- Previous attacks: ***poison the final answer, the type A record with the IP address***
- Kaminsky Attack: goes up one level and ***hijacks the authority records (NS record) instead***
- The attacker (Mallory) requests a random name (usually nonexistent) within the target domain, something unlikely to be in cache:
 - ``
 - ``
 - ``
 - **... and so on**
- Attacker's browser continues to make DNS requests for each tag

Kaminsky Attack



Kaminsky Attack: Example

Stub Resolver

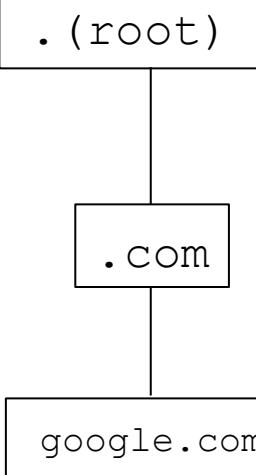
Let's walk through the DNS query for the IP address of `fake_photo1.google.com`

Recursive Resolver

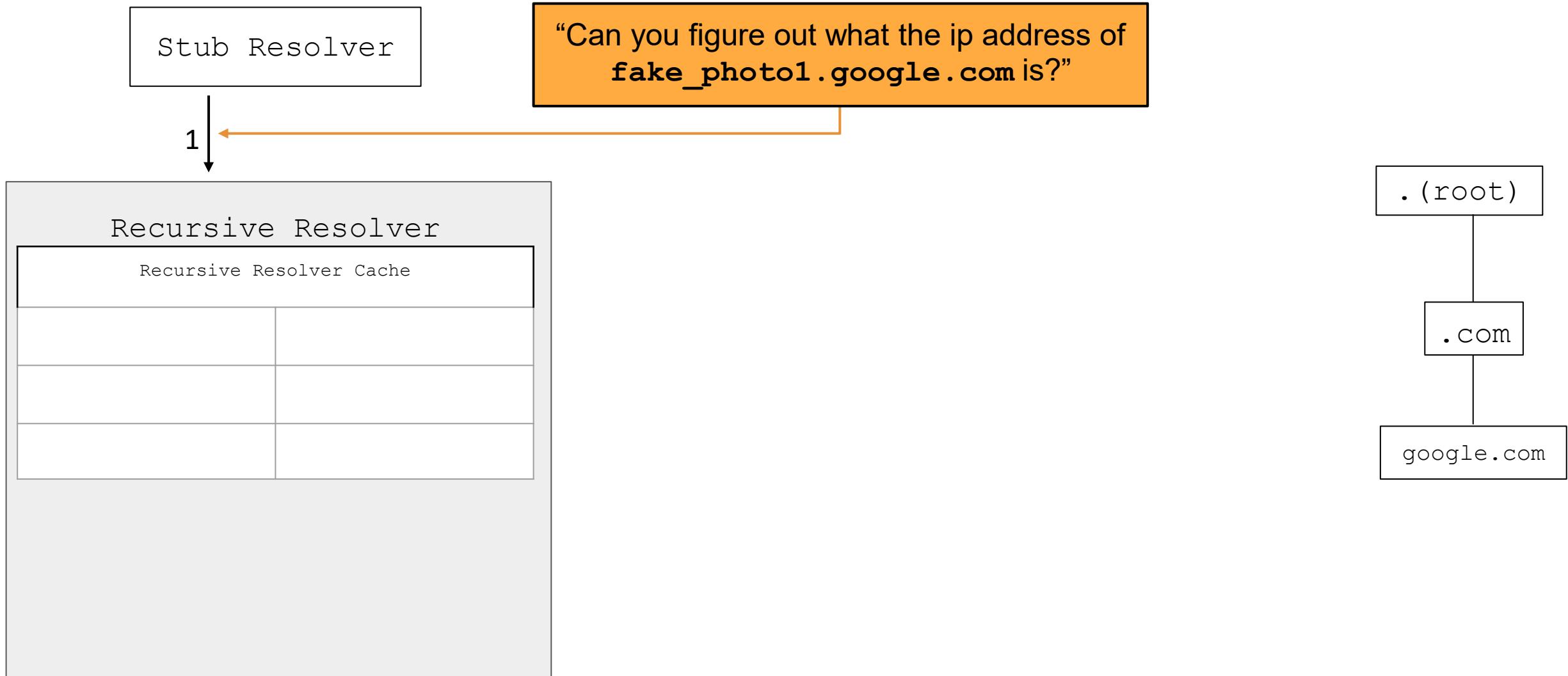
Recursive Resolver Cache

We'll use orange color to represent normal packets across Alice's DNS query.

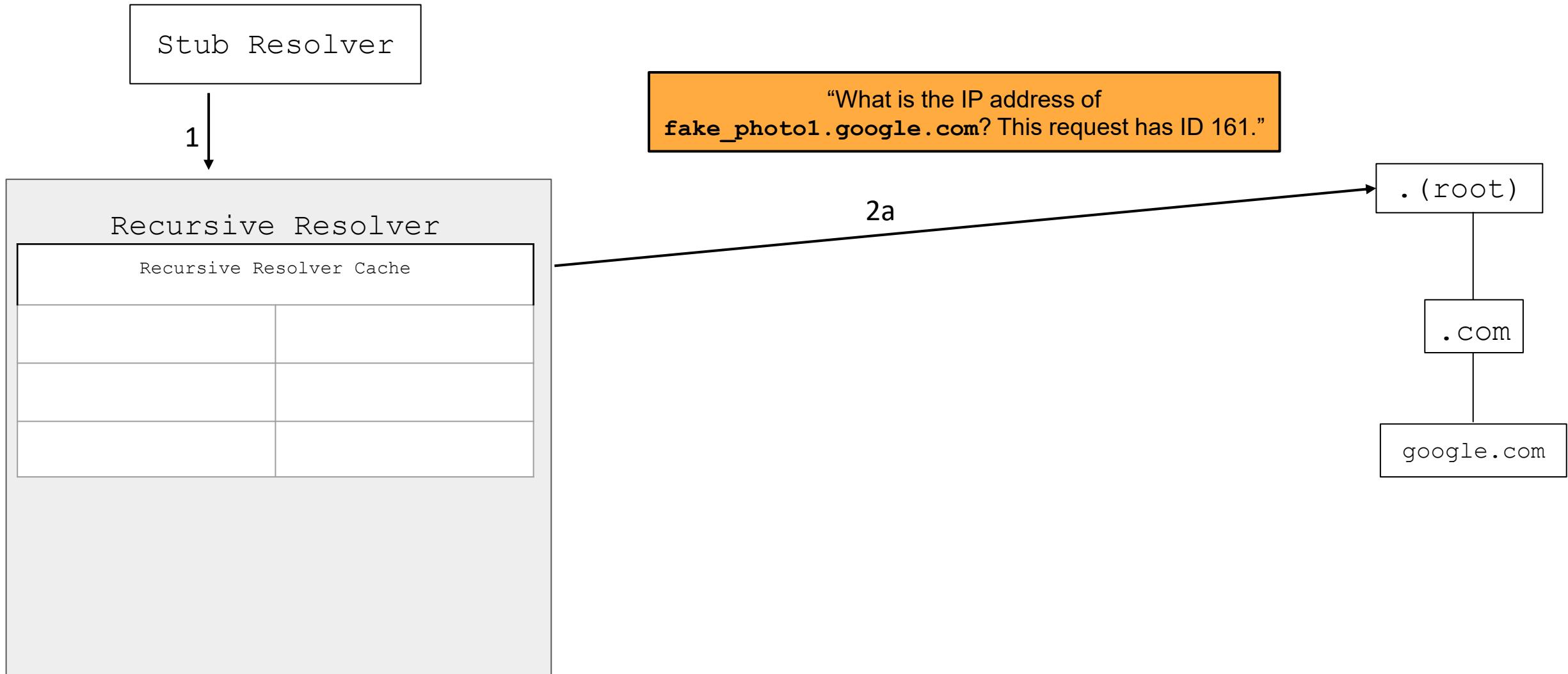
We'll use this color to represent attacker's packets across Alice's DNS query.



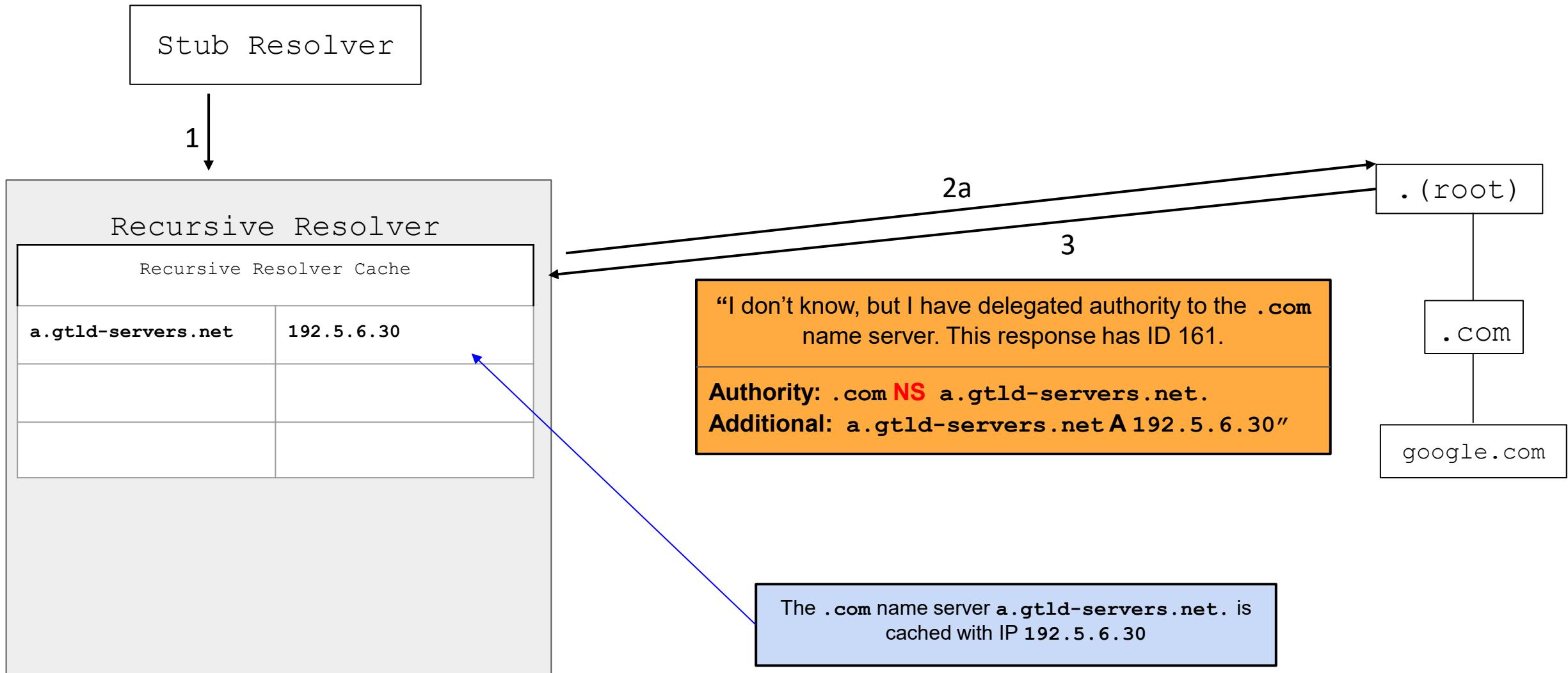
Kaminsky Attack: Example



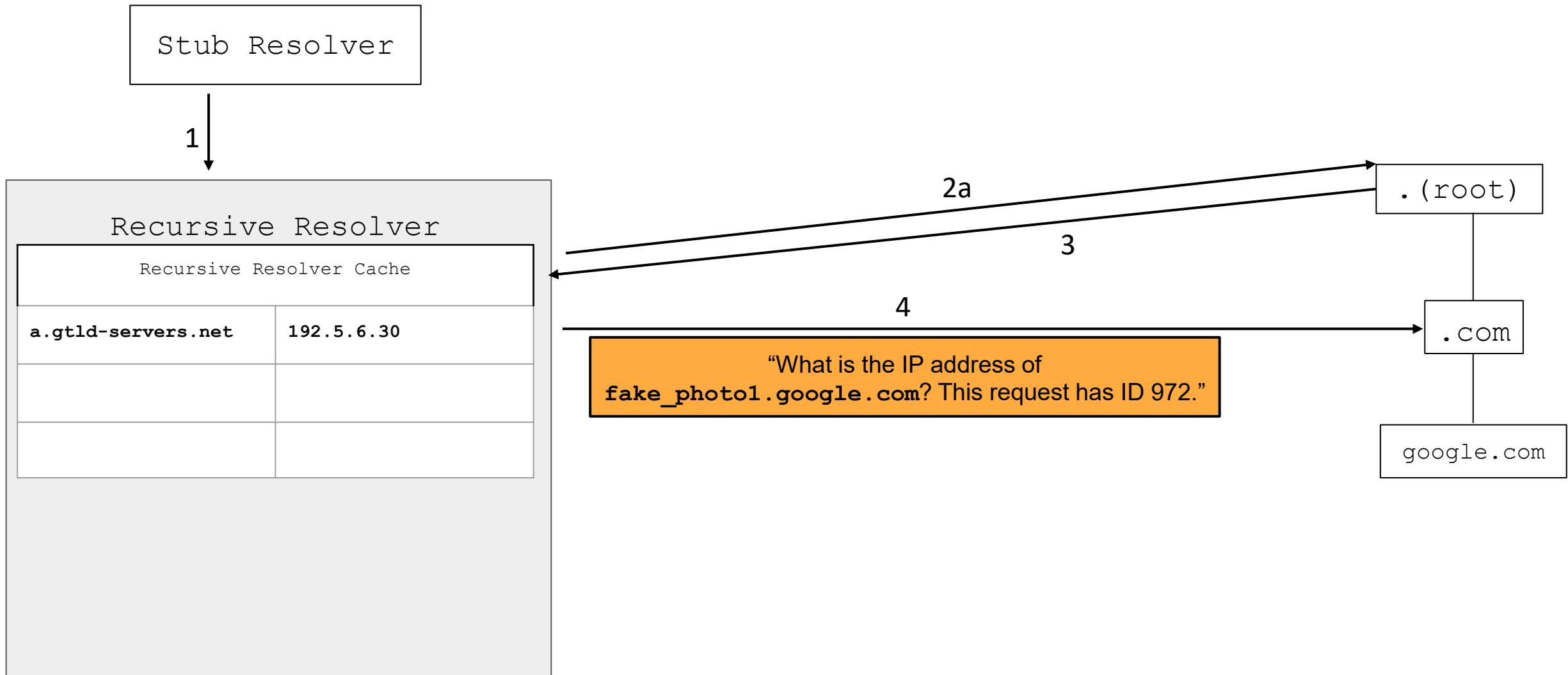
Kaminsky Attack: Example



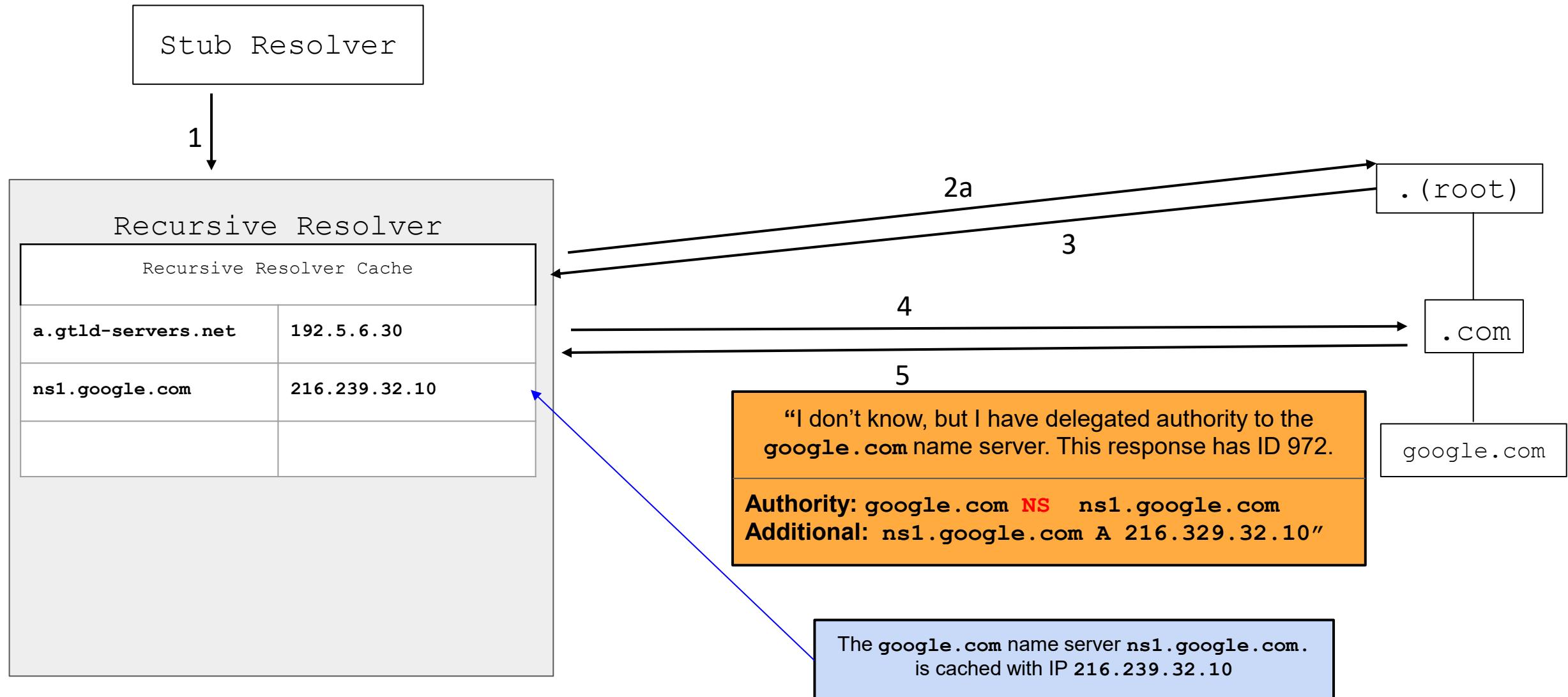
Kaminsky Attack: Example



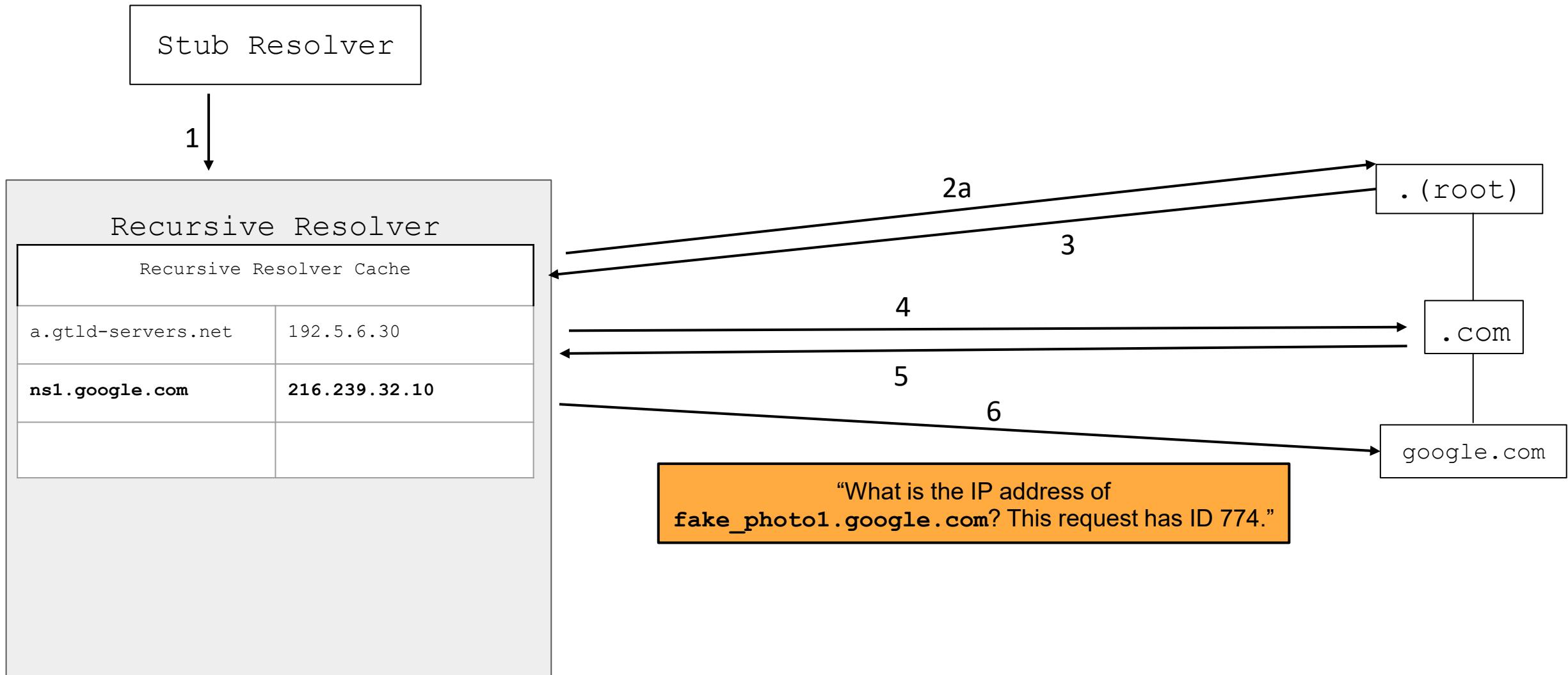
Kaminsky Attack: Example



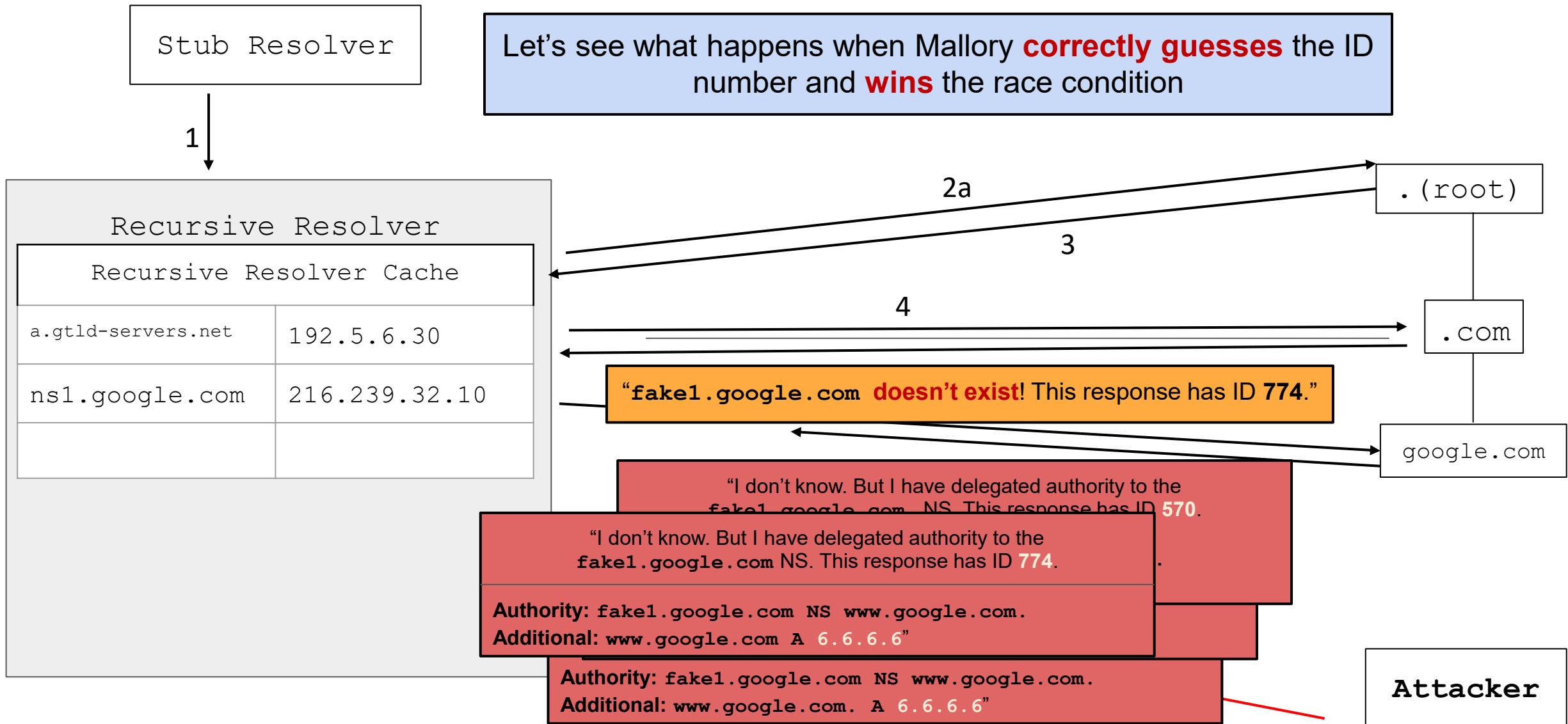
Kaminsky Attack: Example



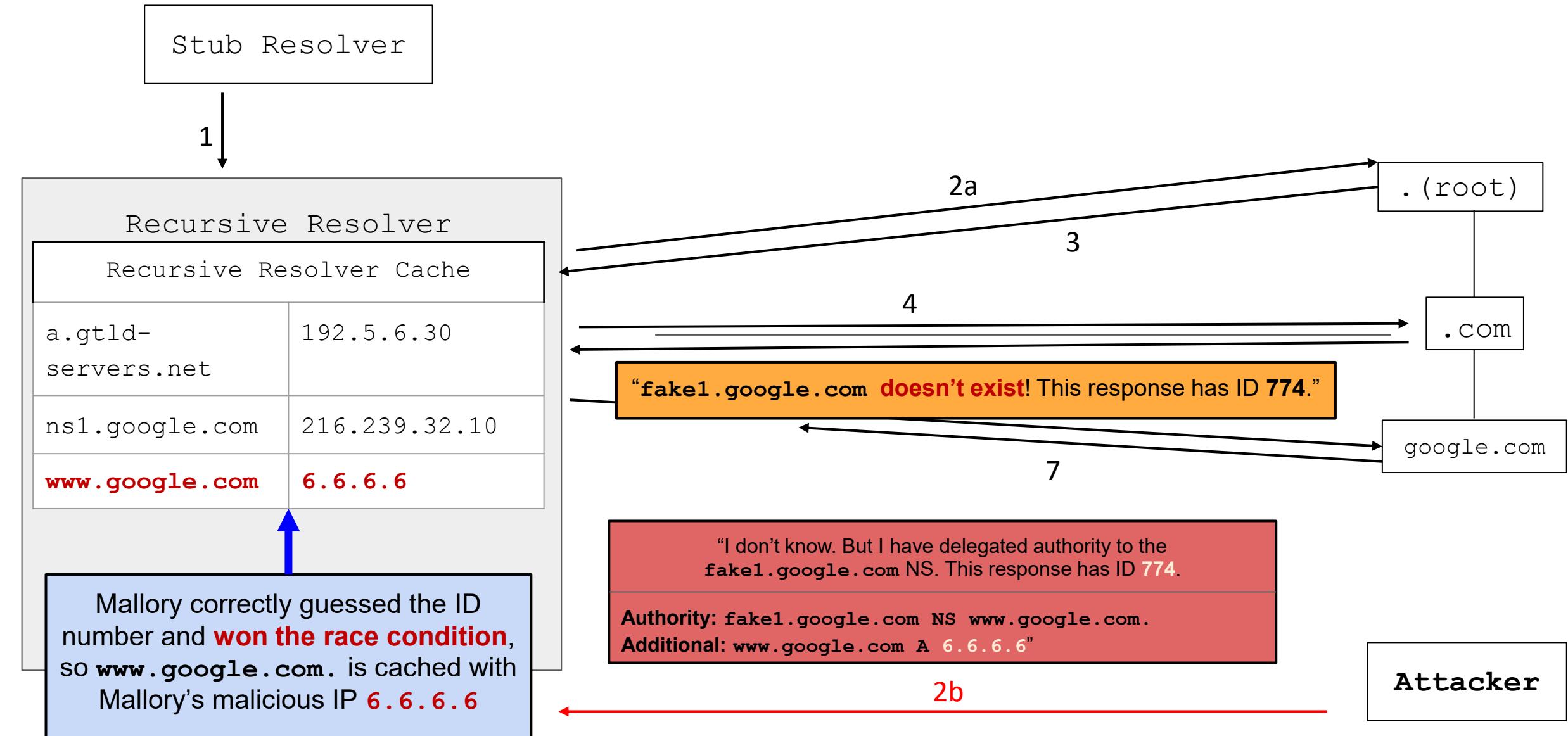
Kaminsky Attack: Example



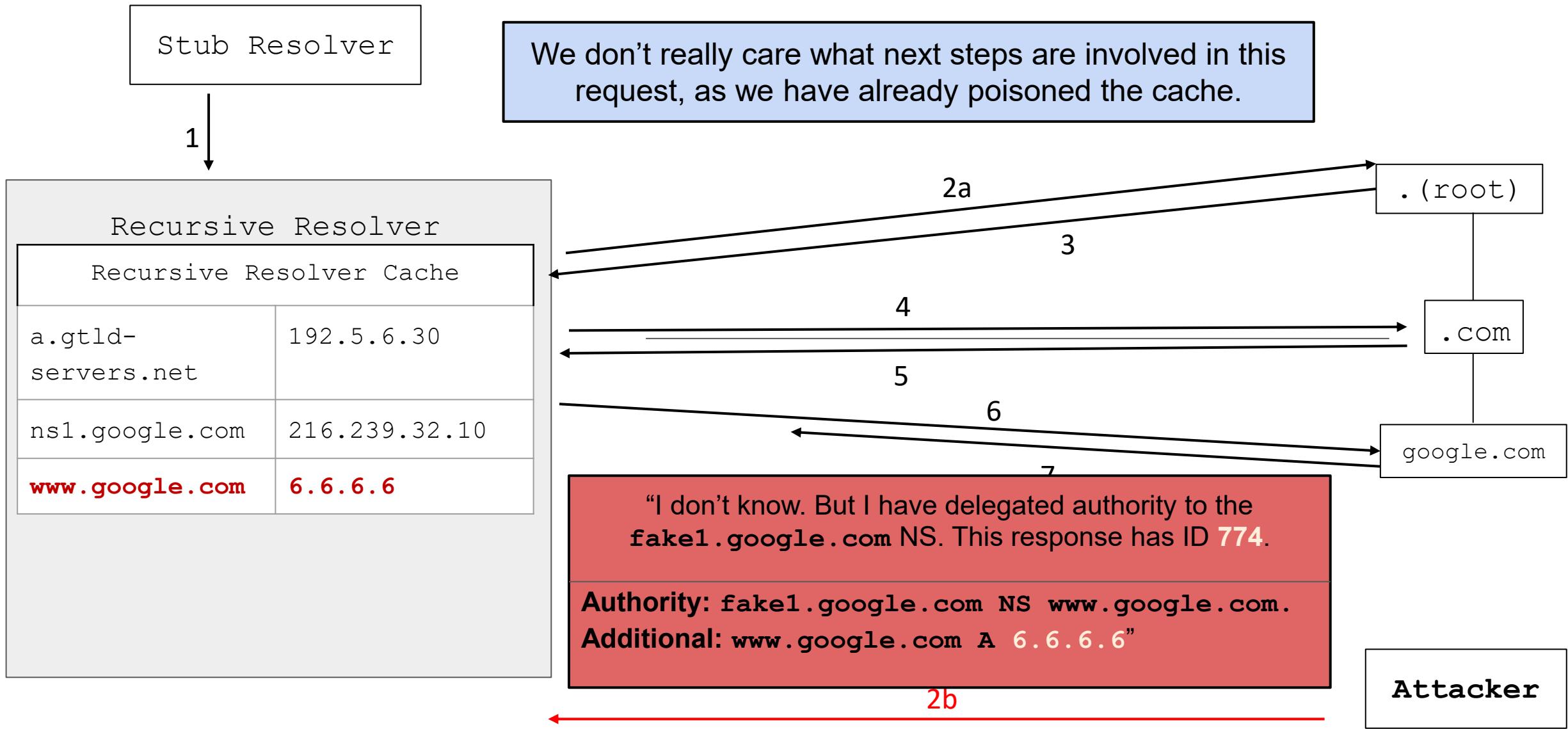
Kaminsky Attack: Example



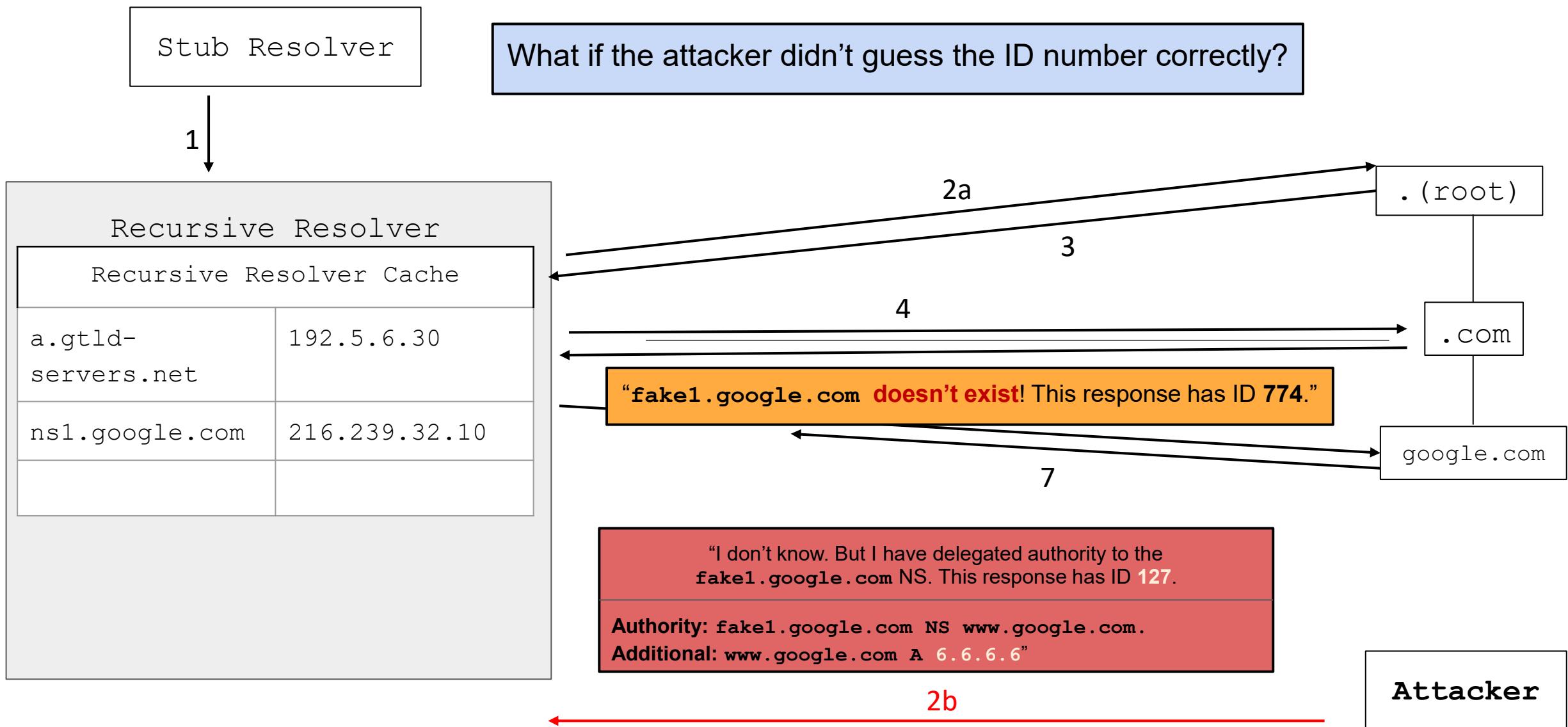
Kaminsky Attack: Example



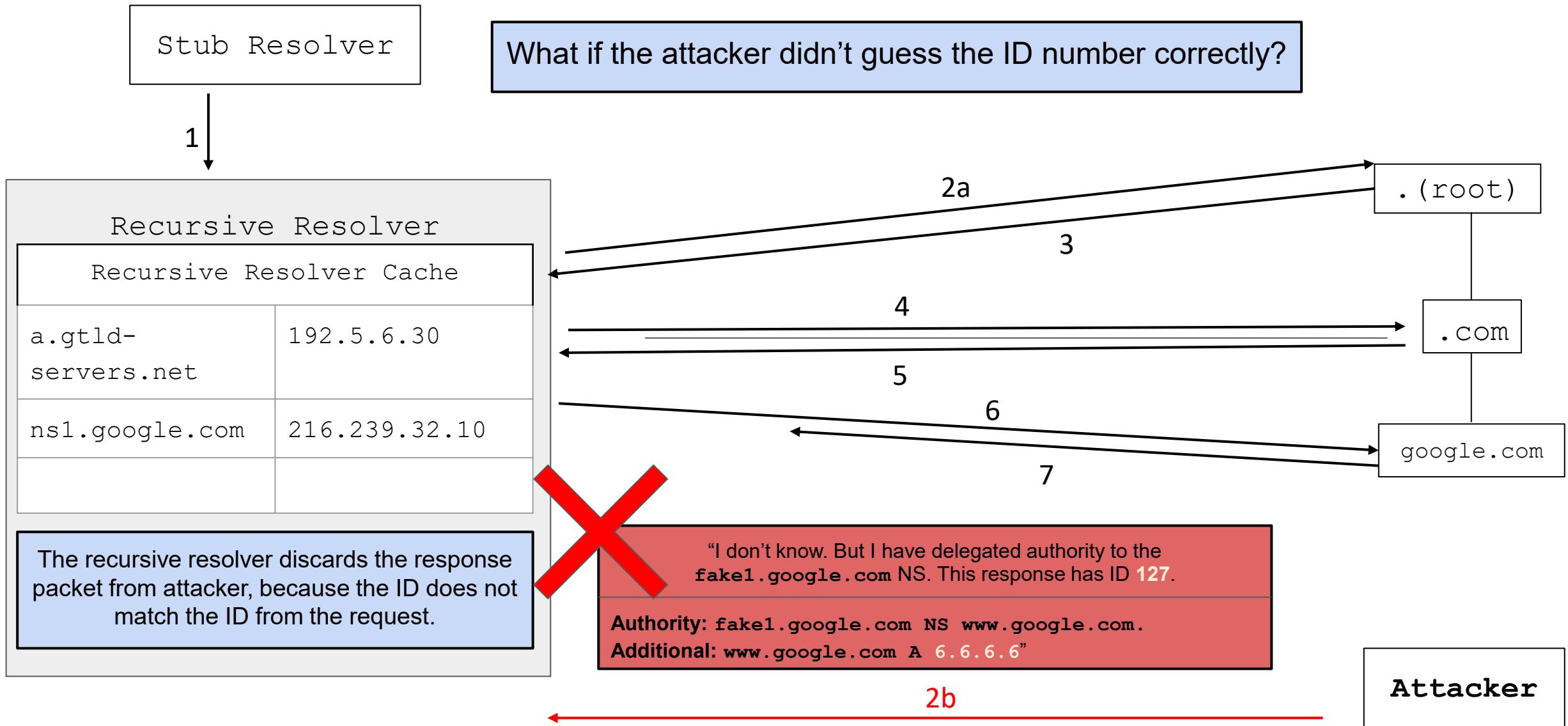
Kaminsky Attack: Example



Kaminsky Attack: Example

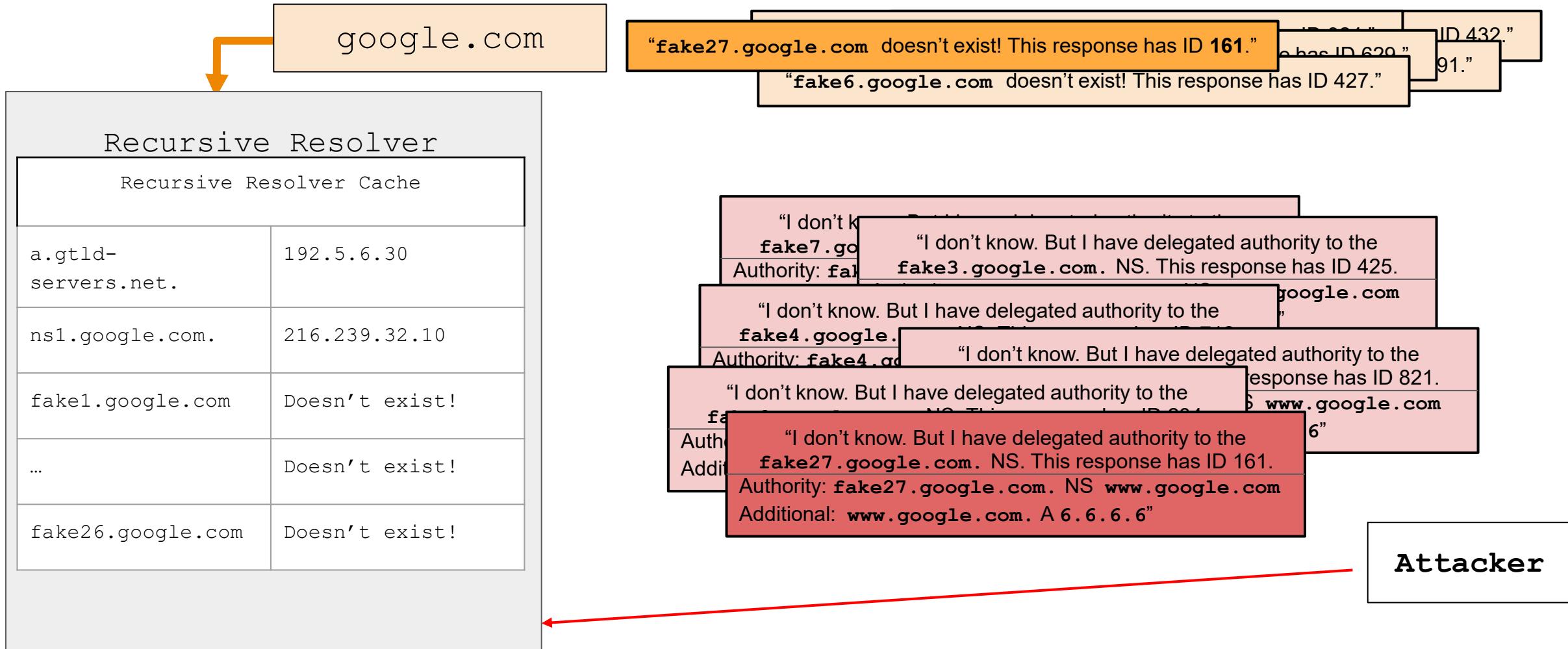


Kaminsky Attack: Example



Kaminsky Attack: Example

Attacker can use the DNS requests from his/her other image tags to try to guess the ID number



Kaminsky Attack: Example

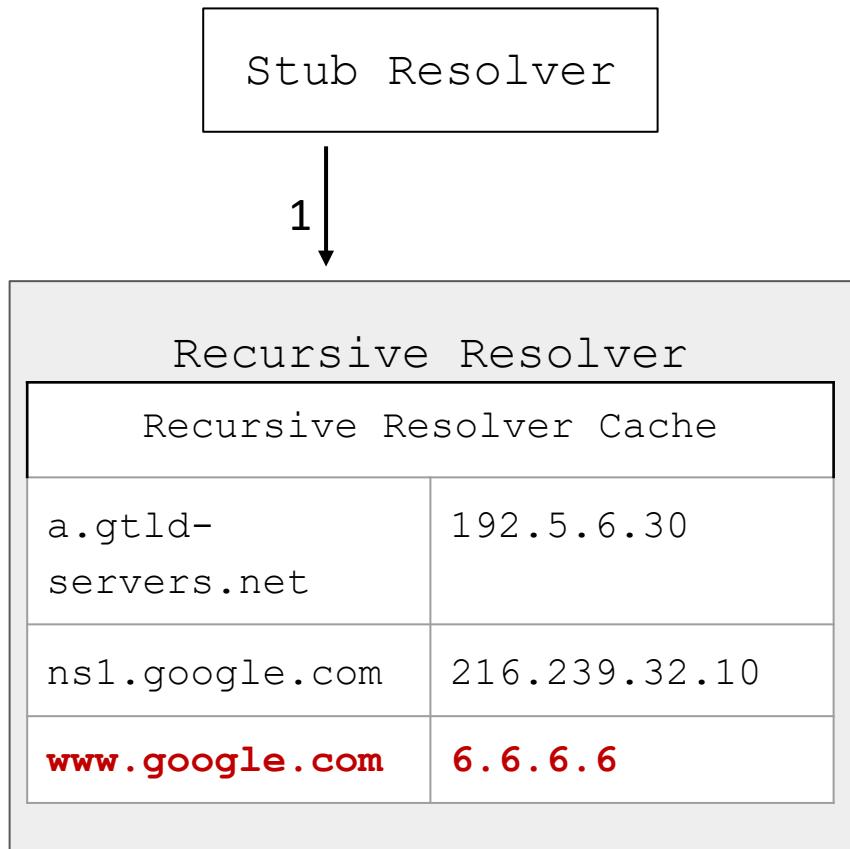
If attacker wins the race condition **and** guesses the correct ID during any one of these DNS queries, attacker's IP will be cached

Recursive Resolver	
Recursive Resolver Cache	
a.gtld-servers.net.	192.5.6.30
ns1.google.com.	216.239.32.10
fake1.google.com	Doesn't exist!
...	Doesn't exist!
fake26.google.com	Doesn't exist!
www.google.com	6.6.6.6

"I don't know. But I have delegated authority to the **fake27.google.com**. NS. This response has ID 161.
Authority: **fake27.google.com**. NS **www.google.com**
Additional: **www.google.com**. A **6.6.6.6**"

Attacker

Kaminsky Attack: Example



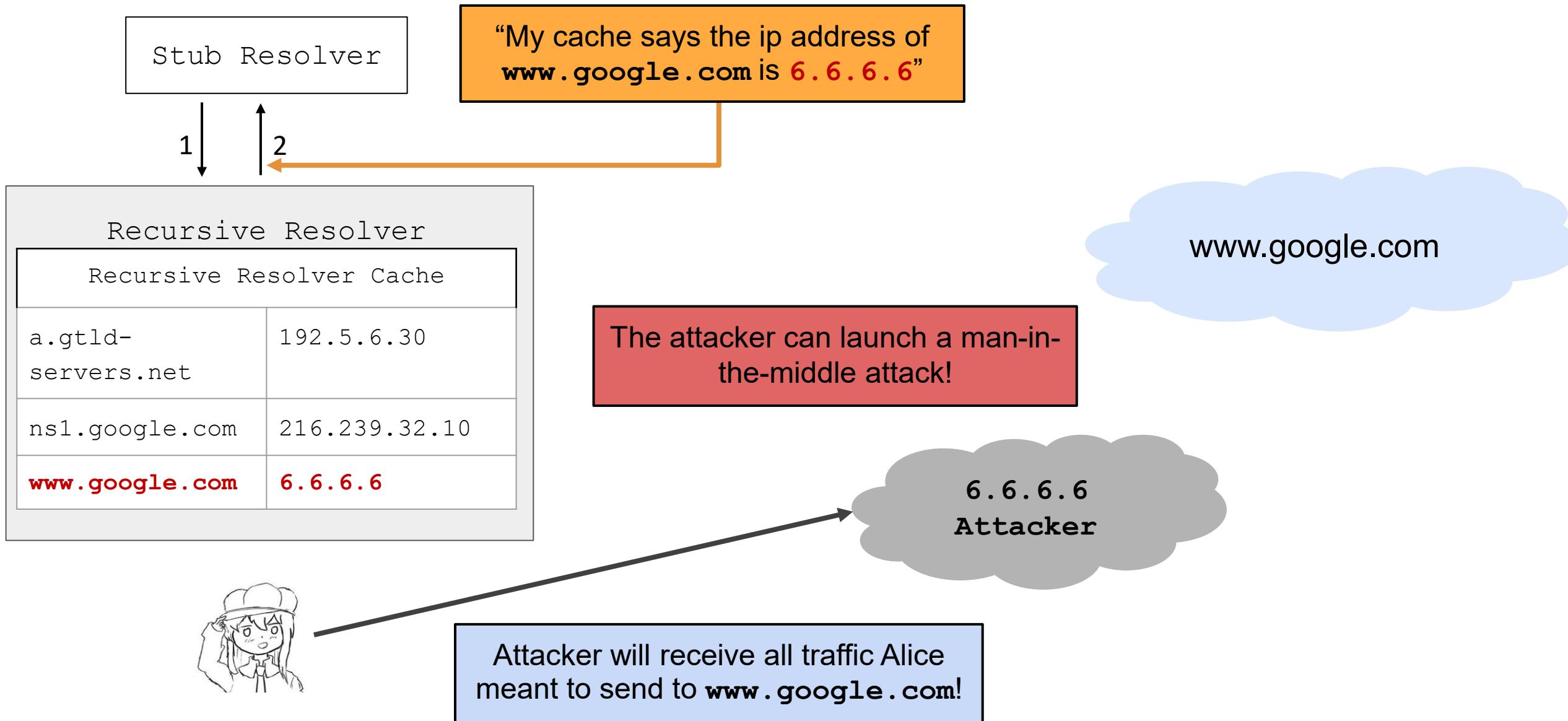
Now that the cache is poisoned, what happens when Alice wants to know the ip address of **www.google.com**?

www.google.com

6.6.6.6
Attacker

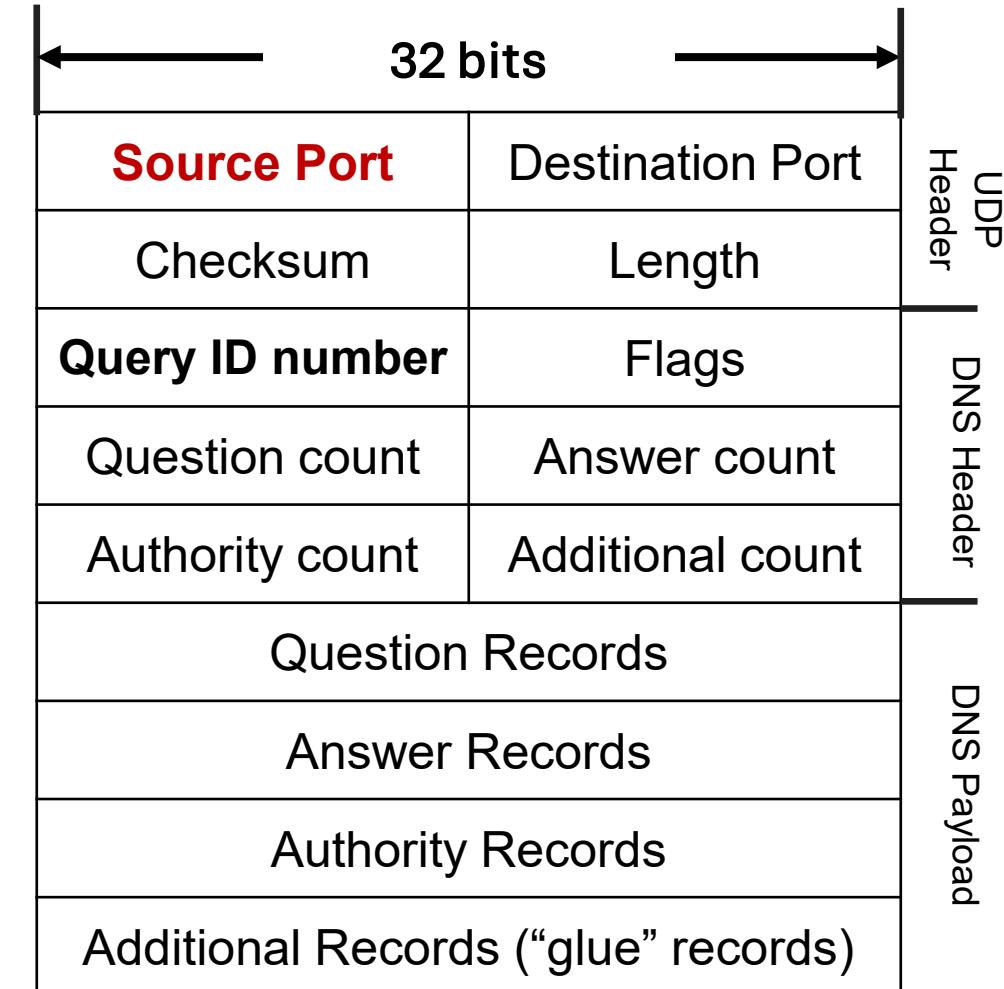


Kaminsky Attack: Example



Defense: Source Port Randomization

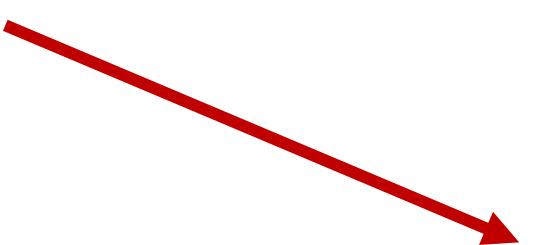
- Why does the attacker's reply work?
 - The Query ID matches the value in the sender's request package
- **Randomize the source port of the DNS query**
 - The attacker must guess the destination port of the response in addition to the query ID
 - This adds 16 bits of entropy, to total 2^{32} possibilities
 - What is information entropy? → The higher the uncertainty, the harder it is for an attacker to successfully launch an attack
 - Attack now takes several hours



Defense: Glue Validation

- Don't cache **glue records** (additional section) as part of DNS lookups
 - They are necessary, since NS records are given in terms of domain names, not IP addresses
 - If you want to cache, you can perform a separate recursive DNS lookup to validate the authority of glue records
- Issue: This was not implemented by all DNS software

It costs you one extra DNS request, but at least it makes Kaminsky attack **impossible**



```
$ dig +norecurse .photos.google.com @128.42.156.7

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26144
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; AUTHORITY SECTION:
.photos.google.com      172800      IN      NS    a.photos.google.com
...
;; ADDITIONAL SECTION:
a.photos.google.com      172800      IN      A     192.5.6.30
...
```

DNS Related Attacks and Risks: Summary

- Cache poisoning attack: Send a malicious record to the resolver, which caches the record
 - Causes packets to be sent to the wrong place (e.g. to the attacker, who becomes a MITM)
- Risk: Malicious name servers
 - Defense: Bailiwick checking: Resolver only accepts records in the name server's zone
- Risk: Network attackers
 - MITM attackers can poison the cache without detection
 - On-path attackers can race the legitimate response to poison the cache
 - Off-path attackers must guess the ID field (Defense: Make the ID field random)
- Kaminsky attack: Query (usually) non-existent domains and put the poisoned record in the additional section (which will still be cached). Lets the off-path attacker try repeatedly until succeeding
 - Defense: Source port randomization (more bits for the off-path attacker to guess)

Cache Poisoning Attack: Response Possibilities

	Attacker correctly guesses the ID number	Attacker does not correctly guess the ID number
Attacker beats the race condition against the legitimate NS	The recursive resolver caches a mapping from a legitimate domain name to the attacker's IP address.	Failure. The recursive resolver caches the actual responses and we have to wait until the record expires to try again.
Attacker does not beat the race condition against the legitimate NS	Failure. The recursive resolver caches the actual responses and we have to wait until the record expires to try again.	Failure. The recursive resolver caches the actual responses and we have to wait until the record expires to try again.

Kaminsky Attack: Response Possibilities

	Attacker correctly guesses the ID number	Attacker does not correctly guess the ID number
Attacker beats the race condition against the legitimate NS	The recursive resolver caches a mapping from legitimate domain names to the attacker's desired IP addresses.	<p>The recursive resolver ignores the response because the ID does not match the request sent earlier.</p> <p>The recursive resolver caches something saying "This domain does not exist".</p> <p>Try again with another fake domain!</p>
Attacker does not beat the race condition against the legitimate NS	<p>The recursive resolver caches something saying "This domain does not exist".</p> <p>Try again with another fake domain!</p>	<p>The recursive resolver caches something saying "This domain does not exist".</p> <p>Try again with another fake domain!</p>

CSC 3511 Security and Networking

Week 9, Lecture 2: Introduction to Cryptography

Roadmap

- ***What is Cryptography?***
- *Definitions and Cryptography Roadmap*
- *Symmetric-Key Encryption*
- *Caesar Cipher*

What is cryptography

- Older definition: The study of secure communication over insecure channels
- Newer definition: Provide rigorous guarantees about the *data* and *computation* in the presence of an attacker
 - Not just **confidentiality** but also **integrity** and **authenticity**

Modern cryptography involves a lot of math:

- Discrete Math
- Probability Theory
- Linear Algebra
- Abstract Algebra
- Number Theory, and more...

We will teach you the basic building blocks of cryptography without going into the details

It's very easy to make a mistake that makes your code insecure

- Lots of tricky *edge cases* that we won't cover
- One small bug could compromise the security of your code ([SONY PlayStation 3 Hack](#))

Never try to write your own cryptographic algorithms:

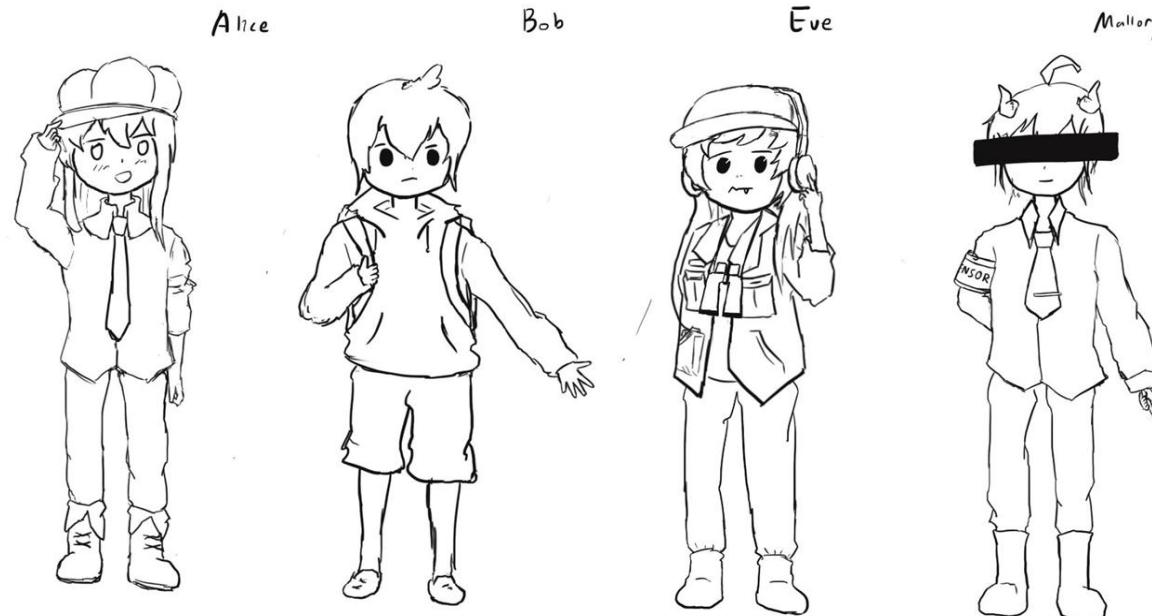
- Use existing well-vetted cryptographic libraries
- This portion of the class is mostly about making you a good consumer of cryptography

Roadmap

- *What is Cryptography?*
- ***Definitions and Cryptography Roadmap***
- *Symmetric-Key Encryption*
- *Caesar Cipher*

Meet Alice, Bob, Eve, and Mallory

- Alice and Bob: The main characters trying to send messages to each other over an insecure communication channel
- Eve: A passive attacker (**eavesdropper**) who can observe and read data sent over the channel, but cannot modify it
- Mallory: An active attacker (man-in-the-middle) who can intercept, read, modify, and even inject data sent over the channel



Meet Alice, Bob, Eve, and Mallory

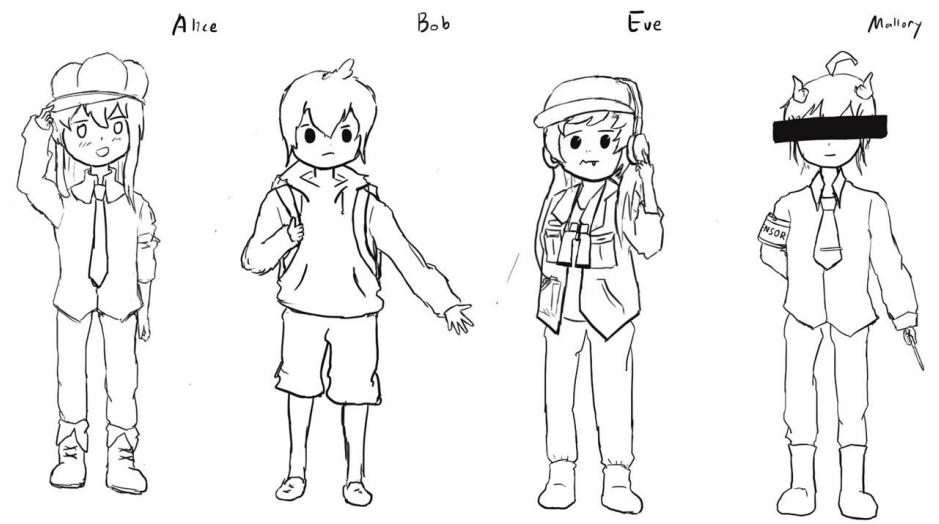
We often describe cryptographic problems using a common cast of characters ([Alice and Bob](#))

For example:

- Alice wants to send a message to Bob.
- However, Eve is going to eavesdrop on the communication channel.
- How does Alice send the message to Bob without Eve learning about the message?

Another example:

- Bob wants to send a message to Alice.
- However, Mallory is going to tamper with the communication channel.
- How does Bob send the message to Alice without Mallory changing the message?

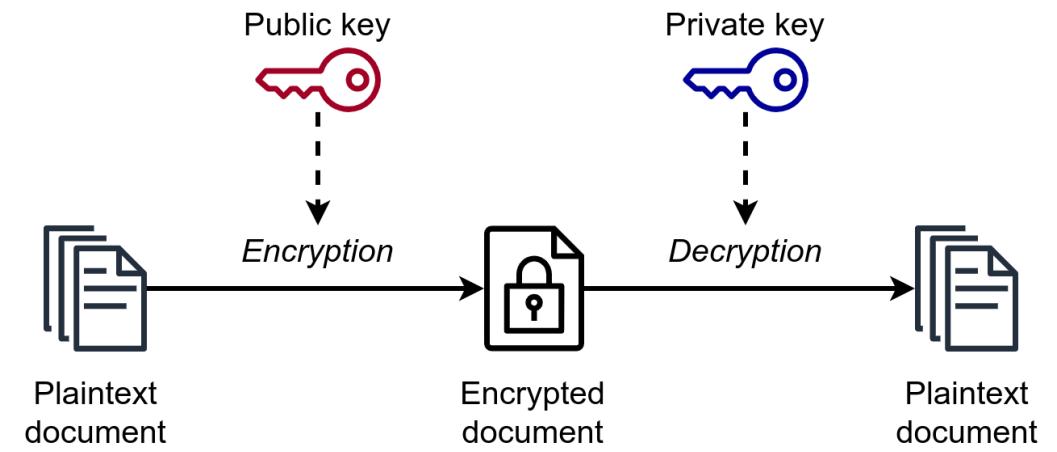
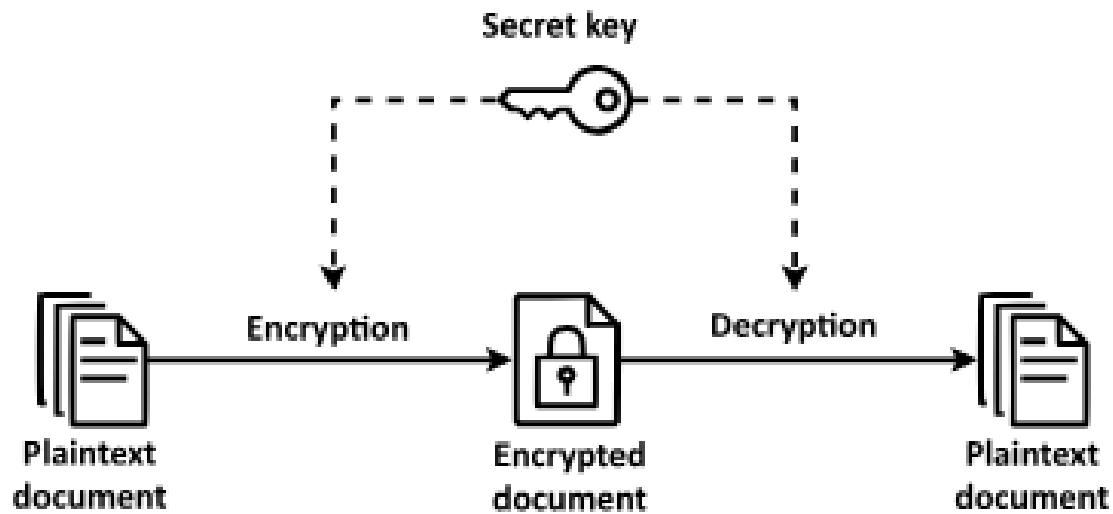


Three Goals of Cryptography

- In cryptography, there are three common properties that we want on our data
- **Confidentiality:** An adversary cannot *read* our messages
- **Integrity:** An adversary cannot *change* our messages without being detected
- **Authenticity:** I can prove that this message came from the person who claims to have written it
 - Integrity and authenticity are closely related properties:
 1. Before I can prove that a message came from a certain person, I have to prove that the message wasn't changed!
 2. But they're not identical properties. Later we'll see some edge cases

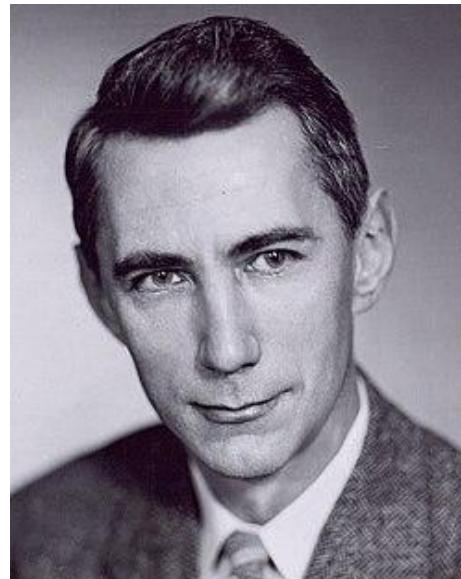
Keys

- The most basic building block of any cryptographic scheme: The **key**
- We can use the key in our algorithms to secure messages
- Two models of keys:
 - **Symmetric key model:** Alice and Bob both know the value of **the same secret key**
 - **Asymmetric key model:** A user has **two keys**, a private key and a public key



Security Principle: Kerckhoff's Principle

- Kerckhoff's principle says:
 - Cryptosystems should remain secure even when the attacker knows **all internal details** of the system
 - The **key** should be the only thing that must be kept secret
 - The system should be designed to make it easy to change keys that are leaked (or suspected to be leaked)
 - If your secrets are leaked, it is usually a lot easier to change the key than to replace every instance of the running software
- Our assumption: The attacker knows all the algorithms we use. The only information the attacker is missing is the **secret key(s)**.
- This principle is closely related to Shannon's Maxim:
 - "one ought to design systems under the assumption that the enemy will immediately gain full familiarity with them"



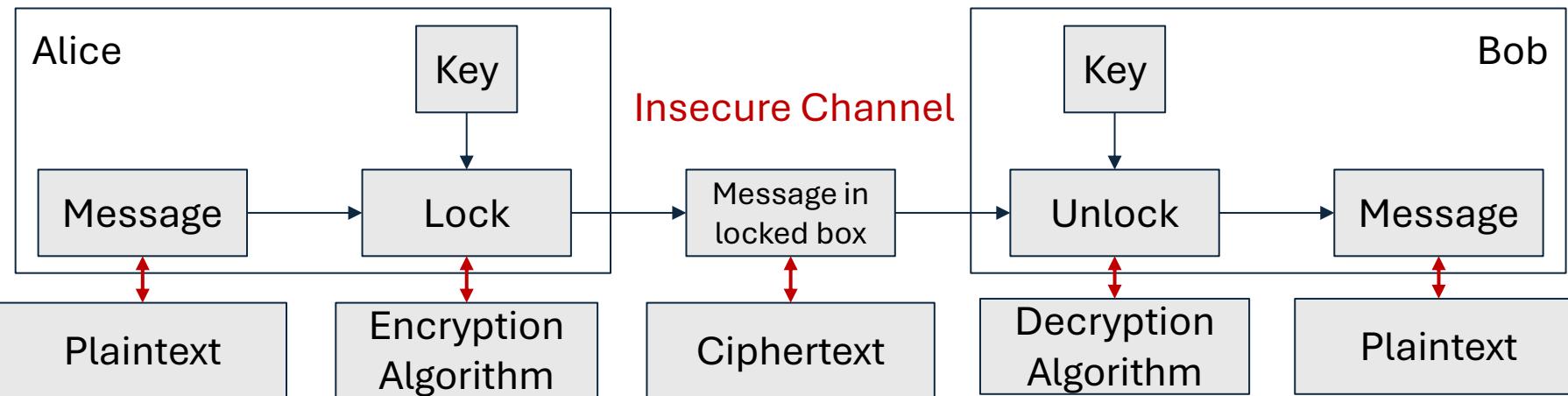
Claude Shannon



Auguste Kerckhoffs

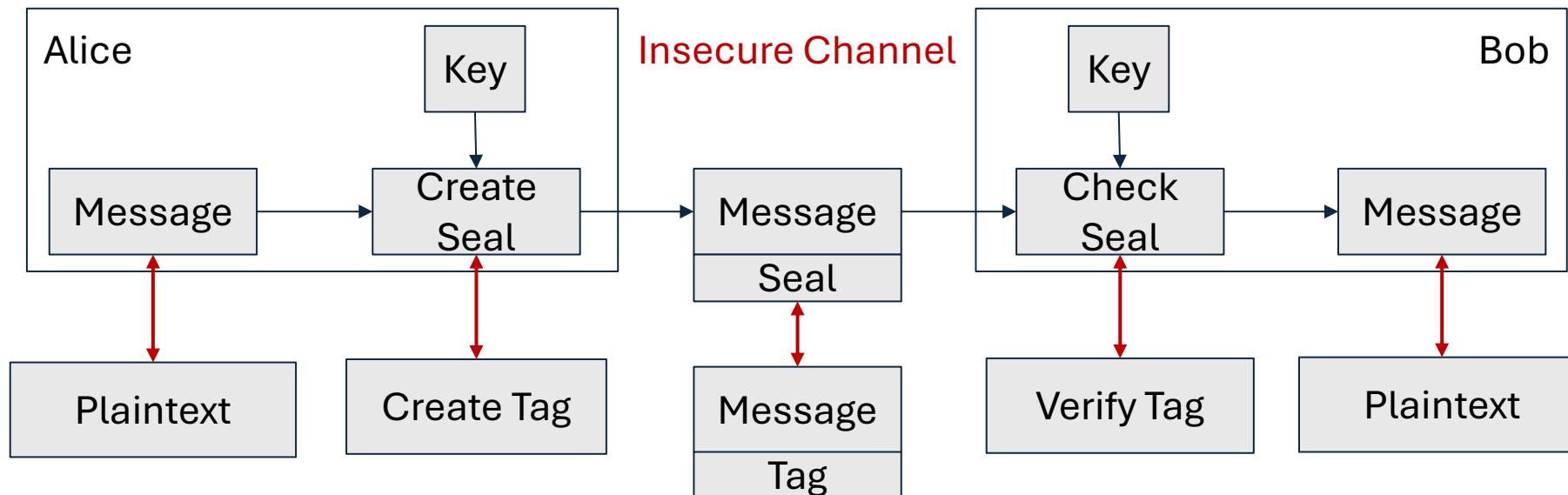
Confidentiality

- **Confidentiality:** An adversary *cannot read* our messages
- Schemes provide confidentiality by **encrypting** messages
- Analogy: Locking and unlocking the message
 - Alice uses the key to lock (**encrypt**) the message (**plaintext**) in a box
 - Alice sends the locked message (**ciphertext**) over the insecure channel
 - Eve sees the locked box, but cannot access the message without the key
 - Bob receives the locked message and uses the key to unlock (**decrypt**) the message



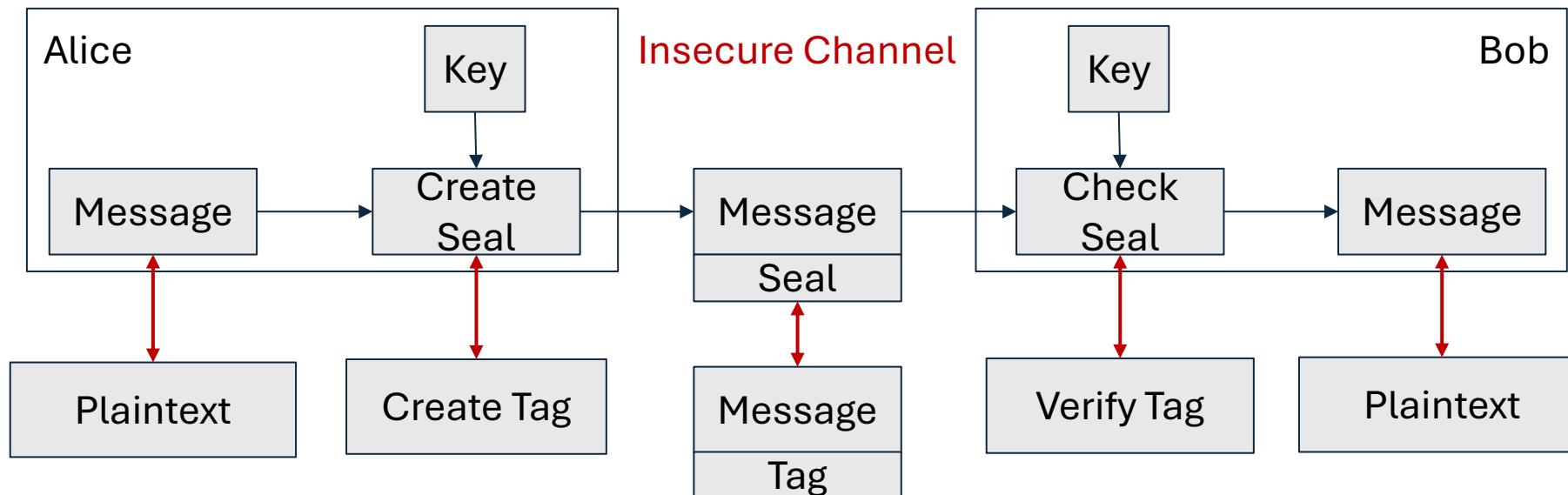
Integrity (and Authenticity)

- Integrity: An adversary *cannot change* our messages without being detected.
- Schemes provide integrity by *adding a tag or signature* on messages
- Analogy: Adding a seal on the message
 - Alice uses the key to add a special seal on the message (e.g. puts tape on the envelope)
 - Alice sends the message and the seal over the insecure channel
 - If Mallory tampers with the message, she'll break the seal (e.g. break the tape on the envelope)
 - Without the key, Mallory cannot create her own seal
 - Bob receives the message and the seal and checks that the seal has not been broken



Integrity (and Authenticity)

- Schemes provide integrity by adding a **tag** or **signature** on messages
 - Alice uses the key to generate a special tag for the message
 - Alice sends the message and the tag over the insecure channel
 - If Mallory tampers with the message, the tag will no longer be valid
 - Bob receives the message and the tag and checks that the tag is still valid
- More on Integrity and Authenticity in a future lecture



Threat Models

- What if Eve can do more than eavesdrop?
- Real-world schemes are often vulnerable to more sophisticated attackers, so cryptographers have created more sophisticated threat models too
- Some threat models for analyzing **confidentiality**:

	Can Eve trick Alice into encrypting messages of Eve's choosing?	Can Eve trick Bob into decrypting messages of Eve's choosing?	Can Eve eavesdrop on Alice's encrypted communications?
Ciphertext-only	No	No	Yes
<u>Chosen-plaintext (AF is short of water)</u>	Yes	No	Yes
Chosen-ciphertext	No	Yes	Yes
Chosen plaintext-ciphertext	Yes	Yes	Yes

Threat Models

- In this class, we'll focus primarily on chosen plaintext attacks, for simplicity
- In practice, cryptographers use the **chosen plaintext-ciphertext model**
 - It's the most powerful
 - It can actually be defended against

	Can Eve trick Alice into encrypting messages of Eve's choosing?	Can Eve trick Bob into decrypting messages of Eve's choosing?	Can Eve eavesdrop on Alice's encrypted communications?
Ciphertext-only	No	No	Yes
Chosen-plaintext	Yes	No	Yes
Chosen-ciphertext	No	Yes	Yes
Chosen plaintext-ciphertext	Yes	Yes	Yes

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	<ul style="list-style-type: none">● One-time pads● Block ciphers with chaining modes (e.g. AES-CBC)● Stream ciphers	<ul style="list-style-type: none">● RSA encryption● ElGamal encryption
Integrity, Authentication	<ul style="list-style-type: none">● MACs (e.g. HMAC)	<ul style="list-style-type: none">● Digital signatures (e.g. RSA signatures)

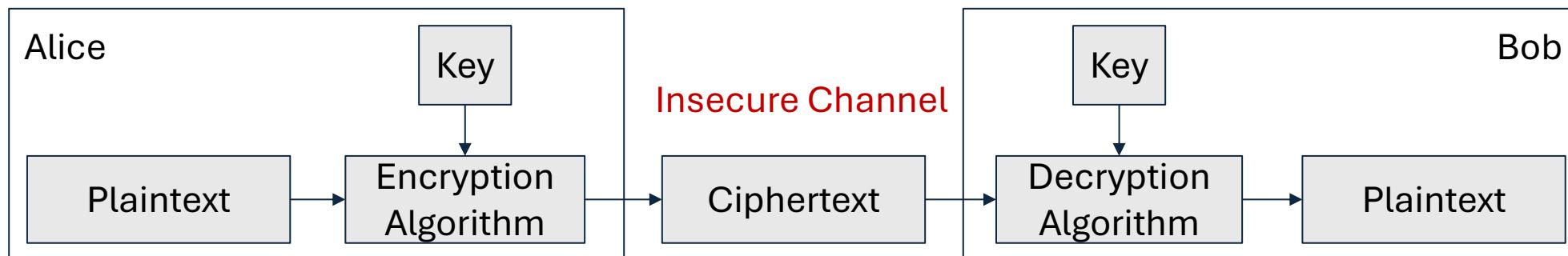
- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)
- Key management (certificates)

Roadmap

- *What is Cryptography?*
- *Definitions and Cryptography Roadmap*
- **Symmetric-Key Encryption**
- *Caesar Cipher*

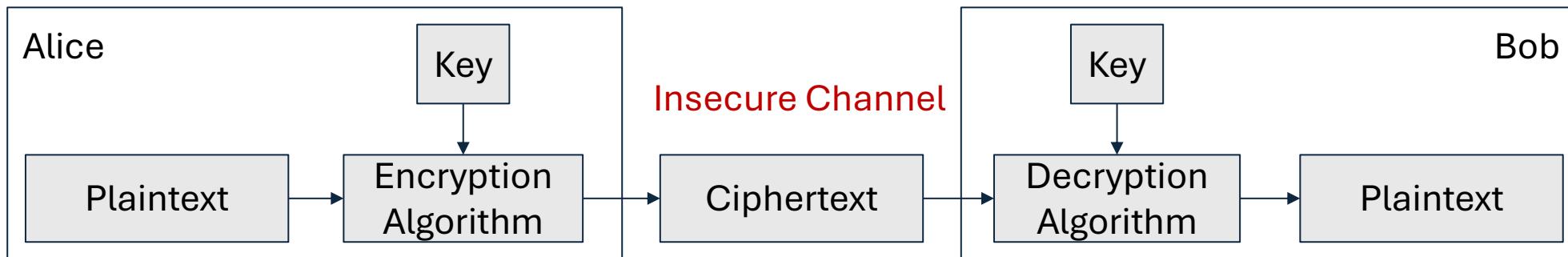
Symmetric-Key Encryption

- The next few schemes are symmetric-key encryption schemes:
 - Encryption schemes aim to provide confidentiality (but not integrity or authentication)
 - Symmetric-key means Alice and Bob share the same secret key that the attacker doesn't know
 - Don't worry about how Alice and Bob share the key for now
- For modern schemes, we're going to assume that messages are bitstrings
 - Bitstring: A sequence of bits (0 or 1), e.g. 11010101001001010
 - Text, images, etc. can usually be converted into bitstrings before encryption



Symmetric-Key Encryption

- A symmetric-key encryption scheme has three algorithms:
 - $\text{KeyGen}() \rightarrow K$: Generate a **key** K
 - $\text{Enc}(K, M) \rightarrow C$: Encrypt a **plaintext** M using the key K to produce **ciphertext** C
 - $\text{Dec}(K, C) \rightarrow M$: Decrypt a ciphertext C using the key K
- What properties do we want from a symmetric encryption scheme?
 - **Correctness**: Decrypting a ciphertext should result in the message that was originally encrypted
 - $\text{Dec}(K, \text{Enc}(K, M)) = M$ for all $K \leftarrow \text{KeyGen}()$ and all M
 - **Efficiency**: Encryption/decryption algorithms should be fast: >1 Gbps on a standard computer
 - **Security**: *Confidentiality* of M from attackers who don't know K



Defining Confidentiality

Recall our definition of confidentiality from earlier: “An adversary cannot read our messages”
This definition isn’t very specific:

- What if Eve can read the first half of Alice’s message, but not the second half?
- What if Eve figures out that Alice’s message starts with “Dear Bob”?

A better definition of confidentiality: The ciphertext should not give the attacker any additional information about the plaintext, beyond what the attacker already knew.

We make this precise with an experiment/security game (aka, indistinguishability game):

1. **Eve chooses** two messages M_0 and M_1 of the same length
2. Alice chooses one message at random M_b , encrypts it, and sends the ciphertext
3. Eve knows either M_0 or M_1 was sent, but doesn't know which
4. **Eve reads the ciphertext** and tries to guess which message was sent
5. If the probability that Eve correctly guesses which message was sent is $1/2$, then the encryption scheme is confidential

Why do we define/play the game this way?

Defining Confidentiality

A better definition of confidentiality (Semantic Security): The **ciphertext should not give the attacker any additional information about the plaintext**, beyond what the attacker already knew.

Intuition

- If the scheme is confidential, Eve can only guess with probability $1/2$, which is no different than if Alice hadn't sent the ciphertext at all
- The ciphertext gave Eve no additional information about which plaintext was sent!
- In other words: The ciphertext is indistinguishable: an adversary cannot tell which of two messages was encrypted, even when they get to choose what those two messages are.

We make this precise with an experiment/security game (aka, indistinguishability game):

1. **Eve chooses** two messages M_0 and M_1 of the same length
2. Alice chooses one message at random M_b , encrypts it, and sends the ciphertext
3. Eve knows either M_0 or M_1 was sent, but doesn't know which
4. **Eve reads the ciphertext** and tries to guess which message was sent
5. If the probability that Eve correctly guesses which message was sent is $1/2$, then the encryption scheme is confidential

Defining Confidentiality: IND-CPA

Let's think about confidentiality from another perspective:

- Recall our threat model: Eve can also perform a chosen plaintext attack
 - Eve can trick Alice into encrypting arbitrary messages of Eve's choice
- We can adapt our experiment to account for this threat model

A specific (formal and more accurate) definition of confidentiality: Even if Eve is able to trick Alice into encrypting other messages, Eve can still only guess what message Alice sent with probability $\frac{1}{2}$

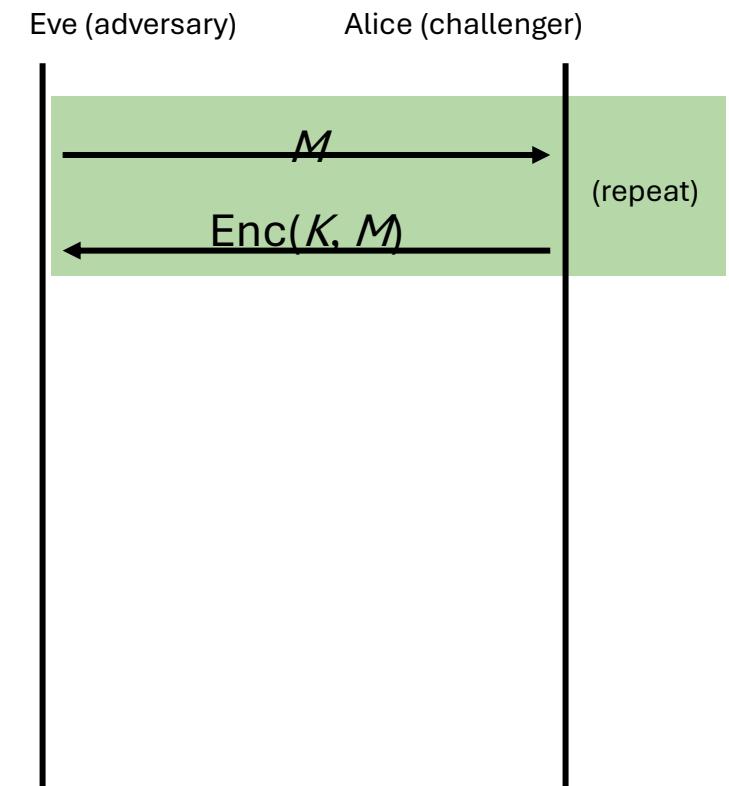
This definition is called **IND-CPA** (indistinguishability under chosen plaintext attack):

- Proven to be equivalent to semantic security
- More commonly used in modern cryptography courses and research: easy to use in proofs; easy to extend to IND-CCA, IND-CCA2, etc.

Let's improve our security game!

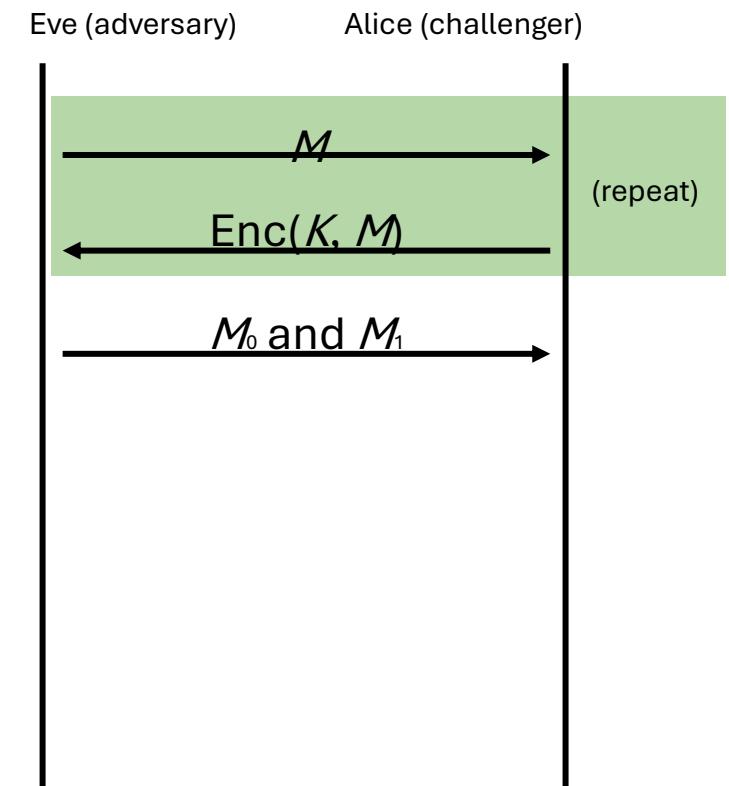
Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts



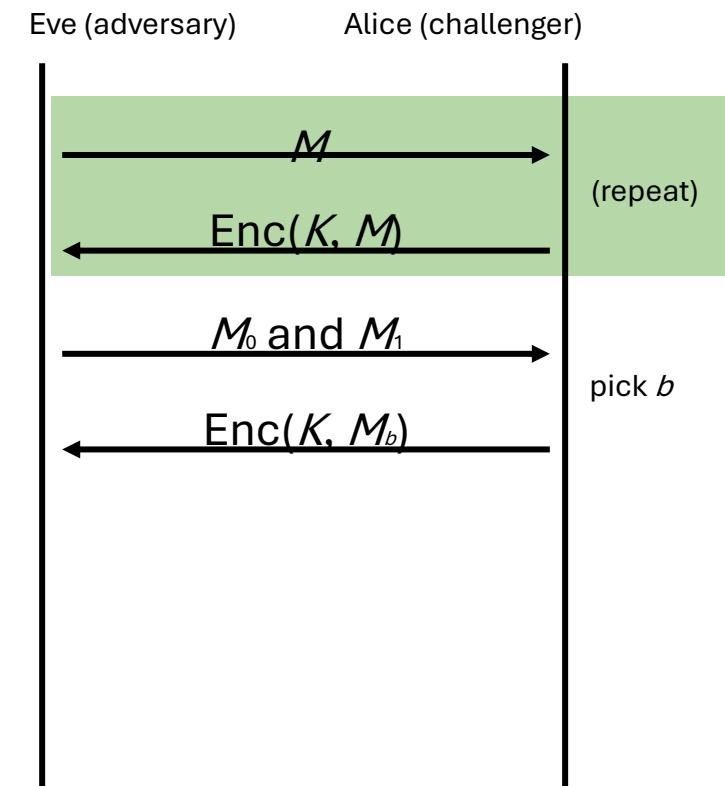
Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice



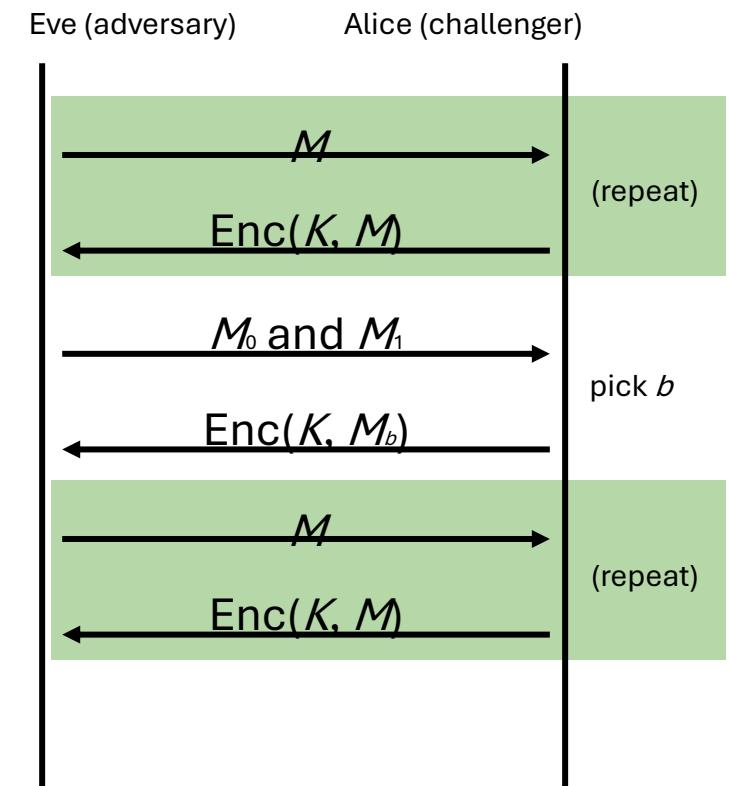
Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice ***randomly chooses*** either M_0 or M_1 to encrypt and sends the encryption back
 - a) Alice does not tell Eve which one was encrypted!



Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice ***randomly chooses*** either M_0 or M_1 to encrypt and sends the encryption back
 - a) Alice does not tell Eve which one was encrypted!
4. Eve may ***again choose plaintexts*** to send to Alice and receives their ciphertexts

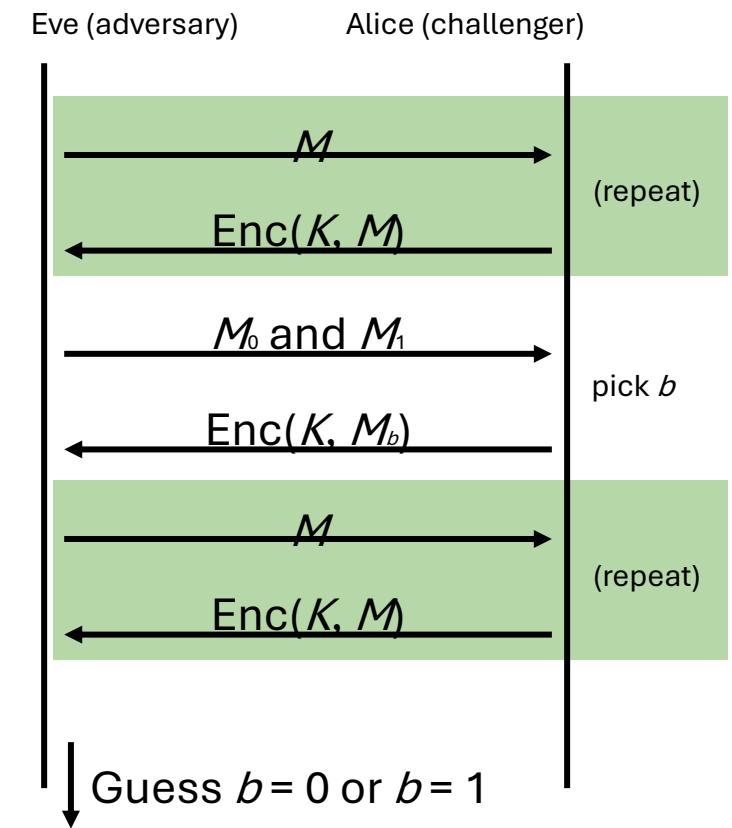


Defining Confidentiality: IND-CPA

1. Eve may choose plaintexts to send to Alice and receives their ciphertexts
2. Eve issues a pair of plaintexts M_0 and M_1 to Alice
3. Alice **randomly chooses** either M_0 or M_1 to encrypt and sends the encryption back
 - a) Alice does not tell Eve which one was encrypted!
4. Eve may **again choose plaintexts** to send to Alice and receives their ciphertexts
5. Eventually, Eve **outputs a guess** as to whether Alice encrypted M_0 or M_1

The security guarantee becomes stronger:

- Even if Eve (adversary) can encrypt whatever they want, whenever they want, before and after seeing the challenge ciphertext, they **STILL** cannot tell which of two messages was encrypted



Defining Confidentiality: IND-CPA

How does Eve guess whether M_0 or M_1 was encrypted? What strategy does she use?

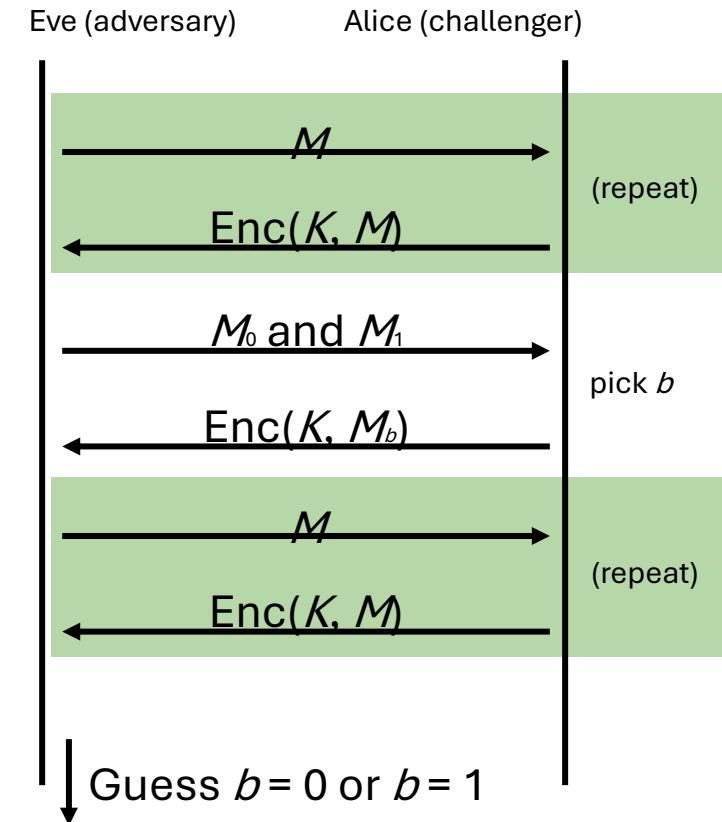
- We don't assume she uses a particular strategy; Eve could use any possible strategy

An encryption scheme is IND-CPA secure if for all polynomial time attackers Eve:

- Eve can win with probability $\leq 1/2 + \varepsilon$, where ε is negligible (e.g., $\varepsilon \leq 1/2^{128}$)

Proving insecurity: There exists at least one strategy that can win the IND-CPA game with probability $> 1/2$

- $1/2$ is the probability of winning by random guessing
- If you can do better than random (e.g., $\varepsilon = 1/100$), then the ciphertext has leaked information, and Eve is able to learn it and use it to gain an advantage!



Roadmap

- *What is Cryptography?*
- *Definitions and Cryptography Roadmap*
- *Symmetric-Key Encryption*
- ***Caesar Cipher***

Caesar Cipher

- Change characters in plaintext to produce ciphertext
- Caesar cipher
 - Used by Julius Caesar over 2,000 years ago
 - Simple idea: Shift each letter by a fixed number of positions in the alphabet
- Example:
 - Key is 3 (right shift each letter by 3 positions)
 - Encryption rule: Replace each letter with the letter 3 positions ahead
 - Plaintext is: HELLO WORLD
 - Ciphertext is: KHOOR ZRUOG
 - H + 3 ➔ K
 - E + 3 ➔ H
 - ...

K = 3			
PT	CT	PT	CT
A	D	N	Q
B	E	O	R
C	F	P	S
D	G	Q	T
E	H	R	U
F	I	S	V
G	J	T	W
H	K	U	X
I	L	V	Y
J	M	W	Z
K	N	X	A
L	O	Y	B
M	P	Z	C

Formalizing Cryptosystems

A cryptosystem is defined by five components ($\mathcal{E}, \mathcal{D}, \mathcal{M}, \mathcal{K}, C$)

- \mathcal{M} : message (plaintexts) space → what can we encrypt?
- \mathcal{K} : key space → what are the possible keys?
- C :set of ciphertexts → what does encryption produce?
- \mathcal{E} :set of encryption functions $e: \mathcal{M} \times \mathcal{K} \rightarrow C$ → How to encrypt the message?
- \mathcal{D} :set of decryption functions $d: C \times \mathcal{K} \rightarrow \mathcal{M}$ → How to decrypt the message?

Example: Caesar cipher

- $\mathcal{M} = \{ \text{sequences of letters} \}$
- $\mathcal{K} = \{ i \mid i \text{ is an integer and } 0 \leq i \leq 25 \}$
- $C = \mathcal{M}$
- $\mathcal{E} = \{ E_k \mid k \in \mathcal{K} \text{ and for all letters } m, E_k(m) = (m + k) \bmod 26 \}$
- $\mathcal{D} = \{ D_k \mid k \in \mathcal{K} \text{ and for all letters } c, D_k(c) = (c - k) \bmod 26 \}$

Attacks on the Caesar Cipher

- Eve sees the ciphertext ***JCKN ECGUCT***, but doesn't know the key K
- If you were Eve, how would you try to break this algorithm?
- Exhaustive search (Brute-force attack)
 - If the key space is small enough, try all possible keys until you find the right one
 - Caesar cipher has 26 possible keys → Try all 26 possible keys!

Attacks on the Caesar Cipher

- Eve sees the ciphertext ***JCKN ECGUCT***, but doesn't know the key K
- If you were Eve, how would you try to break this algorithm?
- Chosen-plaintext attack: Eve tricks Alice into encrypting plaintext of her choice
 - Eve sends a message $M = AAA$ and receives $C = CCC$
 - Eve can deduce the key: C is 2 letters after A, so $K = 2$
 - Eve has the key, so she can decrypt the ciphertext

Attacks on the Caesar Cipher

- Eve sees the ciphertext **JCKN ECGUCT**, but doesn't know the key K
- If you were Eve, how would you try to break this algorithm?
- **Statistical Attack:**

- Compute frequency of each letter in ciphertext:

G	0.1	H	0.1	K	0.1	O	0.3
R	0.2	U	0.1	Z	0.1		

- Apply 1-gram model of English
 - Frequency of characters (1-grams) in English is on next slide

K= 3			
PT	CT	PT	CT
A	D	N	Q
B	E	O	R
C	F	P	S
D	G	Q	T
E	H	R	U
F	I	S	V
G	J	T	W
H	K	U	X
I	L	V	Y
J	M	W	Z
K	N	X	A
L	O	Y	B
M	P	Z	C

Character Frequencies in English

a	0.07984	h	0.06384	n	0.06876	t	0.09058
b	0.01511	i	0.07000	o	0.07691	u	0.02844
c	0.02504	j	0.00131	p	0.01741	v	0.01056
d	0.04260	k	0.00741	q	0.00107	w	0.02304
e	0.12452	l	0.03961	r	0.05912	x	0.00159
f	0.02262	m	0.02629	s	0.06333	y	0.02028
g	0.02013					z	0.00057

Breaking Caesar Cipher: Frequency Analysis Attack

The Weakness: **Letter patterns remain visible**

- Even though letters are shifted, their **relative frequencies** stay the same!

How the Attack Works:

1. Collect ciphertext letter frequencies; Let $f(c)$ = frequency of letter c in the **ciphertext**
2. Compare against known English letter frequencies; Let $p(x)$ = frequency of letter x in the **English**
3. Try all 26 possible keys: For each possible shift/key i ($0 \sim 25$), calculate how well it matches English
 - a) $\varphi(i)$: correlation of frequency of letters in ciphertext with corresponding letters in English, assuming key is i
 - b) $\varphi(i) = \sum_{0 \leq c \leq 25} f(c)p(c-i)$
4. Pick the key with highest correlation: The **i** that gives the highest $\varphi(i)$ is likely the correct key!

Correlation of $\varphi(i)$ and Result

i	$\varphi(i)$	i	$\varphi(i)$	i	$\varphi(i)$	i	$\varphi(i)$
0	0.0469	7	0.0461	13	0.0505	19	0.0312
1	0.0393	8	0.0194	14	0.0561	20	0.0287
2	0.0396	9	0.0286	15	0.0215	21	0.0526
3	0.0586	10	0.0631	16	0.0306	22	0.0398
4	0.0259	11	0.0280	17	0.0386	23	0.0338
5	0.0165	12	0.0318	18	0.0317	24	0.0320
6	0.0676					25	0.0443

Most probable keys, based on φ :

- $i = 6, \varphi(i) = 0.0676$
 - plaintext EBIIL TLOLA
- $i = 10, \varphi(i) = 0.0631$
 - plaintext AXEEH PHKEW
- $i = 14, \varphi(i) = 0.0561$
 - plaintext WTAAD LDGAS
- $i = 3, \varphi(i) = 0.0586$
 - plaintext HELLO WORLD

Only English phrase is for $i = 3$

- That's the key (3 or 'D')

Caesar's Problem

- Key is too short
 - Can be found by exhaustive search
 - Statistical frequencies not concealed well
 - They look too much like regular English letters
- So, make it longer
 - Multiple letters in key
 - Idea is to smooth the statistical frequencies to make cryptanalysis harder

Vigènere Cipher

- Like Caesar cipher, but use a phrase
 - So, it's effectively multiple Caesar ciphers
 - Usually, the key is a repeating keyword
- Example
 - Message: A T T A C K A T D A W N
 - Key: L E M O N L E M O N L E

Encrypt:

- Convert Letters to Numbers

$(0+11) \% 26 = 11$	$(19+4) \% 26 = 23$	$(19+12) \% 26 = 5$	$(0+14) \% 26 = 14$
$(2+13) \% 26 = 15$	$(10+11) \% 26 = 21$	$(0+4) \% 26 = 4$	$(19+12) \% 26 = 5$
$(3+14) \% 26 = 17$	$(0+13) \% 26 = 13$	$(22+11) \% 26 = 7$	$(13+4) \% 26 = 17$
- Convert Numbers back to Letters:
Ciphertext: L X F O P V E F R N H R