Networking Overview + Network Protocol Security

Slides credit: Vern Paxson

Today's Lecture

- Part 1: Networking Overview

- Part 2: Security issues

Keep in mind, networking is:

- Complex topic with many facets
  - We will omit concepts/details that aren't very security-relevant
  - We'll mainly look at IP, TCP, DNS and DHCP

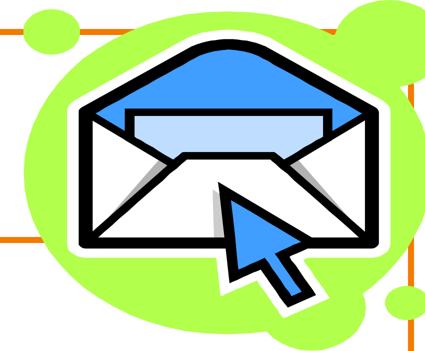
- Networking is full of abstractions
  - Goal is for you to develop apt *mental models / analogies*
  - ASK questions when things are unclear
    - (but we may skip if not ultimately relevant for security, or postpone if question itself is directly about security)

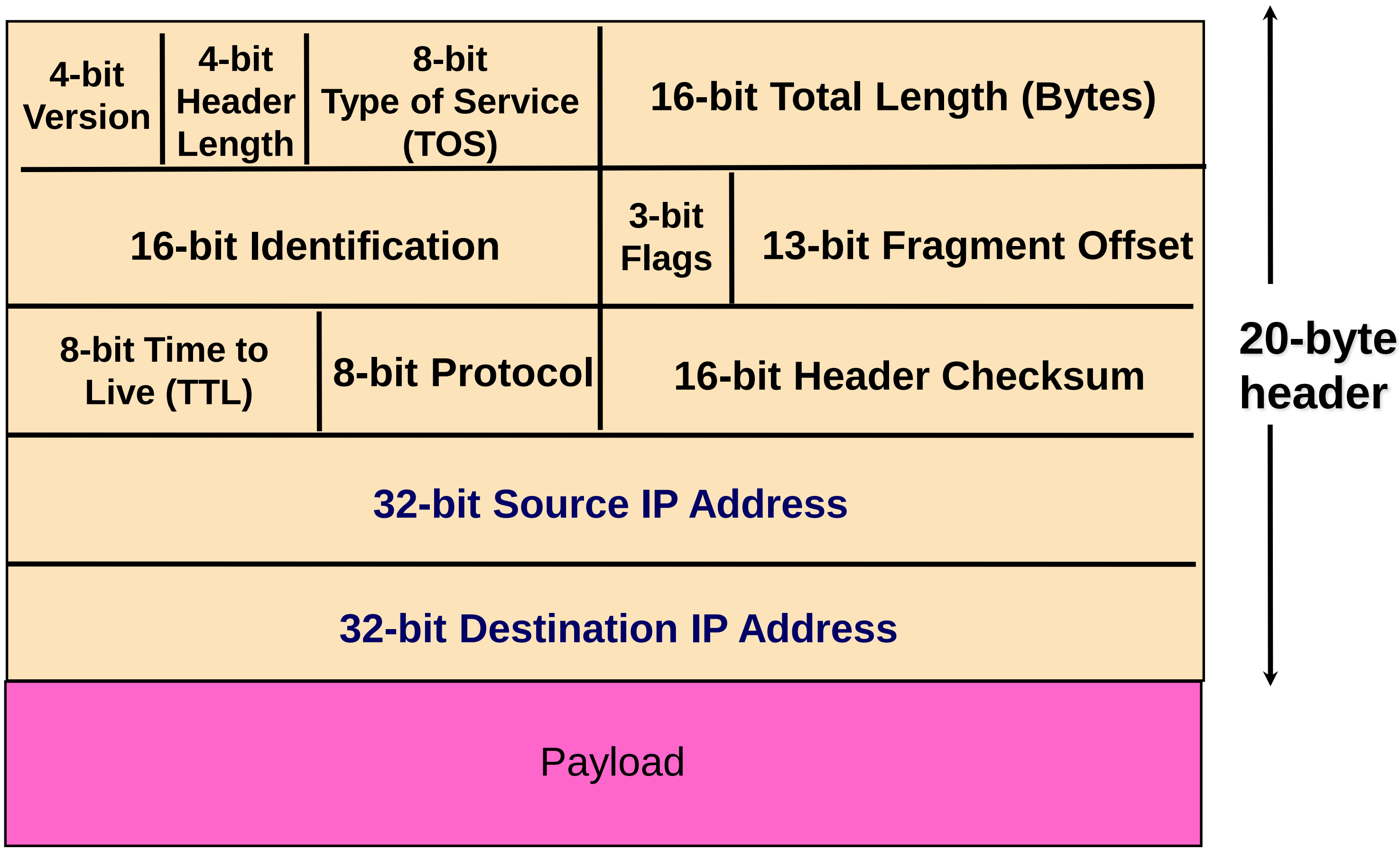
Networking Overview

Key Concept #1: *Protocols*

- A protocol is an agreement on how to communicate
- Includes syntax and semantics
  - How a communication is specified & structured
    - Format, order messages are sent and received
  - What a communication means
    - Actions taken when transmitting, receiving, or timer expires
- E.g.: asking a question in lecture?
  1. Raise your hand.
  2. Wait to be called on.
  3. Or: wait for speaker to **pause** and vocalize
  4. If unrecognized (after timeout): vocalize w/ "excuse me"

Example: IP Packet *Header*





IP = Internet Protocol

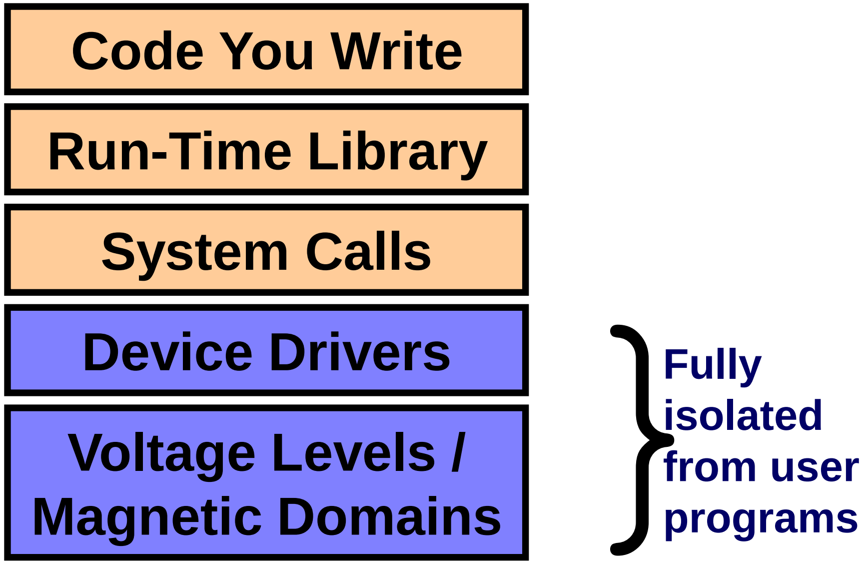
Key Concept #2: *Dumb Network*

- Original Internet design: interior nodes ("routers") have no knowledge* of ongoing connections going through them
- **Not**: how you picture the telephone system works
  - Which internally tracks all of the active voice calls
- Instead: the postal system!
  - Each Internet message ("packet") self-contained
  - Interior "routers" look at destination address to forward
  - If you want smarts, build it "end-to-end"
  - Buys simplicity & robustness at the cost of shifting complexity into end systems

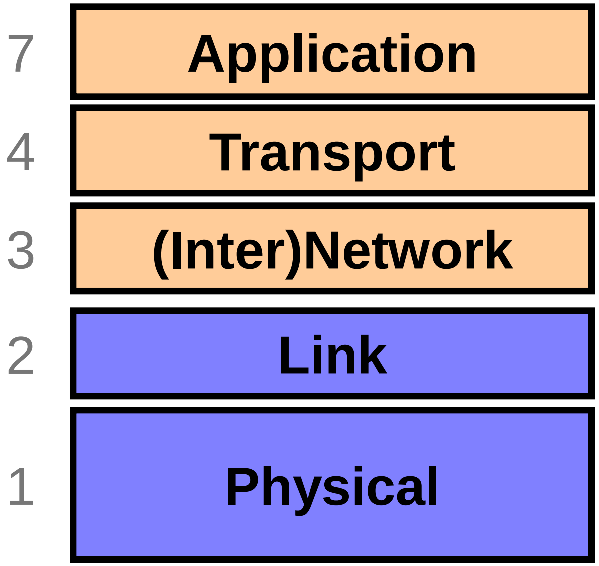
* Today's Internet is full of hacks that violate this

Key Concept #3: *Layering*

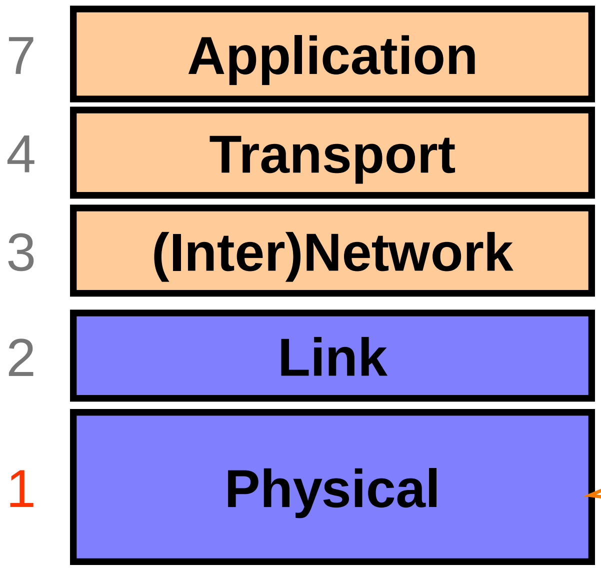
- Internet design is strongly partitioned into layers
  - Each layer relies on services provided by next layer below …
  - … and provides services to layer above it
- Analogy:
  - Consider structure of an application you've written and the "services" each layer relies on / provides



Internet Layering ("Protocol Stack")



Layer 1: Physical Layer

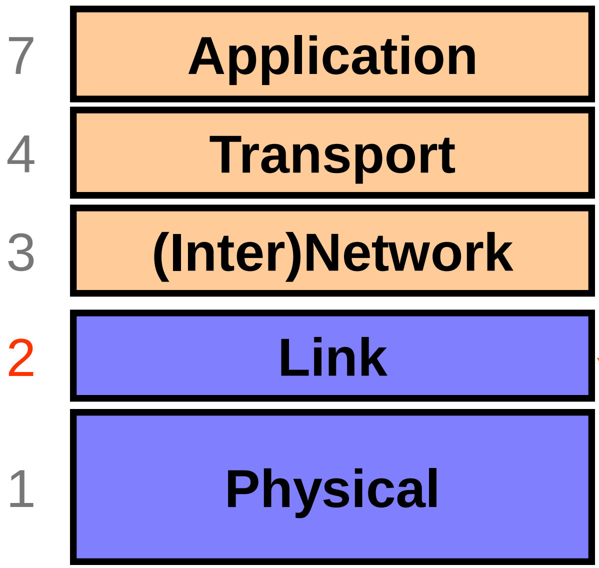


Encoding bits to send them over a single **physical link**

e.g. patterns of

voltage levels /
photon intensities /
RF modulation

Layer 2: Link Layer

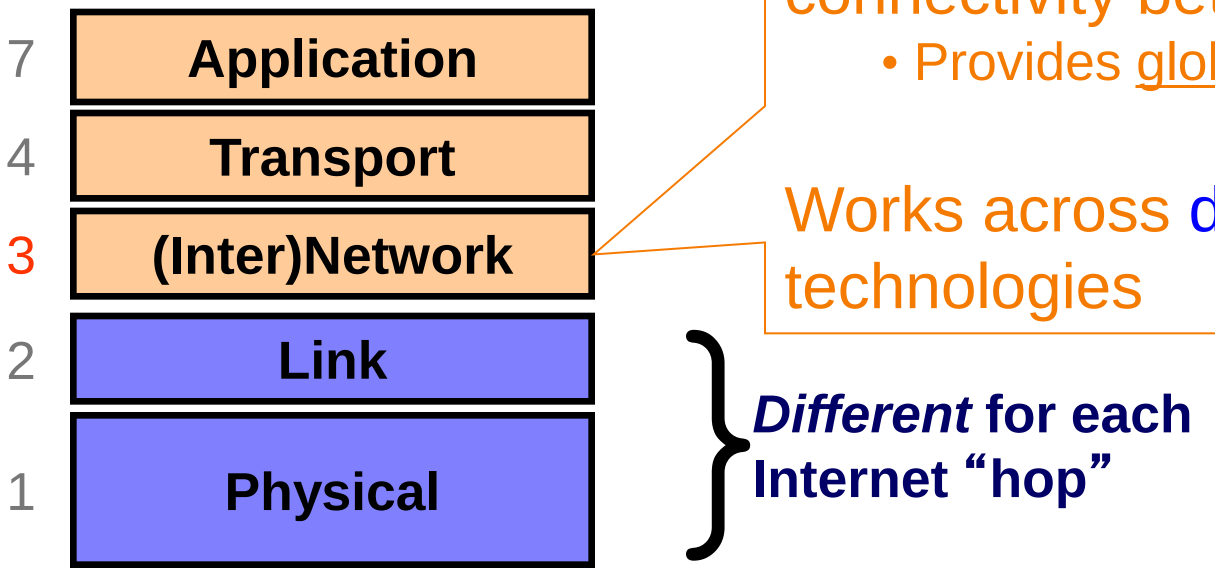


Framing and transmission of a collection of bits into individual messages sent across a single "subnetwork" (one physical technology)

Might involve multiple *physical links* (e.g., modern Ethernet)

Often technology supports broadcast transmission (**every** "node" connected to subnet receives)

Layer 3: (Inter)Network Layer

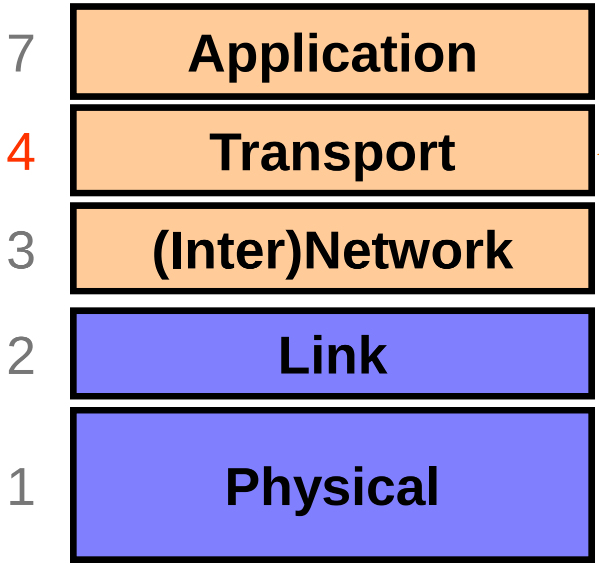


Bridges multiple "subnets" to provide *end-to-end* internet connectivity between nodes

- Provides global addressing

Works across different link technologies

Layer 4: Transport Layer



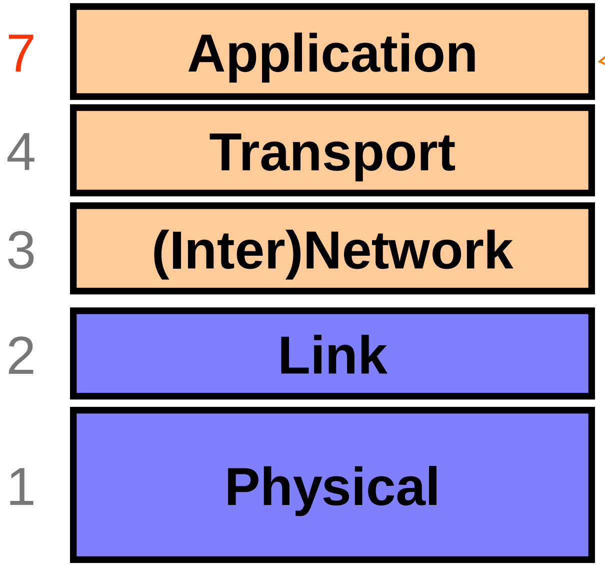
End-to-end communication between processes

Different services provided:

TCP = reliable *byte stream*

UDP = unreliable *datagrams*

Layer 7: Application Layer



Communication of whatever you wish

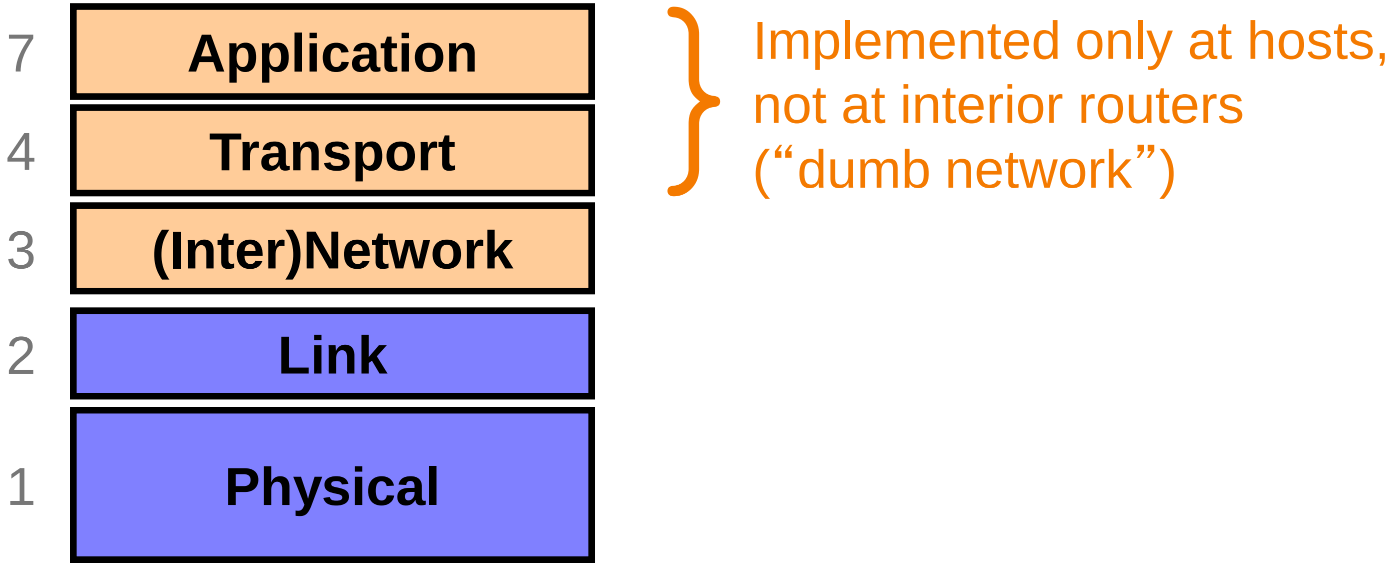
Can use whatever transport(s) is convenient

Freely structured

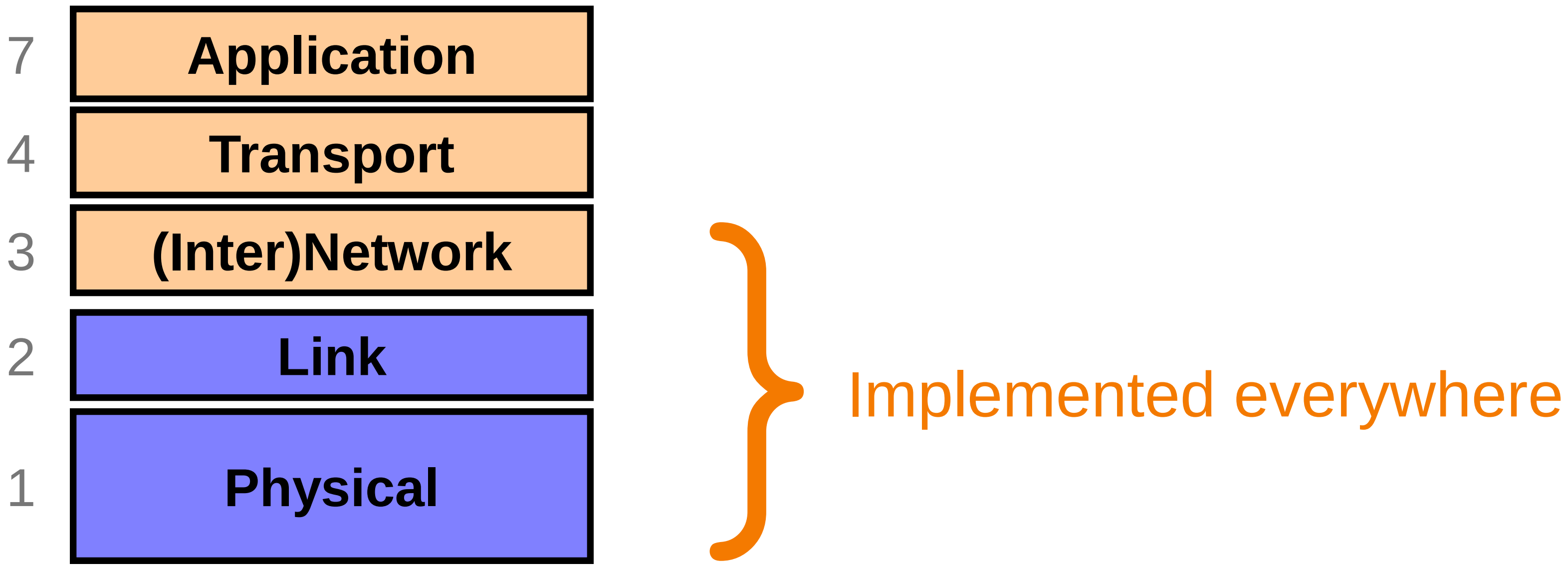
E.g.:

Skype, SMTP (email),
HTTP (Web), Halo, BitTorrent

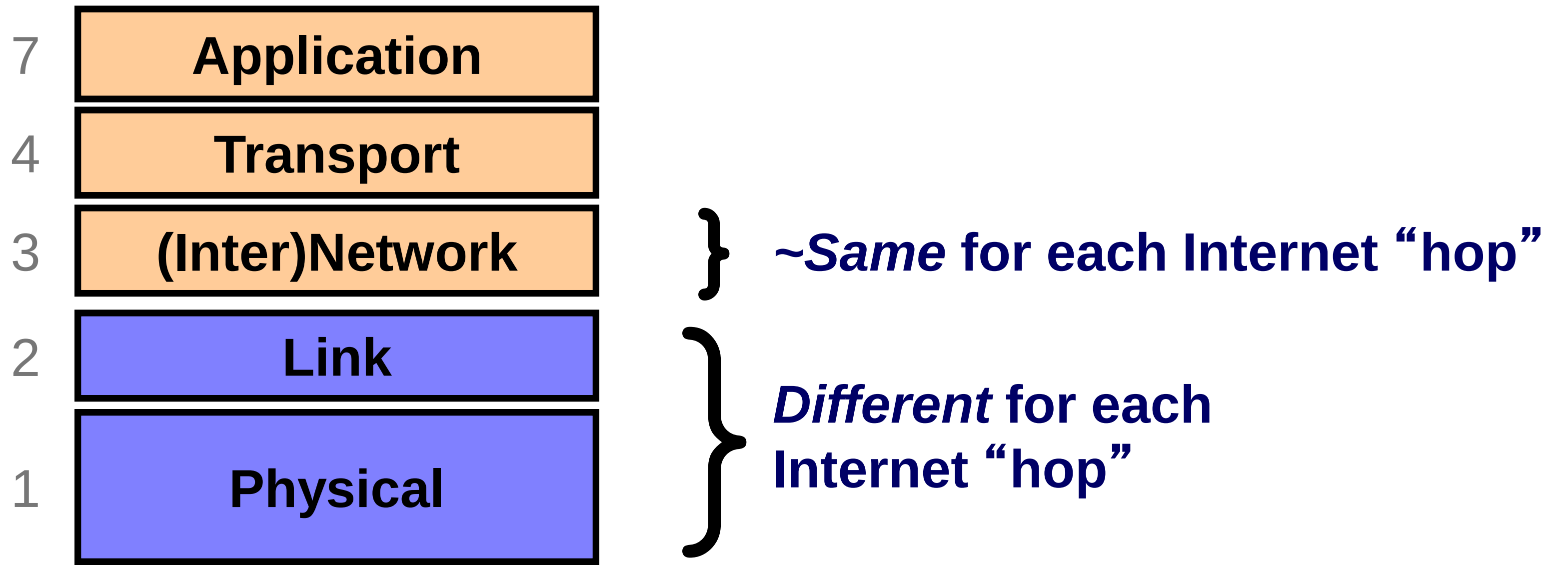
Internet Layering ("Protocol Stack")



Internet Layering ("Protocol Stack")

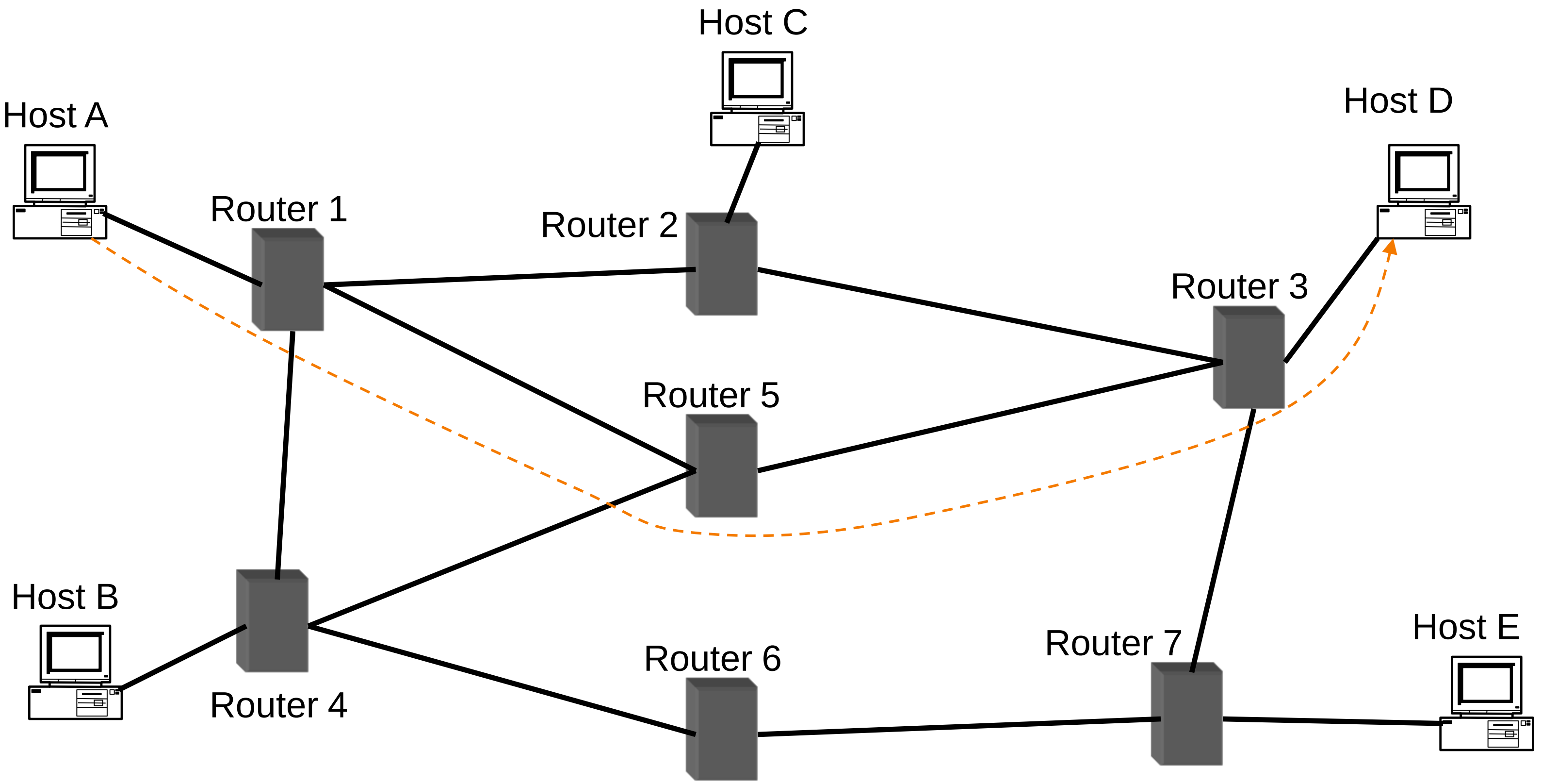


Internet Layering ("Protocol Stack")



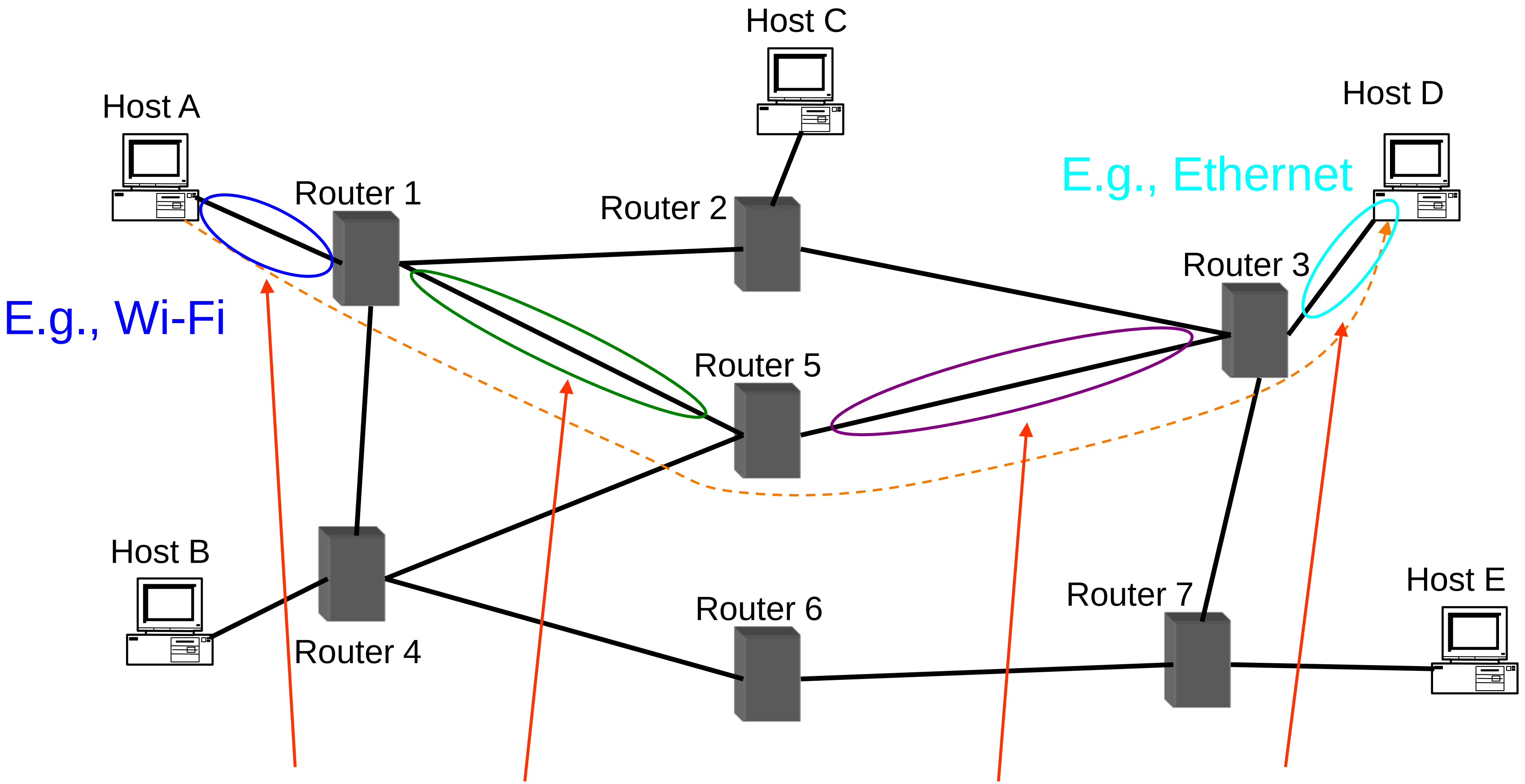
Hop-By-Hop vs. End-to-End Layers

Host A communicates with Host D



Hop-By-Hop vs. End-to-End Layers

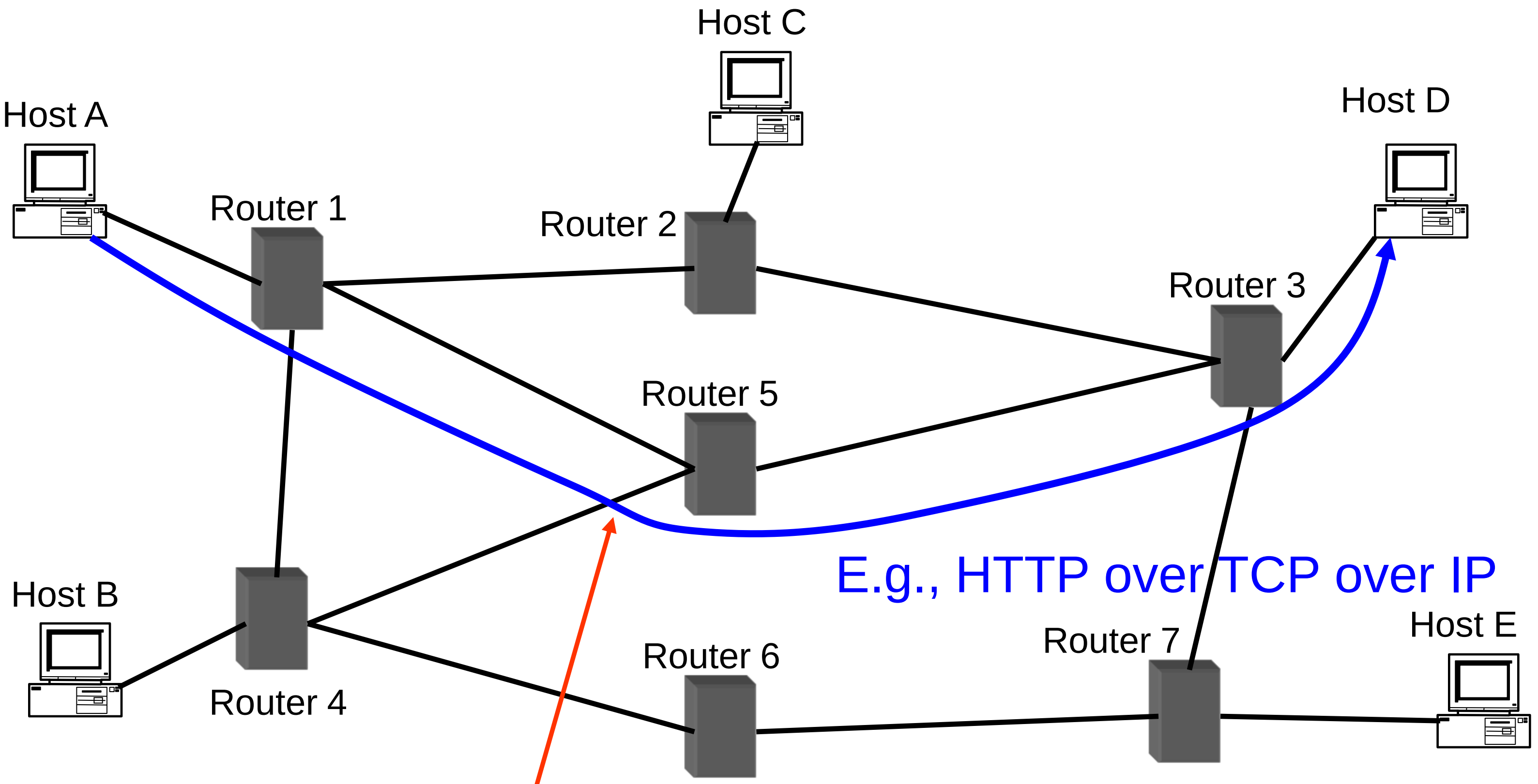
Host A communicates with Host D



***Different* Physical & Link Layers (Layers 1 & 2)**

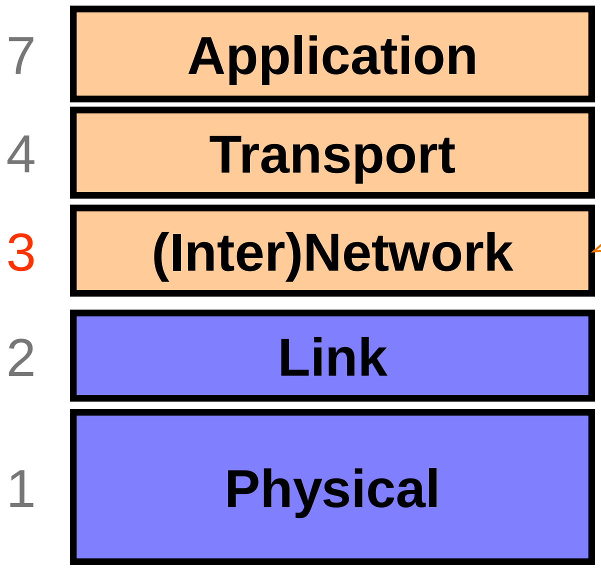
Hop-By-Hop vs. End-to-End Layers

Host A communicates with Host D



Same Network / Transport / Application Layers (3/4/7)
(Routers **ignore** Transport & Application layers)

Layer 3: (Inter)Network Layer

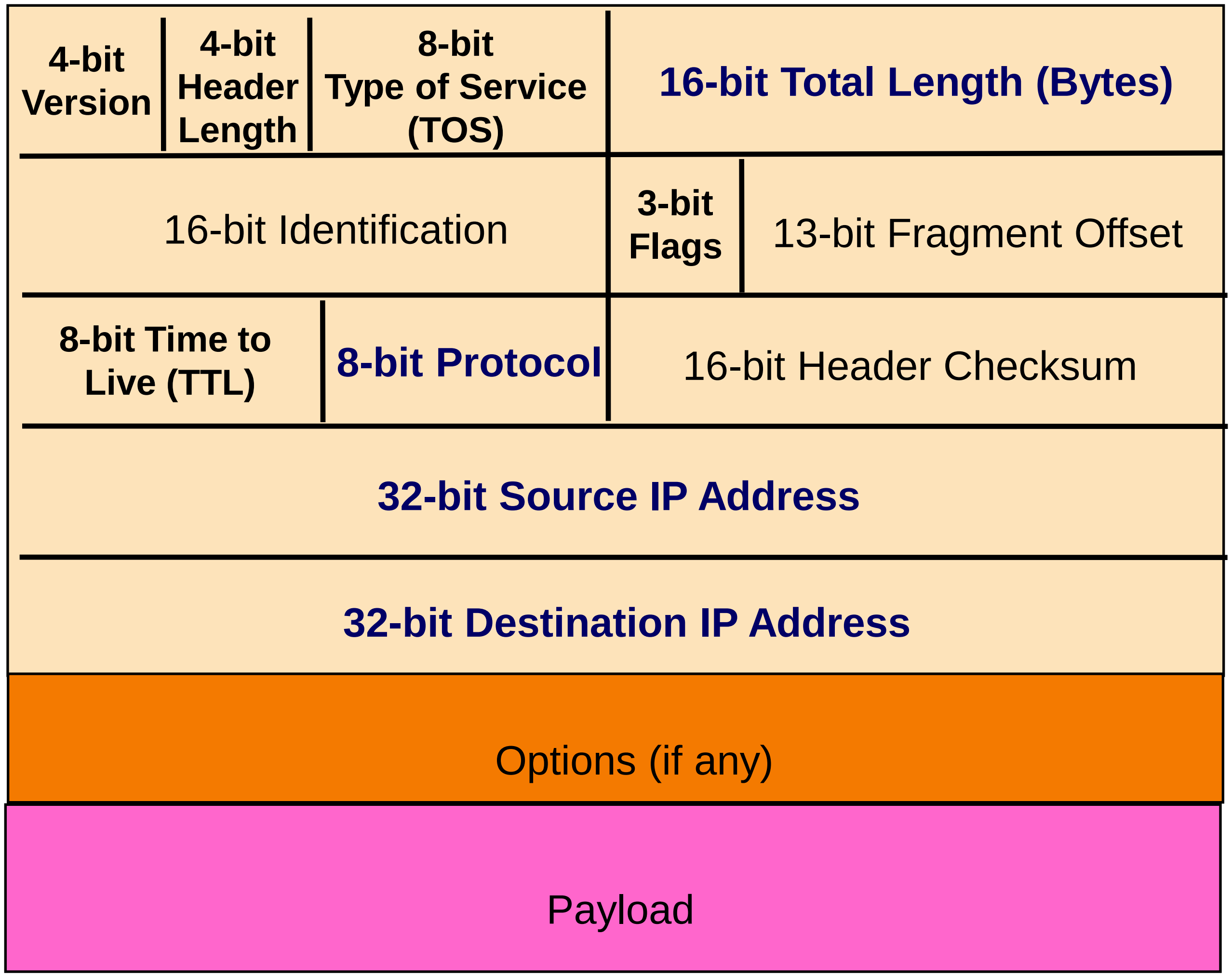


Bridges multiple "subnets" to provide *end-to-end* internet connectivity between nodes

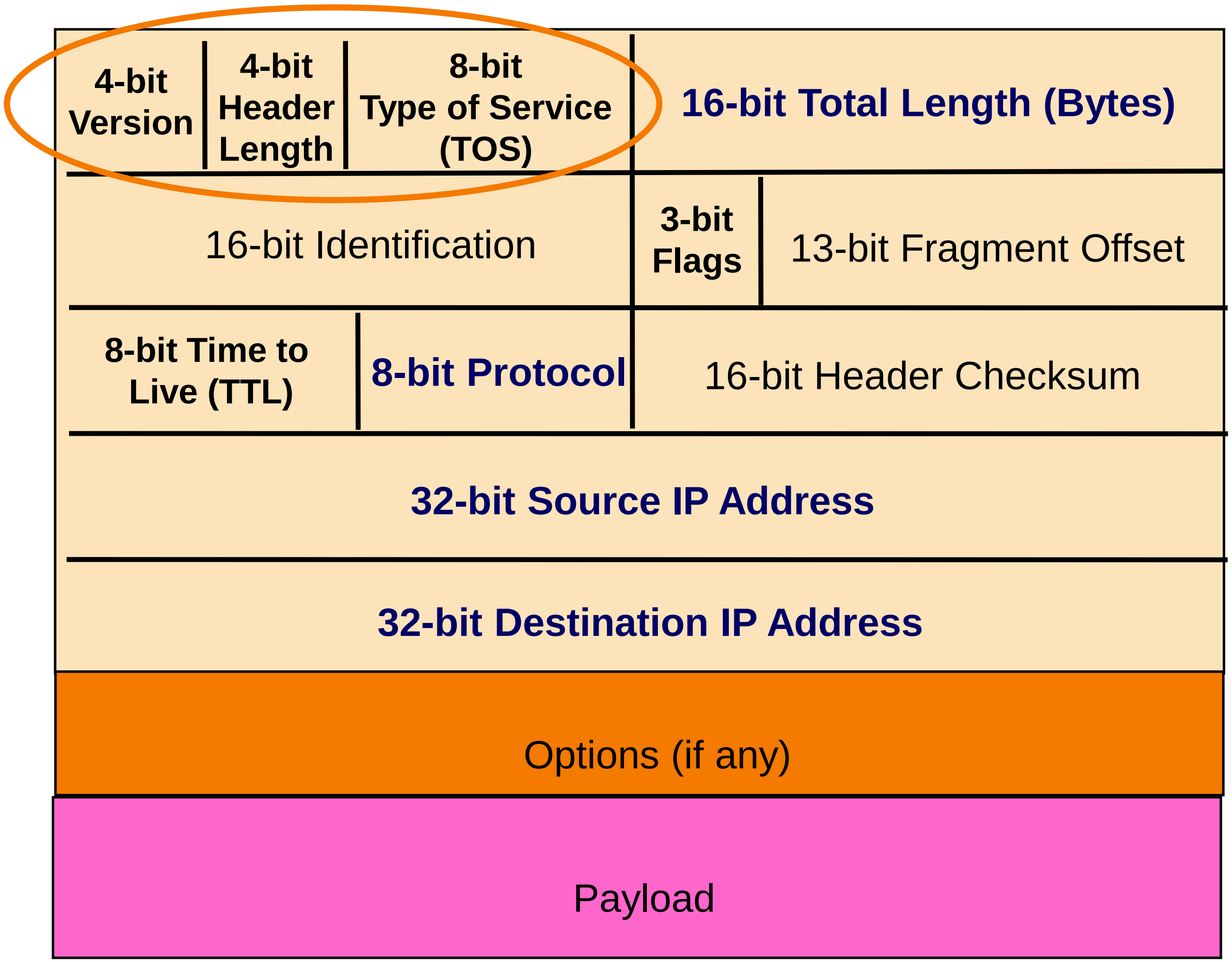
- Provides global addressing

Works across different link technologies

IP Packet Structure



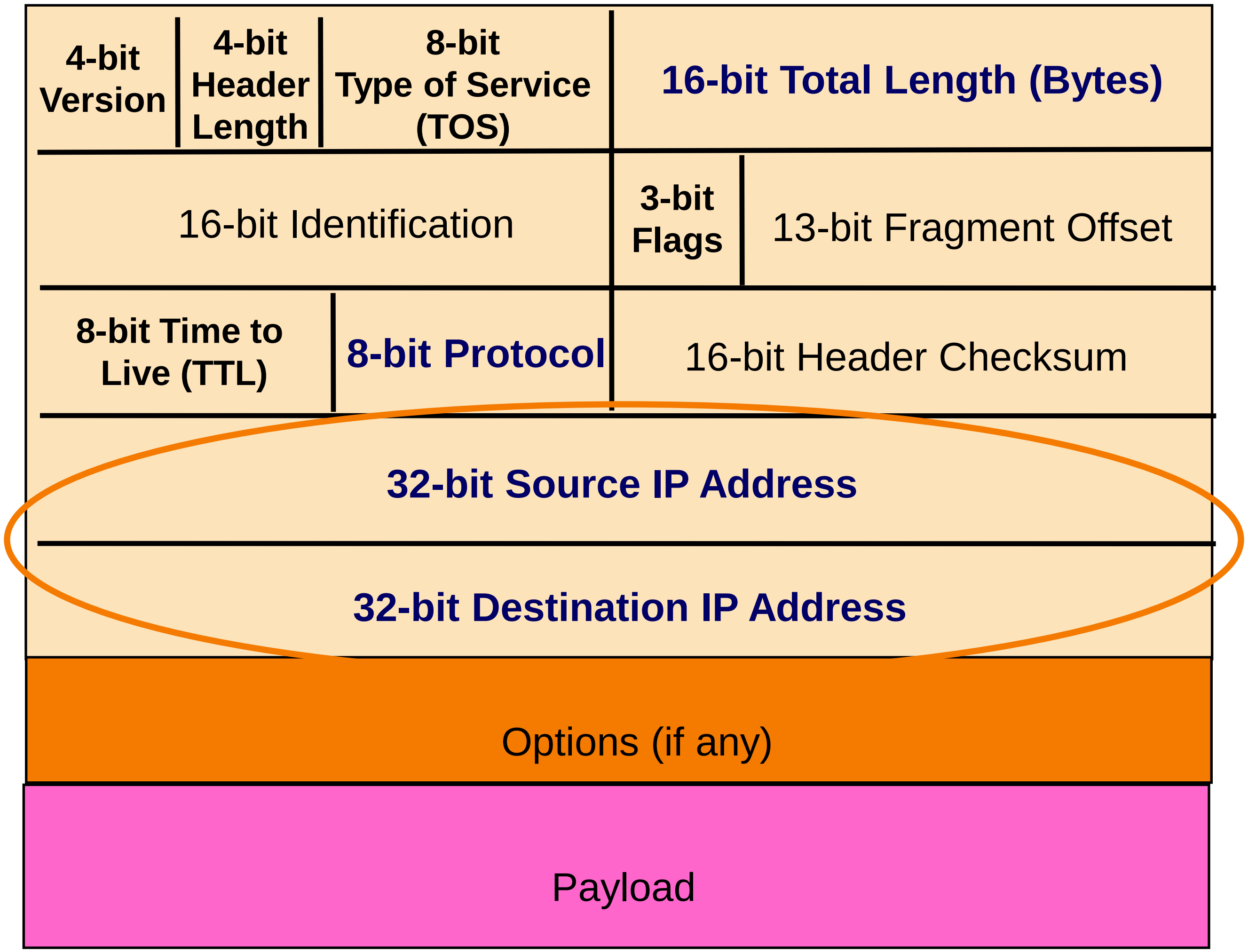
IP Packet Structure



IP Packet Header Fields

- Version number (4 bits)
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically “4” (for IPv4), and sometimes “6” (for IPv6)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically “5” (for a 20-byte IPv4 header)
  - Can be more when IP options are used
- Type-of-Service (8 bits)
  - Allow packets to be treated differently based on needs
  - E.g., low delay for audio, high bandwidth for bulk transfer

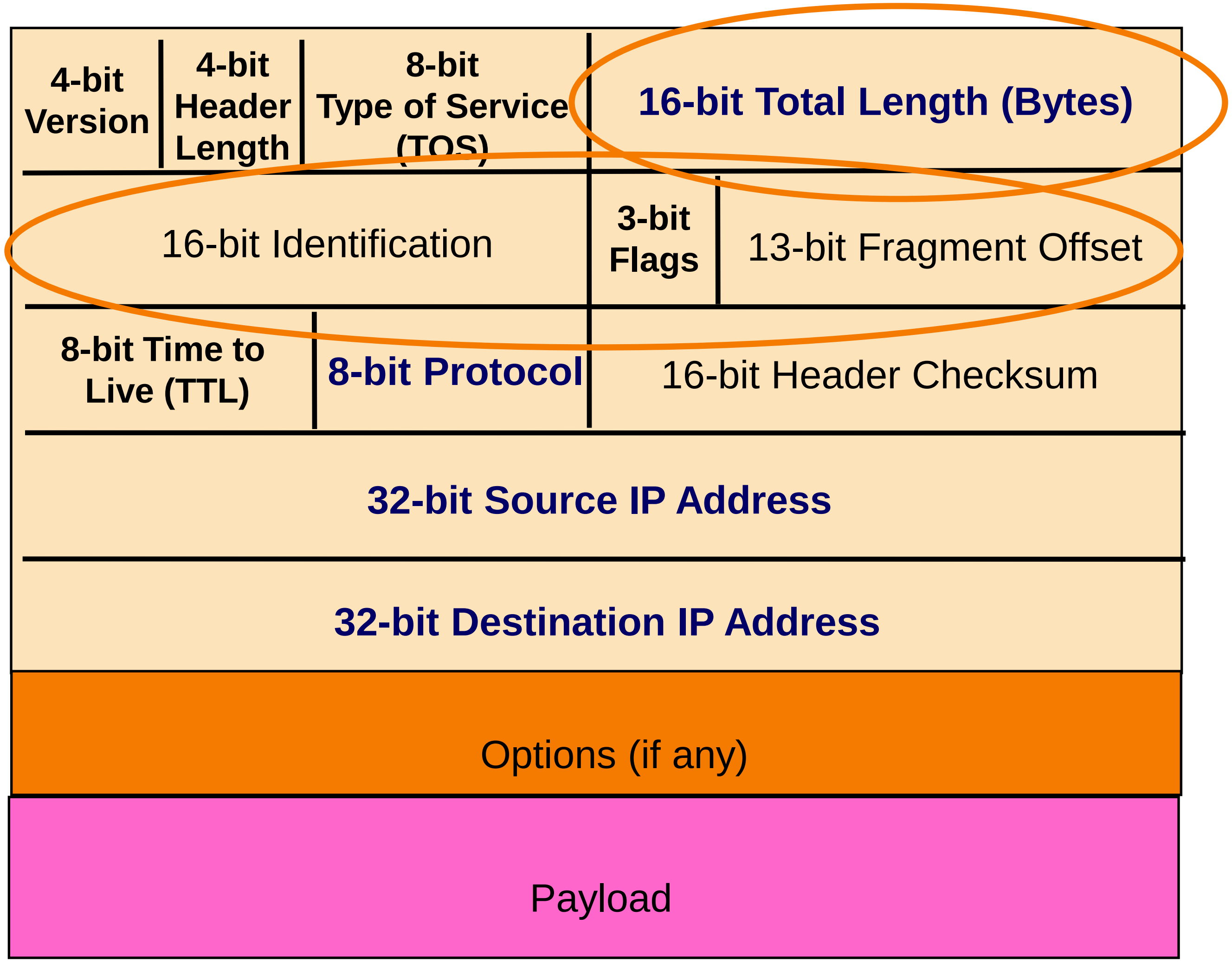
IP Packet Structure



IP Packet Header (Continued)

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

IP Packet Structure

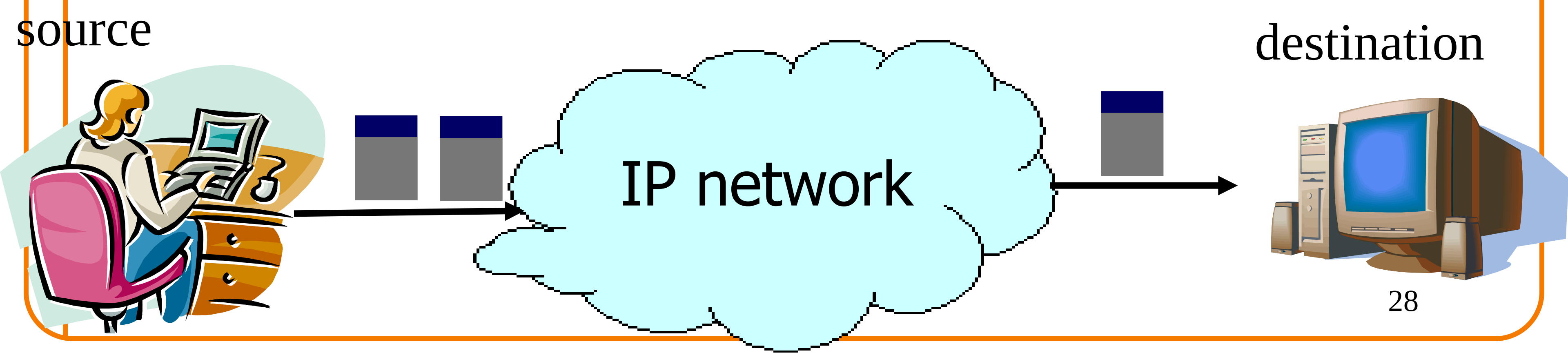


IP Packet Header Fields (Continued)

- Total length (16 bits)
  - Number of bytes in the packet
  - Maximum size is 65,535 bytes (2^{16} -1)
  - … though underlying links may impose smaller limits
- Fragmentation: when forwarding a packet, an Internet router can split it into multiple pieces ("fragments") if too big for next hop link
- End host reassembles to recover original packet
- Fragmentation information (32 bits)
  - Packet identifier, flags, and fragment offset
  - Supports dividing a large IP packet into fragments
  - … in case a link cannot handle a large IP packet

IP: "*Best Effort*" Packet Delivery

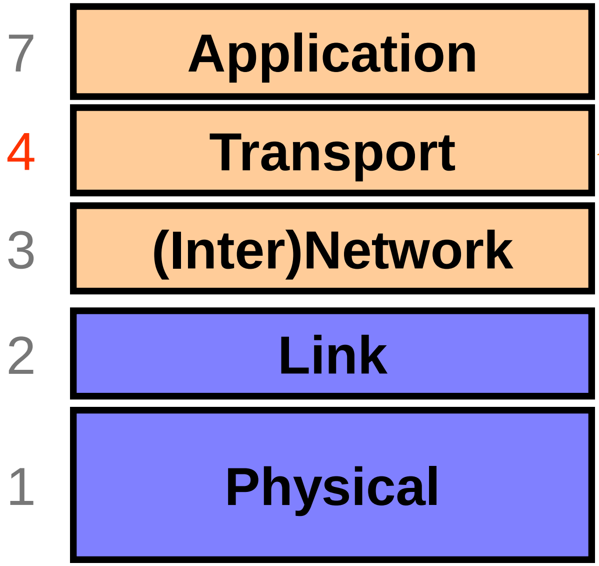
- Routers inspect destination address, locate "next hop" in forwarding table
  - Address = ~unique identifier/locator for the receiving host
- Only provides a "*I'll give it a try*" delivery service:
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order



"Best Effort" is Lame! What to do?

- It's the job of our Transport (layer 4) protocols to build services our apps need out of IP's modest layer-3 service

Layer 4: Transport Layer



End-to-end communication between processes

Different services provided:

TCP = reliable *byte stream*

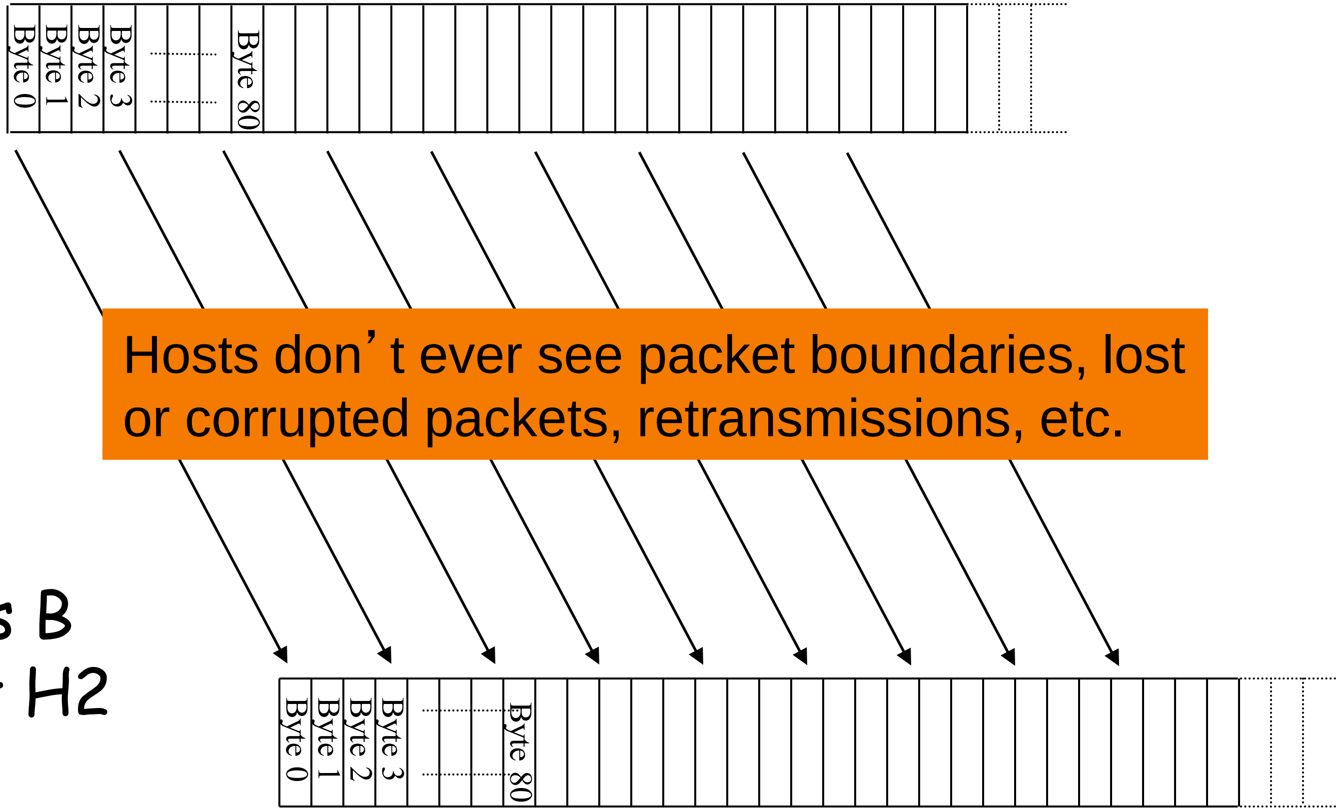
UDP = unreliable *datagrams*

"Best Effort" is Lame! What to do?

- It's the job of our Transport (layer 4) protocols to build services our apps need out of IP's modest layer-3 service
- #1 workhorse: TCP (Transmission Control Protocol)
- Service provided by TCP:
  - Connection oriented (explicit set-up / tear-down)
    - End hosts (processes) can have multiple concurrent long-lived communication
  - **Reliable**, in-order, byte-stream delivery
    - Robust detection & retransmission of lost data

TCP “Bytestream” Service

Process A on host H1

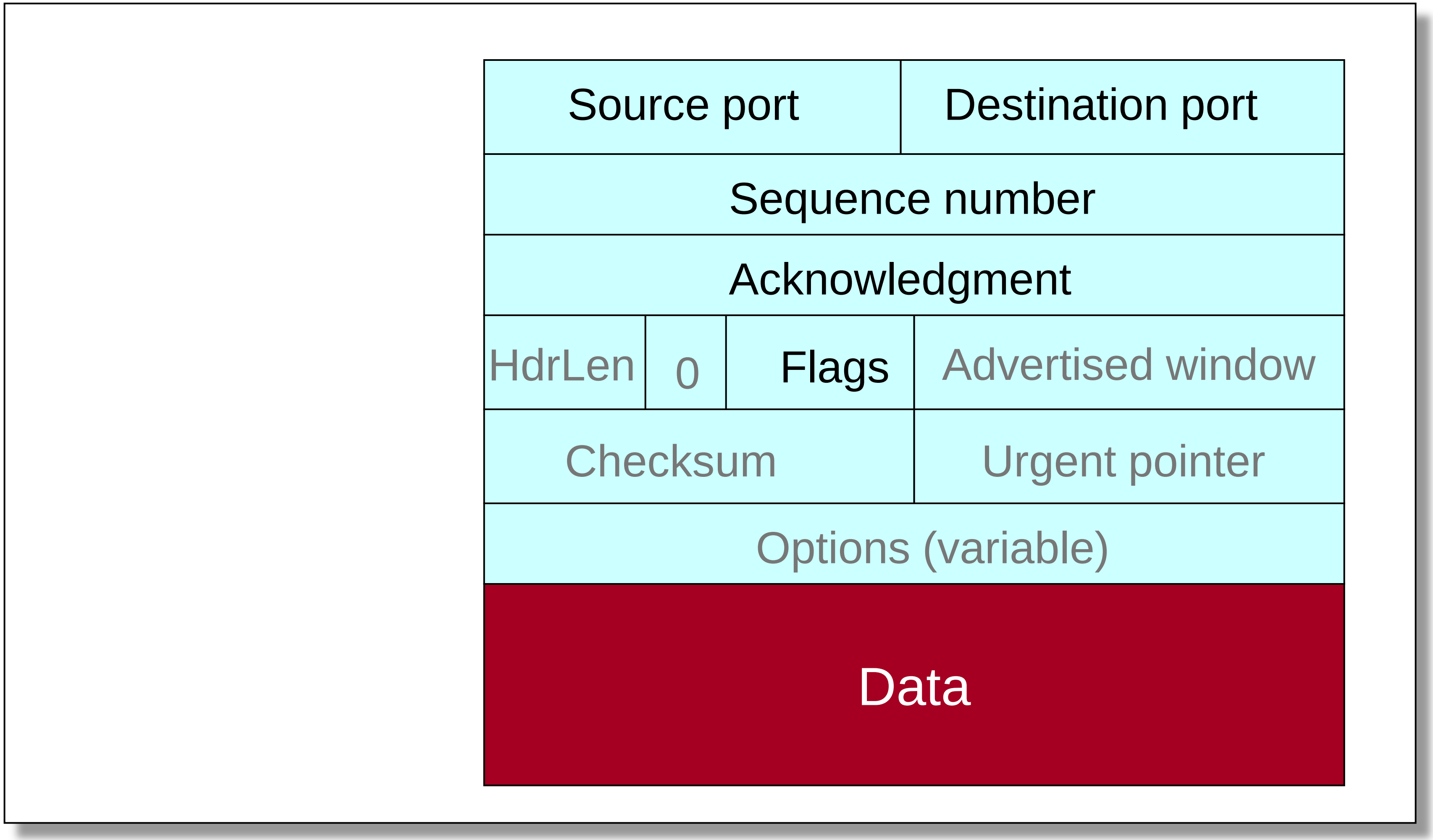


Process B
on host H2

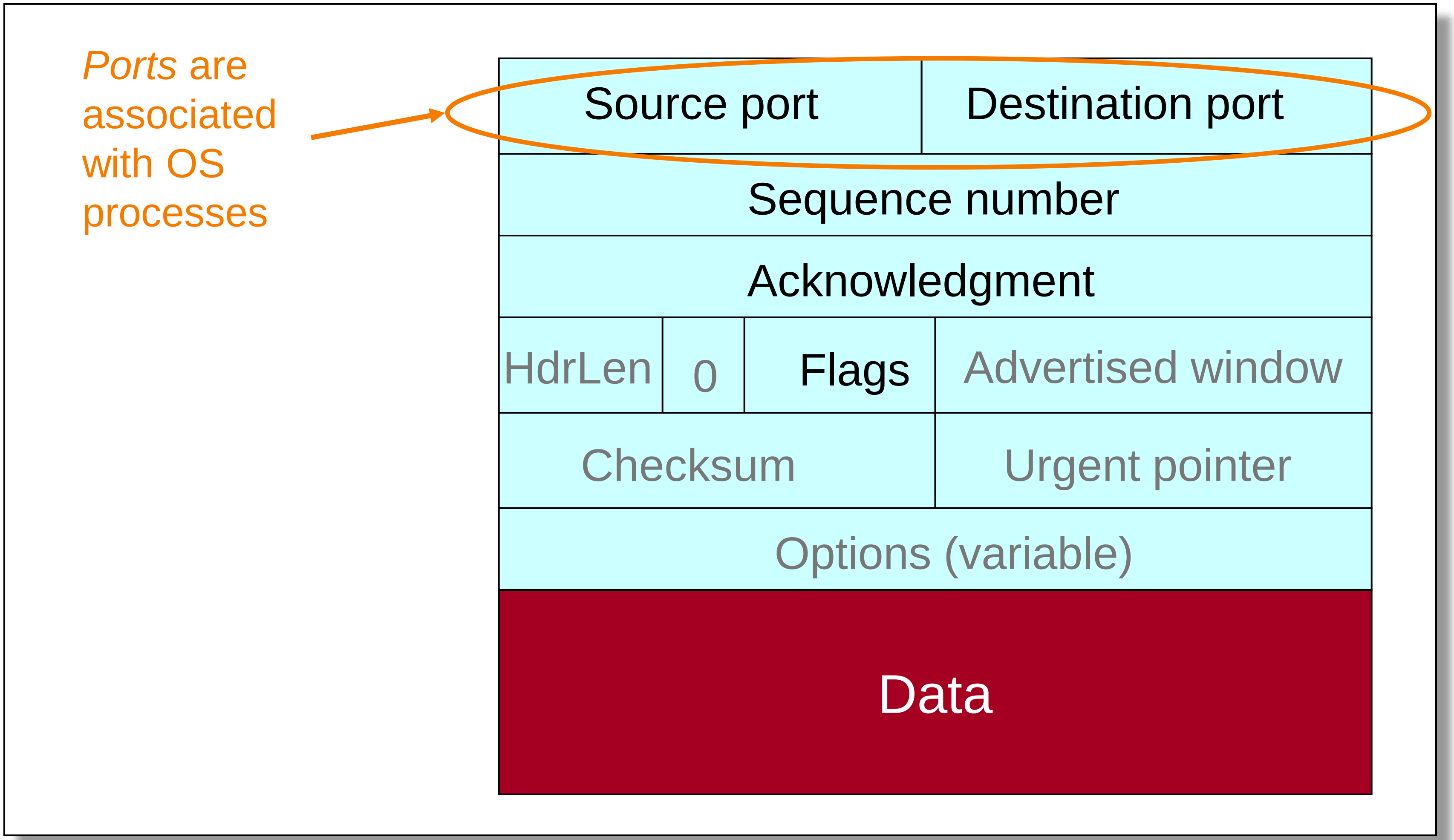
“Best Effort” is Lame! What to do?

- It’s the job of our Transport (layer 4) protocols to build services our apps need out of IP’s modest layer-3 service
- #1 workhorse: TCP (Transmission Control Protocol)
- TCP service:
  - Connection oriented (explicit set-up / tear-down)
    - End hosts (processes) can have multiple concurrent long-lived dialog
  - Reliable, in-order, byte-stream delivery
    - Robust detection & retransmission of lost data
  - Congestion control
    - Dynamic adaptation to network path’s capacity

TCP Header


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

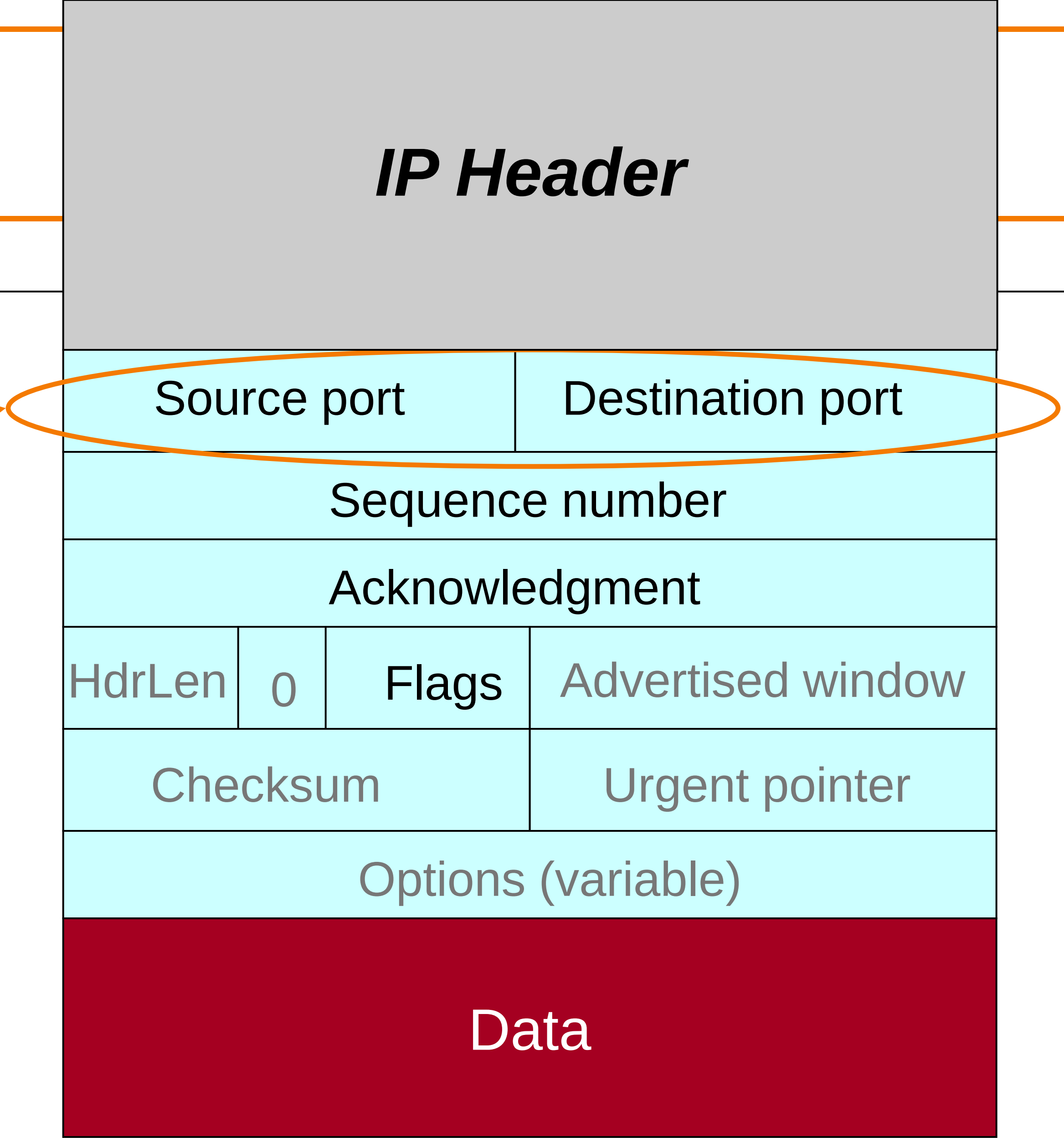
TCP Header



Ports are associated with OS processes
Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data

TCP Header

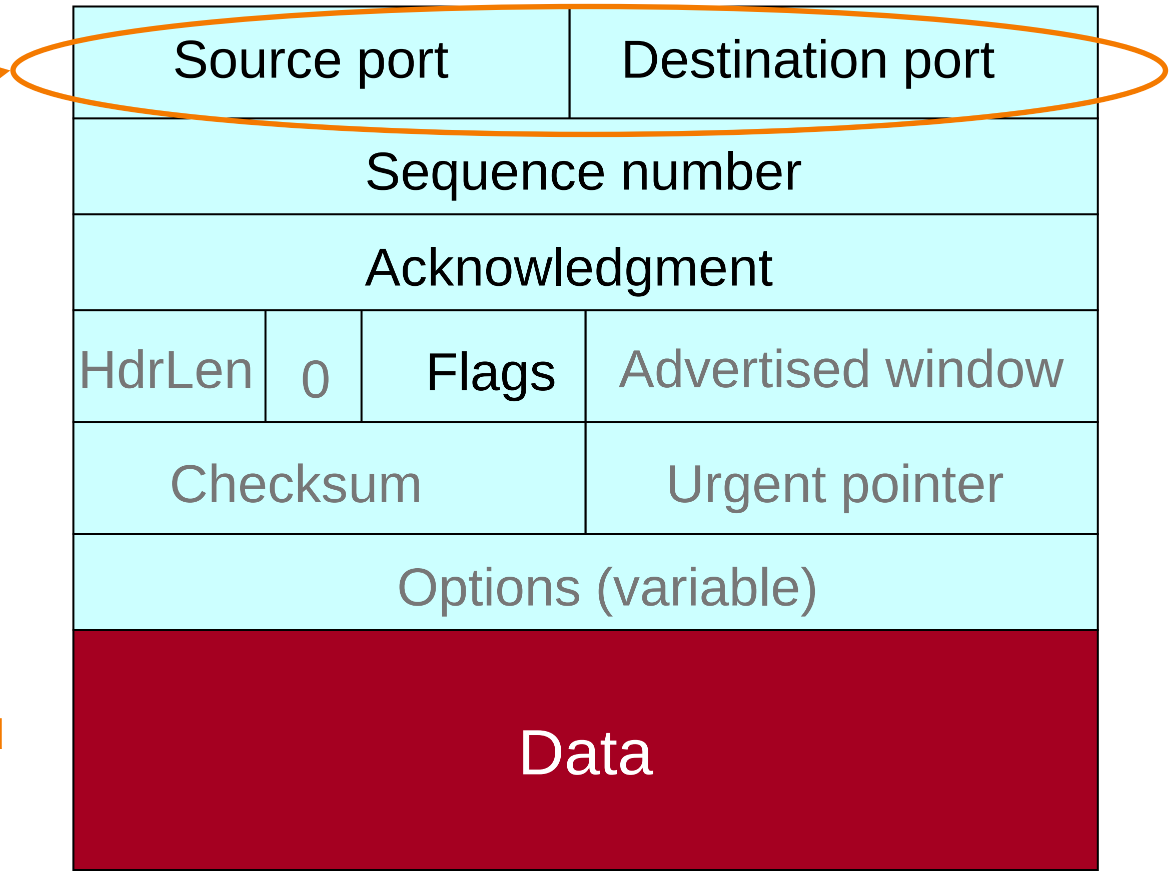
Ports are associated with OS processes



IP source & destination addresses plus TCP source and destination ports uniquely identifies a TCP connection

TCP Header

Ports are associated with OS processes

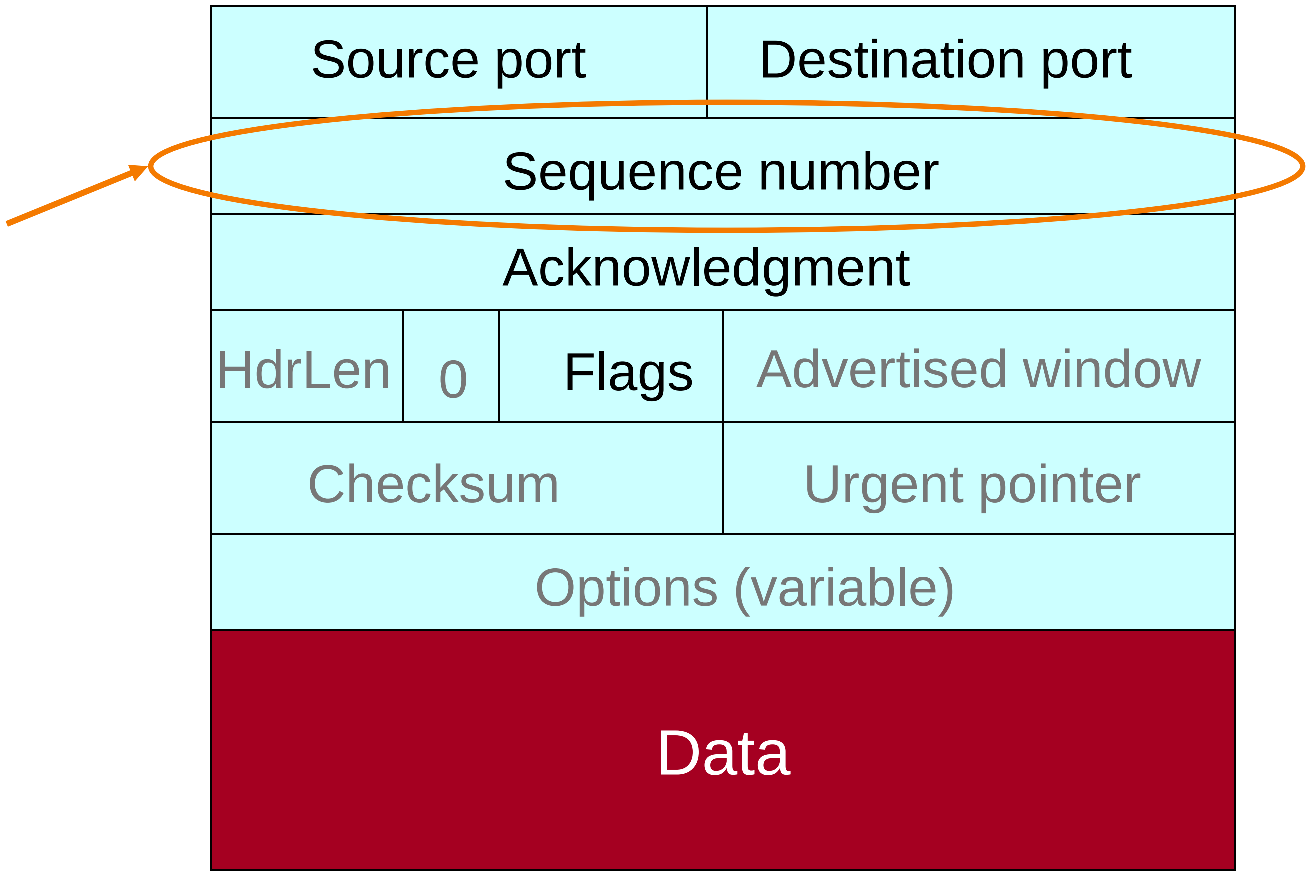


IP source & destination addresses plus TCP source and destination ports uniquely identifies a TCP connection

Some port numbers are "well known" / reserved e.g. port 80 = HTTP

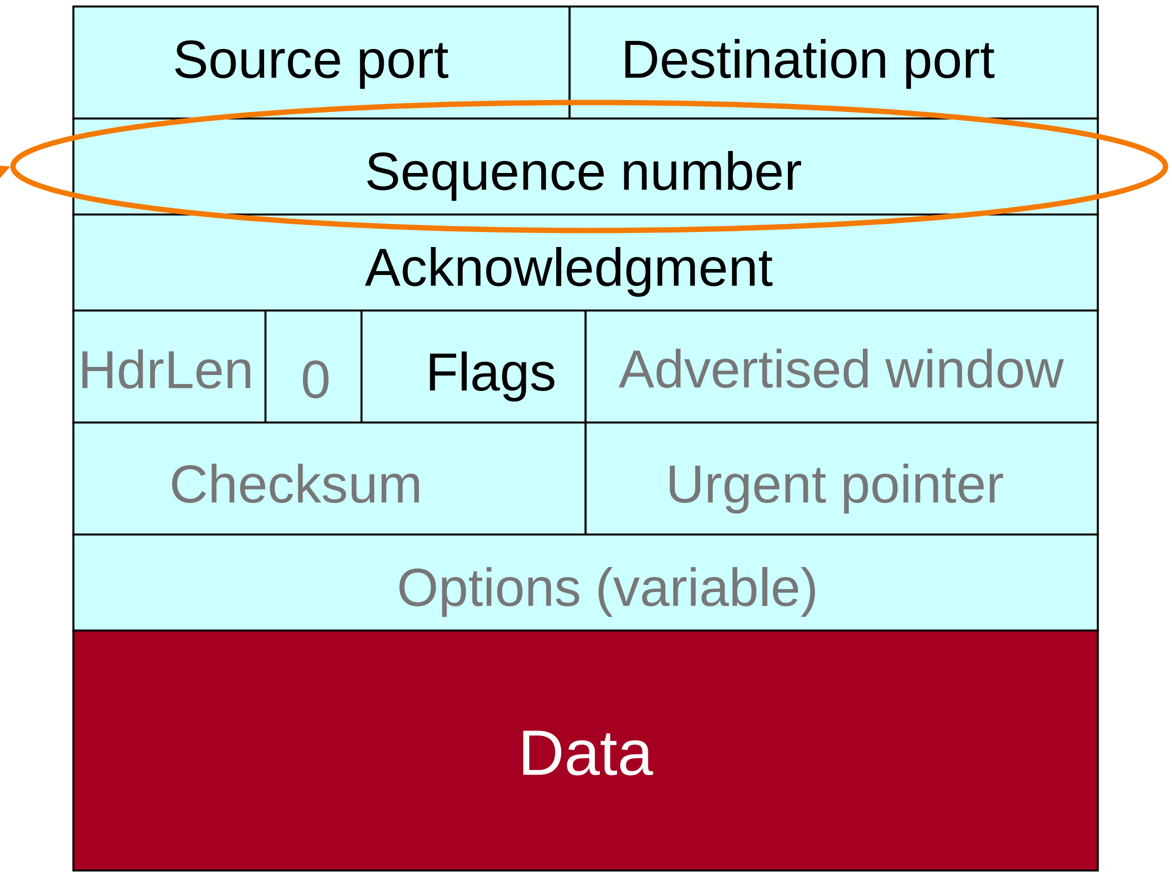
TCP Header

Starting sequence number (byte offset) of data carried in this packet



TCP Header

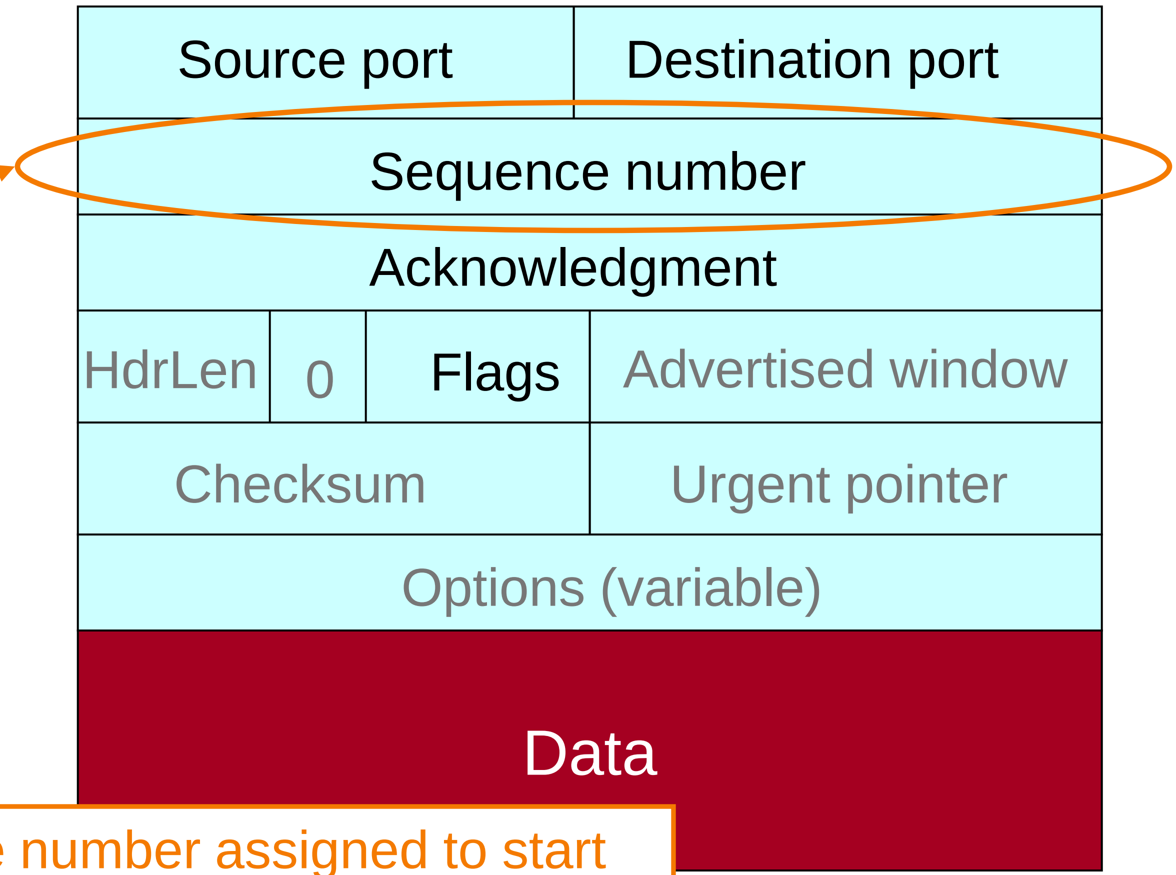
Starting sequence number (byte offset) of data carried in this packet



Byte stream numbered independently in each direction

TCP Header

Starting sequence number (byte offset) of data carried in this packet



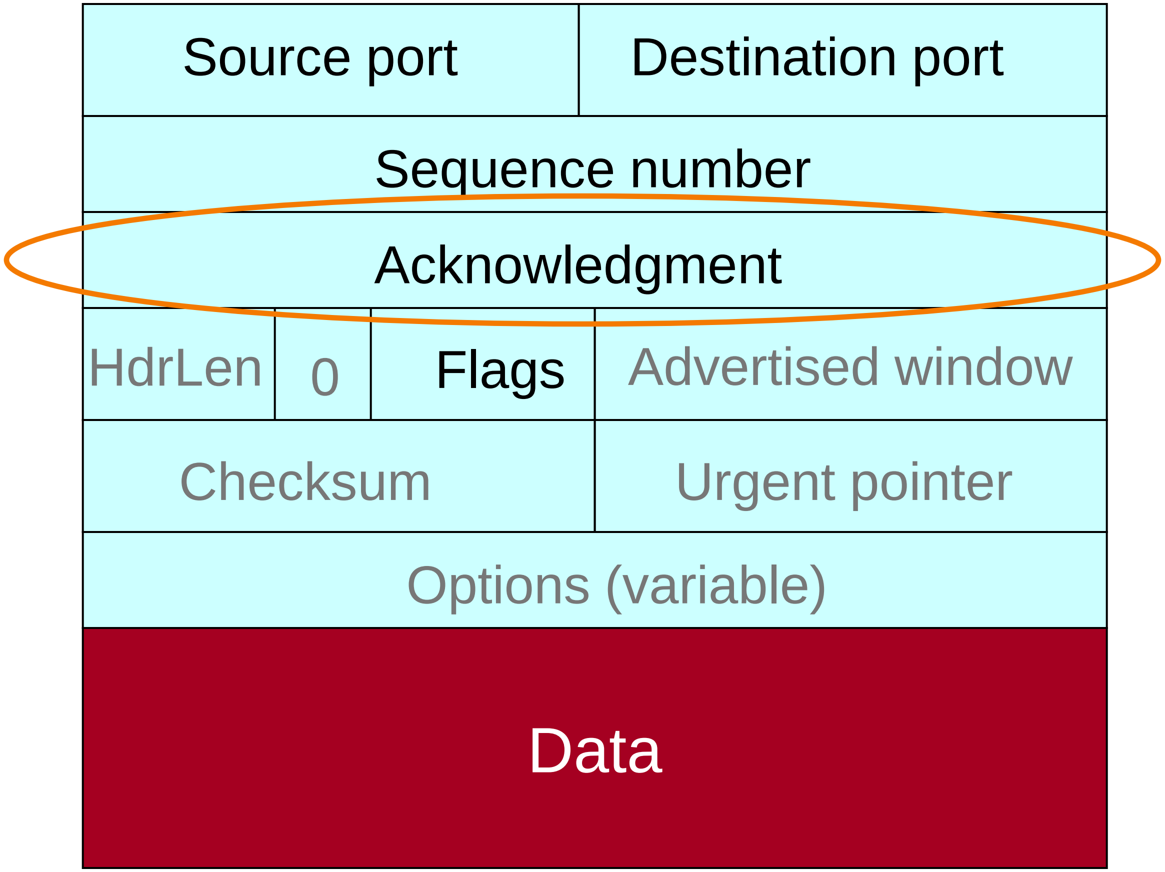
Byte stream numbered independently in each direction

Sequence number assigned to start of byte stream is picked when connection begins; **doesn't** start at 0

TCP Header

Acknowledgment gives seq # **just beyond** highest seq. received **in order**.

If sender sends **N** in-order bytes starting at seq **S** then ack for it will be **S+N**.

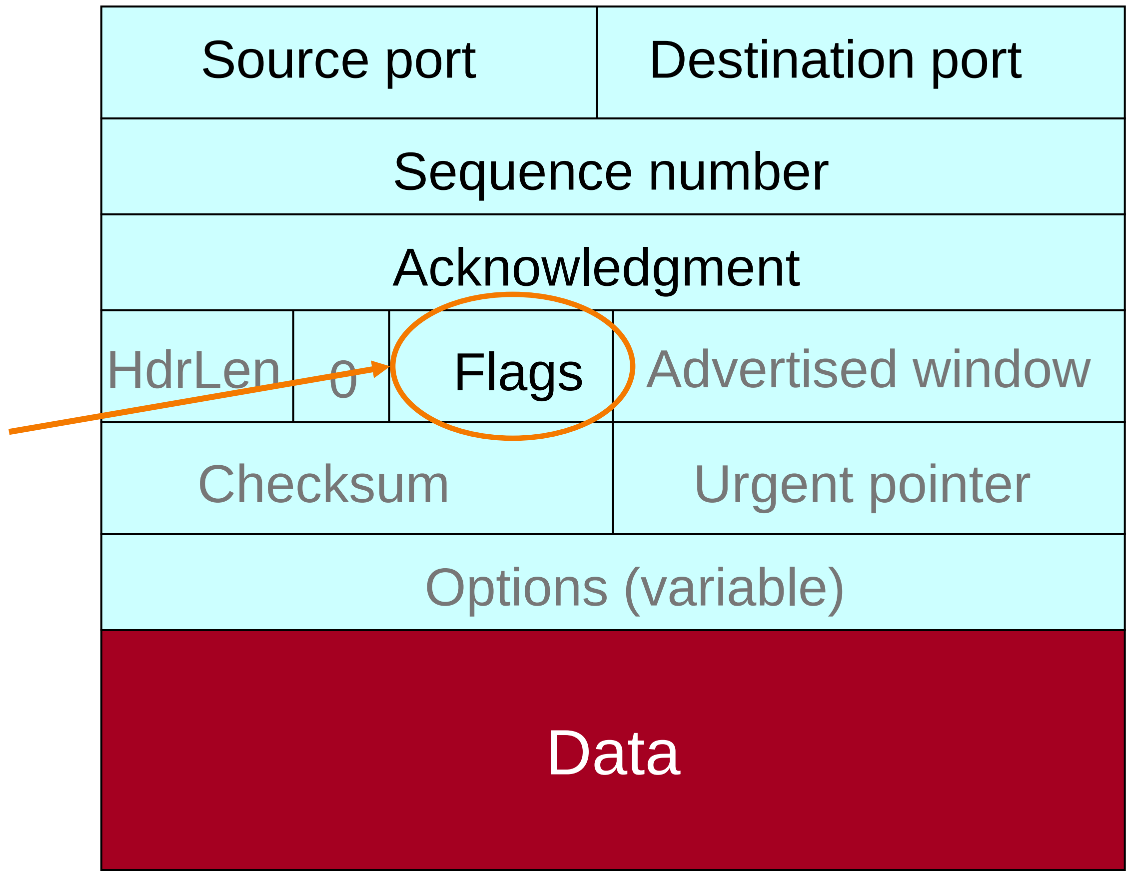


TCP Header

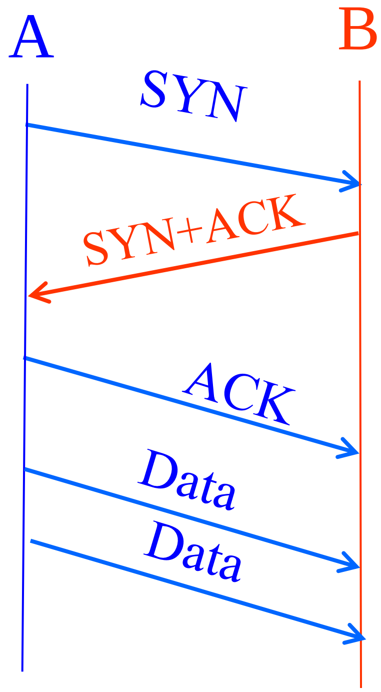
Uses include:

acknowledging data ("**ACK**")

setting up ("**SYN**") and closing connections ("**FIN**" and "**RST**")



Establishing a TCP Connection

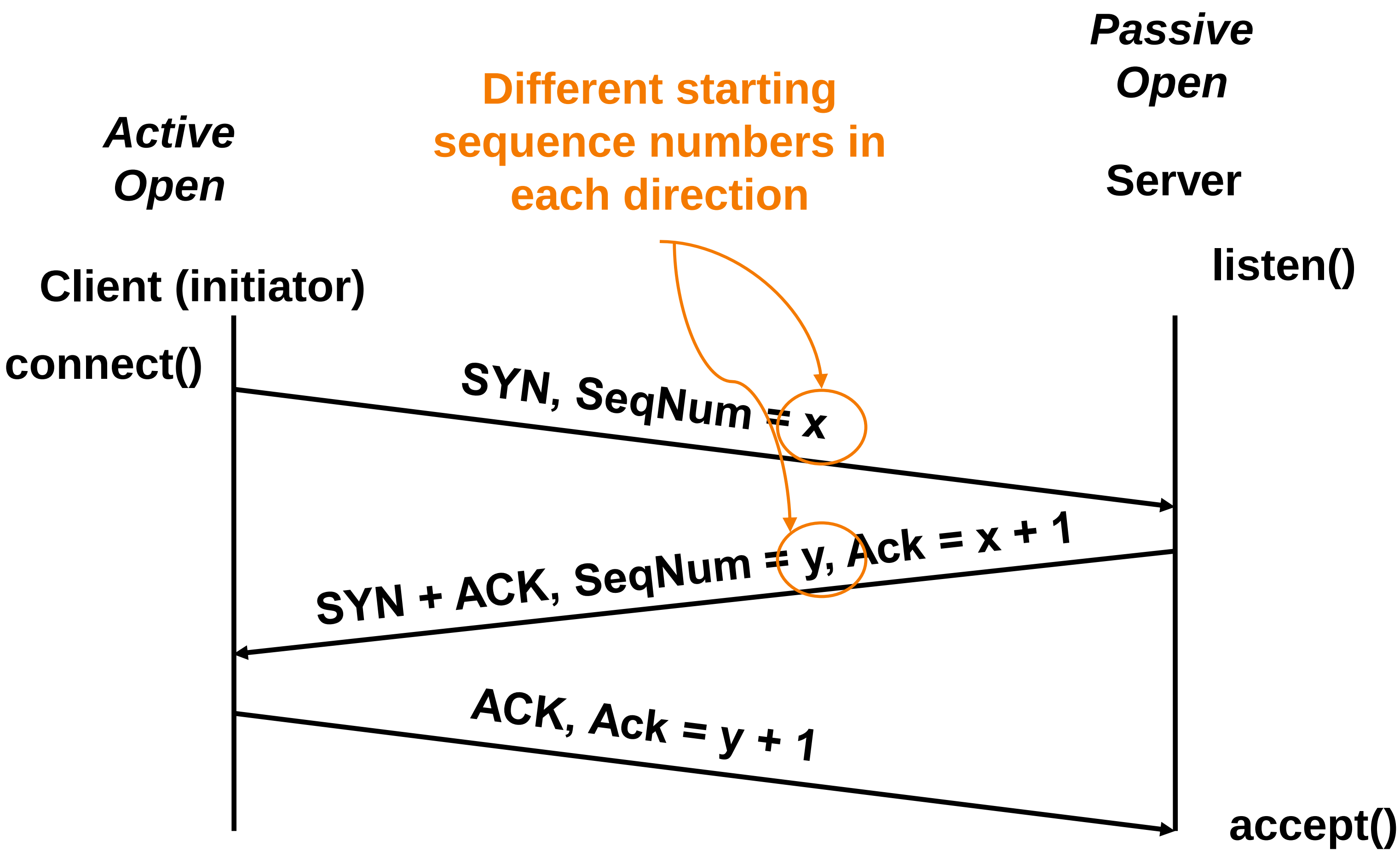


Each host tells its *Initial Sequence Number* (ISN) to the other host.

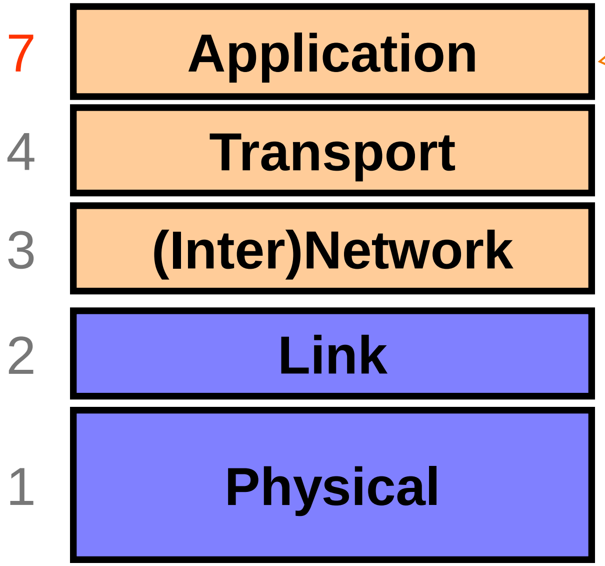
(Spec says to pick based on local clock)

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

Timing Diagram: 3-Way Handshaking



Layer 7: Application Layer



Communication of whatever you wish

Can use whatever transport(s) is convenient

Freely structured

E.g.:

- Skype, SMTP (email),
- HTTP (Web), Halo, BitTorrent

Sample Email (SMTP) interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: From: alice@crepes.fr
C: To: hamburger-list@burger-king.com
C: Subject: Do you like ketchup?
C:
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Email header

Email body

Lone period marks end of message

Web (HTTP) Request

Method **Resource** **HTTP version** **Headers**

```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Mozilla/1.22 (compatible; MSIE 2.0; Windows 95)
Host: www.example.com
Referer: http://www.google.com?q=dingbats

```

Blank line

Data (if POST; none for GET)

GET: download data. POST: upload data.

Web (HTTP) Response

HTTP version **Status code** **Reason phrase** **Headers** **Data**

```
HTTP/1.0 200 OK
Date: Sun, 19 Apr 2009 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Sat, 18 Apr 2009 17:39:05 GMT
Set-Cookie: session=44eb; path=/servlets
Content-Length: 2543

<HTML> Some data... blah, blah, blah </HTML>
```

Questions?

Host Names vs. IP addresses

- Host names
  - Examples: `www.cnn.com` and `bbc.co.uk`
  - Mnemonic name appreciated by humans
  - Variable length, full alphabet of characters
  - Provide little (if any) information about location
- IP addresses
  - Examples: `64.236.16.20` and `212.58.224.131`
  - Numerical address appreciated by routers
  - Fixed length, binary number
  - Hierarchical, related to host location

Mapping Names to Addresses

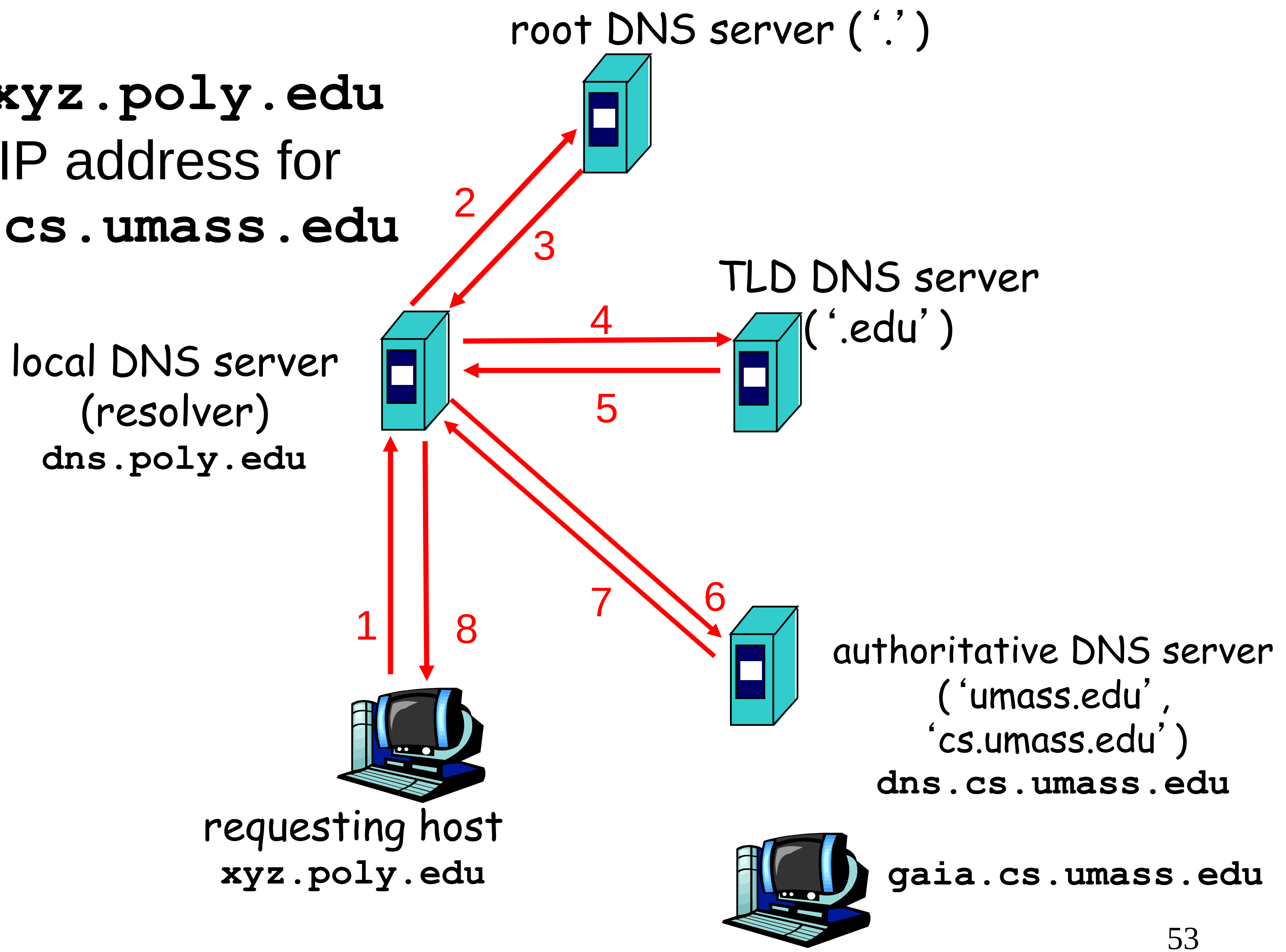
- Domain Name System (DNS)
  - Hierarchical name space divided into zones
  - Zones distributed over collection of DNS servers
  - (Also separately maps addresses to names)
- Hierarchy of DNS servers
  - Root (hardwired into other servers)
  - Top-level domain (TLD) servers
  - “Authoritative” DNS servers (e.g. for *berkeley.edu*)

Mapping Names to Addresses

- Domain Name System (DNS)
  - Hierarchical name space divided into zones
  - Zones distributed over collection of DNS servers
  - (Also separately maps addresses to names)
- Hierarchy of DNS servers
  - Root (hardwired into other servers)
  - Top-level domain (TLD) servers
  - "Authoritative" DNS servers (e.g. for *berkeley.edu*)
- Performing the translations
  - Each computer configured to contact a *resolver*

Example

Host at **`xyz.poly.edu`** wants IP address for **`gaia.cs.umass.edu`**



DNS Protocol

DNS protocol: *query* and *reply* messages, both with same *message format*

(Mainly uses UDP transport rather than TCP)

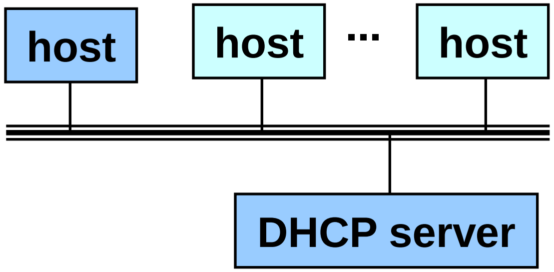
Message header:

- Identification: 16 bit # for query, reply to query uses same #
- Replies can include "Authority" (name server responsible for answer) and "Additional" (info client is likely to look up soon anyway)
- Replies have a Time To Live (in seconds) for **caching**

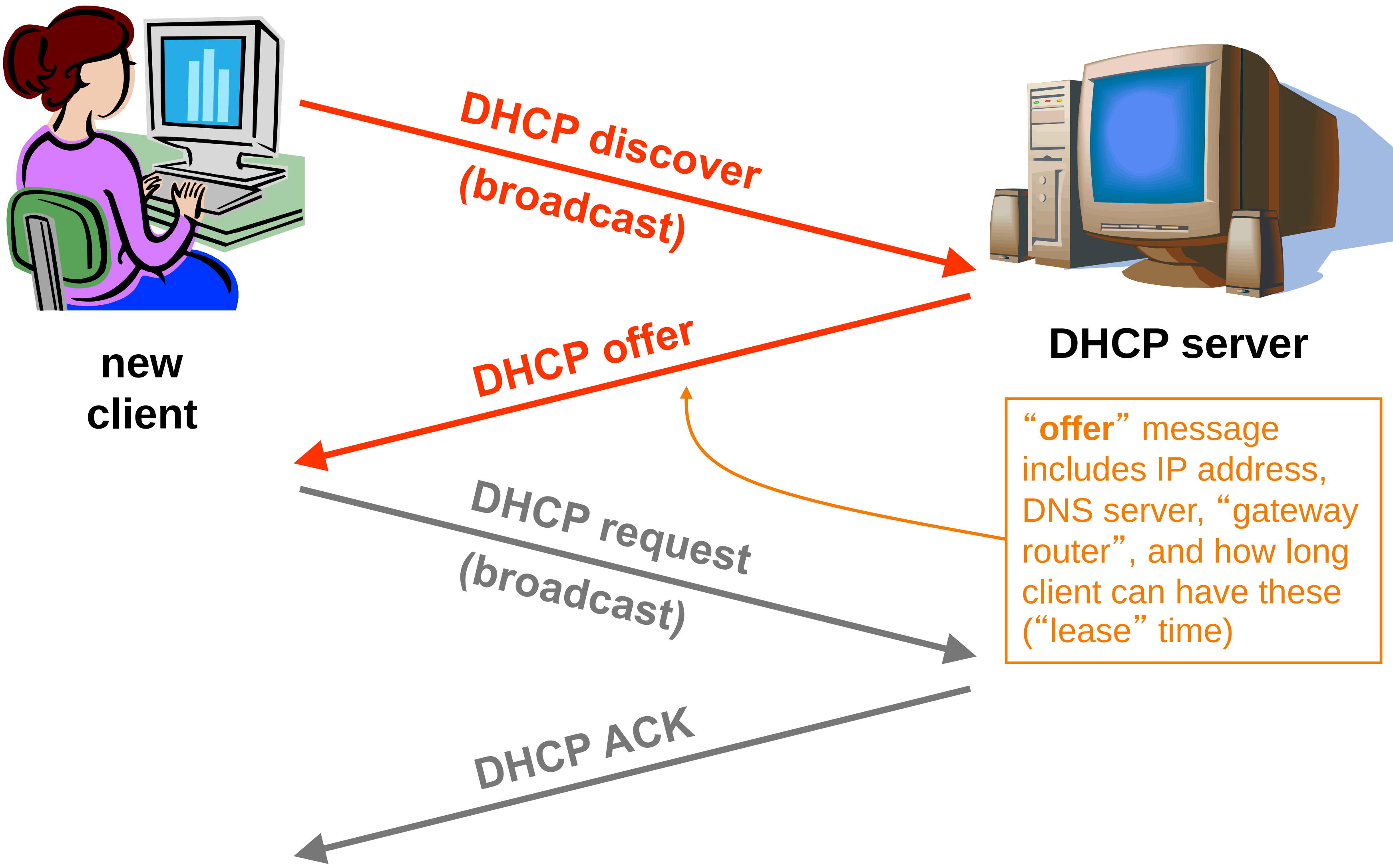
16 bits	*16 bits*
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

Bootstrapping Problem

- New host doesn't have an IP address yet
  - So, host doesn't know what source address to use
- Host doesn't know *who to ask* for an IP address
  - So, host doesn't know what destination address to use
- Solution: shout to "**discover**" server that can help
  - Broadcast a server-discovery message (layer 2)
  - Server(s) sends a reply offering an address



Dynamic Host Configuration Protocol

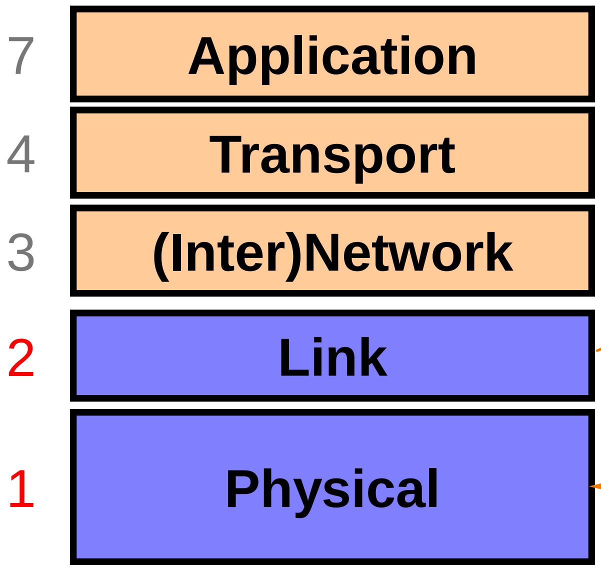


Questions?

Security Issues

Layer 1,2 Threats

Layers 1 & 2: General Threats?



Framing and transmission of a collection of bits into individual messages sent across a single "subnetwork" (one physical technology)

Encoding bits to send them over a single **physical link**

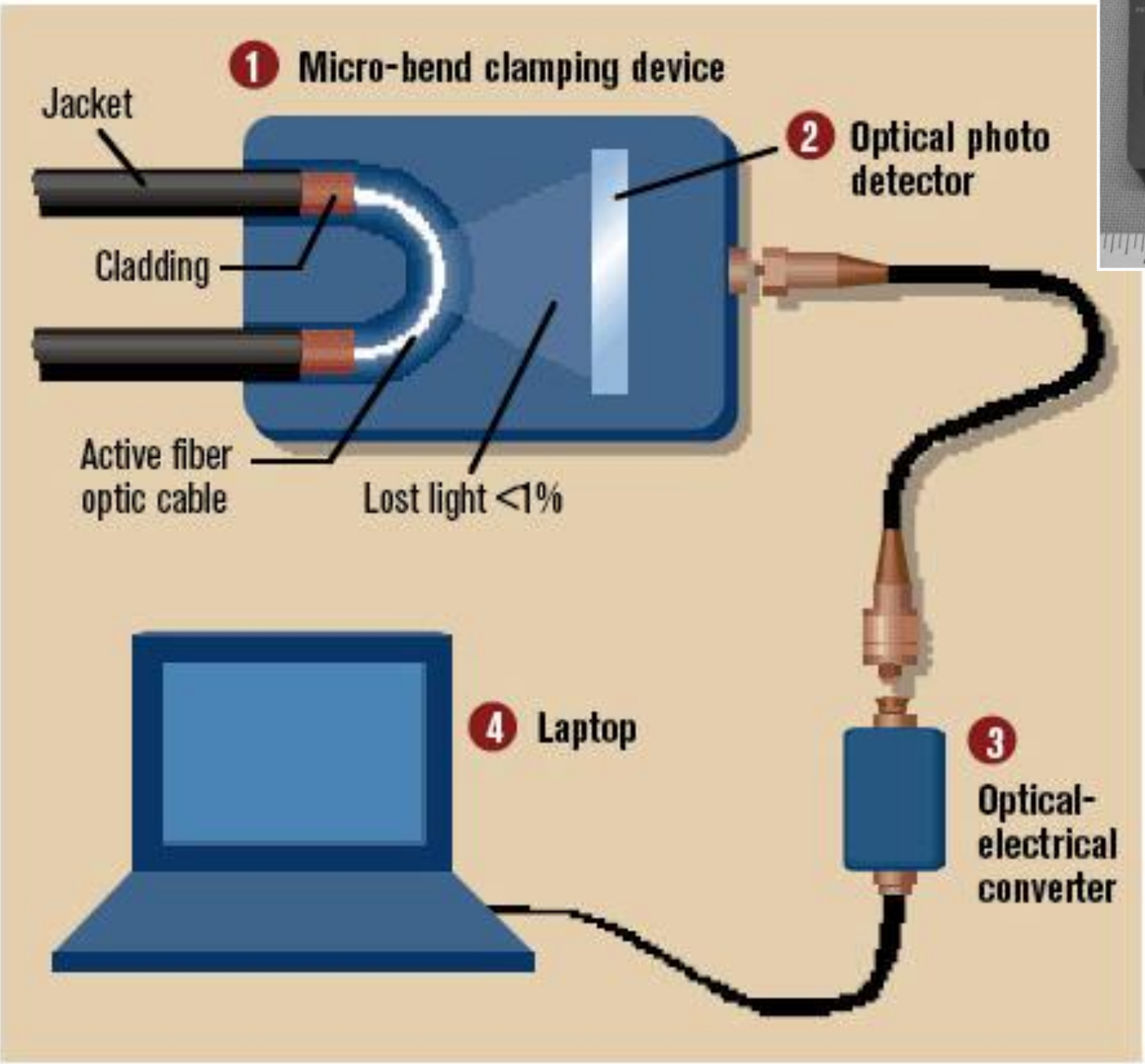
e.g. patterns of

voltage levels /
photon intensities /
RF modulation

Physical/Link-Layer Threats: *Eavesdropping*

- Also termed *sniffing*
- For subnets using broadcast technologies (e.g., WiFi, some types of Ethernet), get it for "free"
  - Each attached system's NIC (= Network Interface Card) can capture any communication on the subnet
  - Some handy tools for doing so
    - o Wireshark
    - o tcpdump / windump
    - o bro
- For any technology, routers (and internal "switches") can look at / export traffic they forward
- You can also "tap" a link
  - Insert a device to mirror physical signal

Stealing Photons



Operation Ivy Bells

By Matthew Carle
Military.com

The Regulus guided missile submarine, USS Halibut (SSN 587) which carried out Operation Ivy Bells.

At the beginning of the 1970's, divers from the specially-equipped submarine, USS Halibut (SSN 587), left their decompression chamber to start a bold and dangerous mission, code named "Ivy Bells".

In an effort to alter the balance of Cold War, these men scoured the ocean floor for a five-inch diameter cable carry secret Soviet communications between military bases.

The divers found the cable and installed a 20-foot long listening device on the cable. designed to attach to the cable without piercing the casing, the device recorded all communications that occurred. If the cable malfunctioned and the Soviets raised it for repair, the bug, by design, would fall to the bottom of the ocean. Each month Navy divers retrieved the recordings and installed a new set of tapes.

Upon their return to the United States, intelligence agents from the NSA analyzed the recordings and tried to decipher any encrypted information. The Soviets apparently were confident in the security of their communications lines, as a surprising amount of sensitive information traveled through the lines without encryption.

prison. The original tap that was discovered by the Soviets is now on exhibit at the KGB museum in Moscow.

Physical/Link-Layer Threats: *Disruption*

- With physical access to a subnetwork, attacker can
  - Overwhelm its signaling
    - E.g., jam WiFi's RF
  - Send messages that violate the Layer-2 protocol's rules
    - E.g., send messages > maximum allowed size, sever timing synchronization, ignore fairness rules
- Routers & switches can simply "drop" traffic
- There's also the heavy-handed approach …

Sabotage attacks knock out phone service

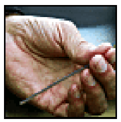
Nanette Asimov, Ryan Kim,Kevin Fagan, Chronicle Staff Writers
Friday, April 10, 2009

PRINT E-MAIL SHARE COMMENTS (477) FONT | SIZE:

(04-10) 04:00 PDT SAN JOSE --

Police are hunting for vandals who chopped fiber-optic cables and killed landlines, cell phones and Internet service for tens of thousands of people in Santa Clara, Santa Cruz and San Benito counties on Thursday.

IMAGES


View More Images

MORE NEWS

- Toyota seeks damage control, in public and private 02.09.10
- Snow shuts down federal government, life goes on 02.09.10
- Iran boosts nuclear enrichment, drawing warnings 02.09.10

The sabotage essentially froze operations in parts of the three counties at hospitals, stores, banks and police and fire departments that rely on 911 calls, computerized medical records, ATMs and credit and debit cards.

The full extent of the havoc might not be known for days, emergency officials said as they finished repairing the damage late Thursday.

Whatever the final toll, one thing is certain: Whoever did this is in a world of trouble if he, she or they get caught.

"I pity the individuals who have done this," said San Jose Police Chief Rob Davis.

Ten fiber-optic cables carrying were cut at four locations in the predawn darkness. Residential and business customers quickly found that telephone service was perhaps more laced into their everyday needs than they thought. Suddenly they couldn't draw out money, send text messages, check e-mail or Web sites, call anyone for help, or even check on friends or relatives down the road.

Several people had to be driven to hospitals because they were unable to summon ambulances. Many businesses lapsed into idleness for hours, without the ability to contact associates or customers.

More than 50,000 landline customers lost service - some were residential, others were business lines that needed the connections for ATMs, Internet and bank card transactions. One line alone could affect hundreds of users.

UPDATE LOOK Vancouver's Venues Shine

NEWS | LOCAL BEAT

$250K Reward Out for Vandals Who Cut AT&T Lines

Local emergency declared during outage

By LORI PREUITT

Updated 2:12 PM PST, Fri, Apr 10, 2009

PRINT EMAIL SHARE BUZZ UP! TWITTER FACEBOOK

AT&T is now offering a $250,000 reward for information leading to the arrest of whoever is responsible for severing lines fiber optic cables in San Jose tha left much of the area without phone or cell service Thursday.

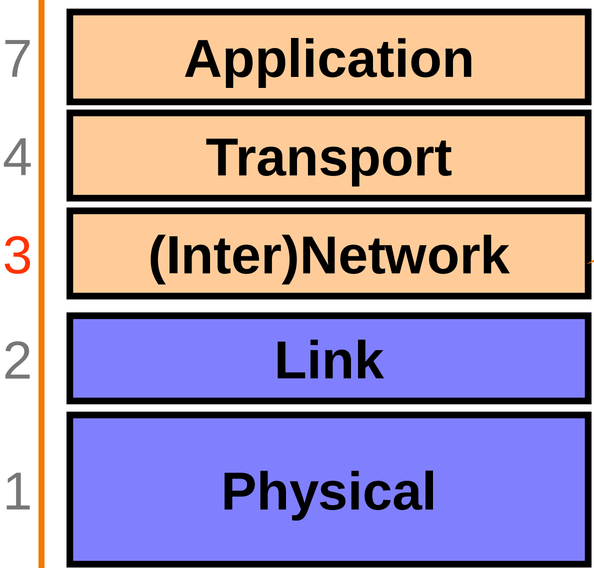
John Britton of AT&T said the reward is the largest ever offered by the company.

Physical/Link-Layer Threats: *Spoofing*

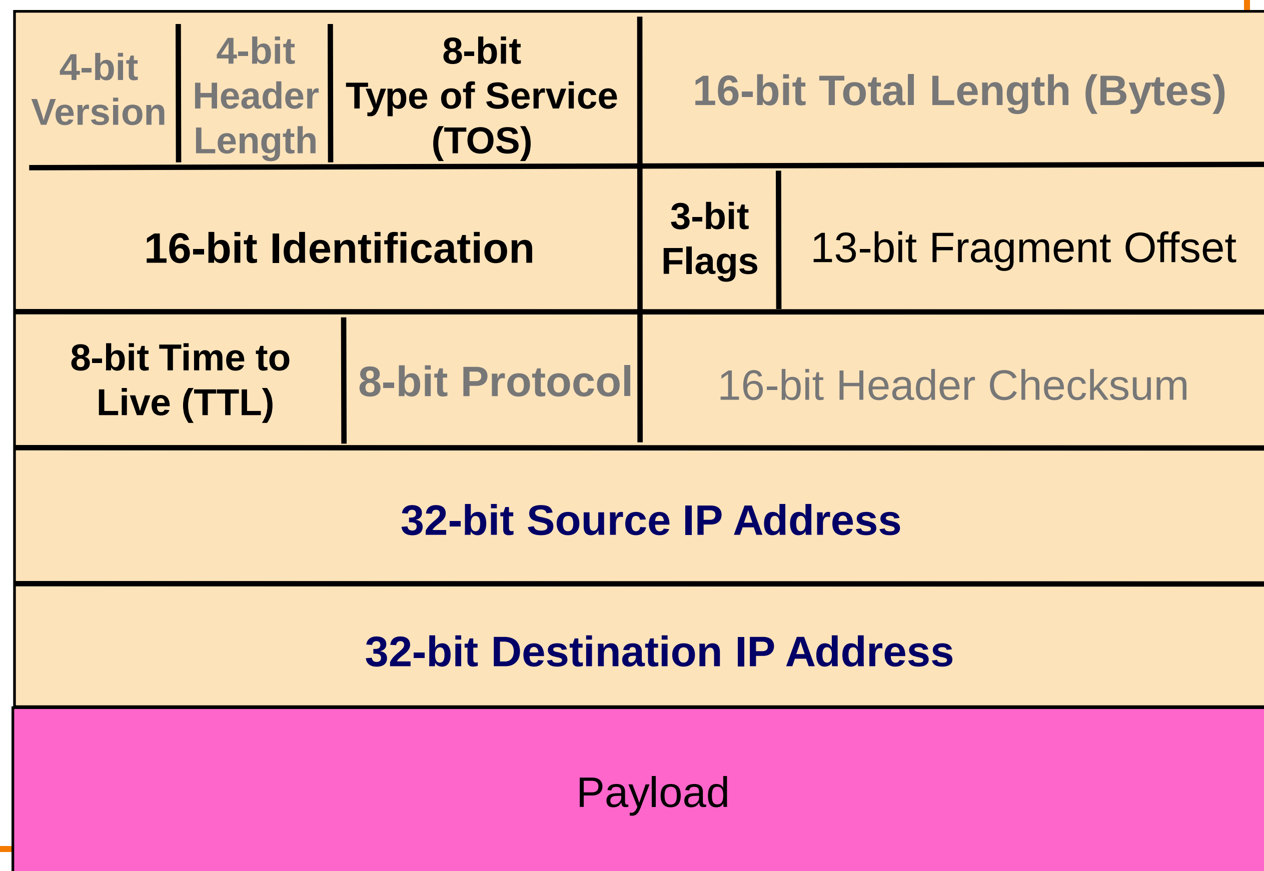
- With physical access to a subnetwork, attacker can create any message they like
  - Termed *spoofing*
- May require root/administrator access to have full freedom
- Particularly powerful when combined with *eavesdropping*
  - Because attacker can understand exact state of victim's communication and craft their spoofed traffic to match it
  - Spoofing w/o eavesdropping = *blind spoofing*

Layer 3 Threats

Layer 3: General Threats?



Bridges multiple "subnets" to provide *end-to-end* internet connectivity between nodes



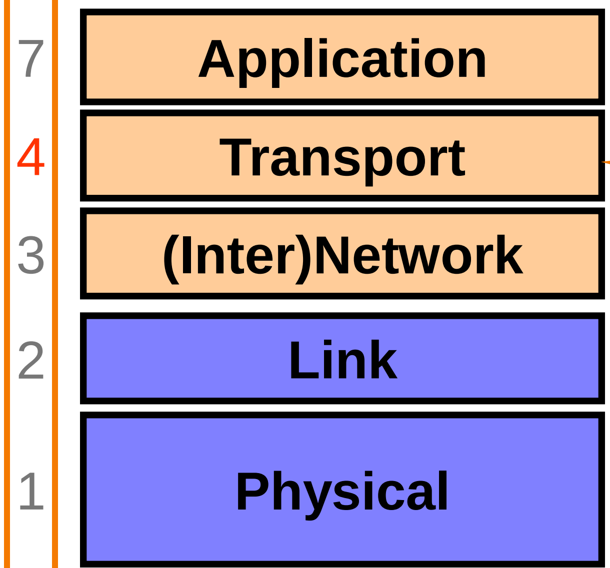
IP = Internet Protocol

Network-Layer Threats

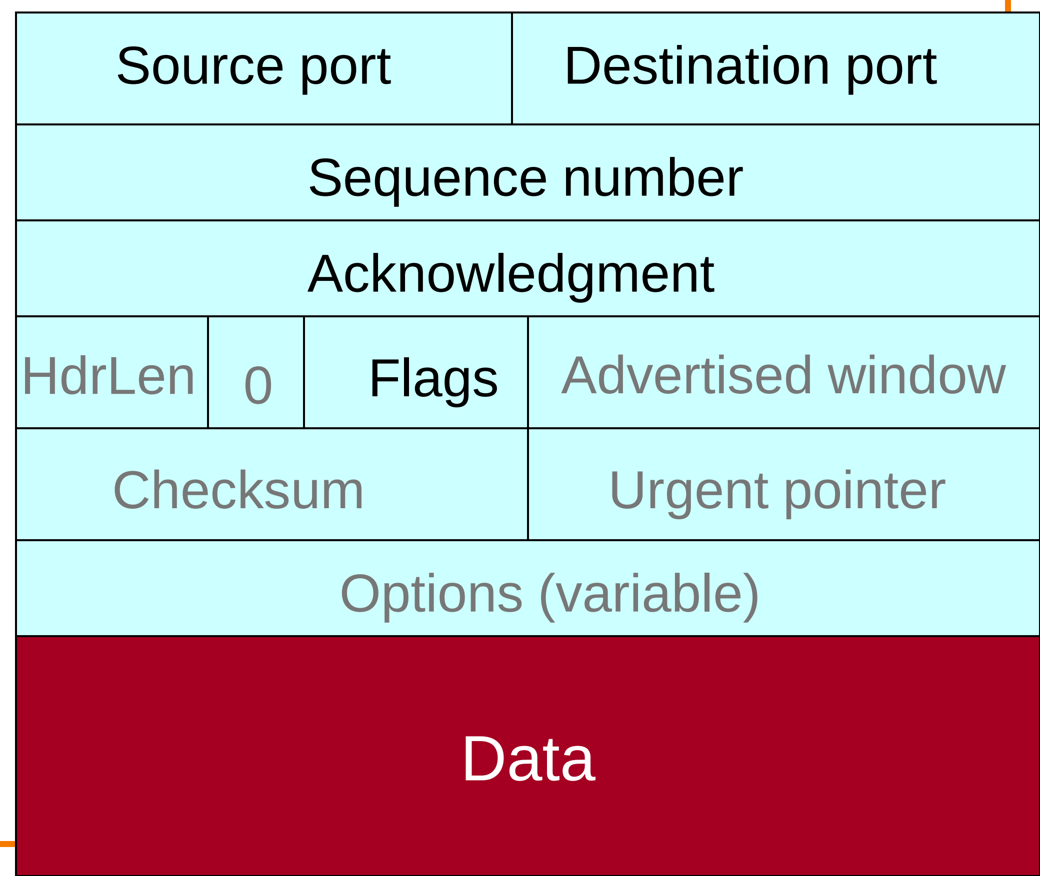
- Major:
  - Can set arbitrary source address
    - “*Spoofing*” - receiver has no idea who you are
    - Could be *blind*, or could be coupled w/ *sniffing*
  - Can set arbitrary destination address
    - Enables “*scanning*” - brute force searching for hosts
- Lesser: **(FYI; don’t worry about unless later explicitly covered)**
  - Fragmentation mechanism can evade network monitoring
  - Identification field leaks information
  - Time To Live allows discovery of topology
  - IP “options” can reroute traffic

Issues with TCP

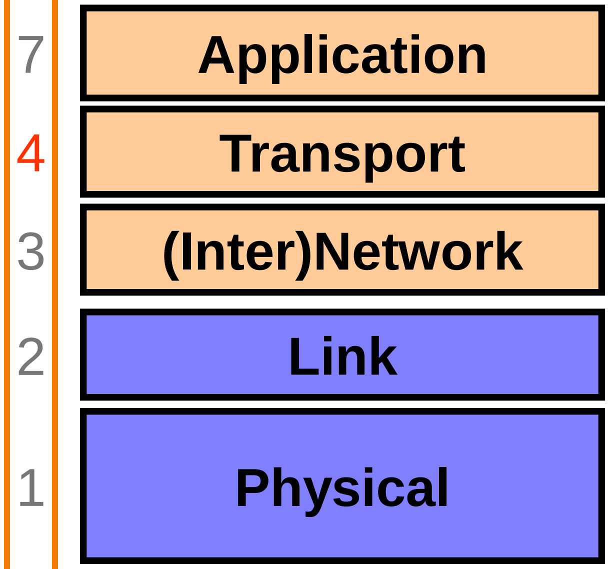
Layer 4: General Threats?



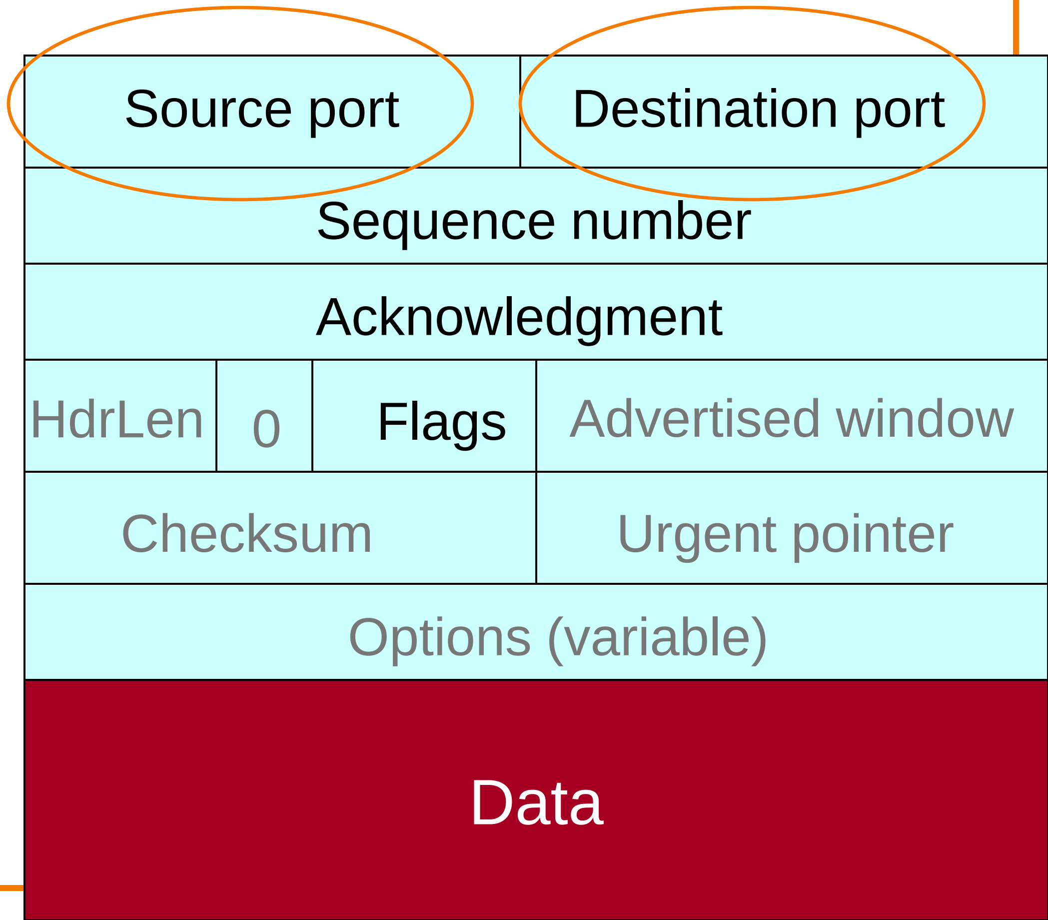
End-to-end communication between processes (TCP, UDP)



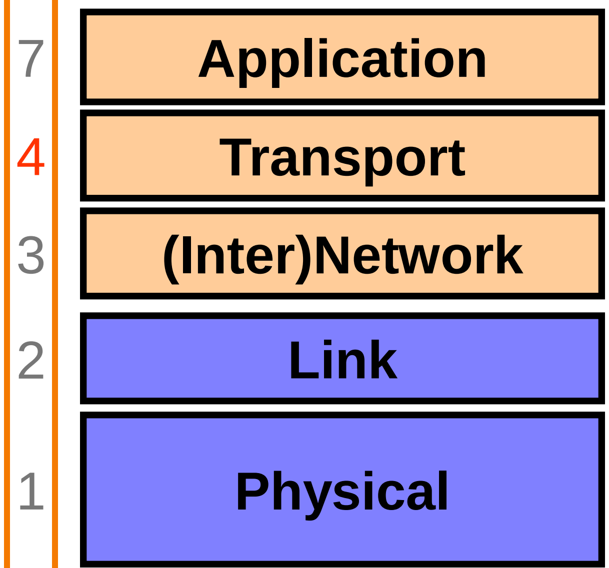
Layer 4: General Threats?



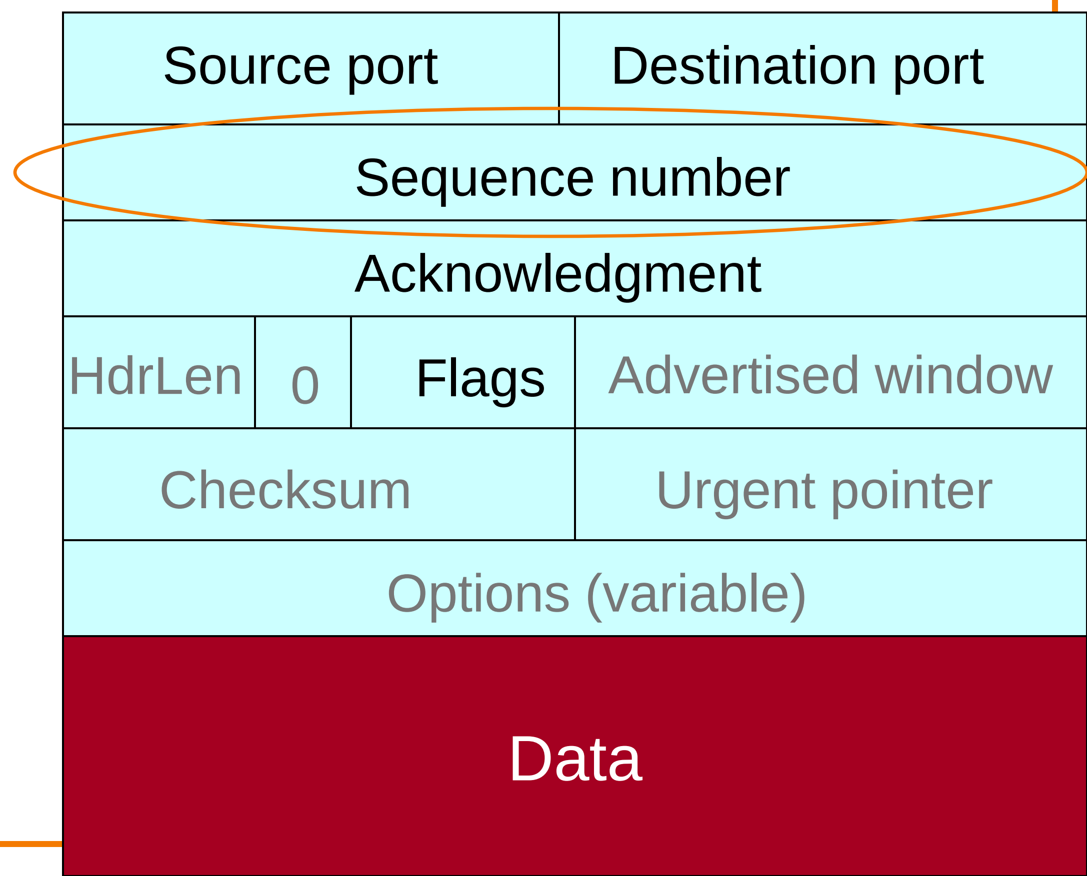
These plus IP addresses define a given connection



Layer 4: General Threats?

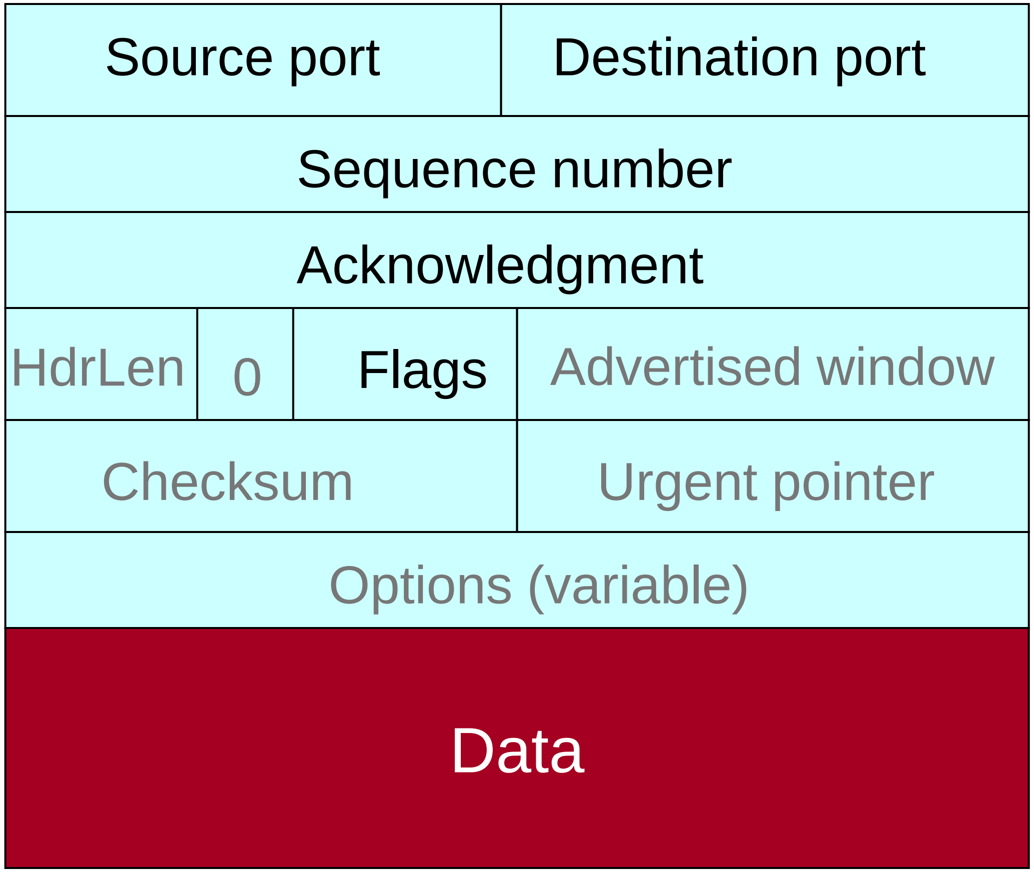


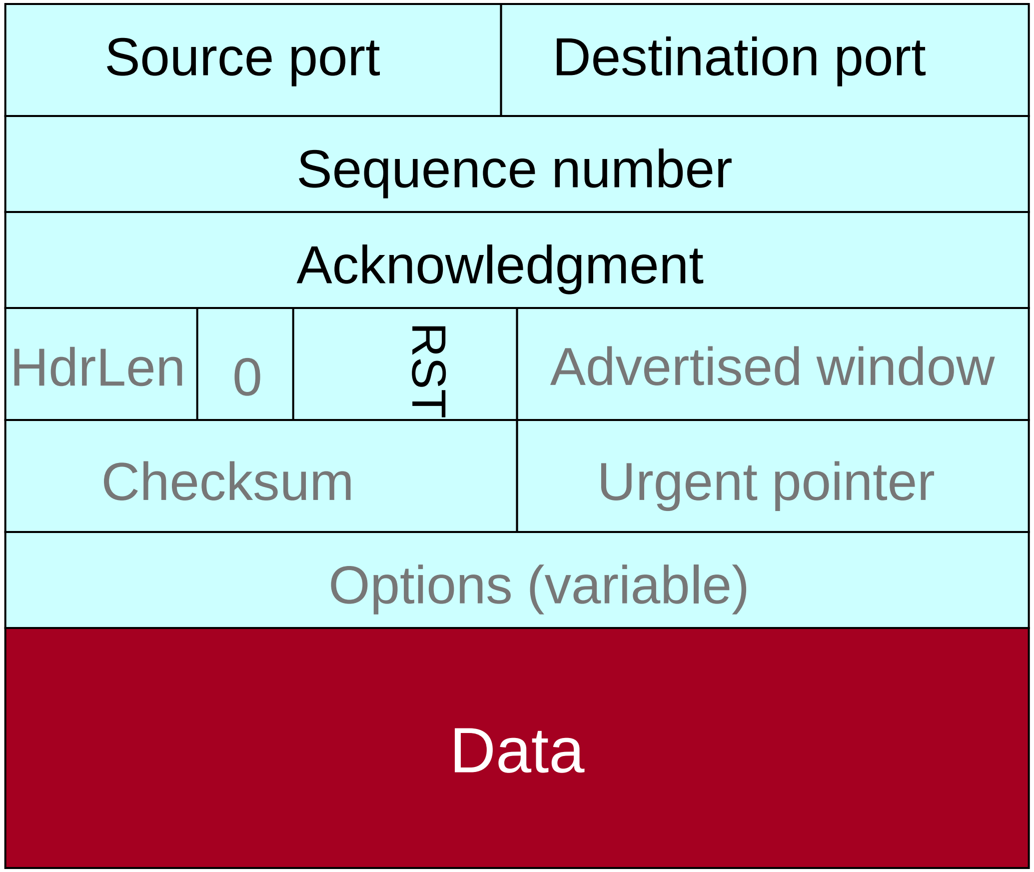
Defines where this packet fits within the sender’s bytestream



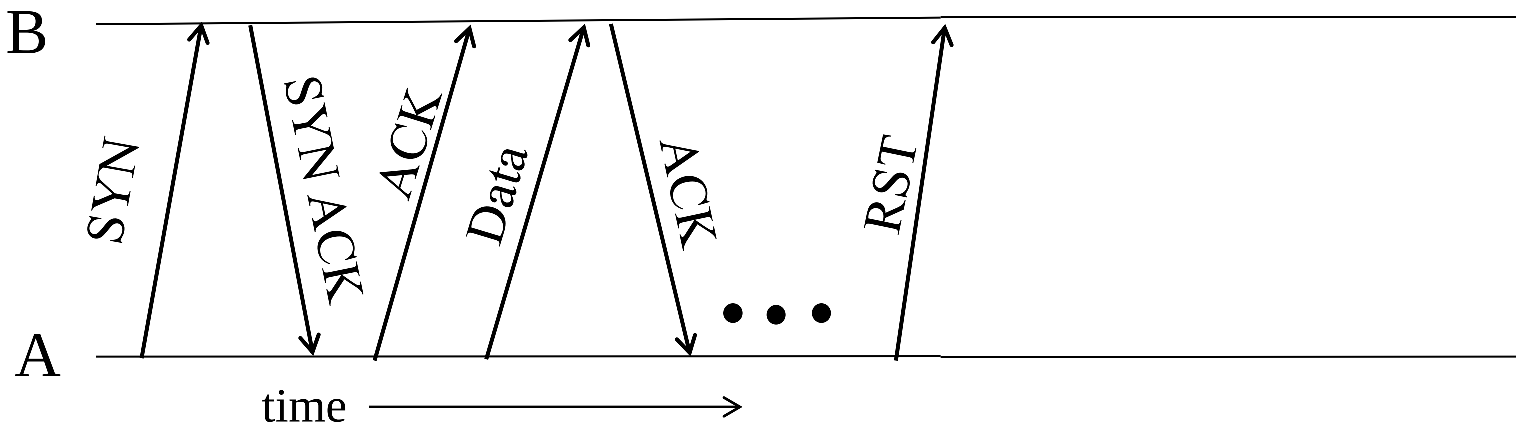
TCP Threat: Disruption

- Normally, TCP finishes ("closes") a connection by each side sending a `FIN` control message
  - Reliably delivered, since other side must ack
- But: if a TCP endpoint finds unable to continue (process dies; info from other "peer" is inconsistent), it abruptly terminates by sending a **`RST`** control message
  - Unilateral
  - Takes effect immediately (no ack needed)
  - Only accepted by peer if has correct* sequence number


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


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

Abrupt Termination

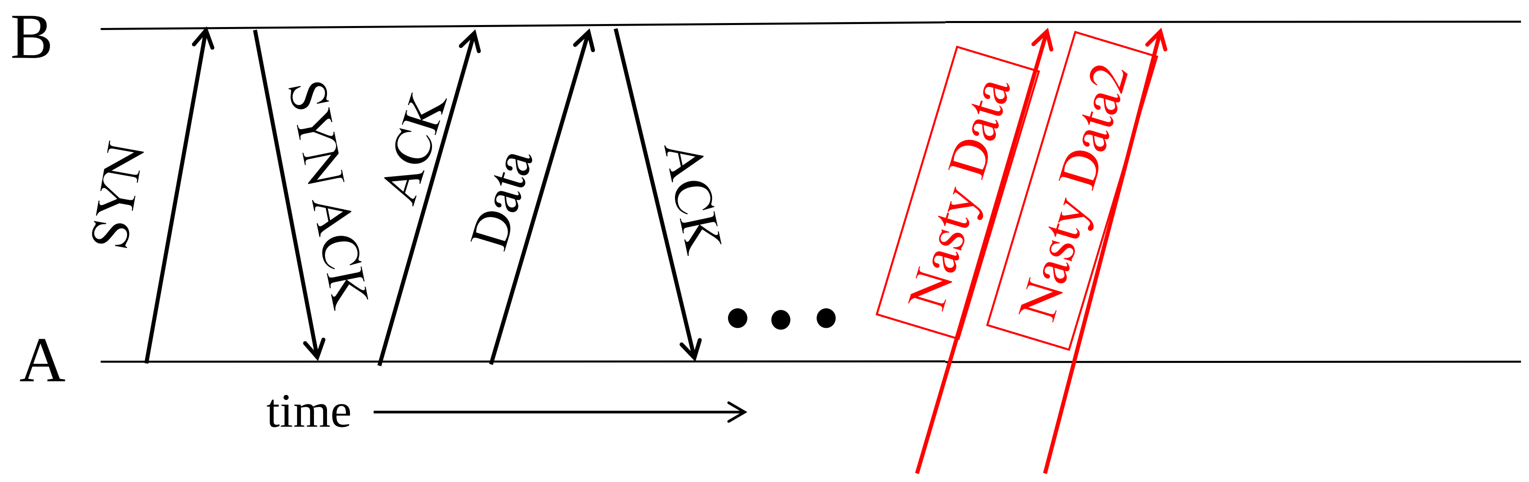


- A sends a TCP packet with RESET (**RST**) flag to B
  - E.g., because app. process on A crashed
- Assuming that the sequence numbers in the **RST** fit with what B expects, That's It:
  - B's user-level process receives: ECONNRESET
  - No further communication on connection is possible

TCP Threat: Disruption

- Normally, TCP finishes ("closes") a connection by each side sending a `FIN` control message
  - Reliably delivered, since other side must ack
- But: if a TCP endpoint finds unable to continue (process dies; info from other "peer" is inconsistent), it abruptly terminates by sending a `RST` control message
  - Unilateral
  - Takes effect immediately (no ack needed)
  - Only accepted by peer if has correct* sequence number
- So: if attacker knows ports & sequence numbers, can disrupt any TCP connection

TCP Threat: Injection



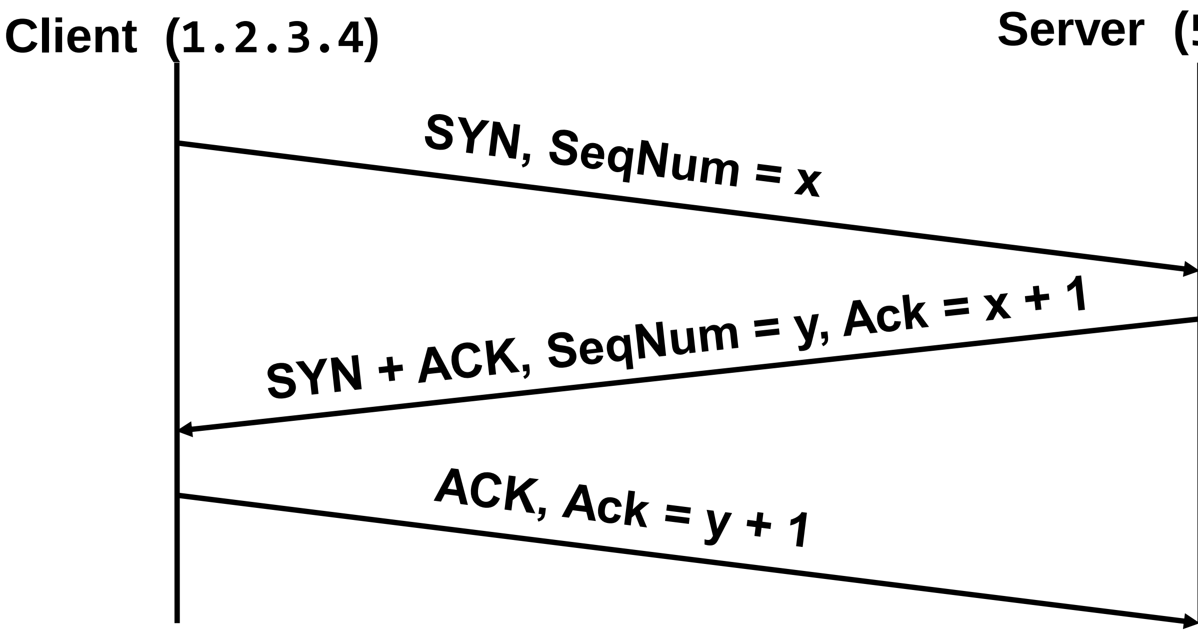
- What about inserting data rather than disrupting a connection?
  - Again, all that's required is attacker knows correct ports, seq. numbers
  - Receiver B is *none the wiser!*
- Termed TCP connection hijacking (or "*session hijacking*")
  - General means to take over an already-established connection!
- We are toast if an attacker can see our TCP traffic!
  - Because then they immediately know the port & sequence numbers

TCP Threat: Blind Spoofing

- Is it possible for an attacker to inject into a TCP connection even if they can't see our traffic?
- YES: if somehow they can guess the port and sequence numbers
- Let's look at a related attack where the goal of the attacker is to create a fake connection, rather than inject into a real one
  - Why?
  - Perhaps to leverage a server's trust of a given client as identified by its IP address
  - Perhaps to frame a given client so the attacker's actions during the connections can't be traced back to the attacker

TCP Threat: Blind Spoofing

- TCP connection establishment:



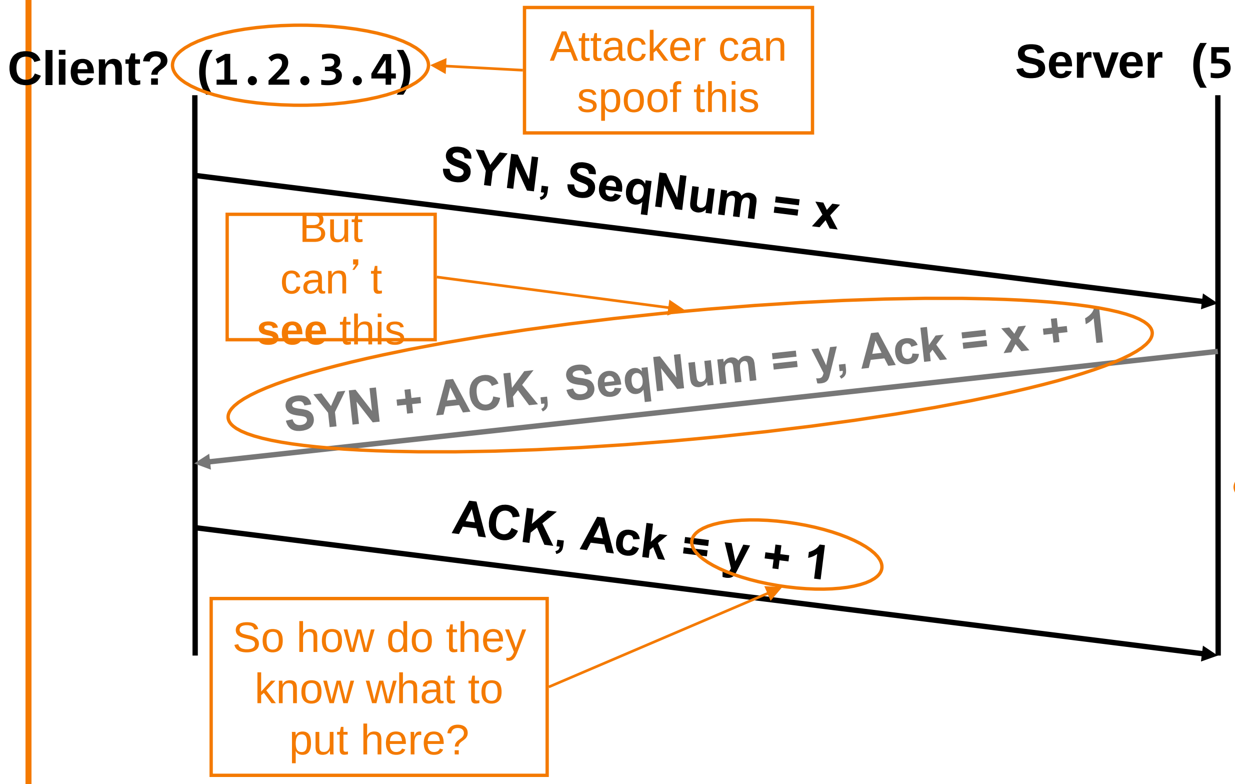
Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on local clock)

- How can an attacker create an *apparent but fake* connection from `1.2.3.4` to `5.6.7.8`?

Blind Spoofing: Attacker’s Viewpoint

Attacker



Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on local clock)

Hmm, any way for the attacker to know ***this?***

How Do We Fix This?

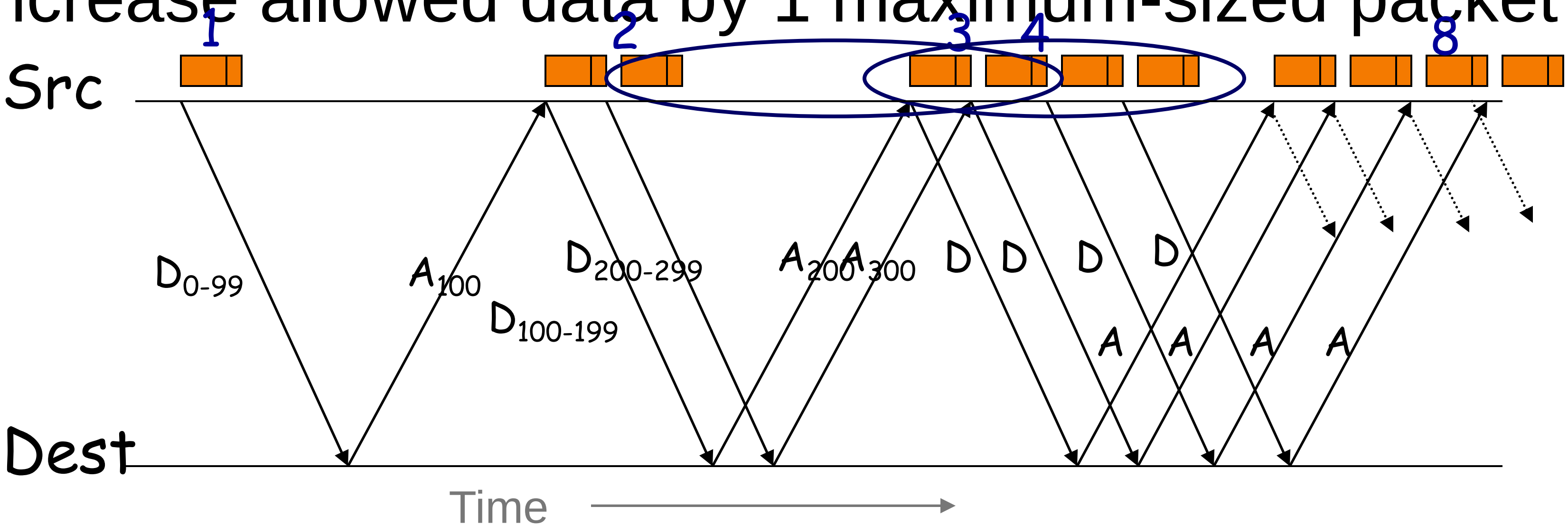
Use A *Random* ISN

Sure - make a non-spoofed connection *first*, and see what server used for ISN y then!

TCP's Rate Management

Unless there's loss, TCP doubles data in flight every "round-trip". All TCPs expected to obey ("fairness").

Mechanism: for each arriving ack for new data, increase allowed data by 1 maximum-sized packet

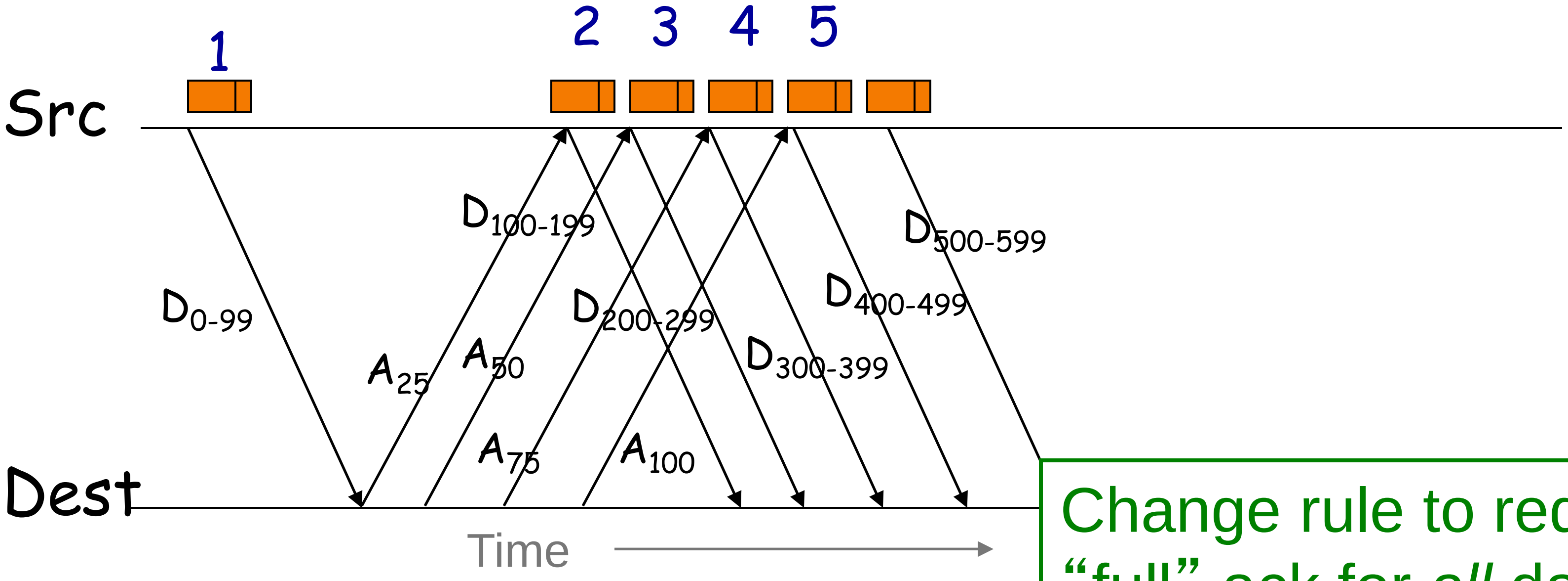


E.g., suppose maximum-sized packet = 100 bytes

Protocol Cheating

How can the destination (receiver) get data to come to them faster than normally allowed?

ACK-Splitting: each ack, even though **partial**, increases allowed data by one maximum-sized packet



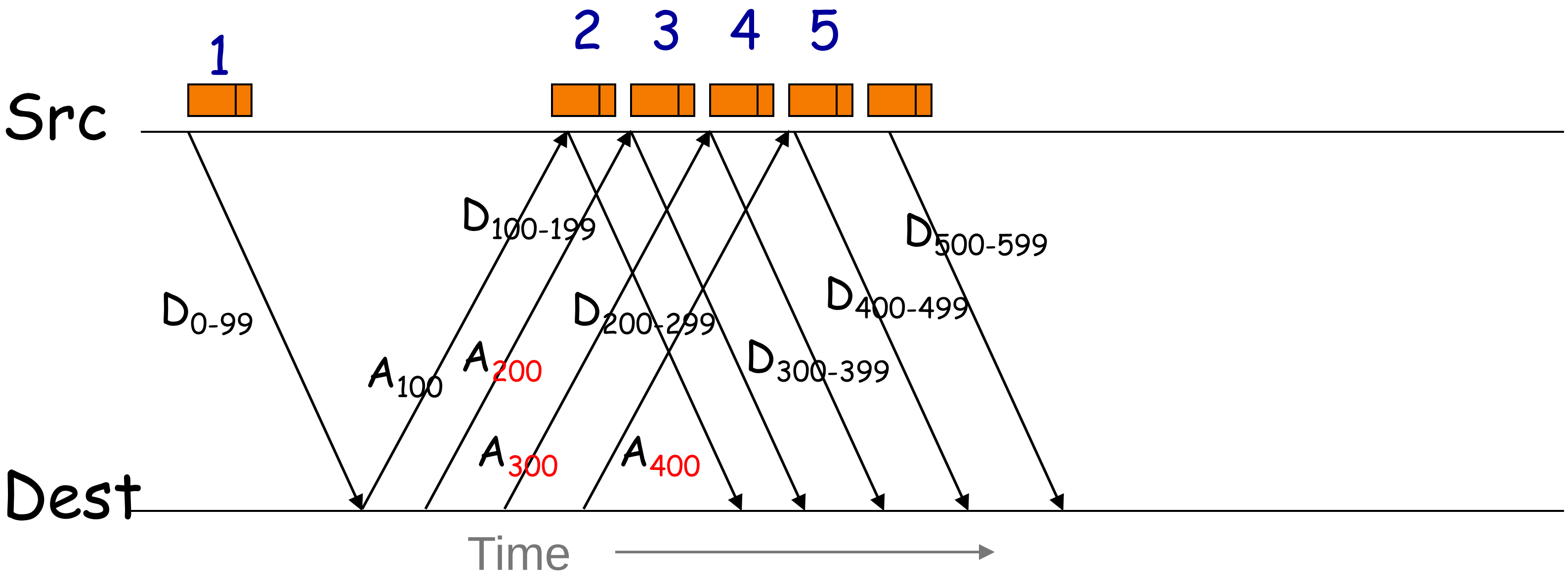
How do we defend against this?

Change rule to require "full" ack for *all* data sent in a packet

Protocol Cheating

How can the destination (receiver) *still* get data to come to them faster than normally allowed?

Opportunistic ack'ing: acknowledge data not yet seen!



How do we defend against *this*?

Keeping Receivers Honest

- Approach #1: if you receive an ack for data you haven't sent, kill the connection
  - Works only if receiver acks too far ahead
- Approach #2: follow the "round trip time" (RTT) and if ack arrives too quickly, kill the connection
  - Flaky: RTT can vary a lot, so you might kill innocent connections
- Approach #3: make the receiver prove they received the data

Note: a *protocol* change

  - Add a **nonce** ("random" marker) & require receiver to include it in ack. Kill connections w/ incorrect nonces
    - (nonce could be function computed over payload, so sender doesn't explicitly transmit, only implicitly)

Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully **terminate** by forging a RST packet
  - **Inject** (*spoof*) data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - *Remains a major threat today*

Summary of TCP Security Issues

- An attacker who can observe your TCP connection can manipulate it:
  - Forcefully **terminate** by forging a RST packet
  - **Inject** (*spoof*) data into either direction by forging data packets
  - Works because they can include in their spoofed traffic the correct sequence numbers (both directions) and TCP ports
  - *Remains a major threat today*
- An attacker who can predict the ISN chosen by a server can "blind spoof" a connection to the server
  - Makes it appear that host ABC has connected, and has sent data of the attacker's choosing, when in fact it hasn't
  - *Undermines any security based on trusting ABC's IP address*
  - Allows attacker to "frame" ABC or otherwise avoid detection
  - **Fixed** today by choosing **random** ISNs

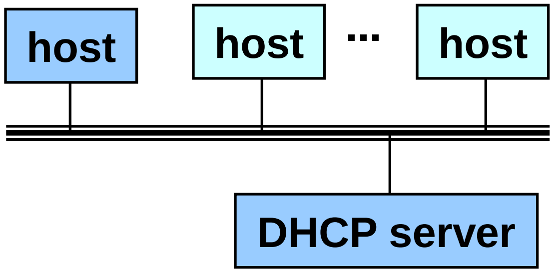
TCP Security Issues, con't

- TCP limits the rate at which senders transmit:
  - TCP relies on endpoints behaving properly to achieve "fairness" in how network capacity is used
  - Protocol lacks a mechanism to prevent cheating
  - Senders can cheat by just not abiding by the limits
    - Remains a significant vulnerability: essentially nothing today prevents
- Receivers can manipulate honest senders into sending too fast because senders trust that receivers are honest
  - To a degree, sender can validate (e.g., partial acks)
  - A **nonce** can force receiver to only act on data they've seen
  - Such rate manipulation remains a vulnerability today
- General observation: tension between ease/power of protocols that assume everyone follows vs. violating
  - Security problems persist due to difficulties of retrofitting …
  - … coupled with **investment in installed base**

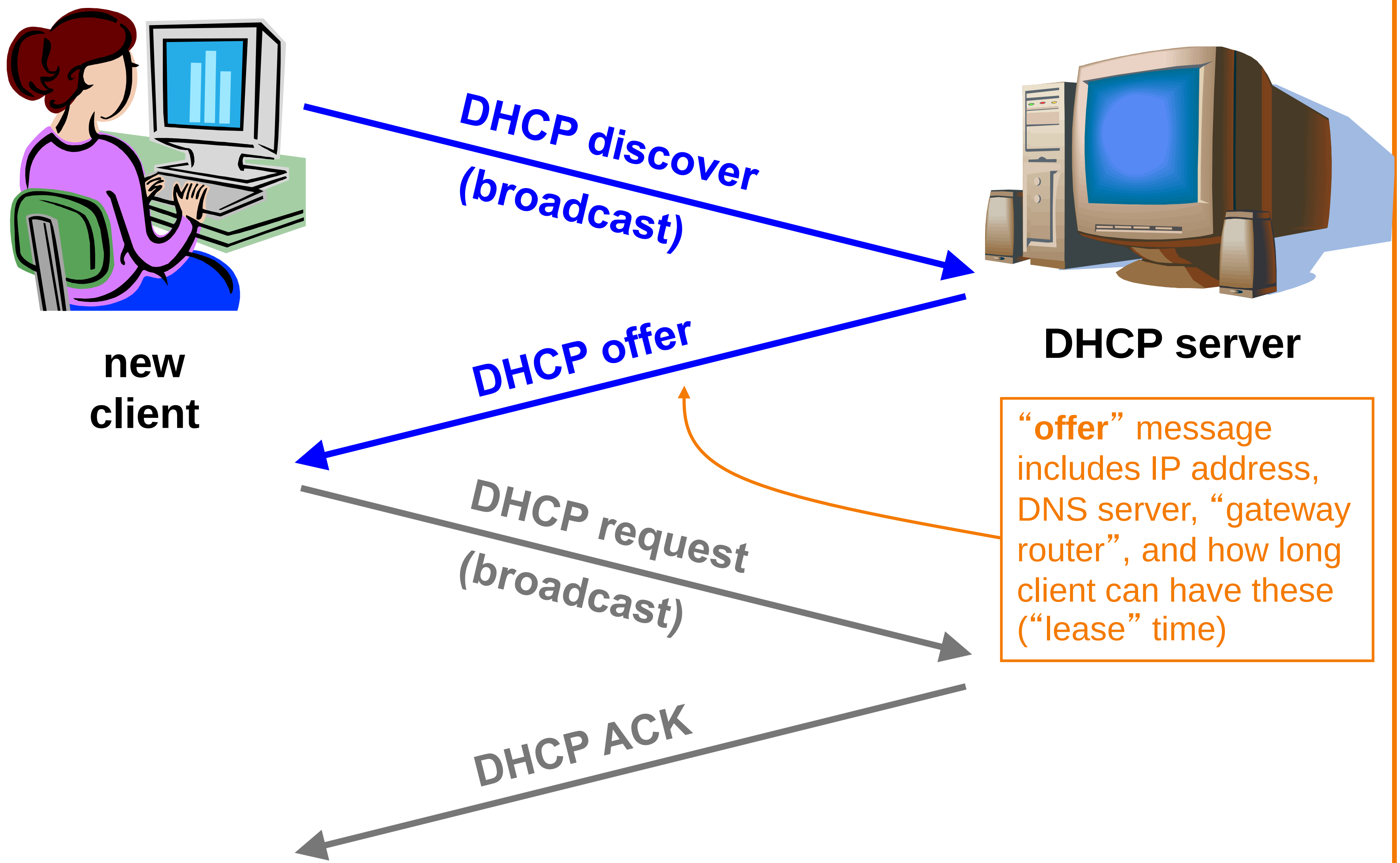
DHCP Problems

Internet Bootstrapping: DHCP

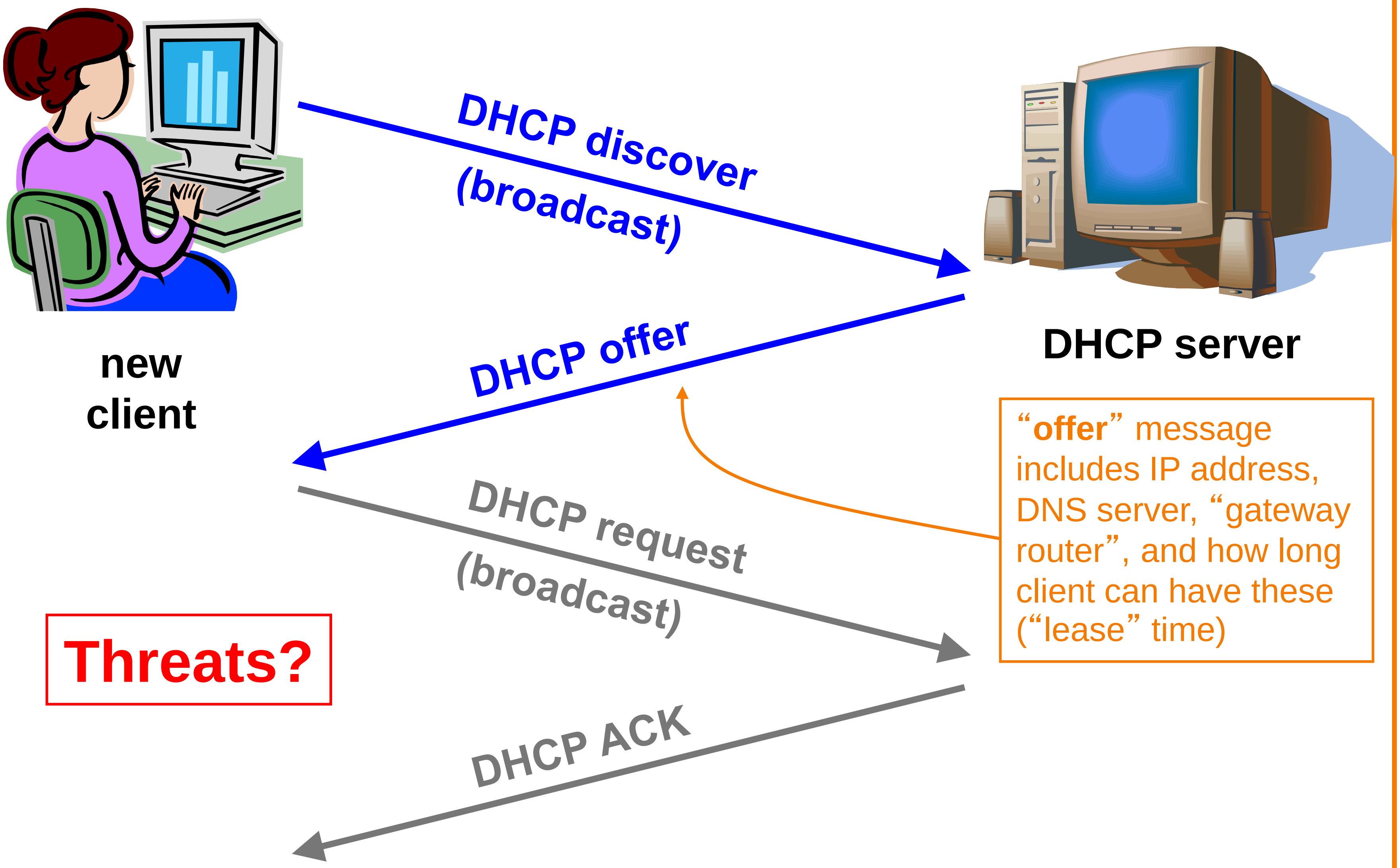
- New host doesn't have an IP address yet
  - So, host doesn't know what source address to use
- Host doesn't know *who to ask* for an IP address
  - So, host doesn't know what destination address to use
- Solution: shout to "**discover**" server that can help
  - Broadcast a server-discovery message (layer 2)
  - Server(s) sends a reply offering an address



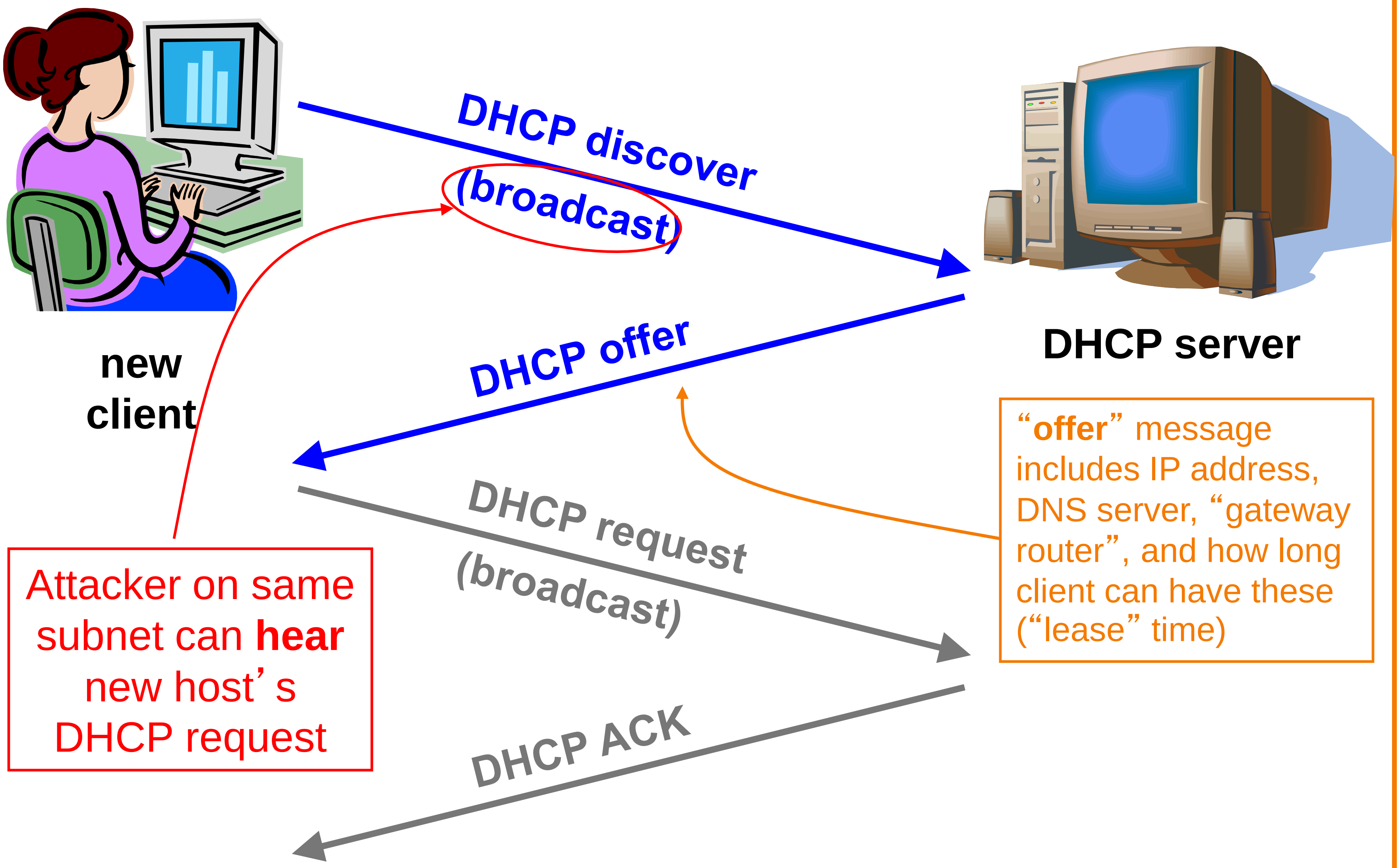
Dynamic Host Configuration Protocol



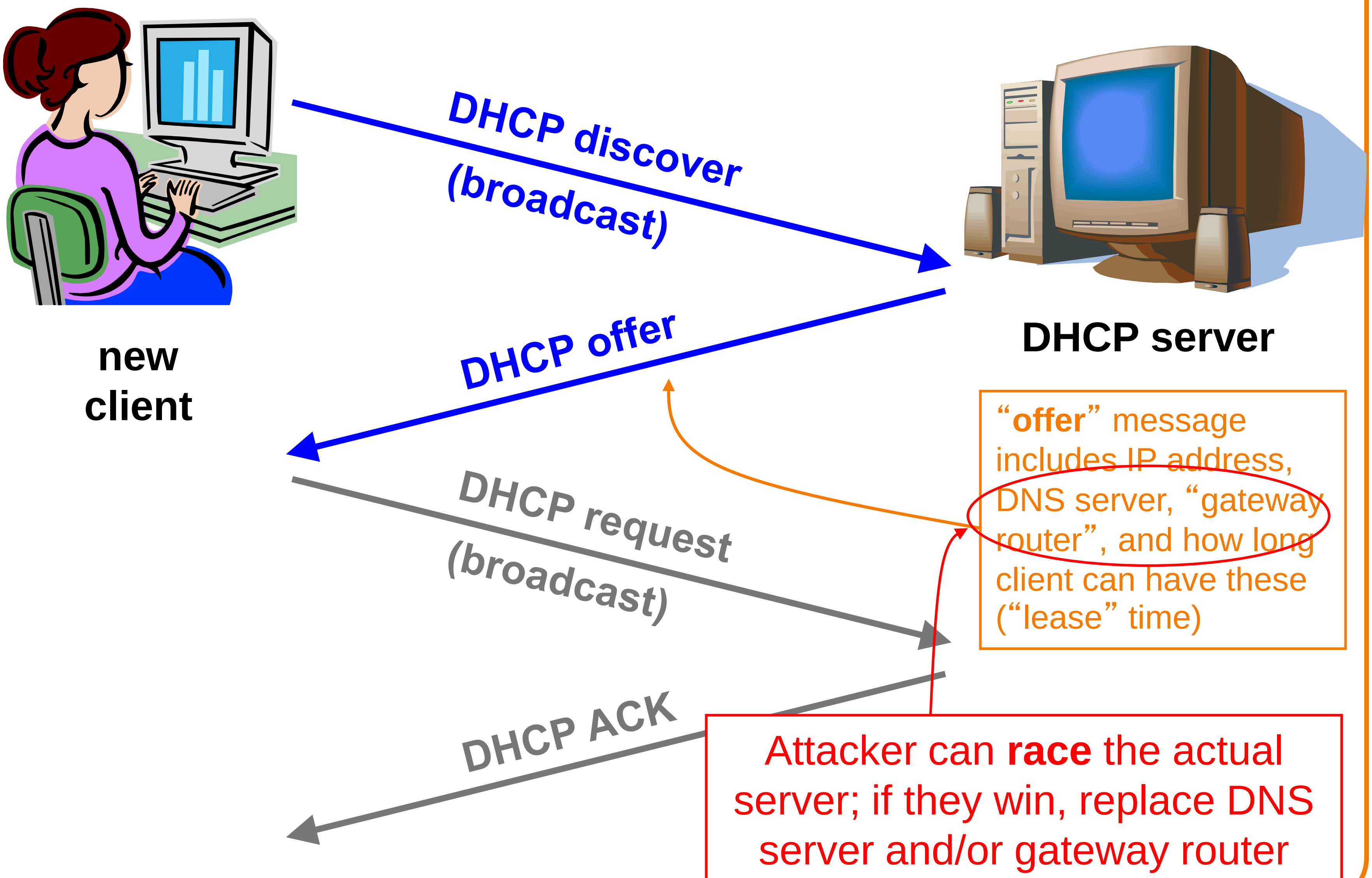
Dynamic Host Configuration Protocol



Dynamic Host Configuration Protocol



Dynamic Host Configuration Protocol



DHCP Threats

- Substitute a fake DNS server
  - Redirect any of a host's lookups to a machine of attacker's choice
- Substitute a fake "gateway"
  - Intercept all of a host's off-subnet traffic
    - (even if not preceded by a DNS lookup)
  - Relay contents back and forth between host and remote server
    - Modify however attacker chooses
- An invisible *Man In The Middle* (MITM)
  - Victim host has no way of knowing it's happening
    - (Can't necessarily alarm on peculiarity of receiving multiple DHCP replies, since that can happen benignly)
- How can we fix this?

Hard

DNS Vulnerabilities

Non-Eavesdropping Threats: DNS

- DHCP attacks show brutal power of attacker who can eavesdrop
- Consider attackers who *can't* eavesdrop - but still aim to manipulate us via how protocols function
- DNS: path-critical for just about everything we do
  - Maps hostnames ⇔ IP addresses
  - Design only **scales** if we can minimize lookup traffic
    - #1 way to do so: caching
    - #2 way to do so: return not only answers to queries, but additional info that will likely be needed shortly
- Directly interacting w/ DNS: `dig` program on Unix
  - Allows querying of DNS system
  - Dumps each field in DNS responses

dig eecs.mit.edu A

Use Unix "`dig`" utility to look up DNS address ("`A`") for hostname `eecs.mit.edu`

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

These are just comments from `dig` itself with details of the request/response

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

Transaction identifier

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.                                   62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

Here the server echoes back the question that it is answering

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QU…IONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                IN        A

;; ANSWER SECTION:
eecs.mit.edu.         21600   IN        A        18.62.1.6

;; AUTHORITY SECTION:
mit.edu.              11088   IN        NS       BITSY.mit.edu.
mit.edu.              11088   IN        NS       W20NS.mit.edu.
mit.edu.              11088   IN        NS       STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.       126738  IN        A        18.71.0.151
BITSY.mit.edu.        166408  IN        A        18.72.0.3
W20NS.mit.edu.        126738  IN        A        18.70.0.160
```

"Answer" tells us its address is 18.62.1.6 and we can cache the result for 21,600 seconds

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.         126738  IN      A       18.71.0.151
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

"Authority" tells us the *name servers* responsible for the answer. Each record gives the *hostname* of a different name server ("`NS`") for names in `mit.edu.` We should cache each record for 11,088 seconds.

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                  11088   IN      NS      BITSY.mit.edu.
mit.edu.                  11088   IN      NS      W20NS.mit.edu.
mit.edu.                  11088   IN      NS      STRAWB.mit.edu.

;; ADDITIONAL SECTION:
STRAWB.mit.edu.           126738  IN      A       18.71.0.151
BITSY.mit.edu.            166408  IN      A       18.72.0.3
W20NS.mit.edu.            126738  IN      A       18.70.0.160
```

"Additional" provides extra information to save us from making separate lookups for it, or helps with bootstrapping.

Here, it tells us the IP addresses for the hostnames of the name servers. We add these to our cache.

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.           21600   IN      A       18.62.1.6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30      IN      NS      eecs.berkeley.edu.

;; ADDITIONAL SECTION:
eecs.berkeley.edu.      30      IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

What happens if the mit.edu server returns the following to us instead?

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30      IN      NS      eecs.berkeley.edu.

;; ADDITIONAL SECTION:
eecs.berkeley.edu.      30      IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

We dutifully store in our cache a mapping of `eecs.berkeley.edu` to an IP address under MIT's control. (It could have been any IP address they wanted, not just one of theirs.)

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                  IN      A

;; ANSWER SECTION:
eecs.mit.edu.                                           .6

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30      IN      NS      eecs.berkeley.edu.

;; ADDITIONAL SECTION:
eecs.berkeley.edu.      30      IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

In this case they chose to make the mapping *disappear* after 30 seconds. They could have made it persist for weeks, or disappear even quicker.

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 19901
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;eecs.mit.edu.                   IN      A

;; ANSWER SECTION
eecs.mit.edu.

;; AUTHORITY SECTION:
mit.edu.                11088   IN      NS      BITSY.mit.edu.
mit.edu.                11088   IN      NS      W20NS.mit.edu.
mit.edu.                30      IN      NS      eecs.berkeley.edu.

;; ADDITIONAL SECTION:
eecs.berkeley.edu.      30      IN      A       18.6.6.6
BITSY.mit.edu.          166408  IN      A       18.72.0.3
W20NS.mit.edu.          126738  IN      A       18.70.0.160
```

How do we fix such ***cache poisoning***?

dig eecs.mit.edu A

```
; ; <<>> DiG 9.6.0-APPLE-P2 <<>> eecs.mit.edu a
;; global options: +c
;; Got answer:
;; ->>HEADER<<- opcod
;; flags: qr rd ra; Q

;; QUESTION SECTION:
;eecs.mit.edu.

;; ANSWER SECTION:
eecs.mit.edu.          21600   IN   A    18.62.1.6

;; AUTHORITY SECTION:
mit.edu.               11088   IN   NS   BITSY.mit.edu.
mit.edu.               11088   IN   NS   W20NS.mit.edu.
mit.edu.               30      IN   NS   eecs.berkeley.edu.

;; ADDITIONAL SECTION:
eecs.berkeley.edu.     30      IN   A    18.6.6.6
BITSY.mit.edu.         166408  IN   A    18.72.0.3
W20NS.mit.edu.         126738  IN   A    18.70.0.160
```

Don't accept Additional records unless they're for the domain we're looking up

E.g., looking up `eecs.mit.edu` ⇒ only accept additional records from `*.mit.edu`

No extra risk in accepting these since server could return them to us directly in an Answer anyway.

DNS Threats, con't

What about *blind spoofing*?

- Say we look up `mail.google.com`; how can an off-path attacker feed us a bogus A answer before the legitimate server replies?

- How can such an attacker even know we are looking up `mail.google.com`?

16 bits	*16 bits*
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

```
<img src="http://mail.google.com" …>
```

DNS Blind Spoofing, con't

Fix?

Once they know we're looking it up, they just have to guess the Identification field and reply before legit server.

How hard is that?

Originally, identification field incremented by 1 for each request. How does attacker guess it?

16 bits	*16 bits*
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

<img src="http://badguy.com" …> ← They observe ID k here

<img src="http://mail.google.com" …> ← So this will be k+1

DNS Blind Spoofing, con’t

Once we randomize the Identification, attacker has a 1/65536 chance of guessing it correctly.
Are we pretty much safe?

Attacker can send *lots* of replies, not just one …

However: once reply from legit server arrives (with correct Identification), it’s **cached** and no more opportunity to poison it. Victim is innoculated!

16 bits	*16 bits*
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

Unless attacker can send 1000s of replies before legit arrives, we’re likely safe - phew! **?**

DNS Blind Spoofing (Kaminsky 2008)

- Two key ideas:
  - Spoof uses Additional field (rather than Answer)
  - Attacker can get around caching of legit replies by generating a series of different name lookups:

```
<img src="http://random1.google.com" …>
<img src="http://random2.google.com" …>
<img src="http://random3.google.com" …>
                  . . .
<img src="http://randomN.google.com" …>
```

Kaminsky Blind Spoofing, con't

For each lookup of `randomk.google.com`, attacker returns a bunch of records like this, each with a different Identifier

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

;; ANSWER SECTION:
randomk.google.com       21600       IN      A       doesn't matter

;; AUTHORITY SECTION:
google.com.              11088       IN      NS      mail.google.com

;; ADDITIONAL SECTION:
mail.google.com          126738      IN      A       6.6.6.6
```

Once they win the race, not only have they poisoned `mail.google.com` …

Kaminsky Blind Spoofing, con't

For each lookup of `randomk.google.com`, attacker returns a bunch of records like this, each with a different Identifier

;; QUESTION SECTION:				
;randomk.google.com.		**IN**	**A**	
;; ANSWER SECTION:				
randomk.google.com	**21600**	**IN**	**A**	*doesn't matter*
;; AUTHORITY SECTION:				
google.com.	**11088**	**IN**	**NS**	**mail.google.com**
;; ADDITIONAL SECTION:				
mail.google.com	**126738**	**IN**	**A**	**6.6.6.6**

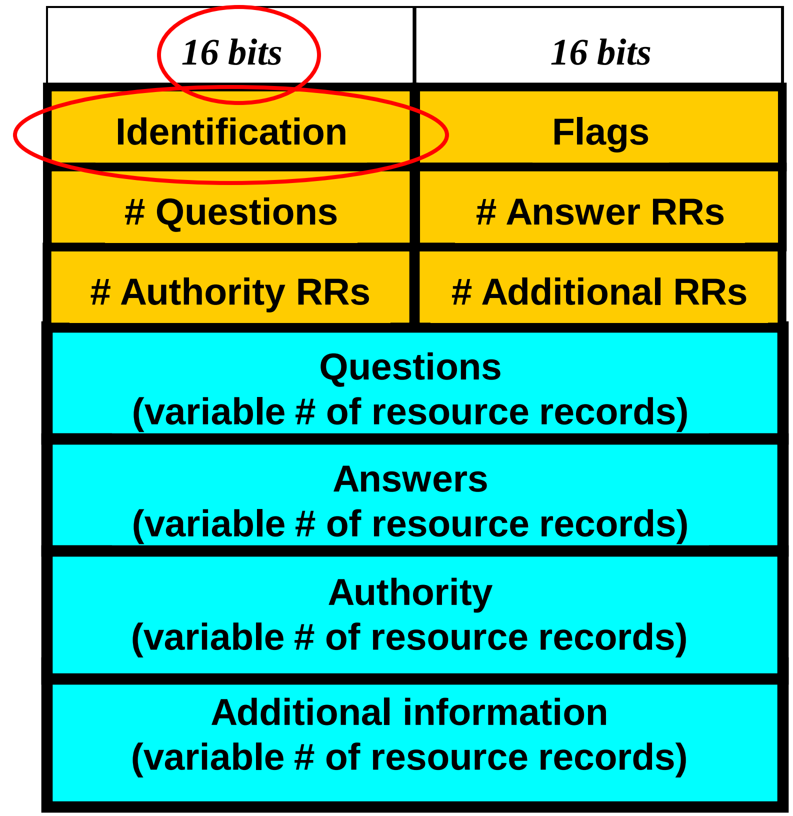
Once they win the race, not only have they poisoned `mail.google.com` … but also the cached `NS` record for `google.com`'s name server - so any **future** *X*`.google.com` lookups *go through the attacker's machine*

Defending Against Blind Spoofing

Central problem: all that tells a client they should accept a response is that it matches the Identification field.

With only 16 bits, it lacks sufficient entropy: even if truly random, the *search space* an attacker must *brute force* is too small.

Where can we get more entropy? (*Without* requiring a protocol change.)



Defending Against Blind Spoofing

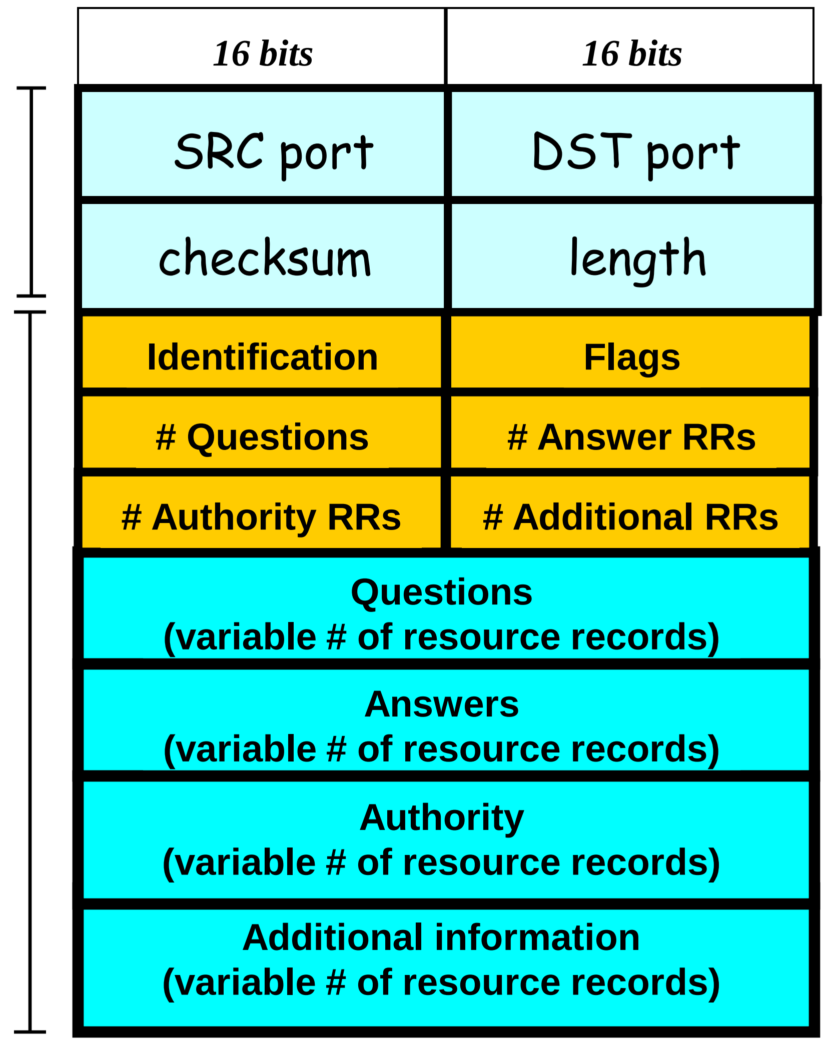
DNS (primarily) uses UDP for transport rather than TCP.

UDP header has:

- 16-bit Source & Destination ports (identify processes, like w/ TCP)
- 16-bit checksum, 16-bit length

UDP Header

UDP Payload



Defending Against Blind Spoofing

Total *entropy*: 16 bits

DNS (primarily) uses UDP for transport rather than TCP.

UDP header has:

- 16-bit Source & Destination ports (identify processes, like w/ TCP)
- 16-bit checksum, 16-bit length

For requestor to receive DNS reply, needs both correct Identification and correct ports.

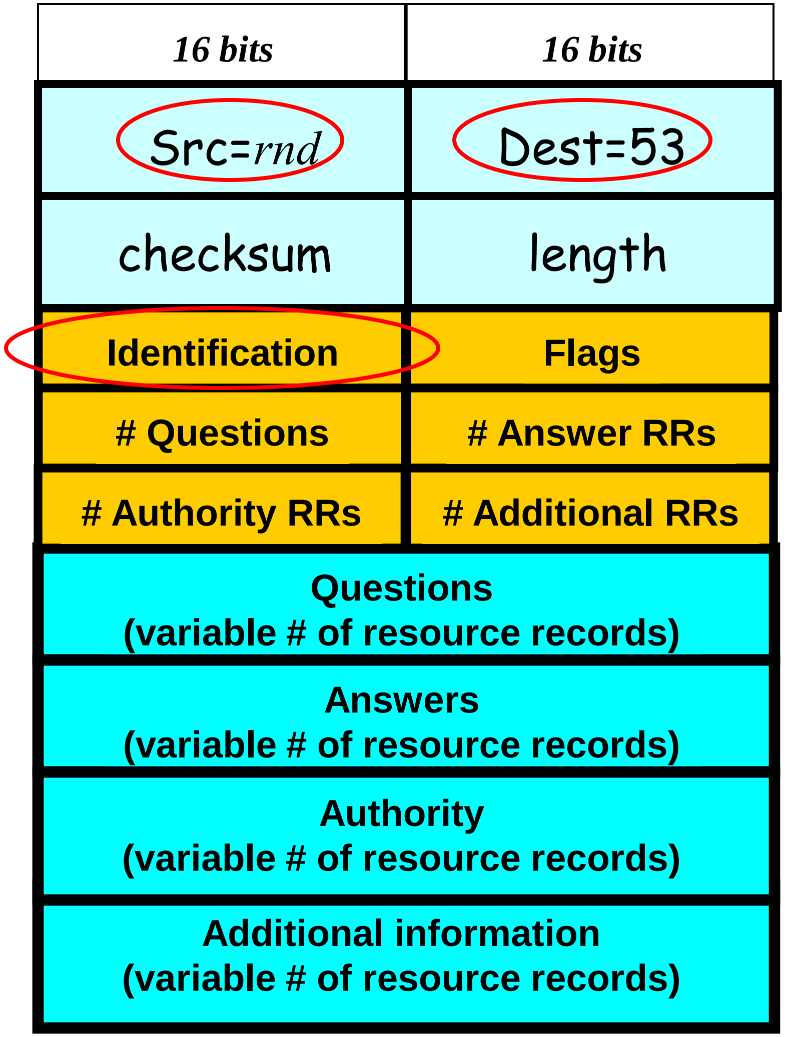
On a request, DST port = 53. SRC port usually also 53 - but not fundamental, just convenient

16 bits	*16 bits*
Src=53	Dest=53
checksum	length
Identification	**Flags**
# Questions	**# Answer RRs**
# Authority RRs	**# Additional RRs**
Questions (variable # of resource records)	
Answers (variable # of resource records)	
Authority (variable # of resource records)	
Additional information (variable # of resource records)	

Defending Against Blind Spoofing

Total *entropy*: ? bits

"Fix": use random source port



Defending Against Blind Spoofing

"Fix": use random source port

32 bits of entropy makes it orders of magnitude harder for attacker to guess all the necessary fields and dupe victim into accepting spoof response.

This is what primarily "secures" DNS today. (Note: not all resolvers have implemented random source ports!)