Lecture 18

ARP, DHCP, NAT, TLS, and End-to-End

CS 168, Fall 2024 @ UC Berkeley

Slides credit: Sylvia Ratnasamy, Rob Shakir, Peyrin Kao

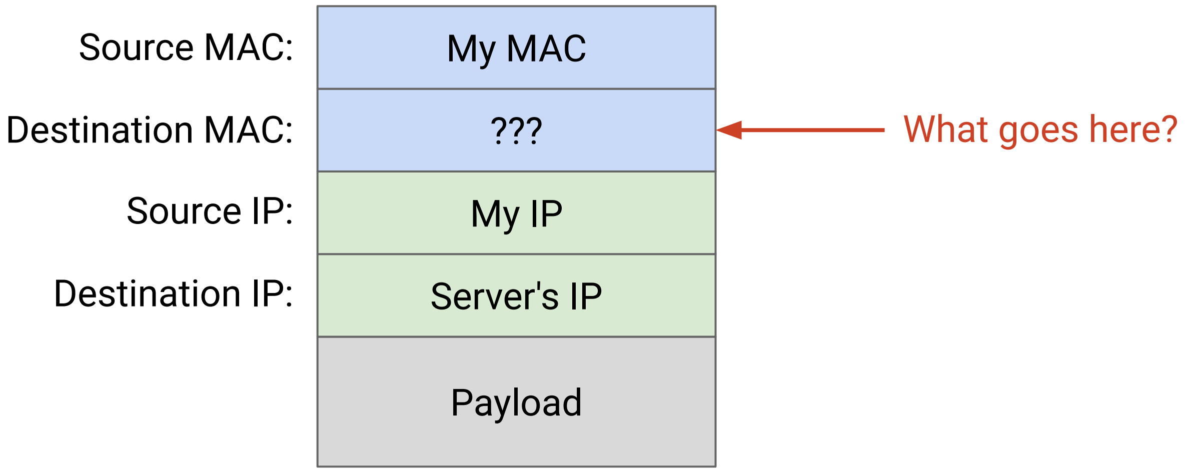
ARP: Connecting Layers 2 and 3

Lecture 18, CS 168, Fall 2024

Connecting Layers 2 and 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.



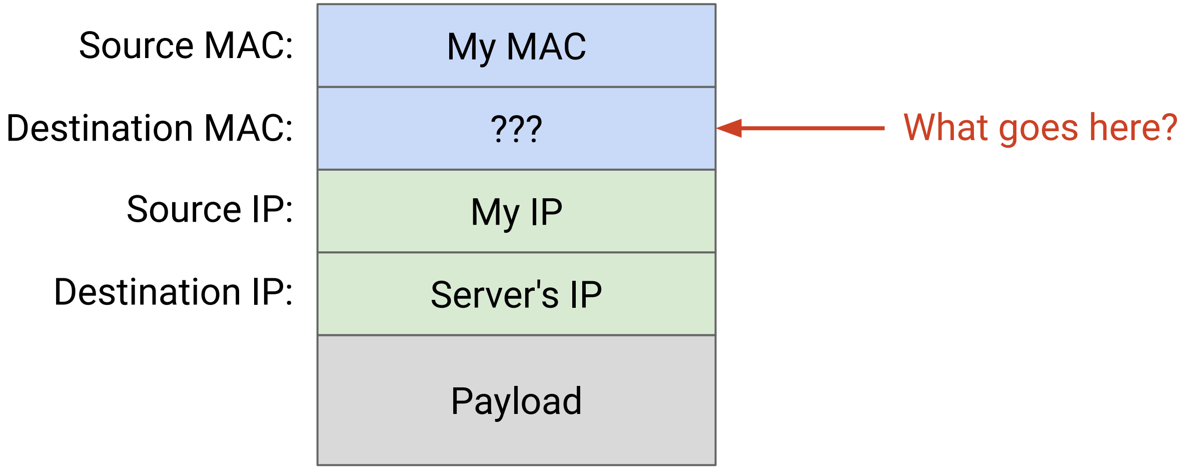
Connecting Layers 2 and 3

If the destination IP is in our local network:

- Find the destination's MAC address, and send to destination on Layer 2.

If the destination IP is *not* in our local network:

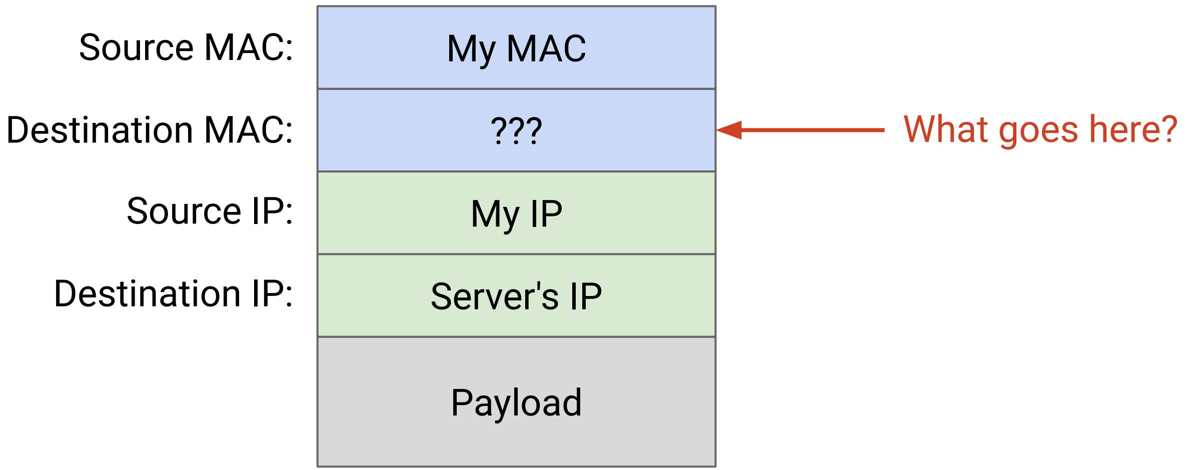
- Find the router's MAC address, and send to the router on Layer 2.
- Router can forward our packet toward the destination.



Connecting Layers 2 and 3

How do we send packets to the destination (local) or the router (non-local)?

- We could broadcast: Put `FF:FF:FF:FF:FF:FF` as destination MAC.
  - But now, everybody else has to process this packet.
  - Need extra bandwidth to send the packet to everyone on local network.
- We really want to *unicast* the packet to the right MAC address.
  - We need some way to translate IP addresses to MAC addresses.



ARP (Address Resolution Protocol) – Steps

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.

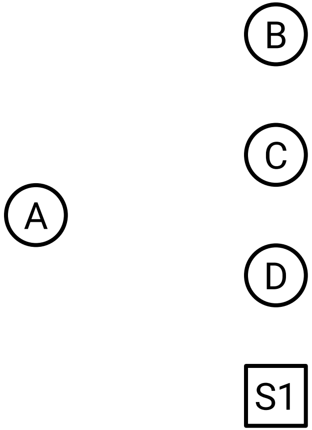
Steps of the protocol:

1. Alice checks her cache 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) – Step 1/4

Alice knows Bob's IP address is 1.2.3.4. She wants to learn Bob's MAC address.

Alice's cache		
IP	MAC	TTL



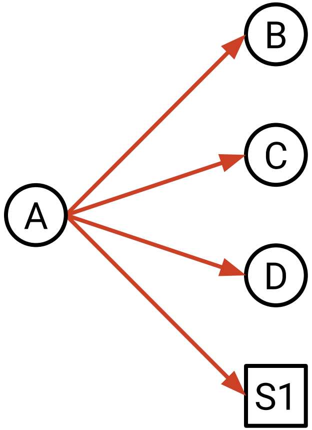
1. Alice checks her cache to see if she already knows the MAC address corresponding to **1.2.3.4**.

Since her cache is empty, she must make a request to find out.

ARP (Address Resolution Protocol) – Step 2/4

Alice knows Bob's IP address is 1.2.3.4. She wants to learn Bob's MAC address.

Alice's cache		
IP	MAC	TTL

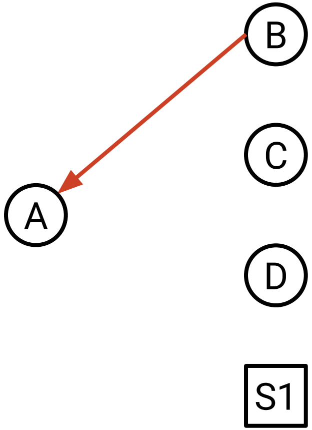


2. Alice asks everyone else on the local network: "What is the MAC address of **1.2.3.4**?"

ARP (Address Resolution Protocol) – Step 3/4

Alice knows Bob's IP address is 1.2.3.4. She wants to learn Bob's MAC address.

Alice's cache		
IP	MAC	TTL



3. Bob responds: "My IP is **1.2.3.4** and my MAC address is `ca:fe:f0:0d:be:ef`."

Everybody else ignores the request.

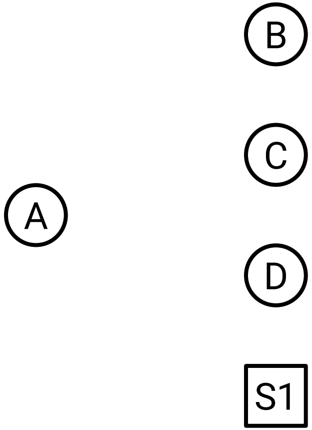
ARP (Address Resolution Protocol) – Step 4/4

Alice knows Bob's IP address is 1.2.3.4. She wants to learn Bob's MAC address.

Alice's cache		
IP	MAC	TTL
1.2.3.4	`ca:fe:f0:0d:be:ef`	1 hr

4. Alice adds Bob's MAC address to her cache.

This mapping can be cached for some time (TTL).



Address Resolution Protocol (ARP)

ARP runs directly on Layer 2 (not IP).

Source MAC:	Alice's MAC
Destination MAC:	`FF:FF:FF:FF:FF:FF`

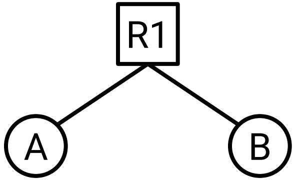
ARP *requests* (aka *solicitations*) are broadcast.

Source MAC:	Bob's MAC
Destination MAC:	Alice's MAC

ARP *responses* (aka *advertisements*) are unicast.

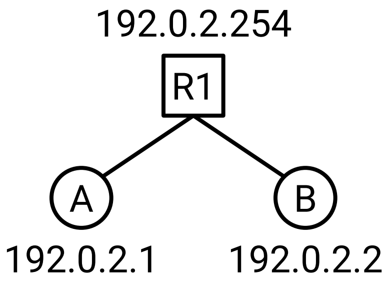
Note: You can also broadcast an unsolicited response:
"My IP is 1.2.3.4, and my MAC is `ca:fe:f0:0d:be:ef`...even though no one asked."

Using ARP in Routers



R1's Table (Conceptual)	
Destination	Next Hop
A	Direct
B	Direct

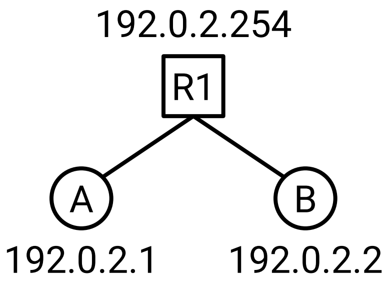
In routing, we showed direct routes like this.



R1's Table (Actual)	
Destination	Next Hop
192.0.2.1	Direct
192.0.2.2	Direct

In reality, the table contains IP addresses...

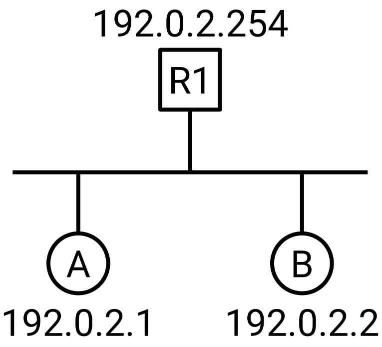
Using ARP in Routers



R1's Table (Actual)	
Destination	Next Hop
192.0.2.0/24	Direct

Our **subnet**: The range of all IP addresses in our local network.

...and addresses can be aggregated!



R1's Table (Actual)	
Destination	Next Hop
192.0.2.0/24	Direct

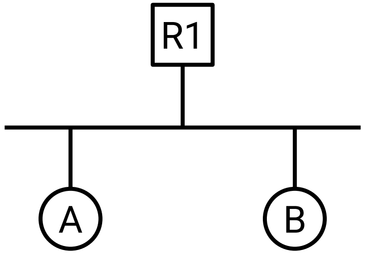
What if multiple hosts are on the same link?

Use ARP to send to correct MAC address.

Using ARP in Routers

"Direct" really means: Send to the right Layer 2 address for the local destination.

- Perform an ARP lookup (or look in our cached ARP table) to find the destination's MAC address.
- Send unicast Ethernet frames to that MAC address.

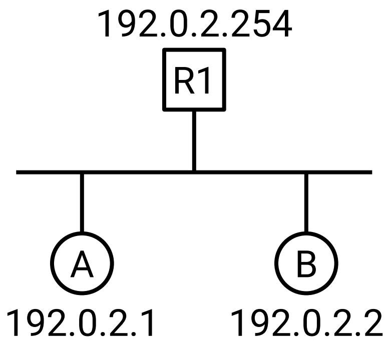


R1's Table (Actual)	
Destination	Next Hop
192.0.2.0/24	Direct

Using ARP in Hosts

How do hosts forward packets?

- If destination IP is local:
  - Use ARP to find MAC of destination. Send to destination.
- If destination IP is non-local:
  - Use ARP to find MAC of router. Send to router.



A's Table (Actual)	
Destination	Next Hop
192.0.2.0/24	Direct
0.0.0.0/0	192.0.2.254

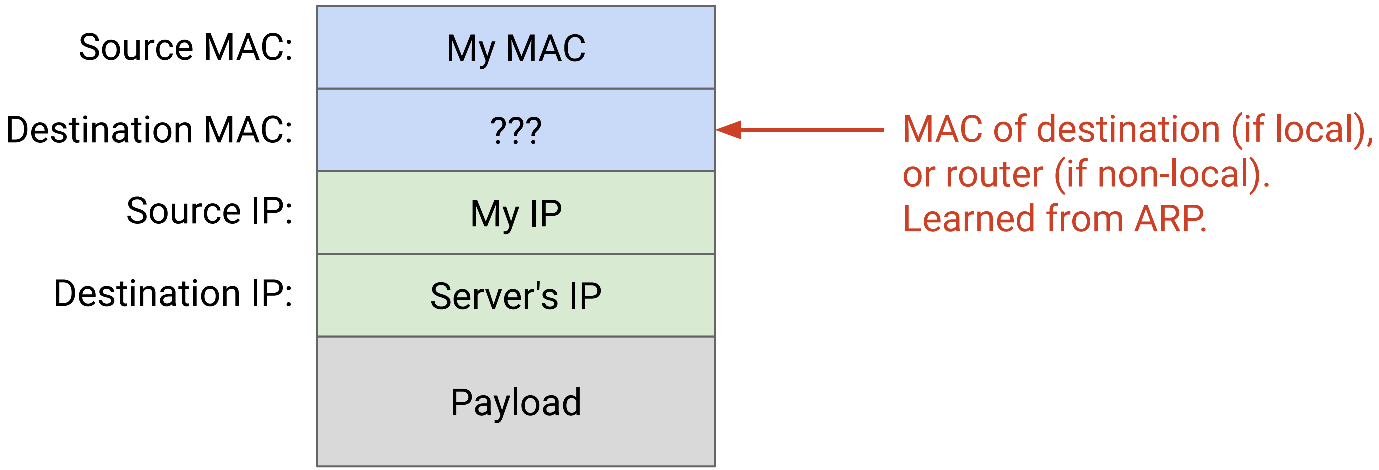
Local destinations are direct.

Non-local destinations get forwarded to router.

Using ARP in Hosts

Notice: Each hop changes the Layer 2 destination, but not the Layer 3 destination.

- Hosts and routers use ARP to find the MAC address of the next hop.



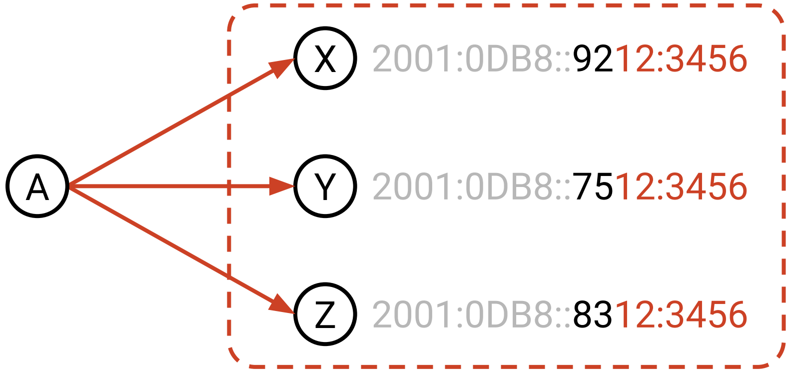
Neighbor Discovery in IPv6

ARP translates IPv4 to MAC. **Neighbor discovery** translates IPv6 to MAC.

- Instead of broadcasting requests, *multicast* request to the group that Bob is in.
- Everyone joins a group based on their IPv6 address.

Everyone with **IP** ending in 78:90AB listens on multicast **MAC** address 33:33:FF:78:90:AB.

Everyone with **IP** ending in 12:3456 listens on multicast **MAC** address 33:33:FF:12:34:56.



If A wants the MAC matching 2001:0DB8::9212:3456, A multicasts to only the 12:3456 group.

DHCP: Joining a New Network

Lecture 18, CS 168, Fall 2024

Joining a New Network

When we connect to a new Ethernet network, we need to learn:

- **Subnet mask**: What range of addresses are local?
- **Default gateway**: Where is the router? So I can send non-local packets to them.
- **DNS server**: Where is the recursive resolver?
- We also need to get an **IP address** that we can use for this network.

Note: We already have a MAC address (burnt into hardware).

We could configure this information manually.

- Need to reconfigure every time we join a different network.
- We want an automatic protocol to learn this information.

Note: Manual configuration is okay for routers, which rarely move.

DHCP (Dynamic Host Configuration Protocol) – Steps

1. **Client Discover**: The client broadcasts a request for a configuration.

2. **DHCP Offer**: One or more DHCP servers respond with a configuration offer.
   Offer includes subnet mask, router's IP address, DNS resolver, and IP for client.

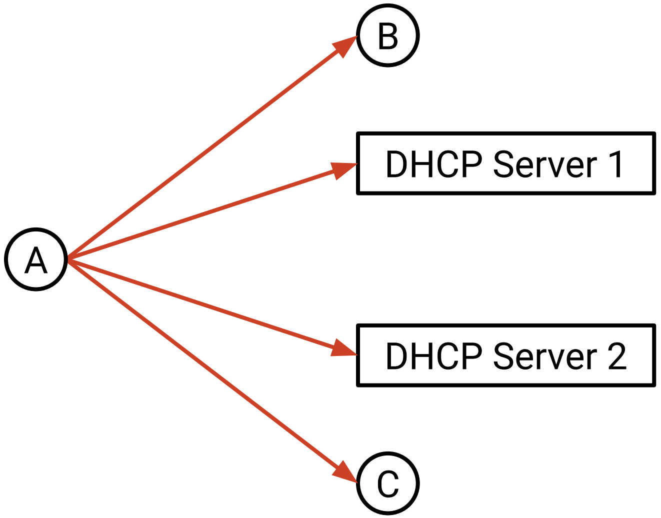
3. **Client Request**: The client broadcasts which configuration it has chosen.
   If multiple DHCP servers made offers, the ones that were not chosen discard their offer.

4. **DHCP Acknowledgement**: The chosen server confirms that its configuration has been given to the client.

DHCP (Dynamic Host Configuration Protocol) – Step 1/4

Alice wants to join the network.

Alice's configuration	
Subnet mask:	???
Router IP:	???
DNS Resolver:	???
My IP:	???

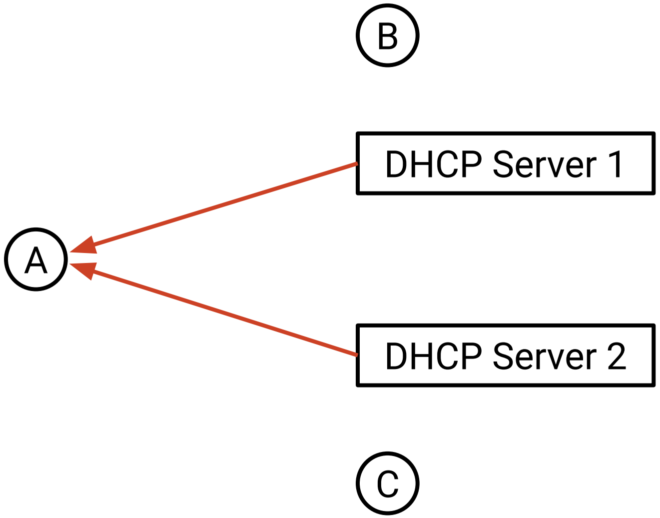


1. **Client discover**: Alice broadcasts a request:
"Can anyone give me a configuration?"

DHCP (Dynamic Host Configuration Protocol) – Step 2/4

Alice wants to join the network.

Alice's configuration	
Subnet mask:	???
Router IP:	???
DNS Resolver:	???
My IP:	???

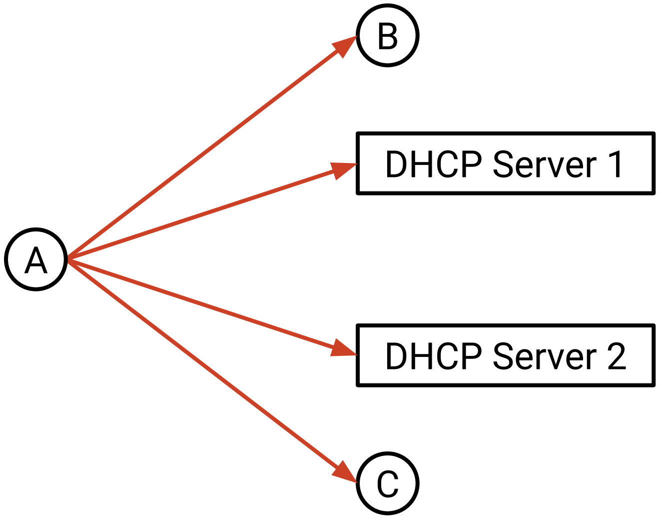


2. **DHCP Offer**: One or more DHCP servers reply with an offer for Alice.

DHCP (Dynamic Host Configuration Protocol) – Step 3/4

Alice wants to join the network.

Alice's configuration	
Subnet mask:	???
Router IP:	???
DNS Resolver:	???
My IP:	???

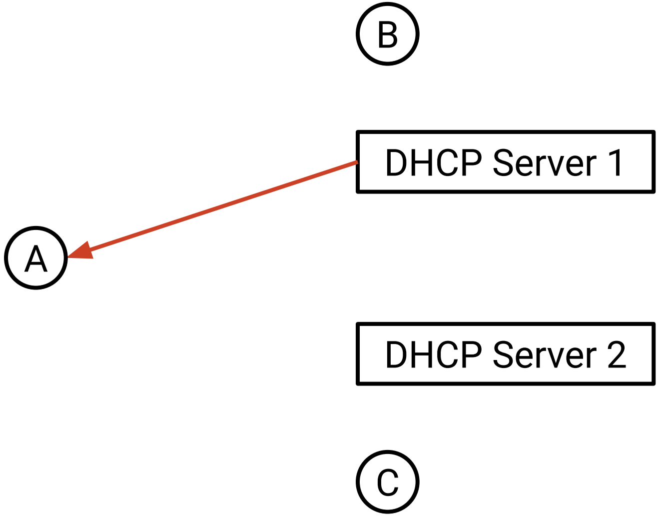


3. **Client Request**: Alice broadcasts which offer she chose: "I'll use the offer from DHCP Server 1."

DHCP (Dynamic Host Configuration Protocol) – Step 3/4

Alice wants to join the network.

Alice's configuration	
Subnet mask:	/24
Router IP:	192.168.86.254
DNS Resolver:	8.8.8.8
My IP:	192.168.86.38



4. **DHCP Acknowledgment**: The chosen DHCP server confirms that the configuration has been set for Alice.

DHCP Servers

DHCP servers offer configurations to new hosts.

- Listen on UDP port 67 for requests.

DHCP servers are configured with required information:

- Subnet mask, gateway router IP address, DNS resolver IP address.
- A *pool* of usable IP addresses.
- DHCP is extensible to provide other information.

Where are the DHCP servers?

- In a small network: Your home router.
- In a larger network: Could be a separate machine.
- Must be in same local network as the clients.
    - In larger networks, router could relay requests to a remote DHCP server.

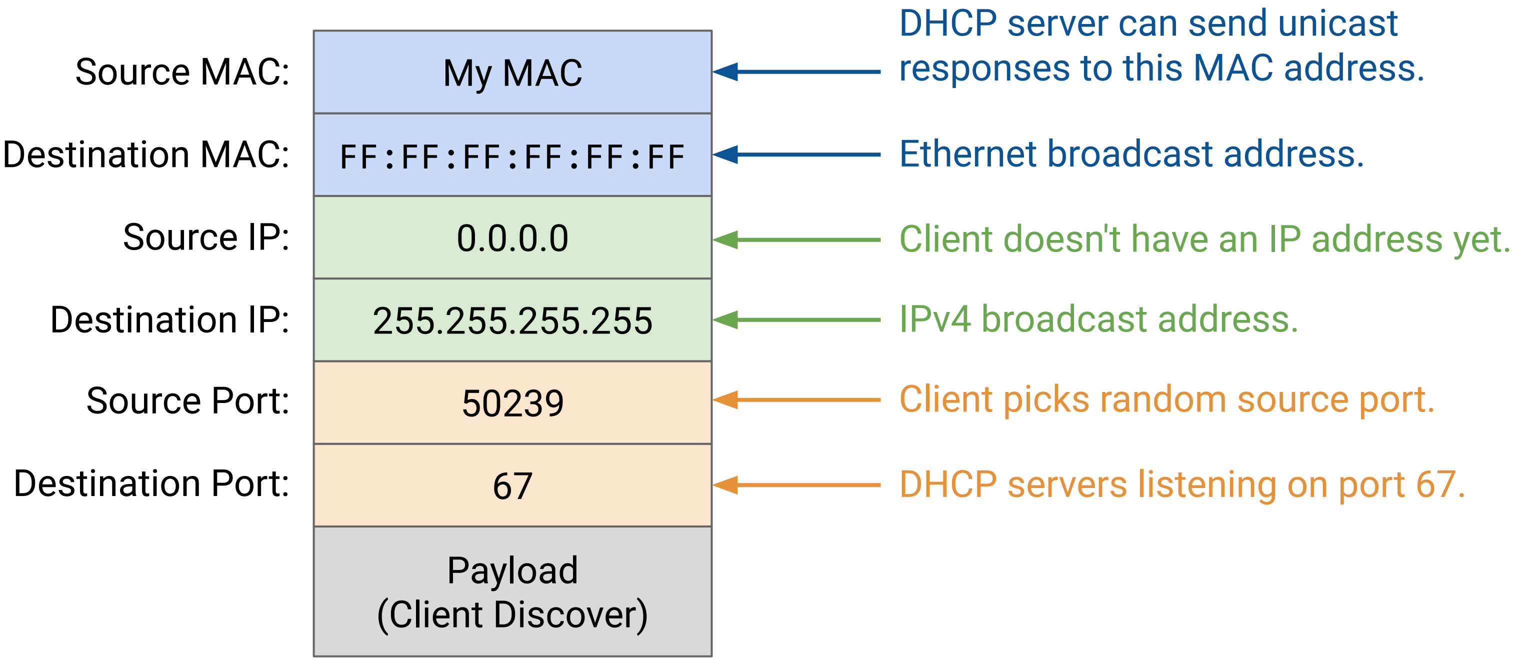
Leasing IP Addresses

DHCP servers temporarily *lease* IP addresses to hosts.

- Host can only use that IP address for a limited time (hours or days).
- Host must renew the lease to keep using the address.
- Servers don't offer the same address to other clients if leased.
    - Avoids two hosts getting the same address.

DHCP Implementation

DHCP is a Layer 7 protocol, running on top of UDP/IP.



Autoconfiguration in IPv6

Step 1: Use Neighbor Discovery (IPv6 ARP) to learn the other information.

- Router address, DNS resolver address, and subnet (aka local network prefix).

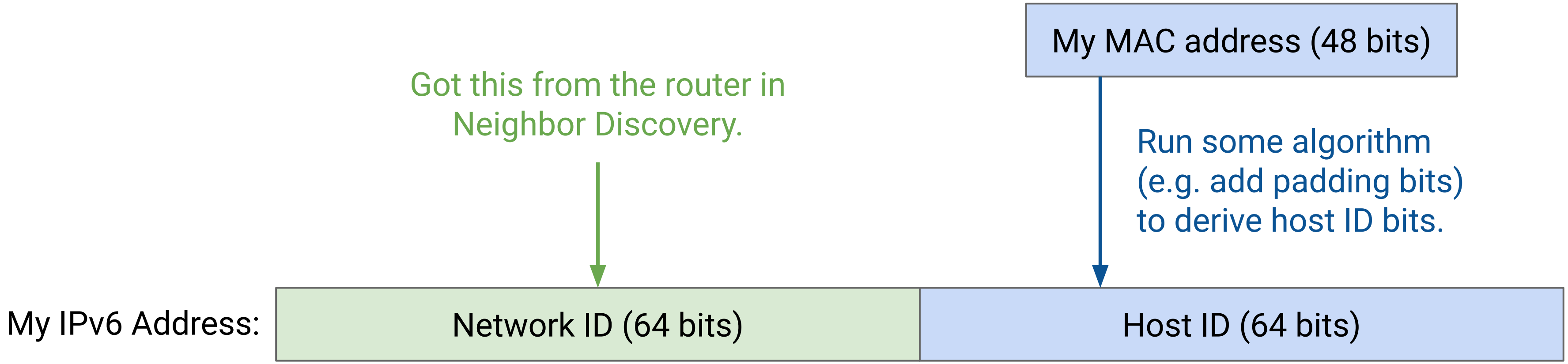
Neighbor Discovery can be extended:

- *Router Solicitation*: Broadcast request for the information.
- *Router Advertisement*: Routers reply with the information.

Autoconfiguration in IPv6

Step 2: Use SLAAC (Stateless Address Autoconfiguration) to give yourself a unique IPv6 address.

- No need for servers to manage and lease addresses!
- Trick: MAC addresses are unique, so we can just copy those bits to form a unique IPv6 address.
- Additional mechanisms in place for duplicate address detection, just in case.



NAT: Network Address Translation

Lecture 18, CS 168, Fall 2024

Problem: IPv4 Address Exhaustion

Recall: 2^{32} IPv4 address is not enough for every host on the Internet.

- IPv6 adoption is slow (and ongoing).

Recall: Private IP addresses help us conserve addresses.

- Private ranges allocated for networks that don't require Internet access.
  - 192.168.0.0/16
  - 10.0.0.0/8
  - 172.16.0.0/12

Weird fact: Your home network uses private IP addresses to conserve addresses.

- But...you do need Internet access!

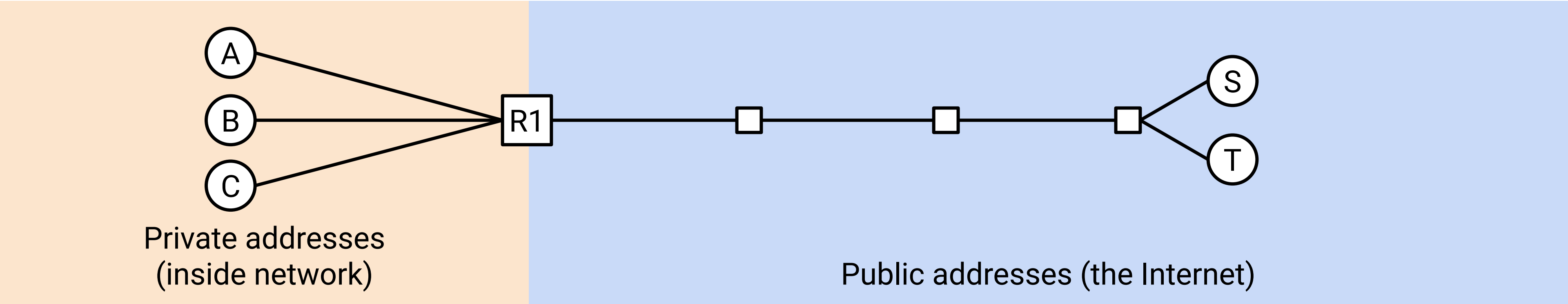
NAT (Network Address Translation)

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.

A, B, C are private addresses.

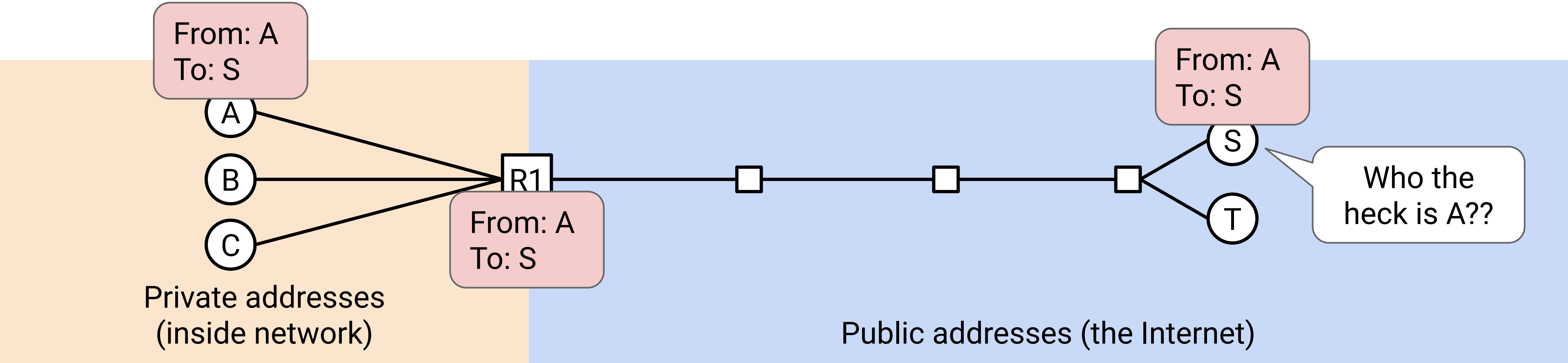
R1, S, T are unique public addresses.



Without NAT

Without NAT, if A sends a packet, replying to A is impossible.

Because A's IP address is private.



Basic NAT

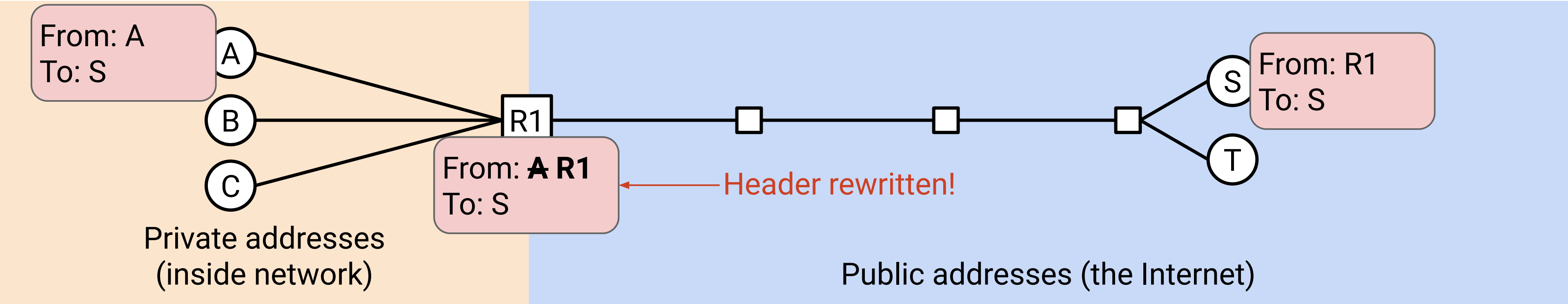
A sends an outgoing packet.

R1 *rewrites* the header so it's coming from a public IP instead of a private IP.

R1 keeps a table, so it remembers where to send any replies.

R1's NAT Table	
Inside	Outside
A	S

R1 leaves a note:
If I get replies from S,
send them to A.

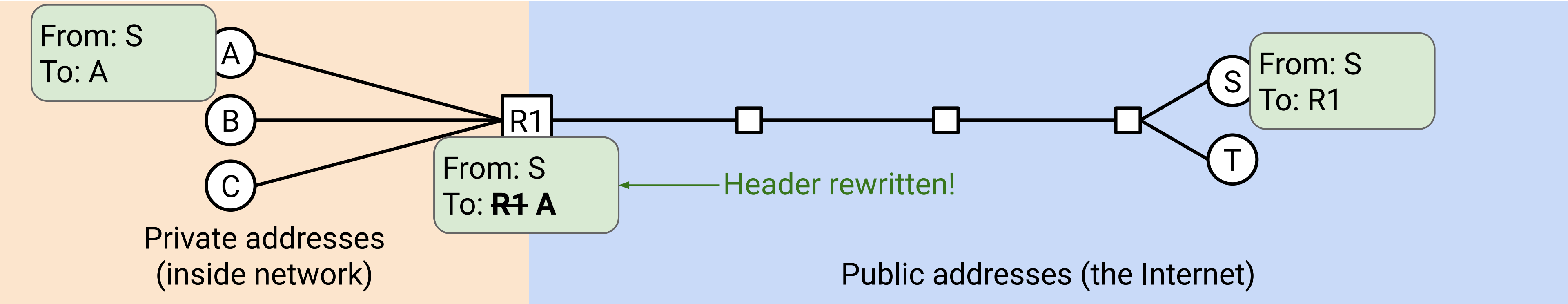


Basic NAT

S sends an incoming reply, addressed to R1 (public).

Router uses the table to replace R1 (public) with A (private).

R1's NAT Table	
Inside	Outside
A	S



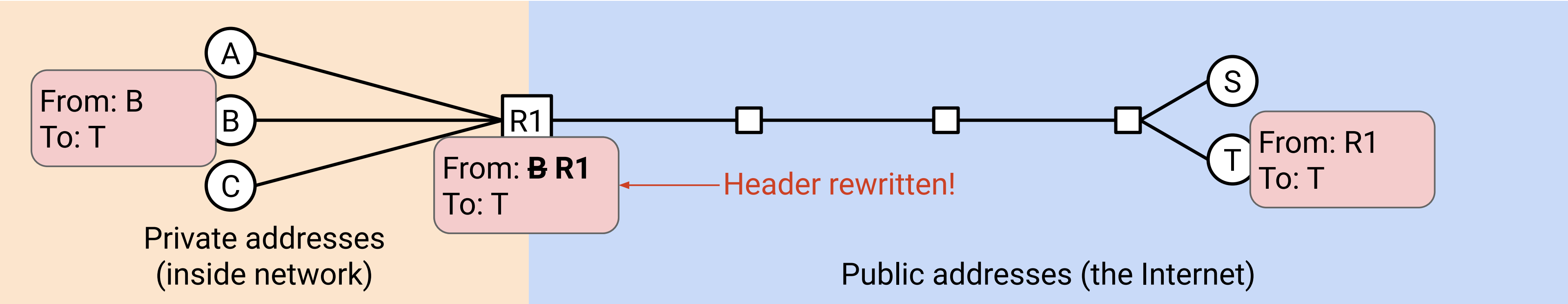
Basic NAT

Later, B sends an outgoing packet.

R1 *rewrites* the header and adds another entry to the table.

R1's NAT Table	
Inside	Outside
A	S
B	T

If I get replies from T, send them to B.

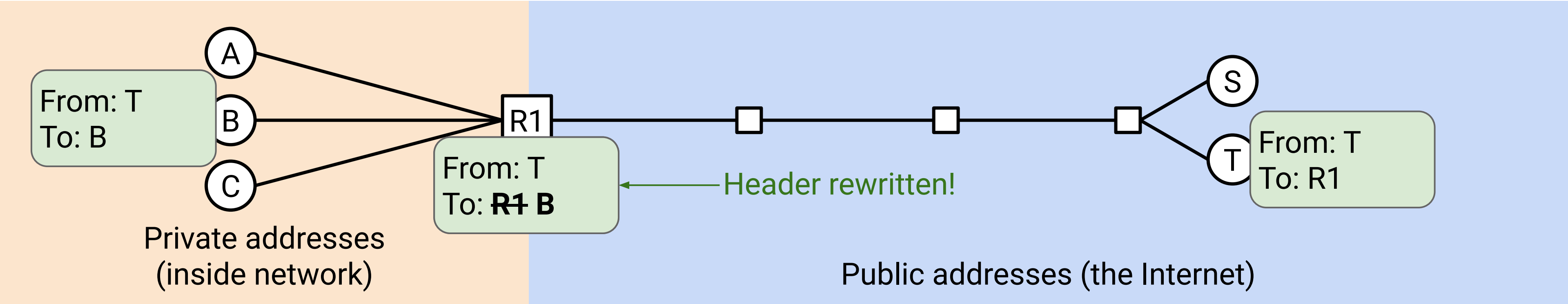


Basic NAT

T sends an incoming reply, addressed to R1 (public).

Router uses the table to replace R1 (public) with B (private).

R1's NAT Table	
Inside	Outside
A	S
B	T



Basic NAT

Outgoing packet: Replace *private* IP with *public* IP.

Incoming packet: Replace *public* IP with *private* IP.

- We have to restore B's original, private IP.
- B has no idea we rewrote the packet.

Router must give B the illusion that it's sending/receiving packets with its own IP.

From: ~~B~~ **R1**
To: T

Outgoing packet

R1's NAT Table	
Inside	Outside
A	S
B	T

From: T
To: ~~R1~~ **B**

Incoming packet

NAT with Ports

Lecture 18, CS 168, Fall 2024

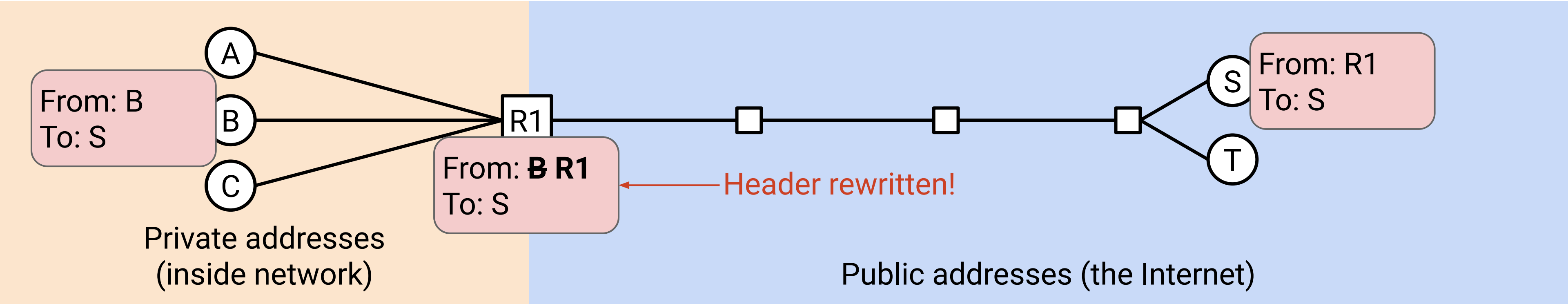
Problem with Basic NAT

What if A and B both wanted to talk to S?

Table is ambiguous! We don't know where to send packets from S.

R1's NAT Table	
Inside	Outside
A	S
B	S

If I get replies from S,
send them to B.

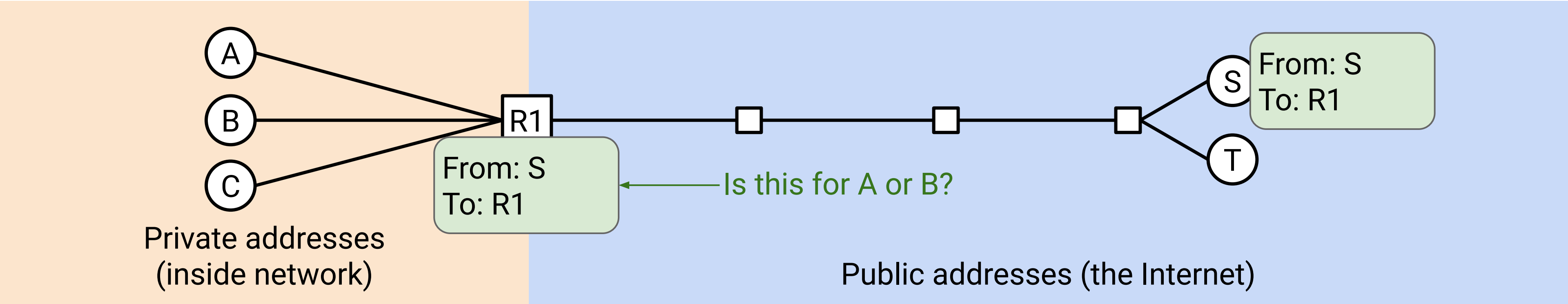


Problem with Basic NAT

What if A and B both wanted to talk to S?

Table is ambiguous! We don't know where to send packets from S.

R1's NAT Table	
Inside	Outside
A	S
B	S

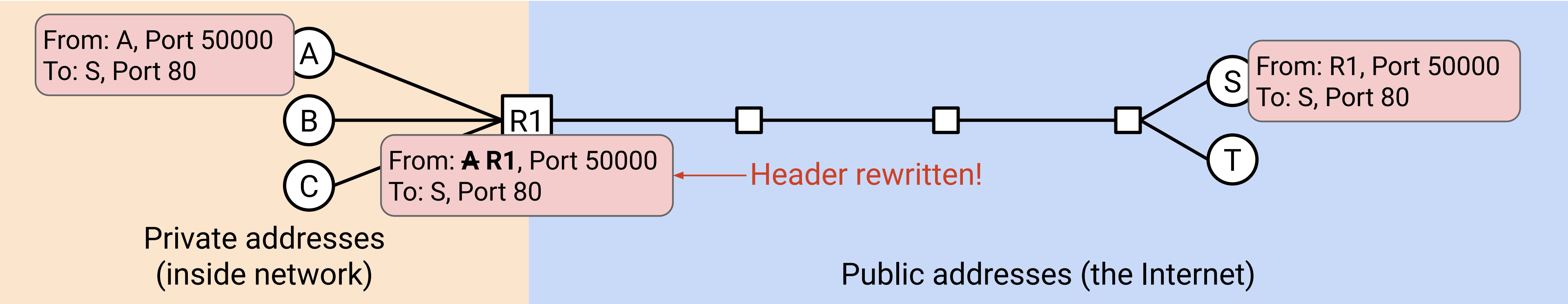


NAT with Ports

Solution: Keep track of port numbers in the table.

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80

If I get replies from S, Port 80, to me, Port 50000, send them to A.

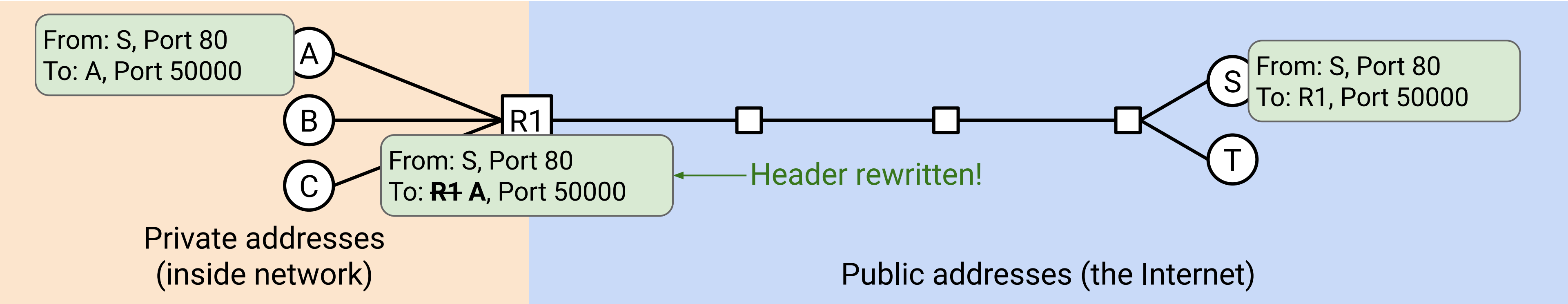


NAT with Ports

Solution: Keep track of port numbers in the table.

Use IPs *and inside source port* to rewrite incoming packets.

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80



NAT with Ports

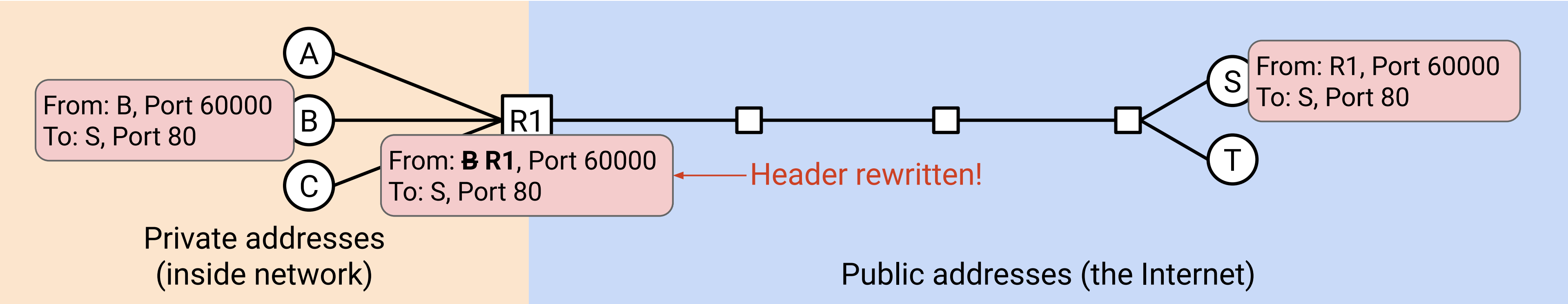
Solution: Keep track of port numbers in the table.

Use IPs *and inside source port* to rewrite incoming packets.

The inside port number will help us distinguish incoming packets.

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80
B, Port 60000	S, Port 80

If I get replies from S, Port 80,
to me, Port 60000, send them to B.



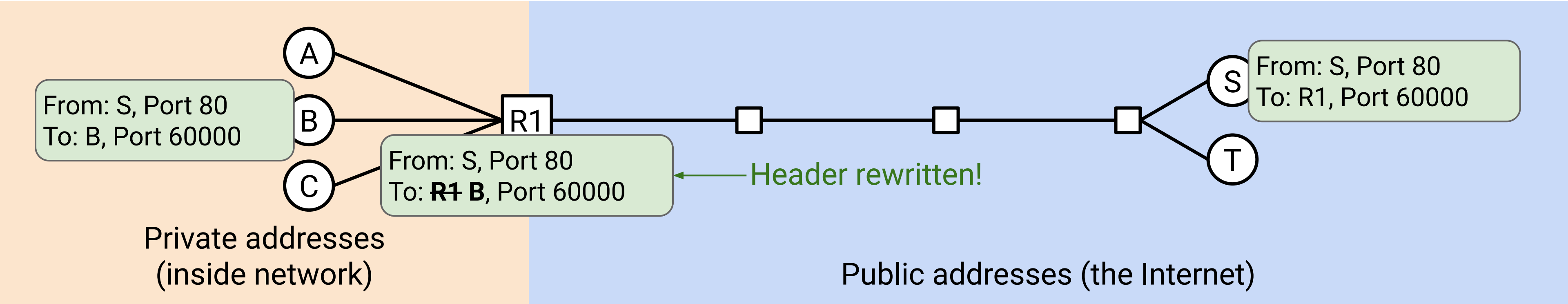
NAT with Ports

Solution: Keep track of port numbers in the table.

Use IPs *and inside source port* to rewrite incoming packets.

The inside port number will help us distinguish incoming packets.

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80
B, Port 60000	S, Port 80



NAT with Ports

Ports help distinguish connections to the same outside server.

- Use IPs *and inside source port* to rewrite incoming packets.

More generally, each entry of the table represents a *connection*.

- Recall: A Layer 4 connection is uniquely identified by a 5-tuple: (inside IP, inside port, protocol, outside IP, outside port).
- Use the table to associate incoming packets with a connection.

R1's NAT Table (Conceptual)	
Inside:	Outside:
A, Port 50000	S, Port 80
B, Port 60000	S, Port 80

R1's NAT Table (Actual)
5-tuples:
(A, 50000, TCP, S, 80)
(B, 60000, TCP, S, 80)

Rewriting Ports

Lecture 18, CS 168, Fall 2024

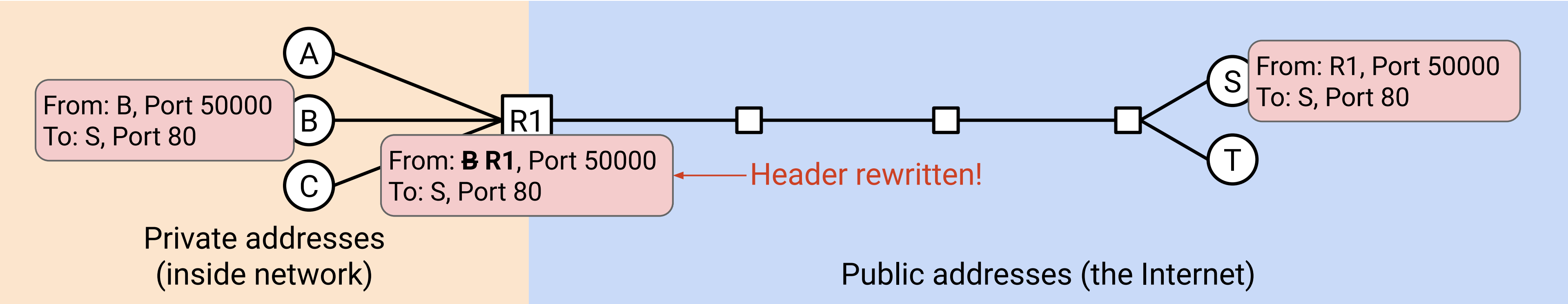
Problem with NAT with Ports

What if A and B both use the *same inside port*?

The only distinguishing value is the inside IP, and the router is rewriting it!

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80
B, Port 50000	S, Port 80

If I get replies from S, Port 80, to me, Port 50000, send them to B.

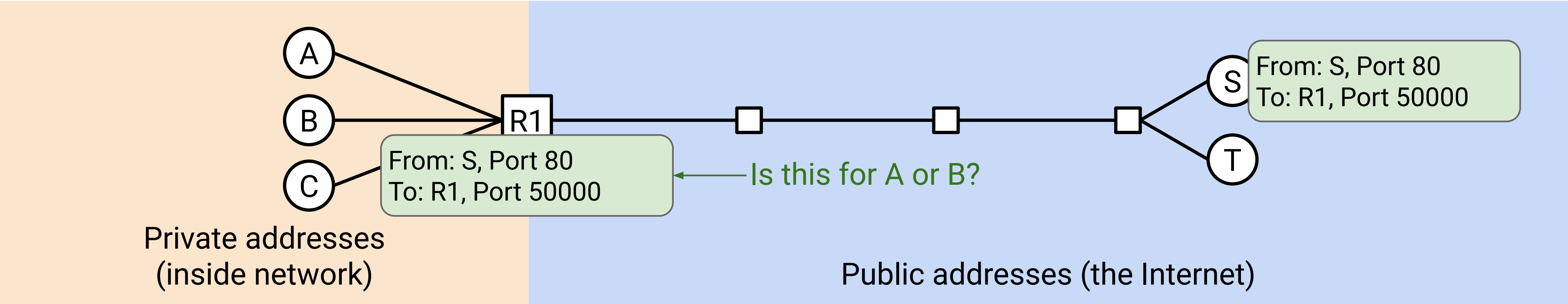


Problem with NAT with Ports

What if A and B both use the *same inside port*?

The only distinguishing value is the inside IP, and the router is rewriting it!

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80
B, Port 50000	S, Port 80



Problem with NAT with Ports

In both cases, outside IP and port are the same.

In both cases, inside port is the same.

Inside IP is the only difference...but the router rewrote it! Incoming packets all say R1.

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80
B, Port 50000	S, Port 80

From: S, Port 80
To: R1, Port 50000

Is this for A or B?

From: S, Port 80
To: R1, Port 50000

Is this for A or B?

Solution: "Fake" Port Numbers

Solution: The router can rewrite the *inside port* too.

- If A and B use the same inside port, make up a "fake" port number for B.
- Rewrite B's outgoing packets to use the fake port number.
- Incoming packets:
  - If they have the fake port number, they're for B.
  - If they have the original port number, they're for A.

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80
B, Port 50000 Fake Port 60000	S, Port 80

From: S, Port 80
To: R1, Port 50000

This is for A!

From: S, Port 80
To: R1, Port 60000

This is for B!

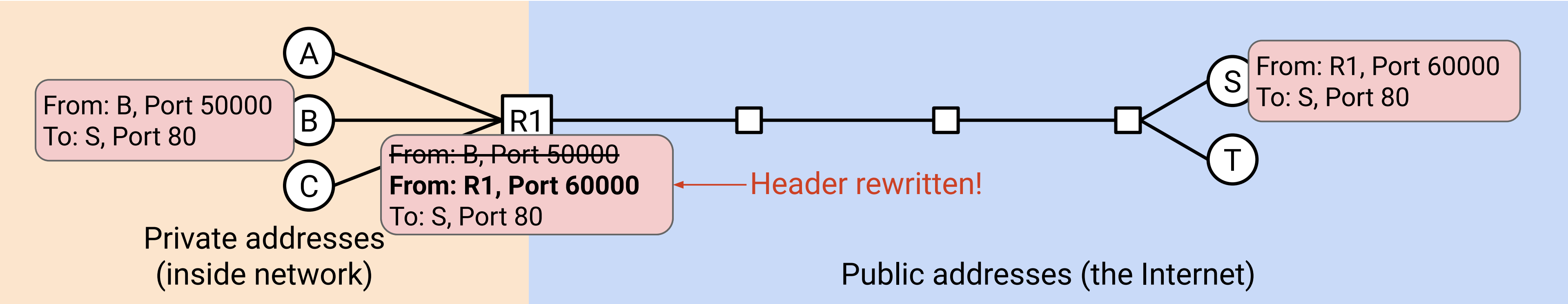
Note: "Fake" is not an official term for this.

NAT with Rewritten Ports

The router rewrites the port number to help distinguish connections that are otherwise identical.

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80
B, Port 50000 Fake Port 60000	S, Port 80

If I get replies from S, Port 80,
to me, Port 60000, send them to B.



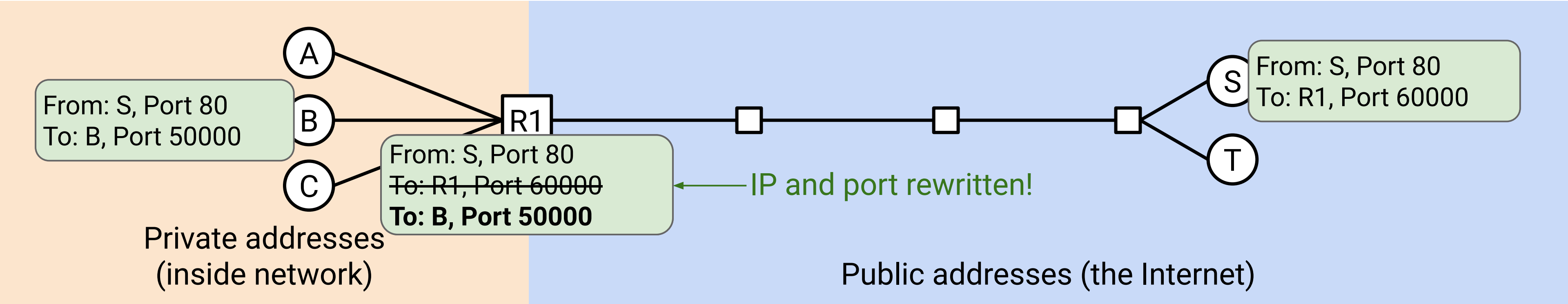
NAT with Rewritten Ports

The router rewrites the port number to help distinguish connections that are otherwise identical.

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80
B, Port 50000 Fake Port 60000	S, Port 80

Incoming packets are S, Port 80 in both cases.

If the inside port is 50000, packet is for A.
If the inside port is 60000 (fake port), packet is for B.



NAT with "Fake" Ports

Outgoing packet: Replace *original* port with *fake* port.

Incoming packet: Replace *fake* port with *original* port.

- We have to restore the original port. B doesn't know we added the fake port.

~~From: B, Port 50000~~
From: R1, Port 60000
To: S, Port 80

Outgoing packet

R1's NAT Table	
Inside	Outside
A, Port 50000	S, Port 80
B, Port 50000 Fake Port 60000	S, Port 80

From: S, Port 80
~~To: R1, Port 60000~~
To: B, Port 50000

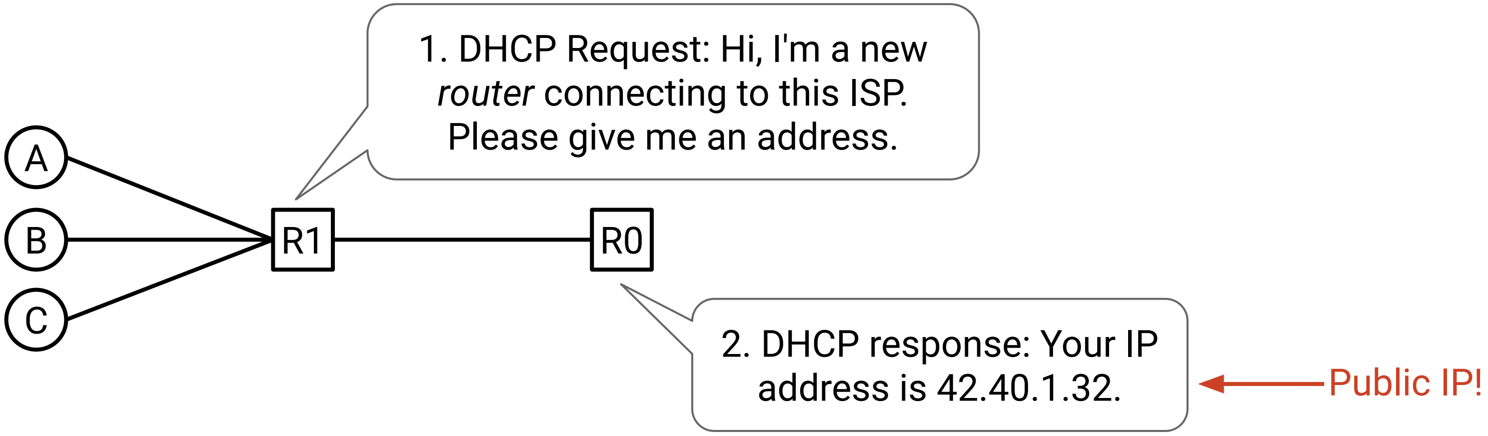
Incoming packet

Implementing NAT

Lecture 18, CS 168, Fall 2024

Implementing NAT

1. Home router sends a DHCP request. Asks ISP for an IP address.
2. ISP sends a DHCP response. Allocates an IP address to the home router.

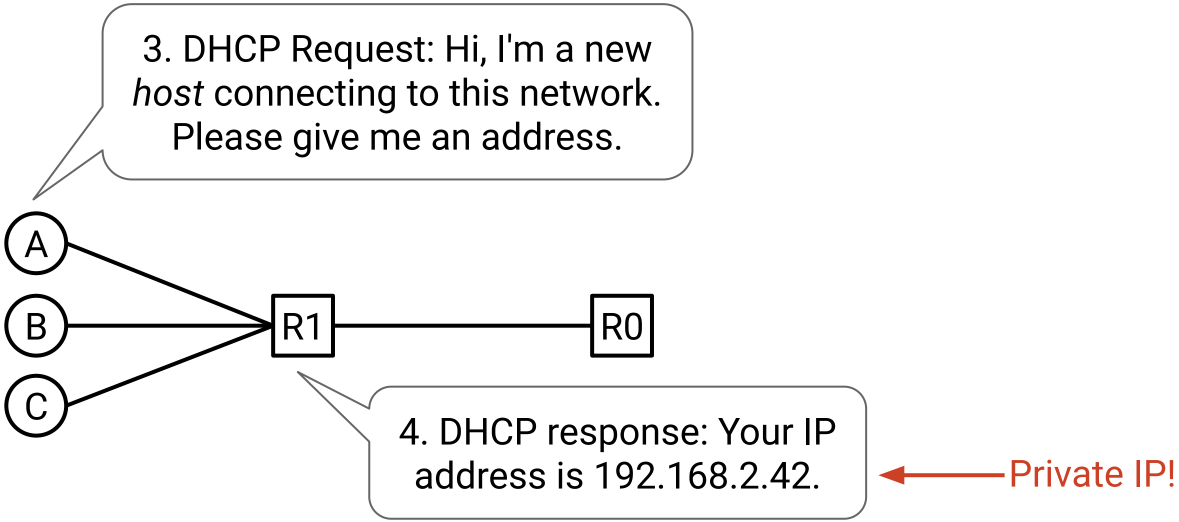


Implementing NAT

3. Host sends a DHCP request. Asks home router for an IP.

4. Home router sends a DHCP response. Gives a private IP to the home router.

Home router uses NAT to convert private IP to public IP.



Types of NAT

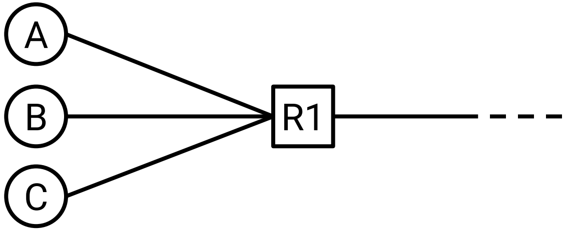
We just saw Port Address Translation (PAT).

- The most complex, widely-used mode of NAT.

Other modes of NAT exist.

- Simpler modes for one-to-one address translation.

R1's NAT Table		
Host	Private	Public
A	10.0.0.1	42.0.2.1
B	10.0.0.2	42.0.2.2
C	10.0.0.3	42.0.2.3



In this simpler NAT mode, every host has its own private and public IP.

So R1 just needs to change the private IP to that host's corresponding public IP.

Implementing NAT

NAT requires routers to do extra work.

- Rewrite headers.
- Read (and maybe rewrite) the Layer 4 header.
- Maintain a *connection state* table.

NAT increases complexity of packet forwarding.

- More CPU cycles per packet.
- More memory per connection.

Where is NAT Used?

NAT increases complexity, so it's performed as close to the edge of the network as possible.

- Small-scale NAT is used in almost every IPv4 network.

As IPv4 addresses ran out, ISPs didn't have enough addresses for each customer.

- ISPs had to run Carrier Grade NAT (CGNAT).
- More complex – many more connections to maintain state for.

NAT is generally not used for IPv6. There are enough addresses!

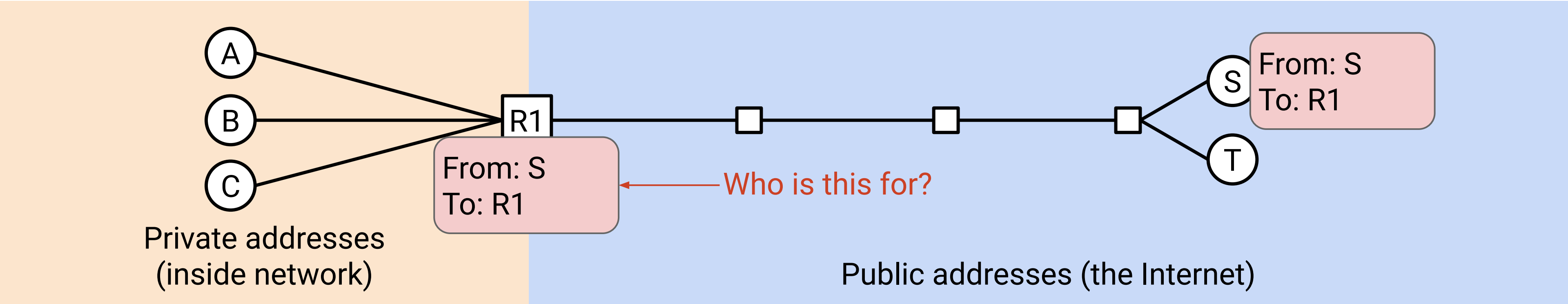
Inbound Connections

So far, we've assumed connections are initiated by the client (inside).

What if someone from outside initiates a connection?

- Basic NAT doesn't support inbound connections!
- Can't initiate a connection to someone with a private address.

R1's NAT Table	
Inside	Outside



Inbound Connections

Basic NAT doesn't support inbound connections.

- Breaks the end-to-end principle!
- This is usually okay.
- On your computer, you usually initiate all the connections.

To support inbound connections, the router needs a *port mapping table*.

- "If someone initiates a connection on Port 50000, it's for A."
- A can manually specify the port it's listening on.
- Dynamic protocols exist for auto-configuring open ports.
    - UPnP *(Universal Plug-n-Play)*.
    - NAT-PMP *(NAT Port Mapping Protocol)*.

NAT: Security Implications

NAT disallows inbound connections by default.

- Could be thought of as a security feature, similar to firewalls.
- More of a side effect than an intentional security policy.

NAT can help preserve client privacy.

- Without NAT: Server sees host's address. *(From: A, To: S)*
  - If IP address is derived from MAC address, server can learn the exact computer being used.
- With NAT: Server sees router IP address. *(From: R1, To: S)*
- Again, more of a side effect.
- Temporary/privacy addresses exist for IPv6.

TLS: Secure Bytestreams

Lecture 18, CS 168, Fall 2024

TCP is Vulnerable

An attacker could read or modify TCP packets.

- Malicious router.
- Attacker sniffing packets on a wire.

An attacker could impersonate a server.

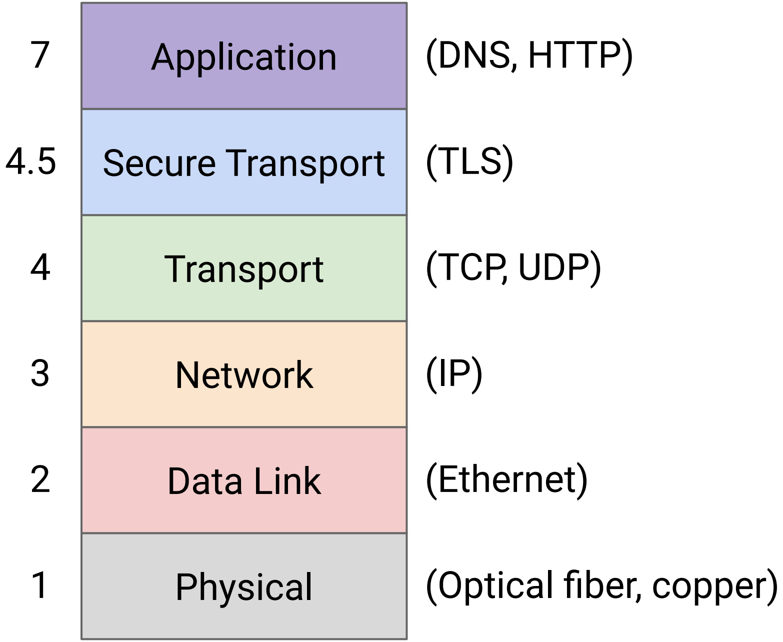
- You do a DNS lookup for www.bank.com.
- Attacker changes the DNS response, mapping www.bank.com to 6.6.6.6.
- When you connect to the website, you're talking to the attacker!

Secure Bytestreams

TLS adds security on top of TCP.

- Relies on Layer 4 bytestream.
- Provides a *secure* bytestream to Layer 7.
- Same abstraction, but data in the bytestream is now encrypted.
- HTTPS runs on top of TLS. *(Uses port 443.)*
  HTTP runs on top of TCP. *(Uses port 80.)*

(Note: It's Layer 4.5 because 5 and 6 are obsolete, and unrelated to security.)



TLS Handshake

TLS uses cryptography to protect messages sent over the bytestream.

- Messages are encrypted.
- Messages are tamper-proof. *(Look up "message authentication codes" if curious.)*

TLS starts with a handshake.

- Exchanges secret keys.
- Verifies identity of the server (stops impersonators).

Handshake runs over TCP bytestream. No need to think about packets!

TLS Handshake (1/5)

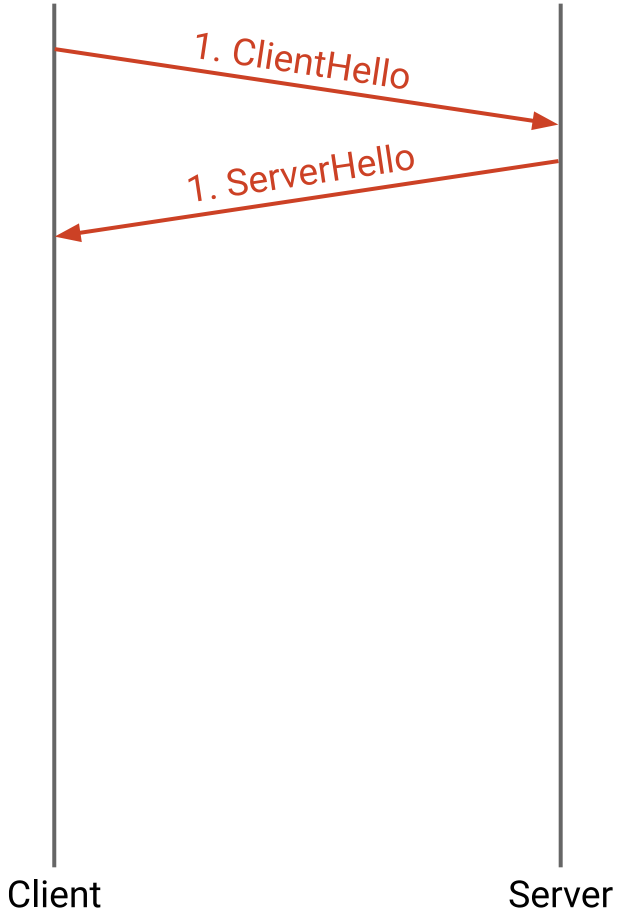
Step 1: Client and server exchange hellos.

Exchange random numbers, to ensure that every handshake results in different keys.

- Don't want to use the same key every time (e.g. if attacker hacks us and learns the key).

Agree on which cryptographic schemes to use.

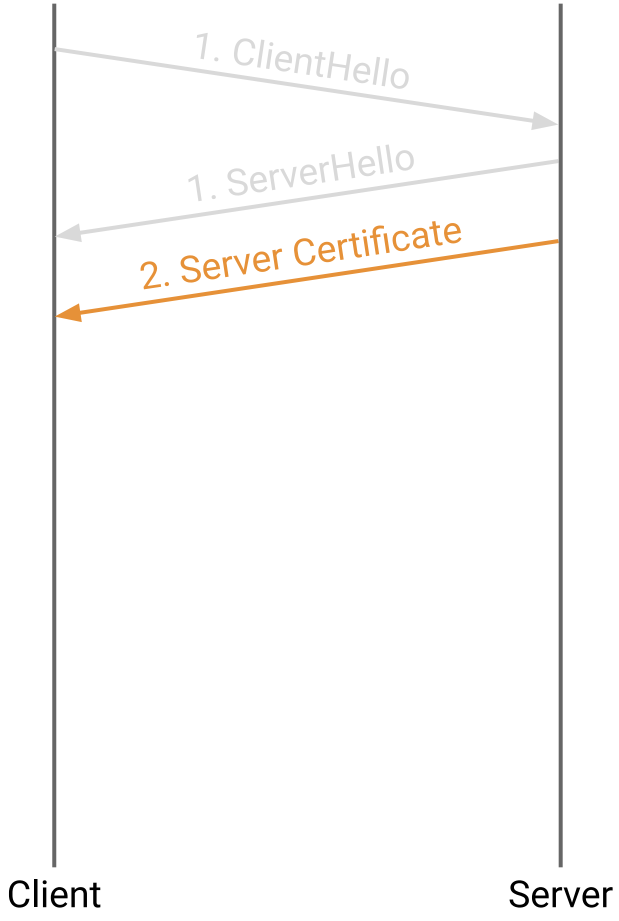
- Client sends a list of schemes it can use.
- Server picks one.



TLS Handshake (2/5)

Step 2: Server sends certificate of authenticity.

- Allows client to verify it's talking to the real server, and not an impersonator.
- Some additional steps needed to verify. Not described here.



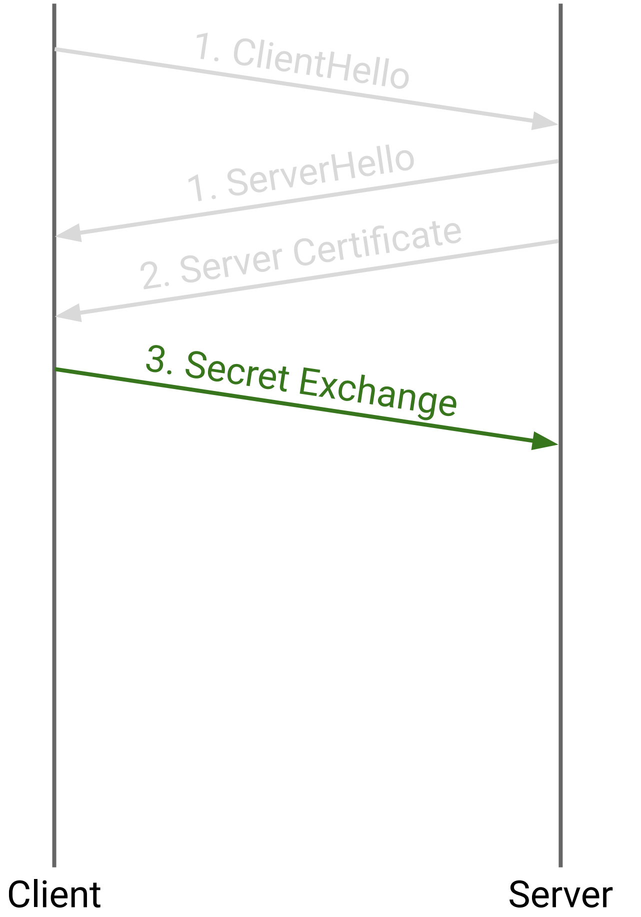
TLS Handshake (3/5)

Step 3: Secret exchange.

- Client and server use cryptography to derive a secret key that only the two of them know.

Example: Use RSA public-key encryption.

- Client encrypts secret with server's public key.
- Server can use private key to learn the secret.
- Attacker cannot learn the secret (doesn't know server's private key).



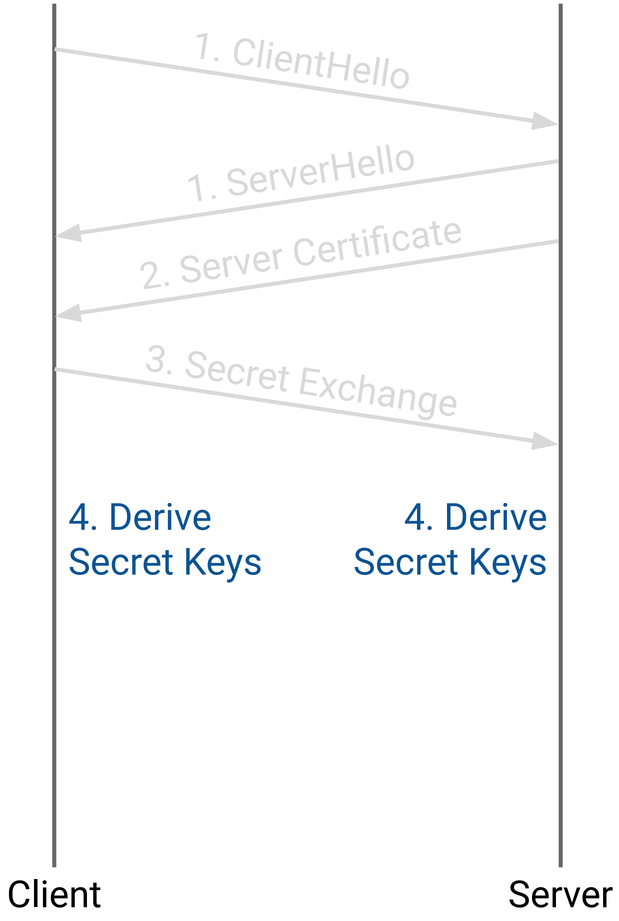
TLS Handshake (4/5)

Step 4: Secret key derivation.

- Server and client each derive key based on random numbers and the shared secret.

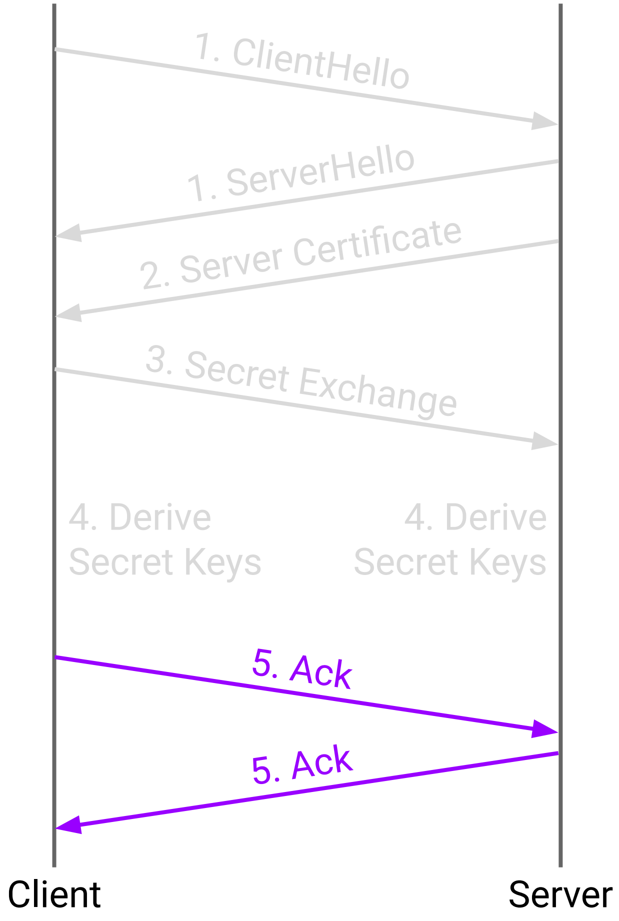
Derivation is done locally and independently by the client and server.

- No messages sent over network in this step!



Step 5: Exchange acknowledgments.

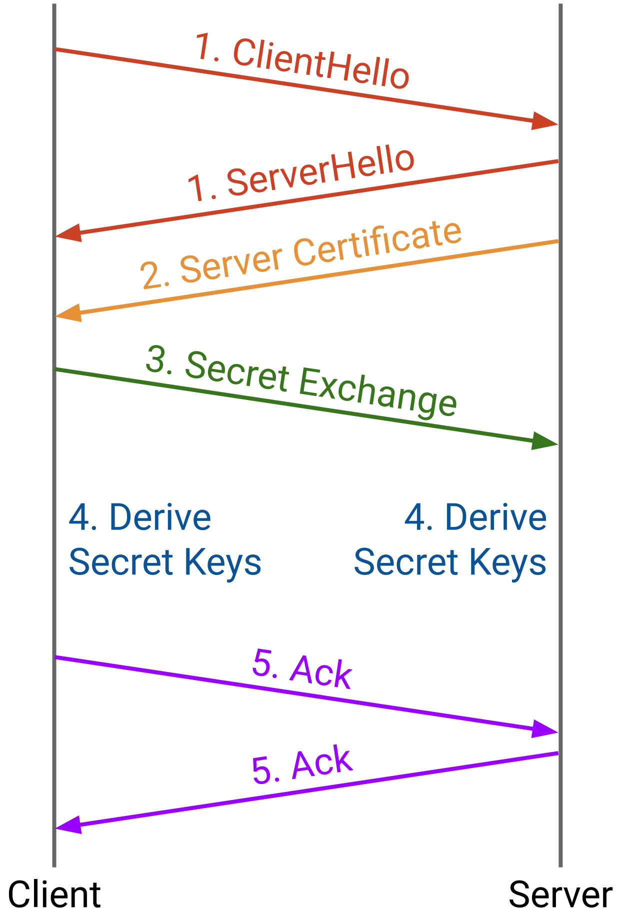
- Use cryptography to make sure client and server derived the same secret keys.



TLS Handshake

After the handshake, all future messages are protected by the secret keys.

- Encrypted and tamper-proof.
- Applications can send data over the secured bytestream.



End-to-End Walkthrough

Lecture 18, CS 168, Fall 2024

ARP: Connecting Layers 2 and 3

DHCP: Joining a New Network

NAT: Network Address Translation

- Basic NAT
- NAT with Ports
- Rewriting Ports
- Implementing NAT

TLS: Secure Bytestreams

End-to-End Walkthrough

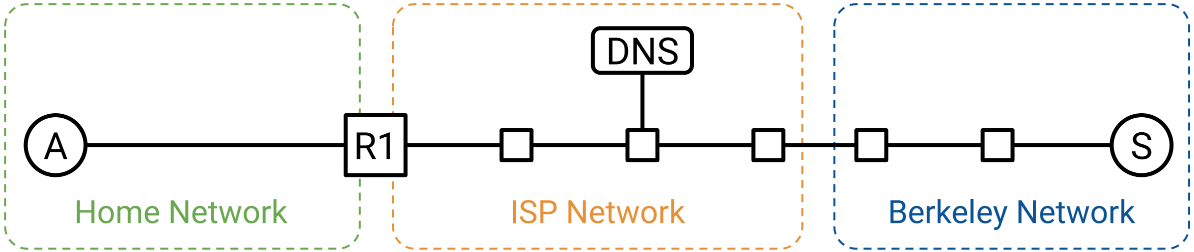
End-to-End Walkthrough

Goal: See exactly what happens when we:

- Turn on our computer, and plug it into an Ethernet network.
- Type www.berkeley.edu into our web browser.

We'll assume we don't need to turn on the Internet from scratch.

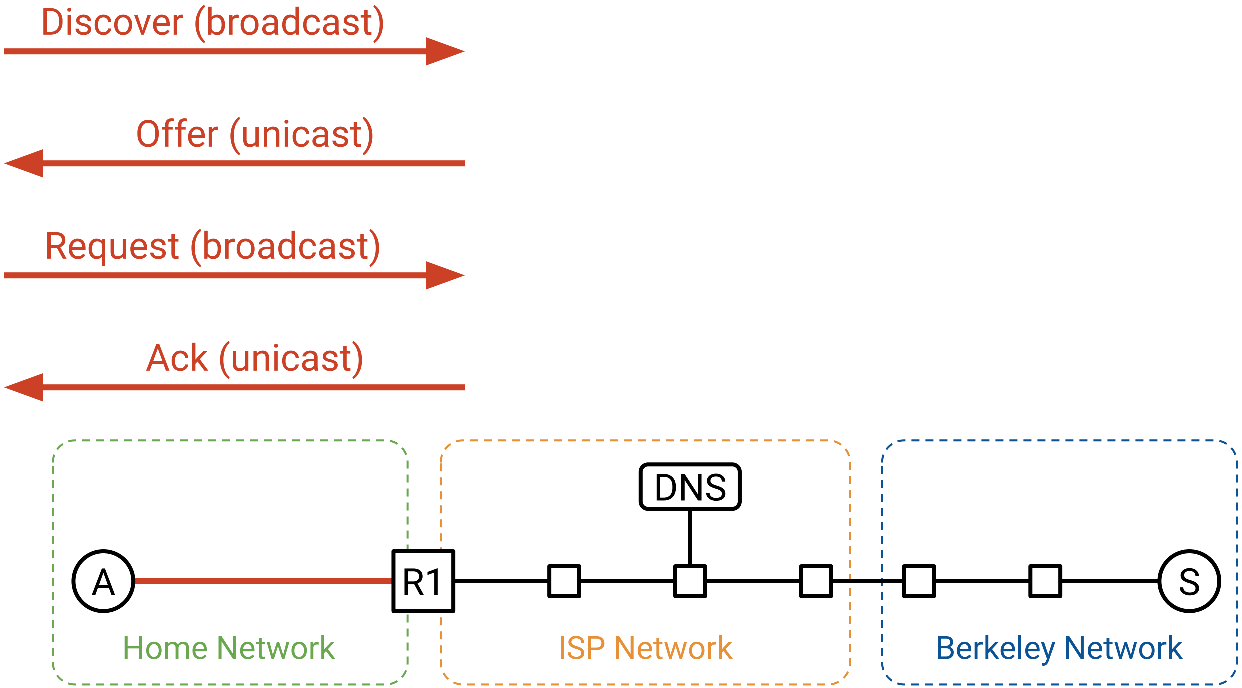
- Routers' forwarding tables already populated.



Step 1/4: DHCP

We connect to the Ethernet network and make a DHCP request.

The router responds with: IP address, subnet mask, default gateway, DNS server.



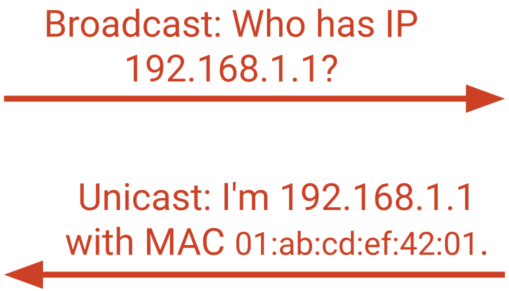
A's Configuration	
My IP:	192.168.1.2
Subnet:	/24
Gateway:	192.168.1.1
DNS:	8.8.8.8

Step 2/4: ARP

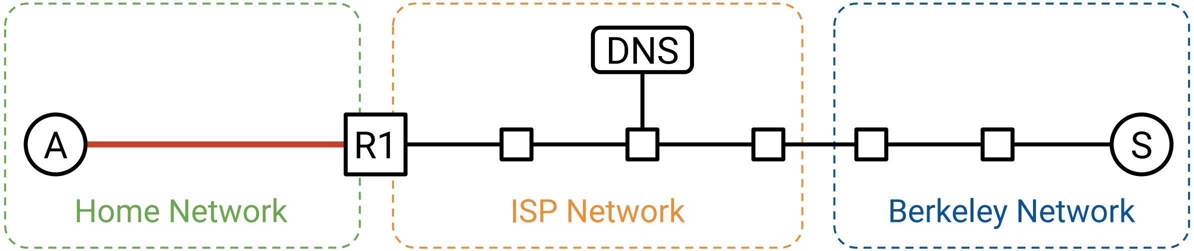
We're about to send some non-local packets, e.g. DNS request to 8.8.8.8.

We need to find the router on our local network. We can use ARP!

A's ARP Table	
IP:	MAC:
192.168.1.1	01:ab:cd:ef:42:01



A's Configuration	
My IP:	192.168.1.2
Subnet:	/24
Gateway:	192.168.1.1
DNS:	8.8.8.8



Step 3/4: DNS Lookup

We can now build a DNS request packet, to find the IP address of www.berkeley.edu.

Source MAC:	`23:45:67:89:ab:cd`	Our MAC (burned into hardware).
Destination MAC:	`01:ab:cd:ef:42:01`	Router MAC (learned from ARP).
Source IP:	192.168.1.2	Our IP (learned from DHCP).
Destination IP:	8.8.8.8	DNS server IP (learned from DHCP).
Source Port:	50239	Random ephemeral source port.
Destination Port:	53	Well-known DNS port.
	DNS Payload	A record in Question section. Name=www.berkeley.edu, Value=blank

Step 3/4: DNS Lookup

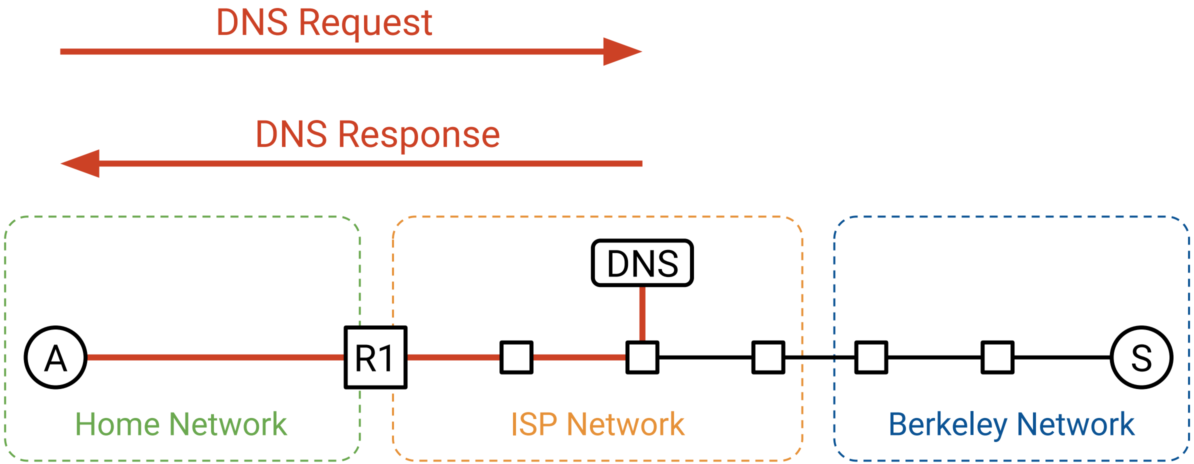
Note: NAT might rewrite headers, but we never see this.

Note: Recursive resolver might have to ask other name servers.

A's DNS Cache	
Domain:	IP:
www.berkeley.edu	141.193.213.21

A's ARP Table	
IP:	MAC:
192.168.1.1	01:ab:cd:ef:42:01

A's Configuration	
My IP:	192.168.1.2
Subnet:	/24
Gateway:	192.168.1.1
DNS:	8.8.8.8



Step 4/4: Connect to Website

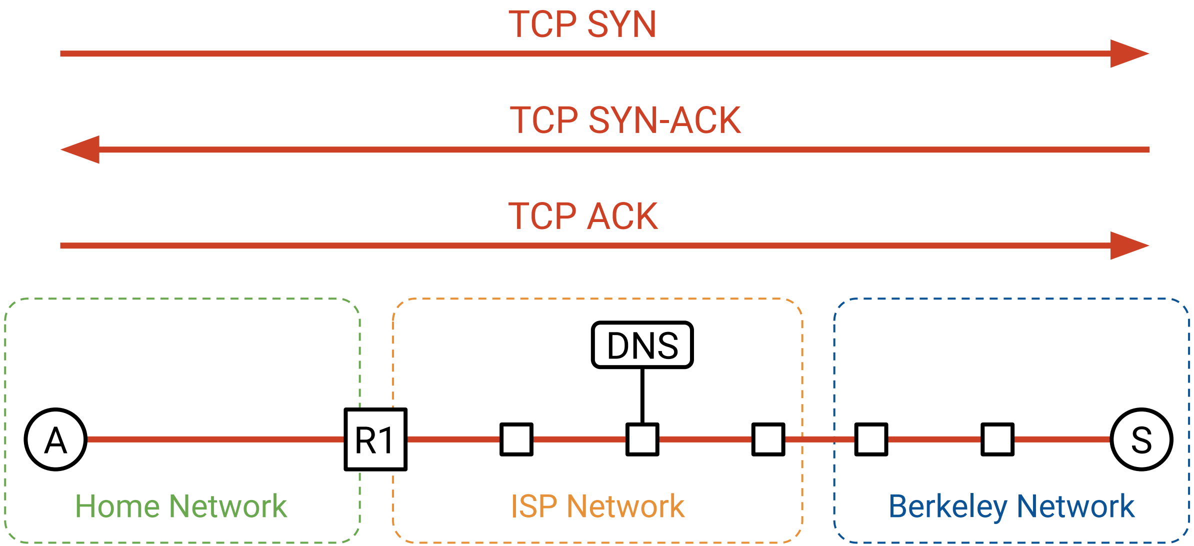
Now that we know www.berkeley.edu's IP address, we can send packets there.

Use 3-way handshake to start a TCP connection.

A's DNS Cache	
Domain:	IP:
www.berkeley.edu	141.193.213.21

A's ARP Table	
IP:	MAC:
192.168.1.1	01:ab:cd:ef:42:01

A's Configuration	
My IP:	192.168.1.2
Subnet:	/24
Gateway:	192.168.1.1
DNS:	8.8.8.8



Step 4/4: Connect to Website

We can now build an HTTP request packet.

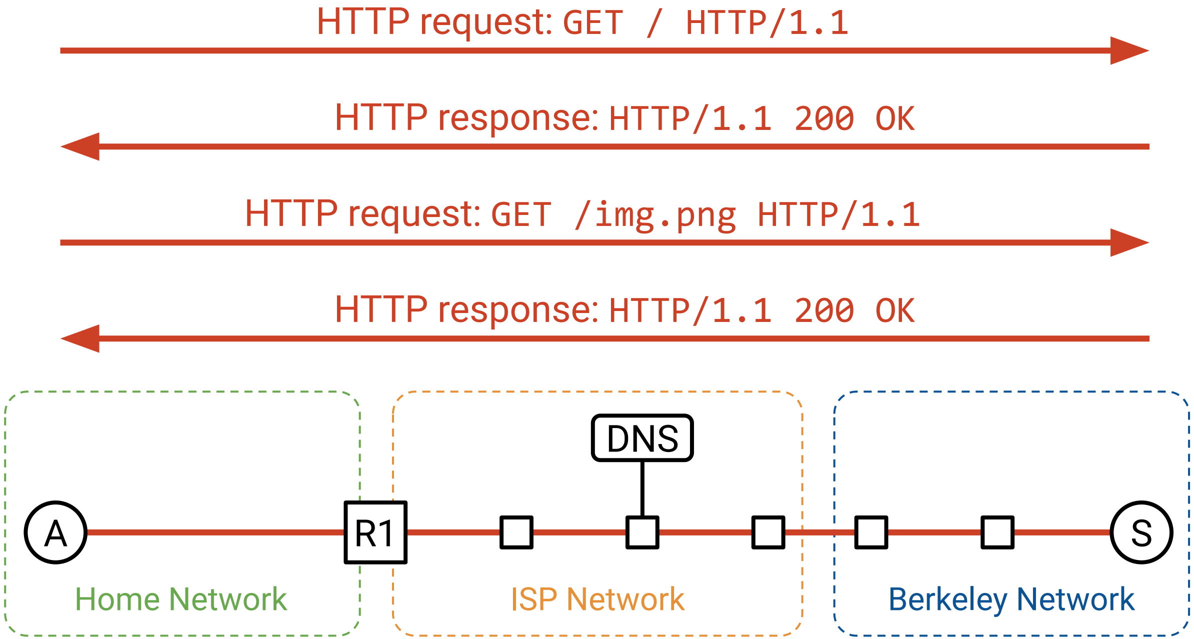
Source MAC:	`23:45:67:89:ab:cd`	Our MAC (burned into hardware).
Destination MAC:	`01:ab:cd:ef:42:01`	Router MAC (learned from ARP).
Source IP:	192.168.1.2	Our IP (learned from DHCP).
Destination IP:	141.193.213.21	www.berkeley.edu IP (learned from DNS).
Source Port:	61098	Random ephemeral source port.
Destination Port:	80	Well-known HTTP port.
	HTTP payload	`GET / HTTP/1.1`

Step 4/4: Connect to Website

Server sends an HTTP response with an HTML page.

Page might trigger further HTTP requests, pipelined in the same TCP connection.

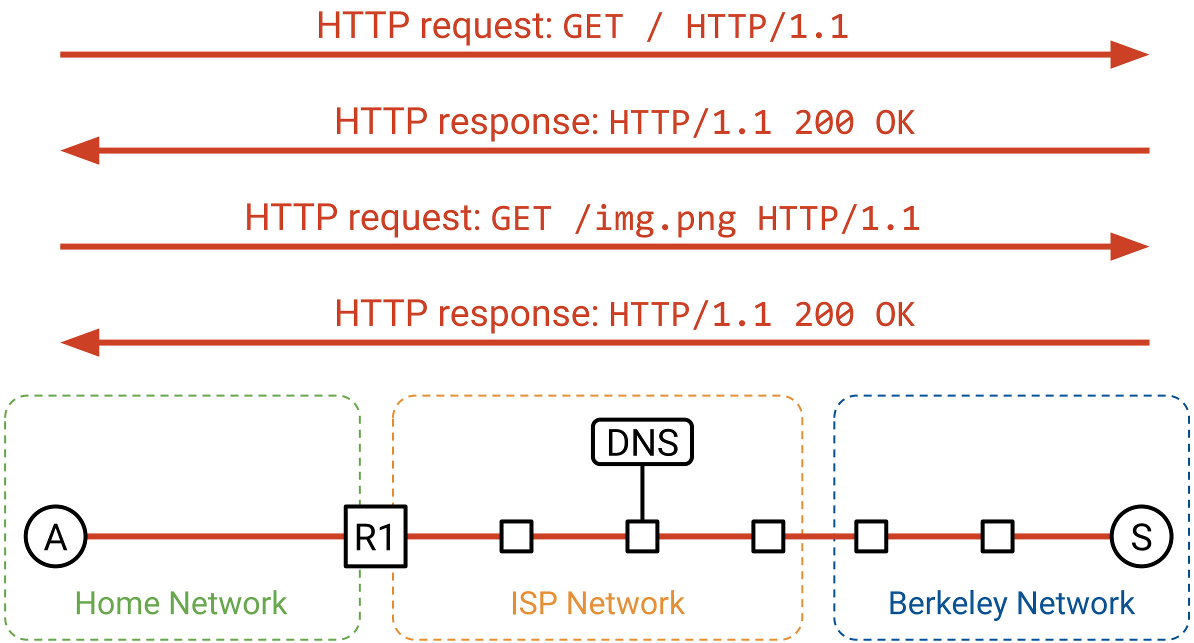
Note: TCP provides a bytestream abstraction, so each HTTP request/response could be multiple packets.



Step 4/4: Connect to Website

HTTP messages end in newlines. Lets us separate messages in the bytestream.

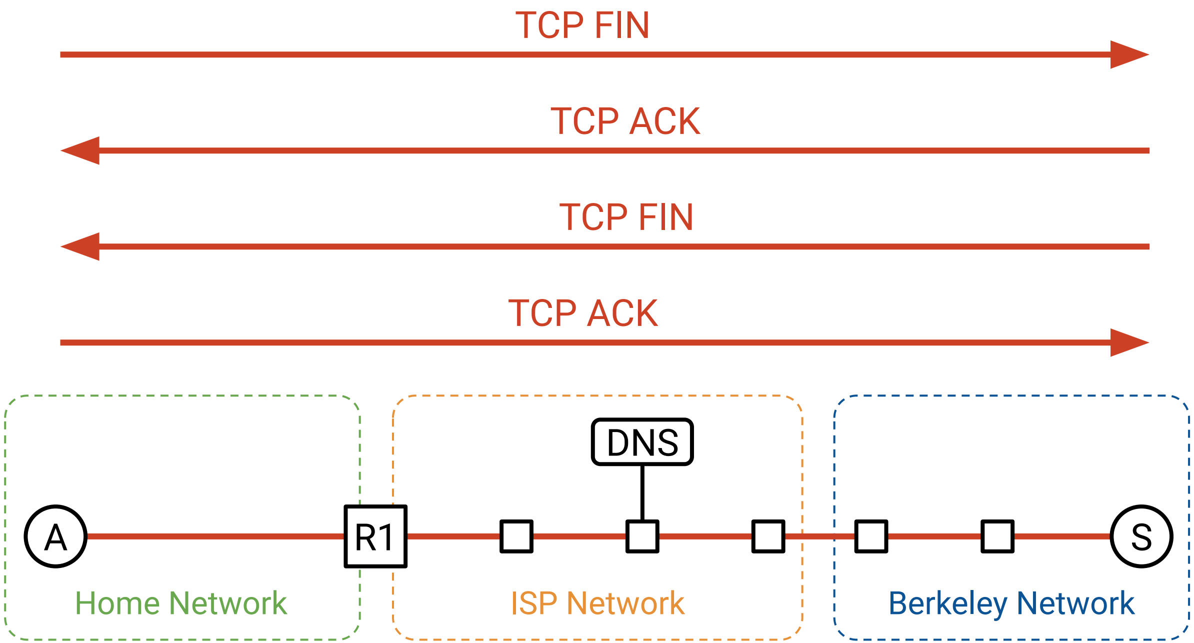
Headers (e.g. Content-Length) tell us how much memory to allocate for the payload.



Step 4/4: Connect to Website

TCP connection stays open for pipelining requests.

Eventually, client or server decides to close the connection.

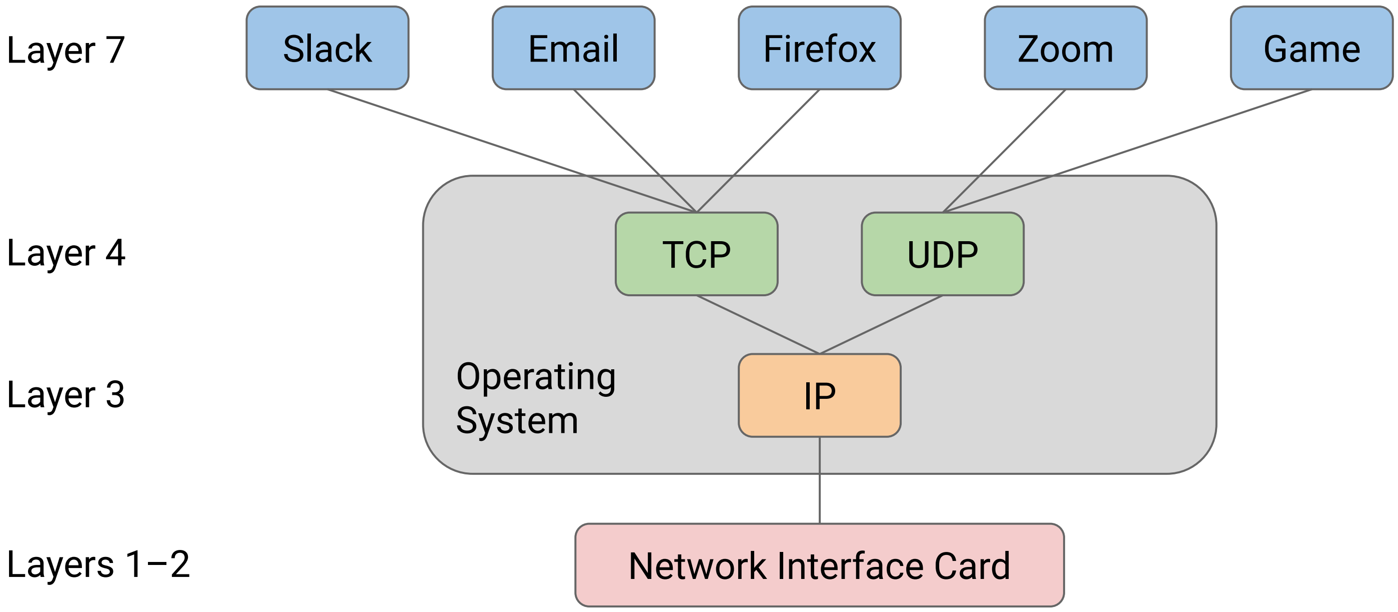


End-to-End Walkthrough – Operating System View

Layer 1–2 are implemented on the Network Interface Card (NIC), in hardware.

Layers 3–4 are implemented in the operating system (OS), in software.

Layer 7 are the applications running on top of the OS, in software.



End-to-End Walkthrough – Operating System View

Step 1: DHCP.

- Done in the operating system (OS). Application (browser) doesn't need to know.

Step 2: ARP.

- Also done in the OS.

Step 3: DNS lookup.

- Browser calls `getaddrinfo` to make the OS perform the lookup.

Step 4: Connect to website.

- Browser opens a TCP connection.
- Browser sends HTTP requests and receives HTTP responses over the bytestream.
- OS implements TCP, e.g. splitting/reordering packets.

Socket API

The **socket** abstraction lets programmers interact with the network.

- **Create**: Constructor for a new socket object.
- **Connect**: Initiate a TCP connection.
- **Listen**: Allows others to connect to us on a specific port.
- **Write**: Send bytes into the bytestream.
- **Read**: Read *N* bytes from the bytestream.

OS associates each socket with a port number.

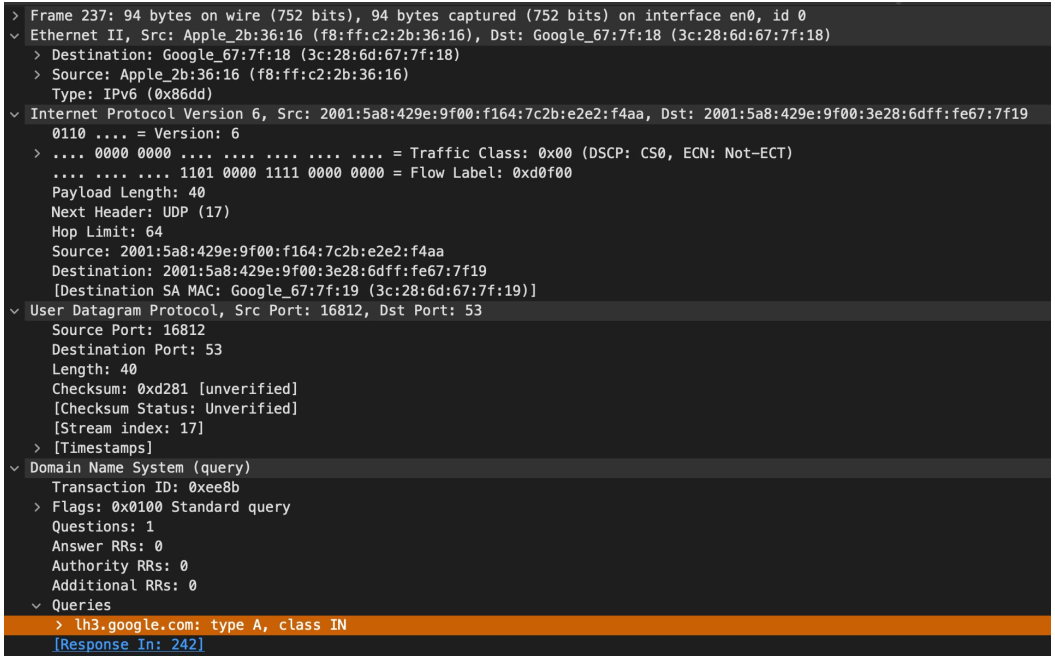
- OS uses port number to send incoming packets to the correct socket.

Note: Sockets exist entirely in software.

Viewing Packets

You can use programs like tshark and wireshark to look at packets.

Often real-world complexities like TLS, or HTTP/3.0 over QUIC.



Revisiting Layering

Layering gives us a powerful way to solve specific problems, without exposing everyone to the complexity of solving them.

- We didn't discuss the electrical engineering and physics at Layer 1.
- But we relied on Layer 1 working when discussing higher layers.

Layering lets us evolve networking for new applications.

Revisiting Layering

We can build more layers on top of Layer 7.

- Many applications build the same things on top of HTTP.
  - Example: Multiplexing multiple data retrievals on the same HTTP connection.
  - Example: Bi-directionally streaming data between a client and a server.
- Common frameworks exist, so you don't always have to start from basic HTTP.

Example: Remote Procedure Call (RPC) libraries.

- RPC: Call a function that runs on a remote computer.
- Examples: Apache Thrift, gRPC.
- Allows us to build applications without repeating ourselves.

Revisiting Layering

Layering allows us to abstract away lower-level details.

- Code to say hello to a remote server is two function calls!
- Did not need to think about: Addressing, headers, DNS, TCP, HTTP, gRPC, etc.

```
func main() {
	flag.Parse()
	// Set up a connection to the server.
	conn, err := grpc.Dial(*addr, grpc.WithTransportCredentials(insecure.NewCredentials()))
	if err != nil { log.Fatalf("did not connect: %v", err) }
	defer conn.Close()
	c := pb.NewGreeterClient(conn)

	// Contact the server and print out its response.
	ctx, cancel := context.WithTimeout(context.Background(), time.Second)
	defer cancel()
	r, err := c.SayHello(ctx, &pb.HelloRequest{Name: *name})
	if err != nil { log.Fatalf("could not greet: %v", err) }
	log.Printf("Greeting: %s", r.GetMessage())
}
```

Programmer can ignore everything at lower layers, and focus on their own application logic.