

Ch 3: Networking Services: NAT, DHCP, DNS, Multicasting, NTP

Magda El Zarki

Prof. of CS

Univ. of CA, Irvine

Email: elzarki@uci.edu

<http://www.ics.uci.edu/~magda>

Network Address Translation - NAT

Translation from internal, hidden, IP addresses to valid external Internet addresses.

Saves costs

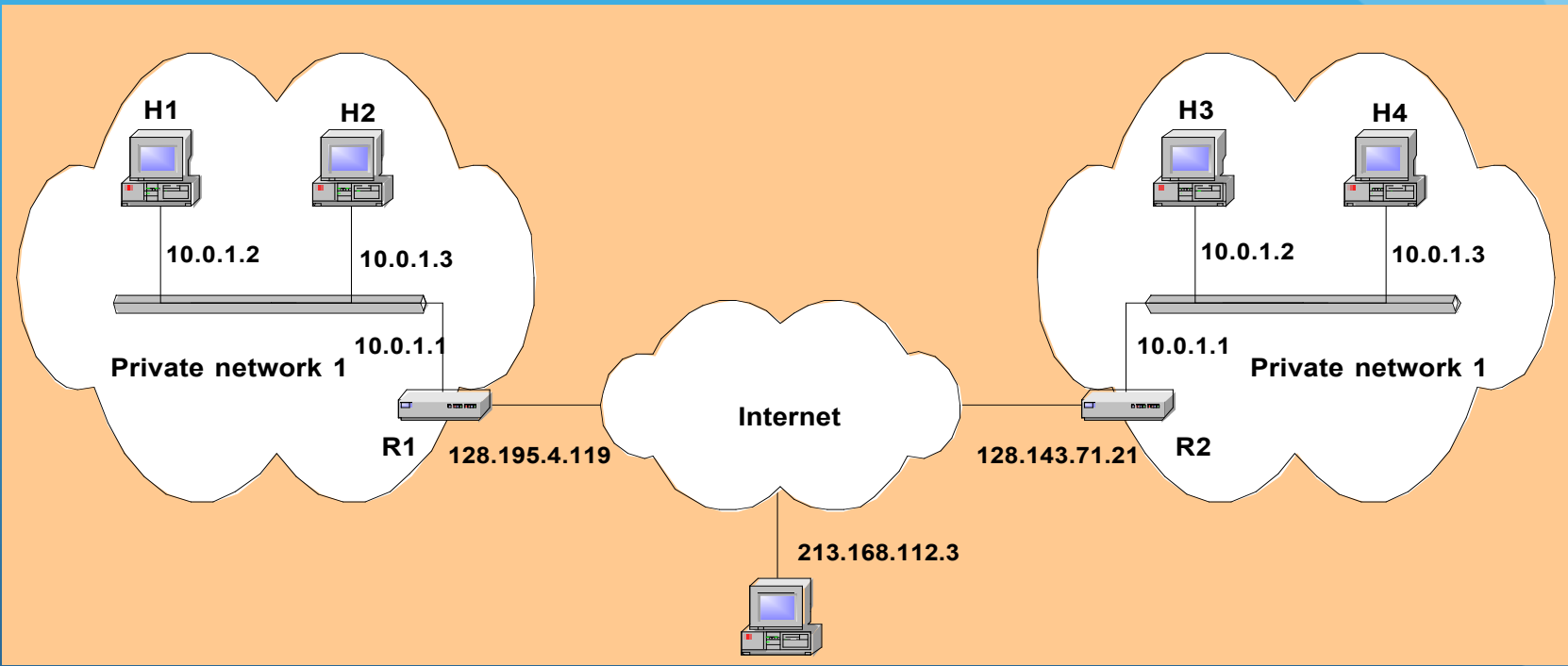
Increases the number of available addresses

Secure - devices behind a firewall

Private Network

- *Private IP* network is an IP network that is not accessible to the Internet. It is closed and self contained.
- IP addresses in a private network can be assigned arbitrarily.
 - Not registered and not guaranteed to be globally unique
- Generally, private networks use addresses from the following experimental address ranges (*non-routable addresses*):
 - 10.0.0.0 – 10.255.255.255
 - 172.16.0.0 – 172.31.255.255
 - 192.168.0.0 – 192.168.255.255
- Private networks can be connected to the Internet via a router but the devices are not directly accessible from the outside.

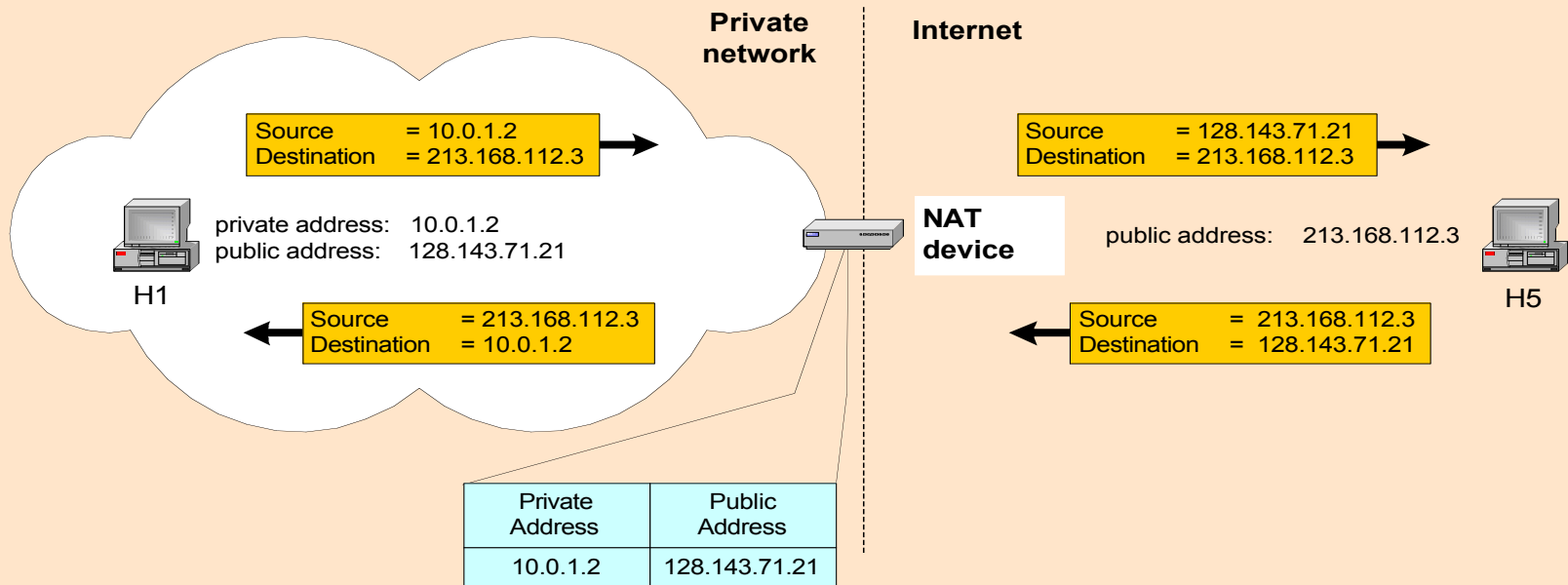
Private Addresses



Network Address Translation (NAT)

- NAT is a router function where IP addresses (and possibly port numbers) of IP datagrams are replaced at the boundary of a private network
- NAT is a method that enables hosts on private networks to communicate with hosts on the Internet
- NAT is run on routers that connect private networks to the public Internet, to replace the IP address-port pair of an IP packet with another IP address-port pair.

Basic operation of NAT

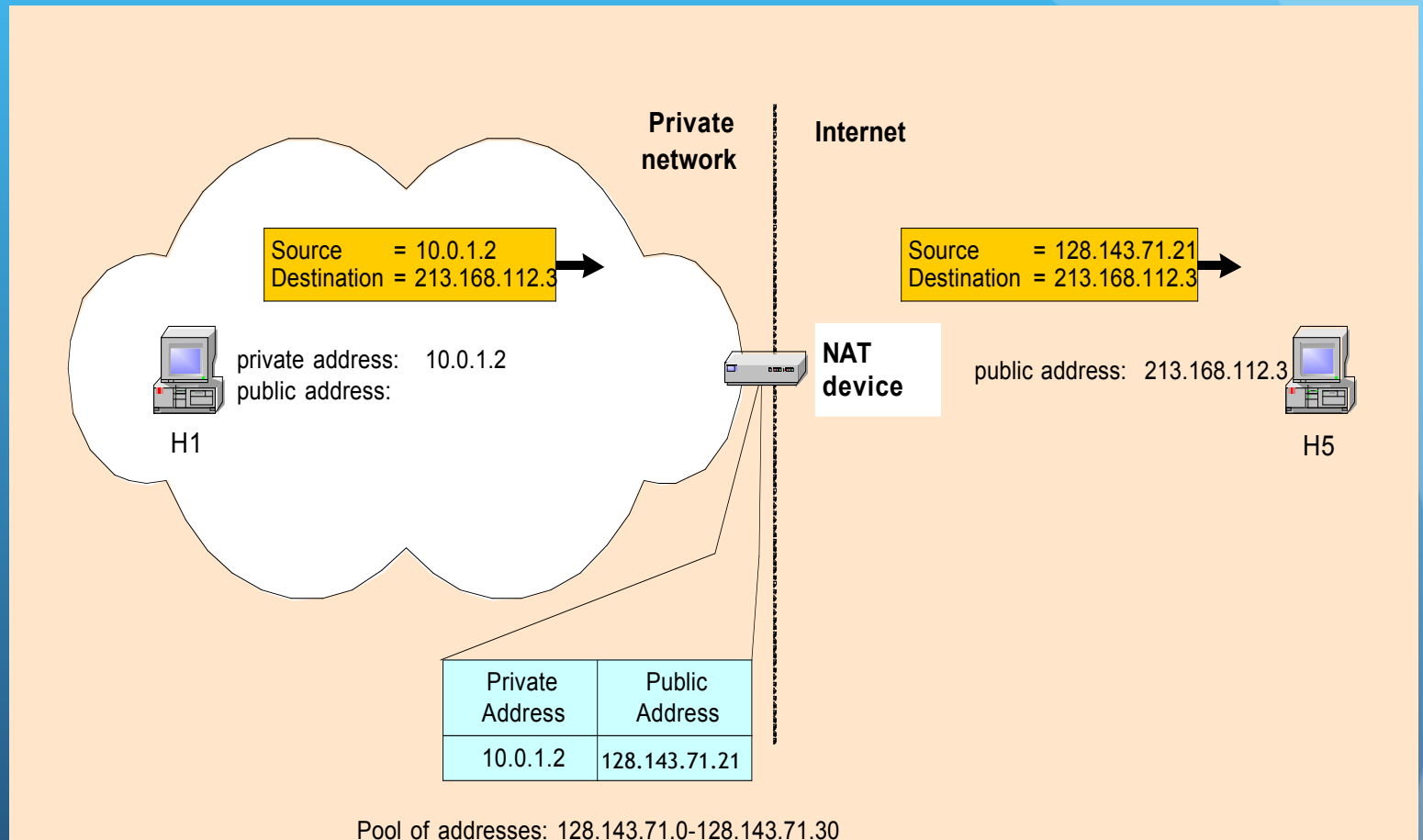


- NAT device has address translation table
- One to one address translation

Pooling of IP addresses

- **Scenario:** Corporate network has many hosts but only a small number of public IP addresses
- **NAT solution:**
 - Corporate network is managed with a private address space
 - NAT device, located at the boundary between the corporate network and the public Internet, manages a pool of public IP addresses
 - When a host from the corporate network sends an IP datagram to a host in the public Internet, the NAT device picks a public IP address from the address pool, and binds this address to the private address of the host

Pooling of IP addresses



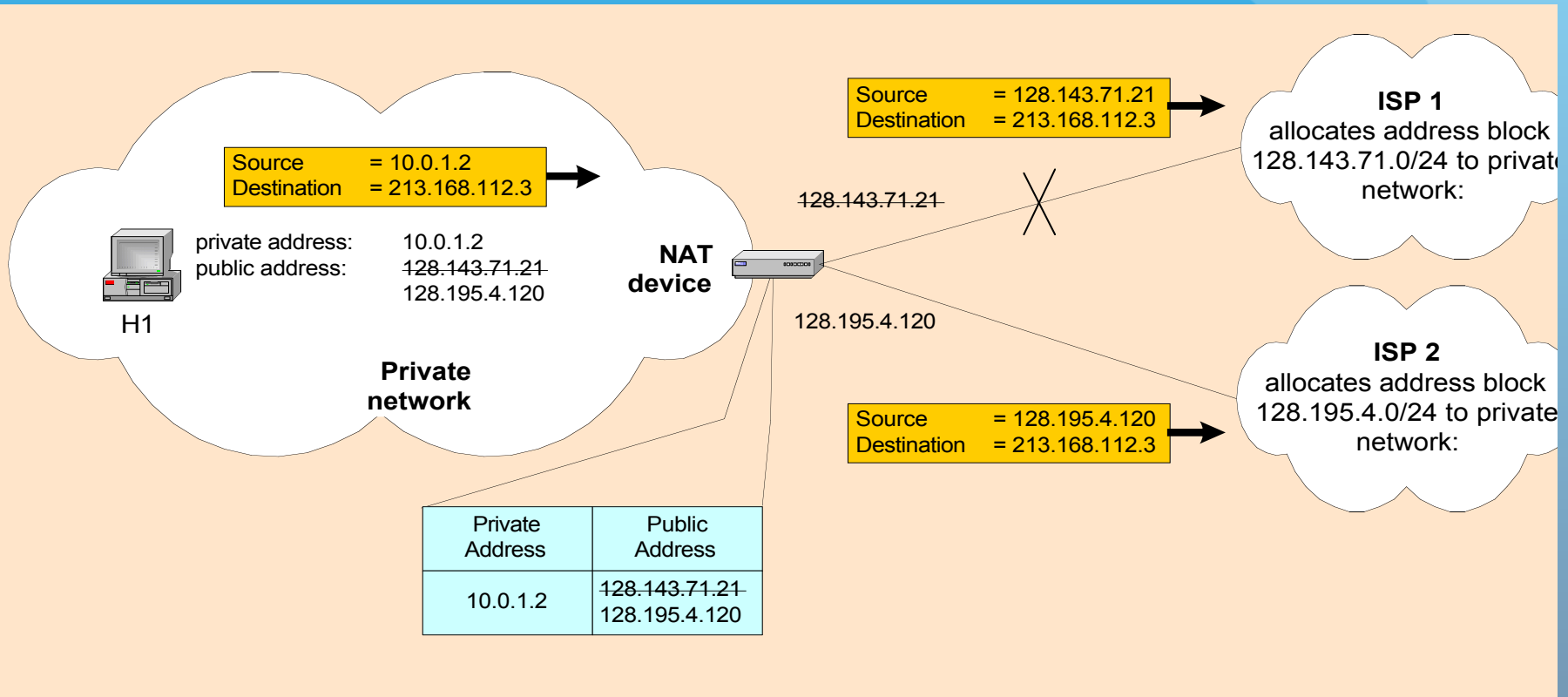
Supporting migration between network service providers

- **Scenario:** In CIDR, the IP addresses in a corporate network are obtained from the service provider. Changing the service provider requires changing all IP addresses in the network.
- **NAT solution:**
 - Assign private addresses to the hosts of the corporate network
 - NAT device has static address translation entries which bind the private address of a host to the public address.
 - Migration to a new network service provider merely requires an update of the NAT device. The migration is not noticeable to the hosts on the network.

Note:

- The difference to the use of NAT with IP address pooling is that the mapping of public and private IP addresses is static.

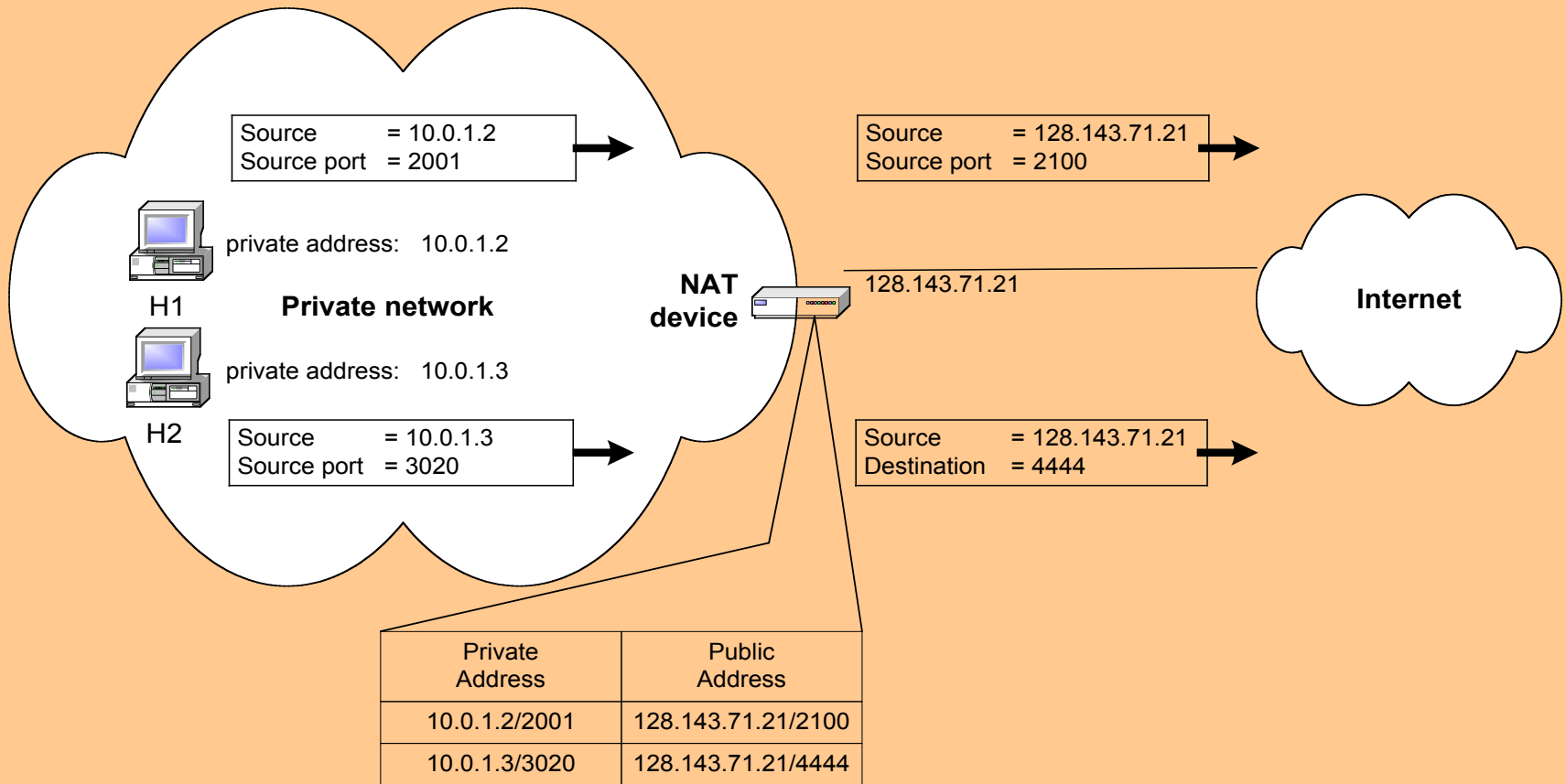
Supporting migration between network service providers



IP masquerading

- **Also called: Network address and port translation (NAPT), port address translation (PAT).**
- **Scenario: Single public IP address** is mapped to **multiple hosts** in a private network.
- **NAT solution:**
 - Assign private addresses to the hosts of the corporate network
 - NAT device modifies the port numbers for outgoing traffic

IP masquerading

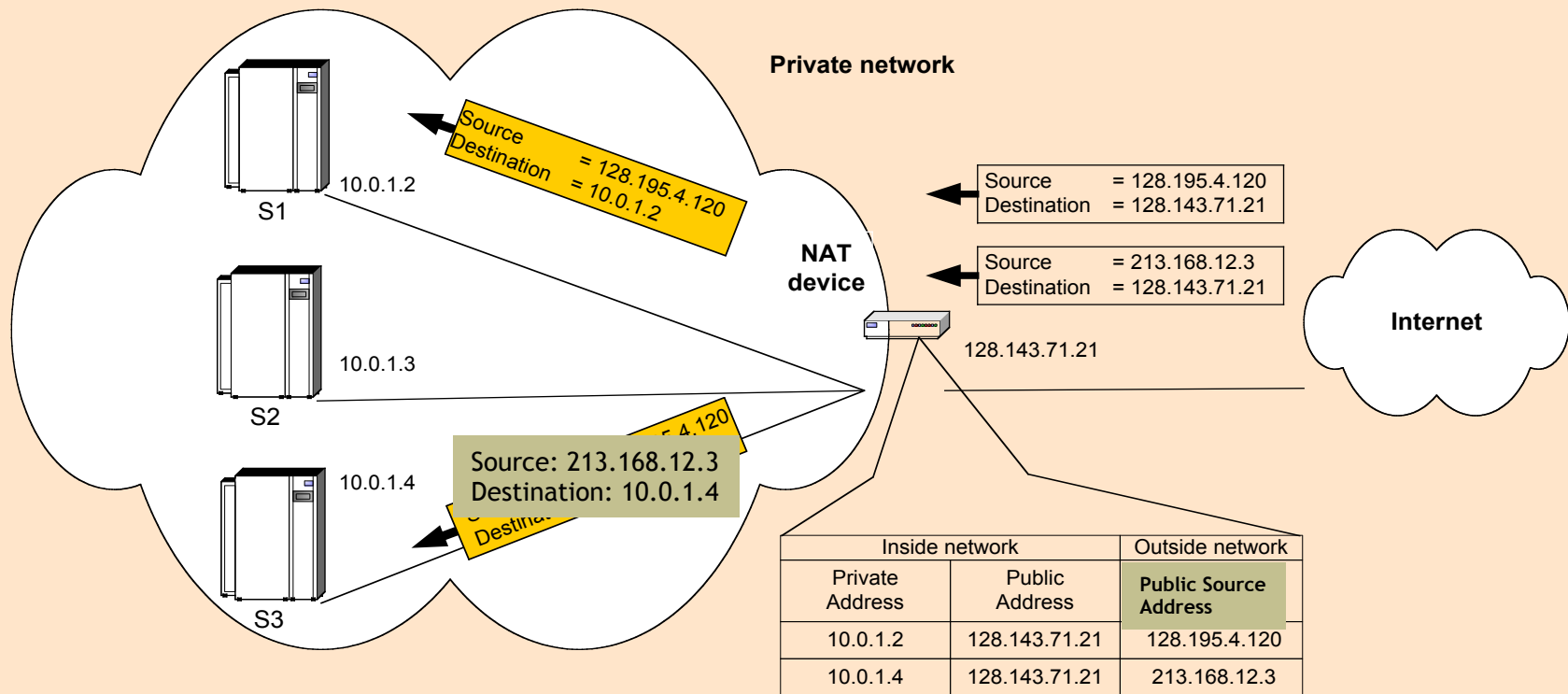


Load balancing of servers

- **Scenario:** Balance the load on a set of identical servers, which are accessible from a single IP address
- **NAT solution:**
 - Here, the servers are assigned private addresses
 - NAT device acts as a proxy for requests to the server from the public network
 - The NAT device changes the destination IP address of arriving packets to one of the private addresses for a server
 - A sensible strategy for balancing the load of the servers is to assign the addresses of the servers in a round-robin fashion.

Load balancing of servers

This is a situation where **one** public address is shared by **several** machines inside a private network for the sole purpose of load balancing.

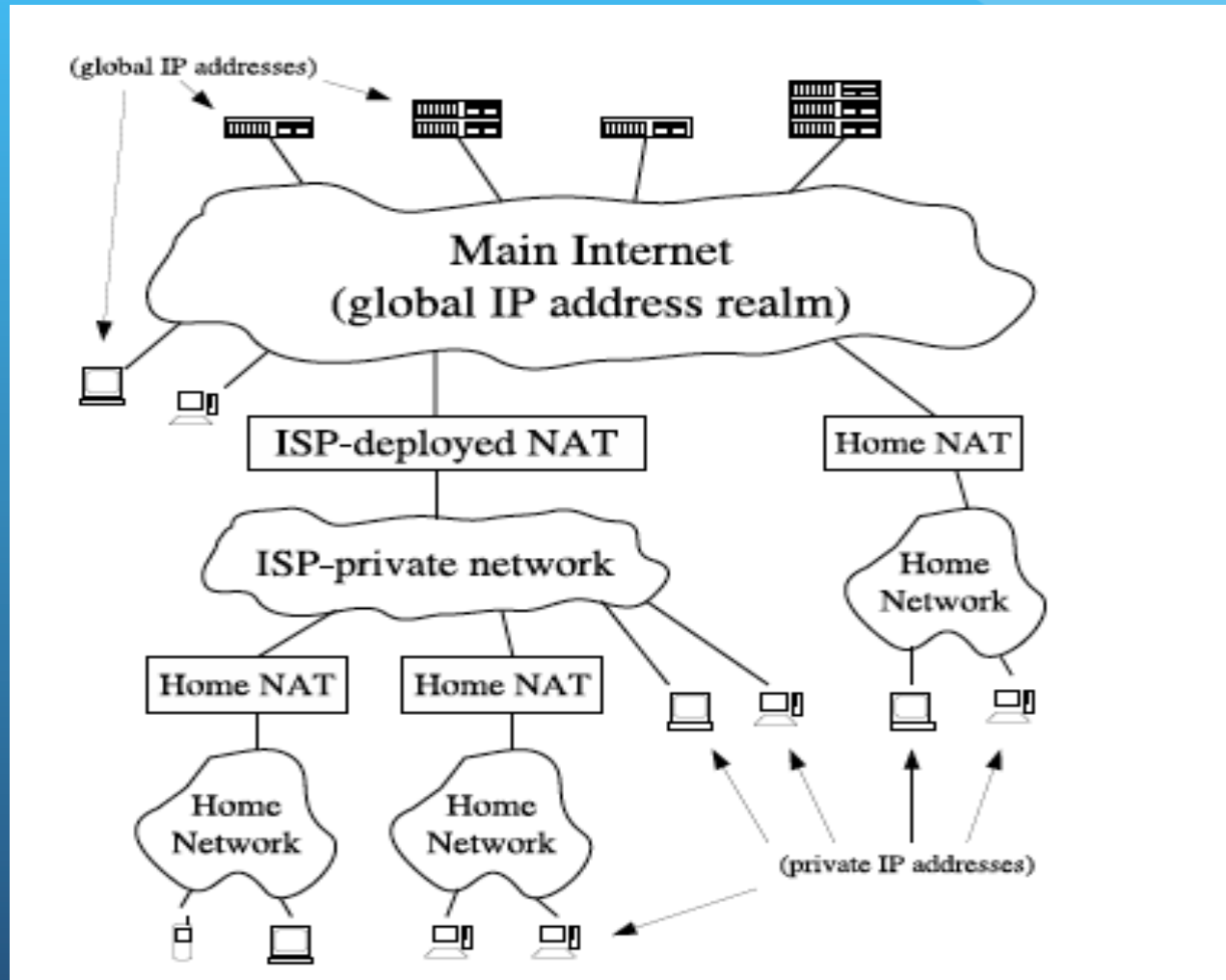


Here we see that IP address 128.143.71.21 is shared by several machines. The NAT device uses a round robin technique to distribute the incoming requests to the servers. The source addresses of incoming requests are used to distinguish between the traffic streams connected to the internal servers.

NAT: Public Services and P2P

- NAT does not allow outside (public) access to any machine behind the NAT device if there is no **unique public IP address associated (directly mapped)** with a **specific internal IP address**.
 - In other words, it can bypass traffic that is addressed to a device that is behind the NAT device if it has a public IP address allocated to it (i.e., mapped to its internal address).
- Services such as Web hosting, FTP, can be addressed by the one to one mapping of a specific IP address.
- P2P services (Skype, Spotify, etc.) cannot be hosted unless a specific server called a moderator/tracker/relay is used to serve as a go between.

Possible NAT Scenarios



Hole Punching: UDP and TCP

- One of the most effective methods of establishing peer-to-peer communication between hosts on different private networks is known as “hole punching.”
- Hole punching enables applications to function *within* the default security policy of most NATs, effectively signaling to NATs on the path that peer-to-peer communication sessions are “solicited” and thus should be accepted.
- Unfortunately, no traversal technique works with **all** existing NATs, because **NAT behavior is not standardized**.
- We will focus on the *simplest* hole punching technique that works cleanly and robustly in the presence of “well-behaved” NATs in any reasonable network topology.

NAT Terminology

- The notion of session:
“A *session endpoint* for TCP or UDP is an (IP address, port number) pair, and a particular *session* is uniquely identified by its two session endpoints.”
- From the perspective of one of the hosts involved, a **session** is effectively **identified** by the **4-tuple (local IP, local port, remote IP, remote port)**.
- The **direction** of a session is normally the flow direction of the packet that initiates the session: the initial SYN packet for TCP, or the first user datagram for UDP.
- The most common type of NAT is *traditional* or **outbound NAT**, which provides an asymmetric bridge between a private network and a public network.
- Outbound NAT has two sub-varieties: **Basic NAT**, which only translates IP addresses, and **Network Address/Port Translation (NAPT)**, which translates entire session endpoints.
- **Hole punching** uses **NAPT**.

Understanding the Problem

- The key to the NAT puzzle lies in the fact that in order for machines behind a NAT gateway to interact with the public Internet, NAT devices necessarily have to **allow inbound traffic**—that is, **replies to requests originating from behind the NAT device**.
- NAT devices **allow** traffic through to a particular host behind a NAT device, provided the traffic is a **reply to a request sent by the NAT device**.
- NAT devices vary widely in operation, and they let through replies coming from other hosts and port numbers, depending on their own notion of **what a reply means**.
- So instead of connecting directly to the host behind NAT (impossible), we need to mimic a scenario in which **the target host originates a connection to us** and then we connect to it as though we are **responding to the request**. In other words, our connection request to the target host should seem like a **reply to the NAT device**.

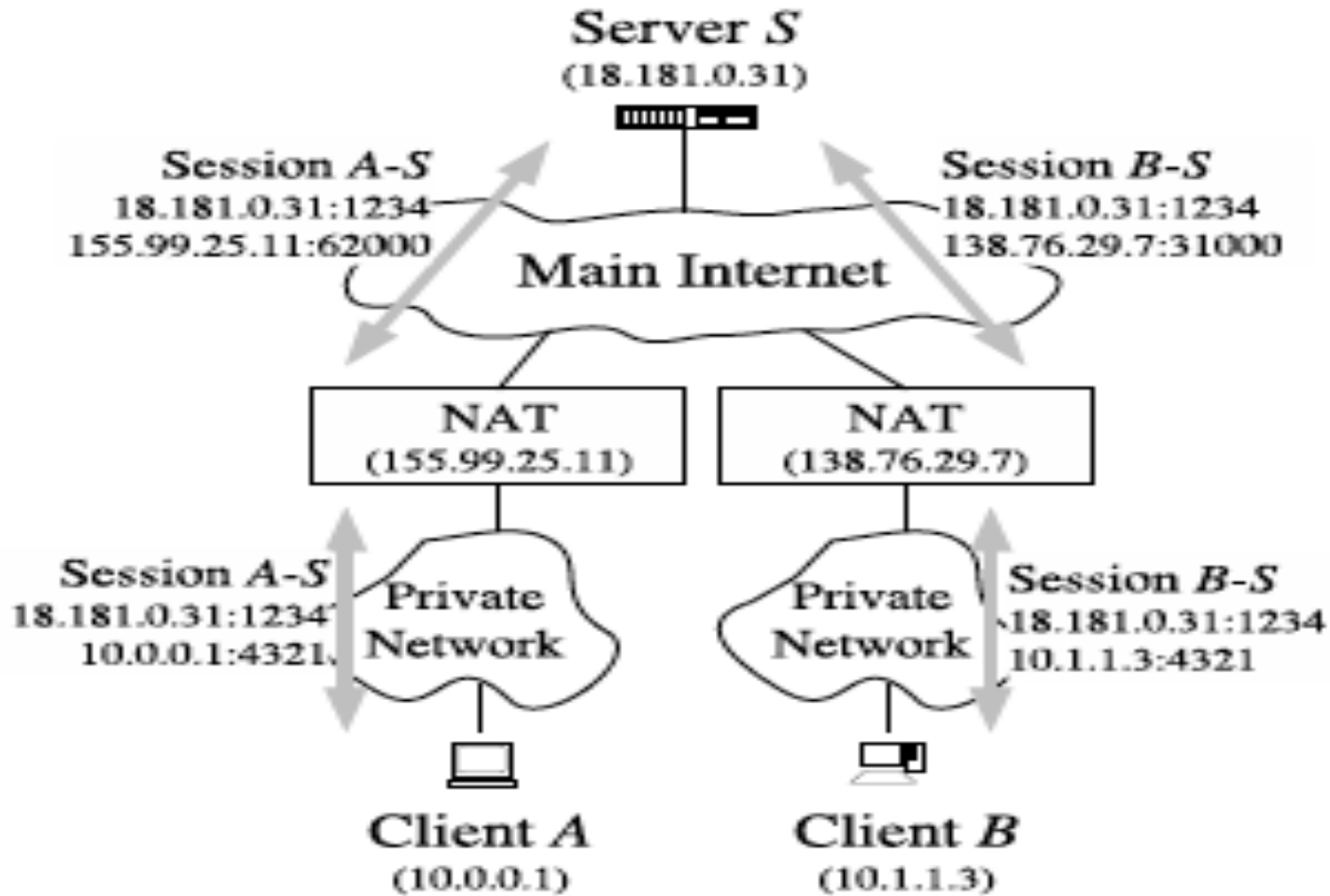
Approach

- First is the issue of how to get the private host to originate traffic so we can send our connection request to it **masquerading as a reply**.
- BUT, NAT devices also have an idle timer, typically of around 60 seconds, they stop waiting for replies once a request originates and no reply comes within 60 seconds. So, it is not enough that the private host originate traffic, but also we have to act fast—we have to send the “reply” before the NAT device removes the “association” with the private host, which will frustrate the connection attempt.
- This solution works fine if the other PEER is on a public network. So if timing is not an issue, i.e., response comes back fast enough, we are set.

Relaying Using an Intermediate Server - P2P both behind NAT

- **Relaying** - A reliable, but least efficient method for P2P communication across NAT is simply to make the communication look to the network like standard client/server communication, through relaying.
- Two client hosts, A and B, each initiate TCP or UDP connections to a well-known “**rendezvous**” server S, at S’s global IP address YY and a predetermined port number XX.
- The two clients simply use the server S to **relay messages** between them along along their already-established client/server connections, and server S forwards the messages.
- Connections (TCP and UDP) have to be persistent. NAT times out UDP NAT connections, so it has to happen fast or host has to continue sending “keep alive” UDP packets to the server to maintain the connection. TCP session has to be kept alive too.
- **Disadvantage:** uses server processing power and network bandwidth.

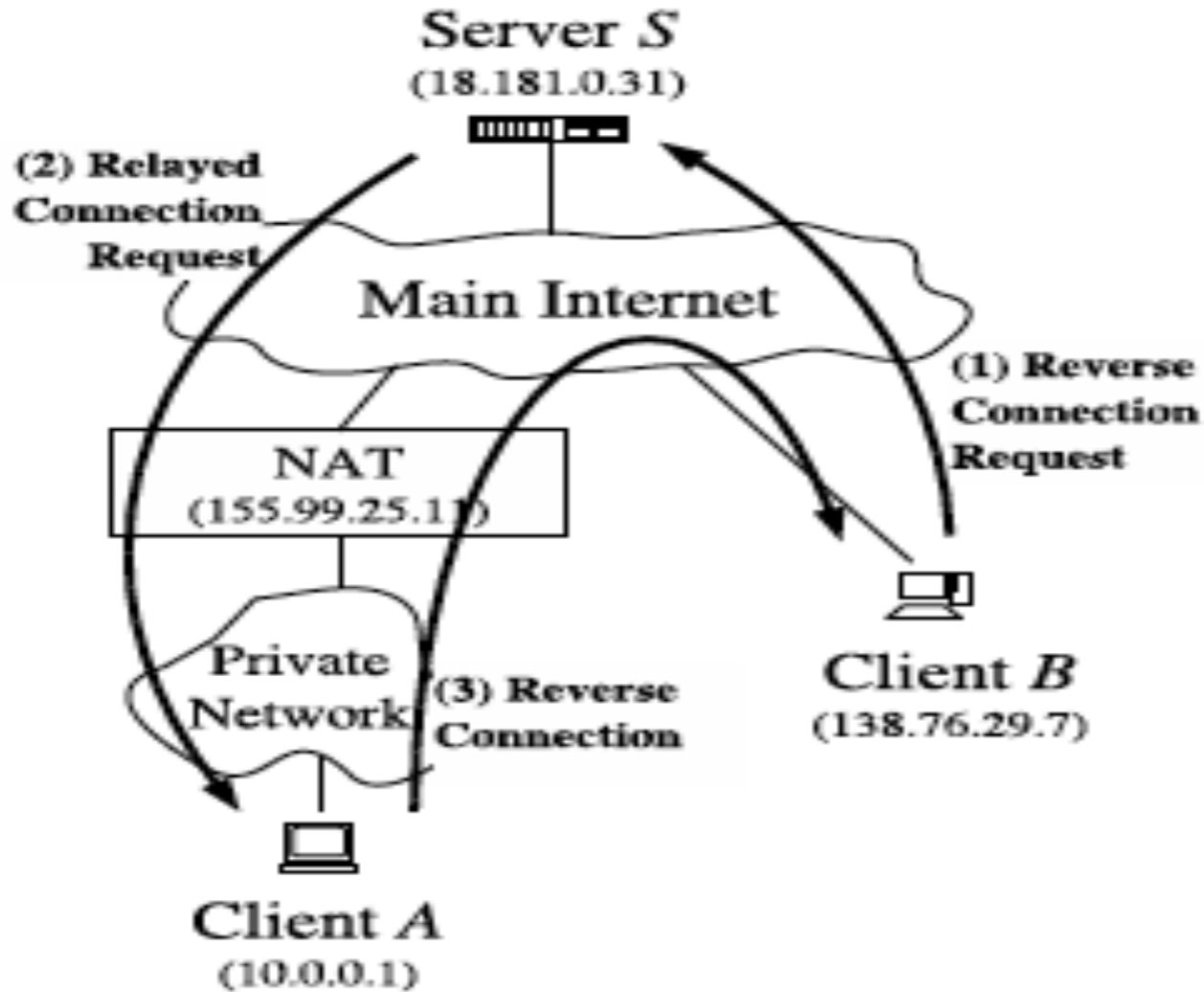
Relaying



Traversal By Connection Reversal Using a Server - Public to Private

- Going from a **host/peer on the public IP internet** to a **host/peer on a private network** with no dedicated public IP address.
- Use a “**rendezvous**” server, listening on a global IP address through a persistent TCP or UDP connection.
- If **A** on a **private network** wants to initiate a connection to **B** on a **public network**, then a direct connection attempt works automatically, because B is not behind a NAT and A’s NAT interprets the connection as an outgoing session.
- If **B** wants to initiate a connection to **A**, however, any direct connection attempt to A is **blocked** by A’s NAT.
- **B** can instead **relay** a connection request **to A** through a well-known **server S**, asking **A** to attempt a “**reverse**” connection back to **B**.

Connection Reversal



UDP Hole Punching - Basic Idea

- UDP hole punching enables two clients to set up a **direct** P2P UDP session with the help of a well-known “rendezvous” server, even if the clients are both behind NATs. I.e, **eliminates** the “Server” function after connection setup.
- Hole punching assumes that the two clients, **A** and **B**, already have active **UDP sessions** with a “rendezvous” **server S**.
- When a client registers with **S**, the server records **two endpoints** for that client: the (**IP address, UDP port**) pair that the **client is using** to talk with **S**, and the (**IP address, UDP port**) pair that the server **observes** the client to be using to talk with it. (private pair and public pair)
- We refer to the first pair as the **client’s private endpoint** and the second as **the client’s public endpoint**.

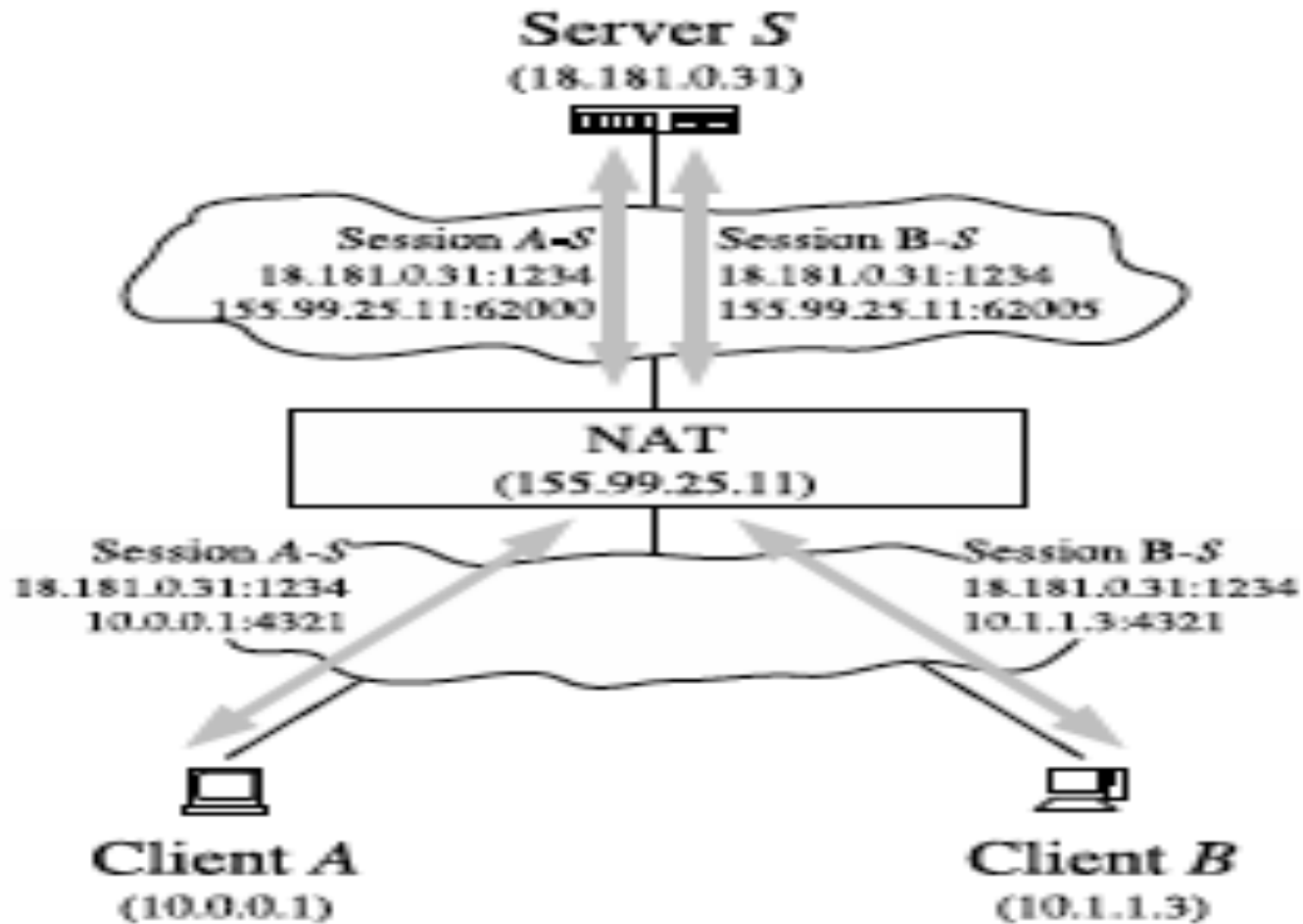
Obtaining the Clients' Private Addresses

- The server might obtain the **client's private** endpoint from the client itself in a field in the **body of the client's registration message**, and obtain the **client's public** endpoint from the **source IP address and source UDP port fields in the IP and UDP headers** of that registration message.
- A few NATs are known to **scan the body of UDP** datagrams for 4-byte fields that look like **IP addresses**, and **translate** them as they would the IP address fields in the IP header.
- To be robust against such behavior, applications may wish to **obfuscate** IP addresses in message bodies, for example:
 - by transmitting the one's complement of the IP address instead of the IP address itself. Or
 - if the application is encrypting its messages, then this behavior is not likely to be a problem.

Establishing P2P Sessions

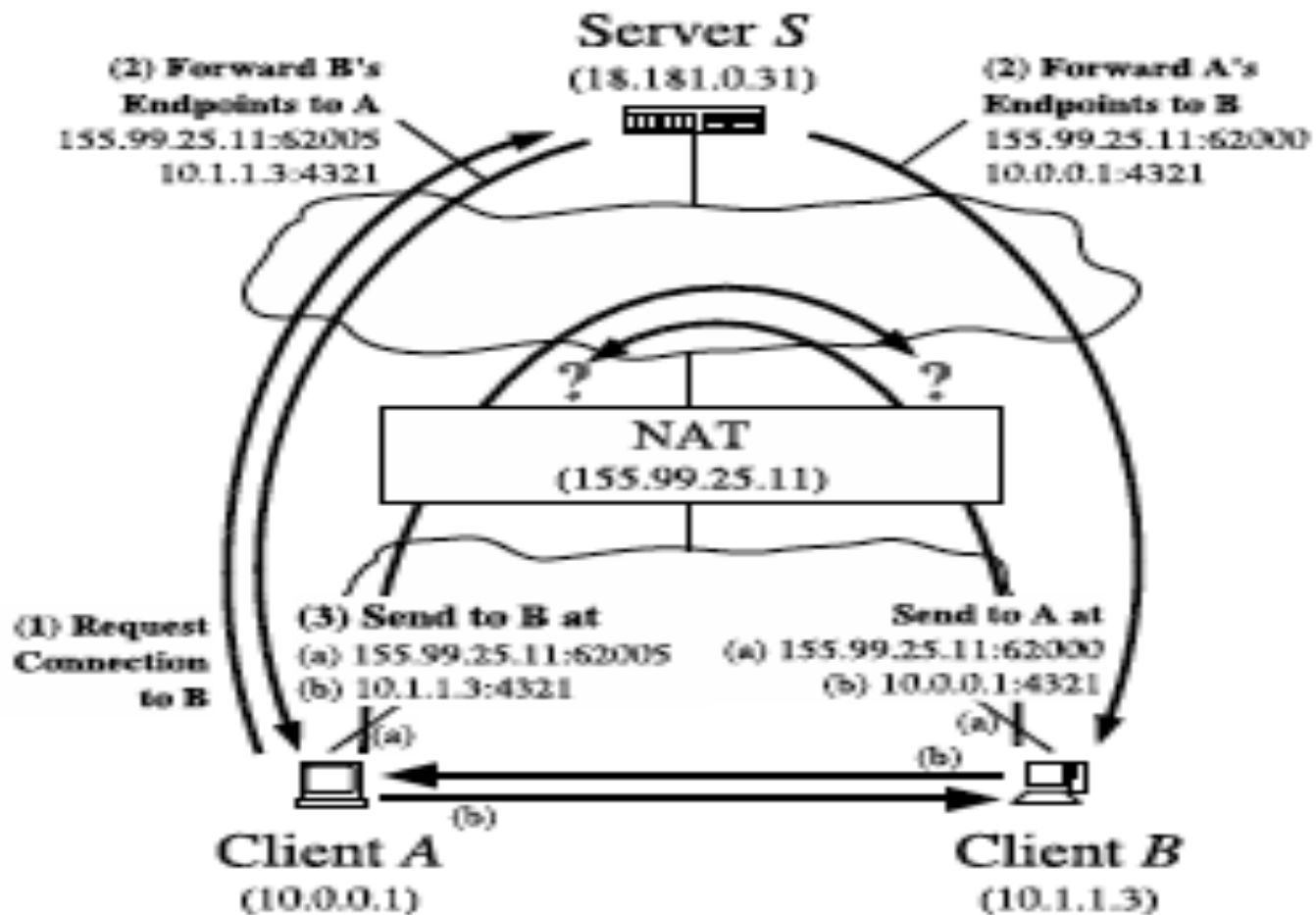
1. **A** initially does not know how to reach **B**, so **A** asks **S** for help establishing a UDP session with **B**.
2. **S** replies to **A** with a message containing **B**'s **public and private endpoints**. At the same time, **S** uses its UDP session with **B** to send **B** a connection request message containing **A**'s **public and private endpoints**. Once these messages are received, **A** and **B** know each other's public and private endpoints.
3. When **A** receives **B**'s public and private endpoints from **S**, **A** starts sending UDP packets to **both** of these **endpoints**, and subsequently "locks in" whichever endpoint first elicits a valid response from **B**.
4. Similarly, when **B** receives **A**'s public and private endpoints in the forwarded connection request, **B** starts sending UDP packets to **A** at each of **A**'s known endpoints, locking in the first endpoint that works.
5. The **order and timing** of these messages are **not critical** as long as they are asynchronous.

Before Hole Punching

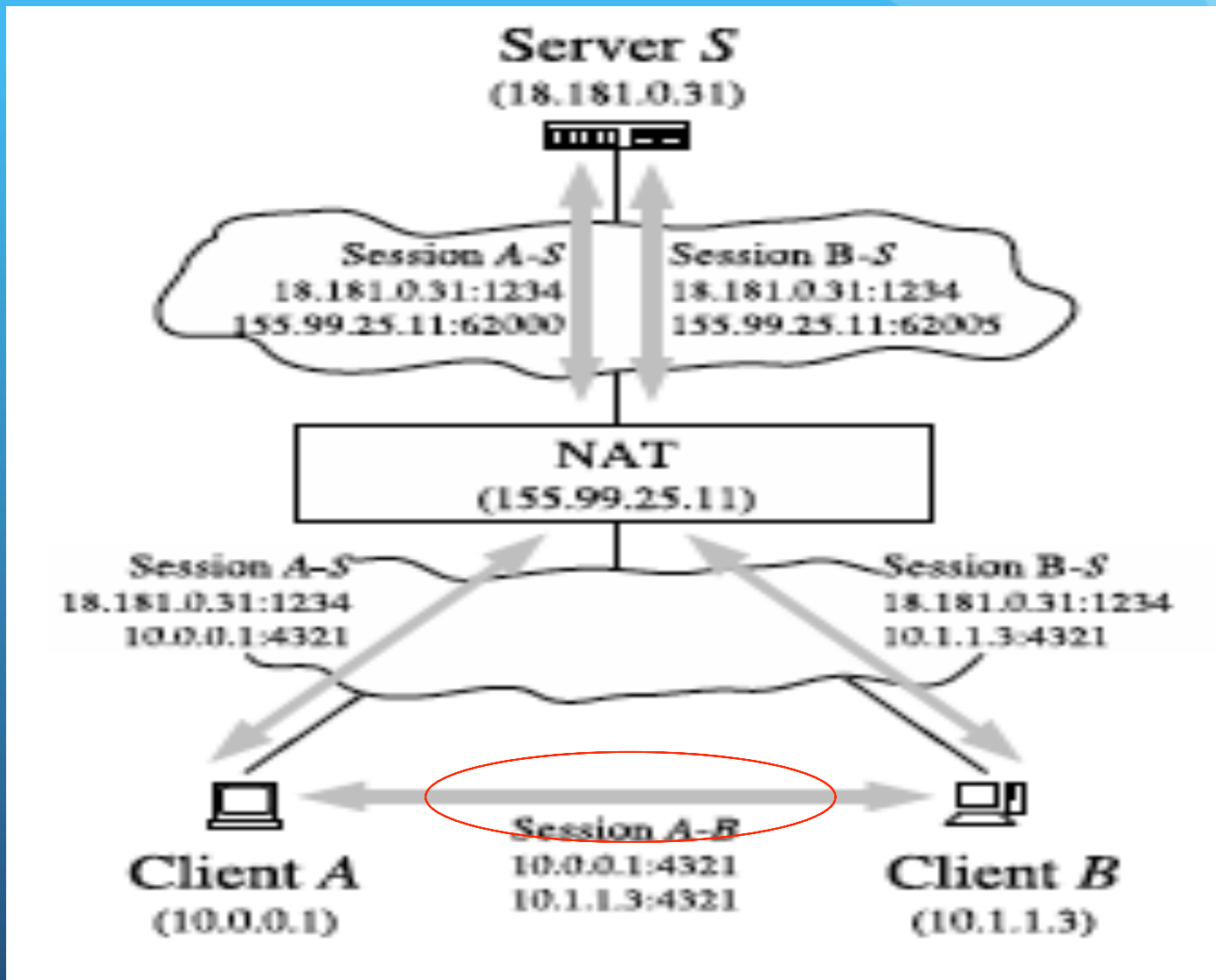


Before Hole Punching

Hole Punching Process



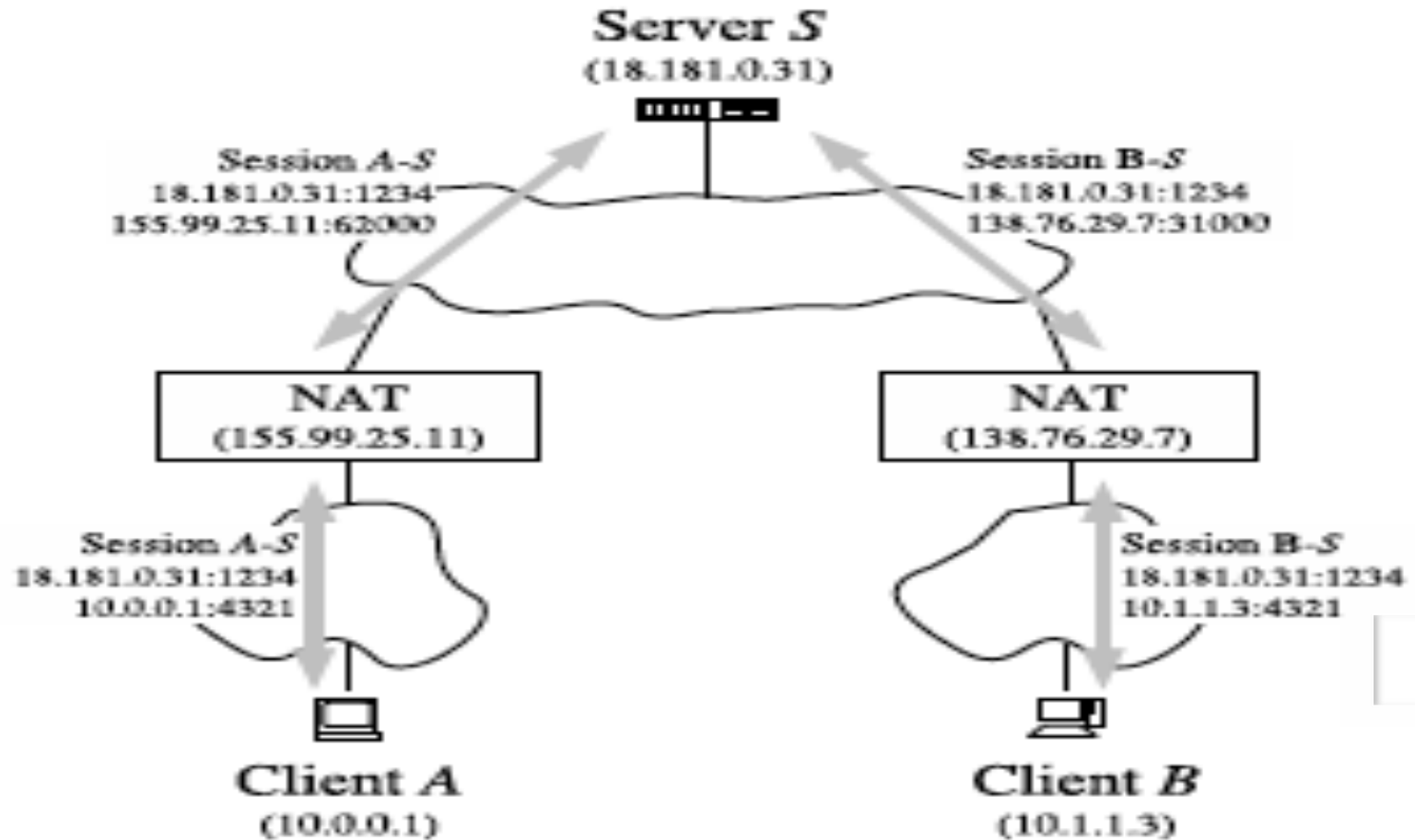
After Hole Punching



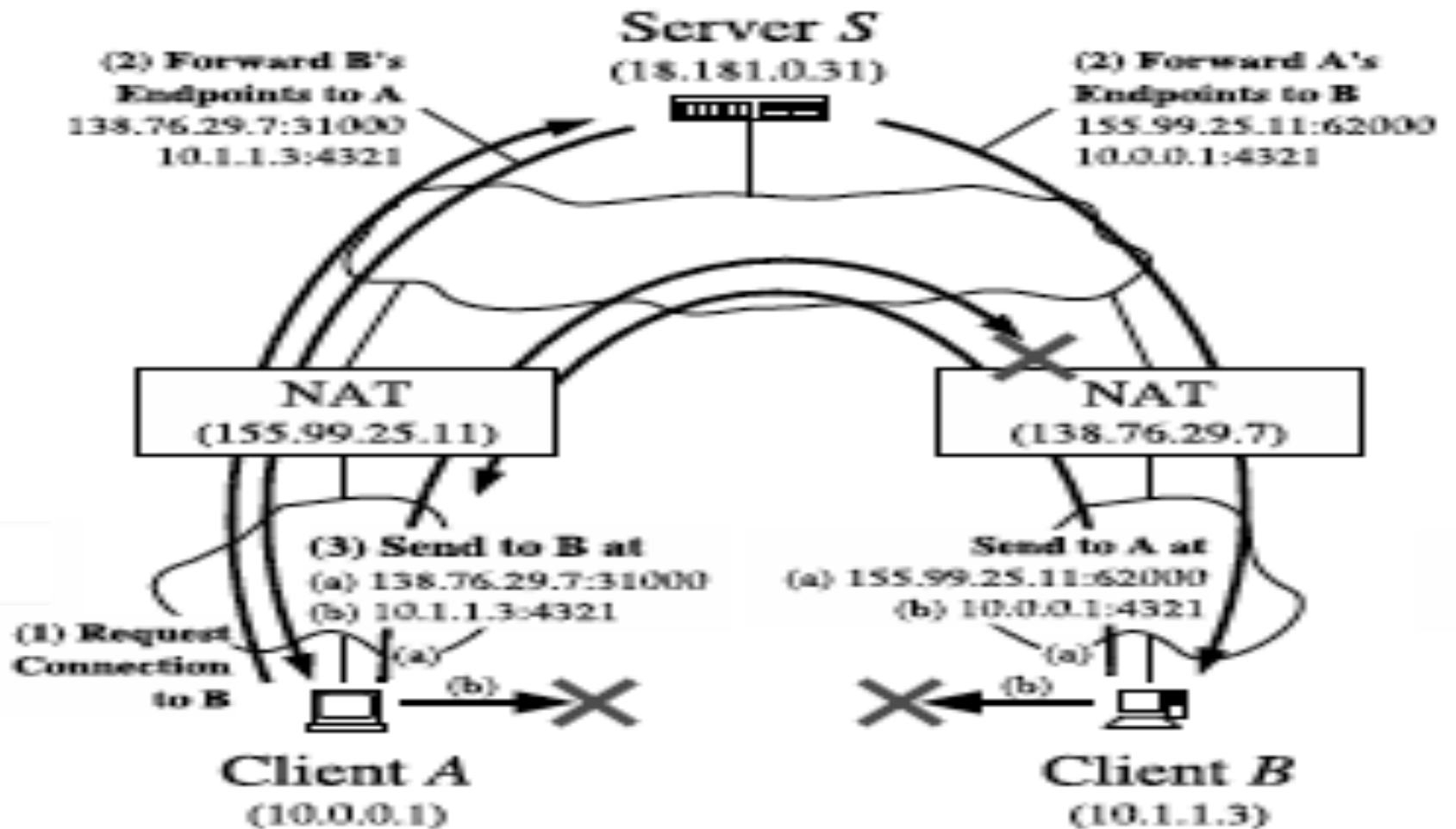
Hole Punching Scenarios

- Three possible scenarios:
 1. Two clients reside behind the same NAT, on one private network (example above)
 2. The clients reside behind different NATs, different private networks
 3. The clients each reside behind *two* levels of NAT: a common “first-level” NAT deployed by an ISP for example, and distinct “second-level” NATs such as consumer NAT routers for home networks
- It is in general difficult or impossible for the application itself to determine the exact physical layout of the network, and thus which of these scenarios (or the many other possible ones) actually applies at a given time.
- Nevertheless, hole punching works automatically in all of these scenarios *without* the application having to know the specific network organization, as long as the NATs involved behave in a “reasonable” fashion, i.e., don’t overwrite IP addresses and port numbers in bodies of messages.

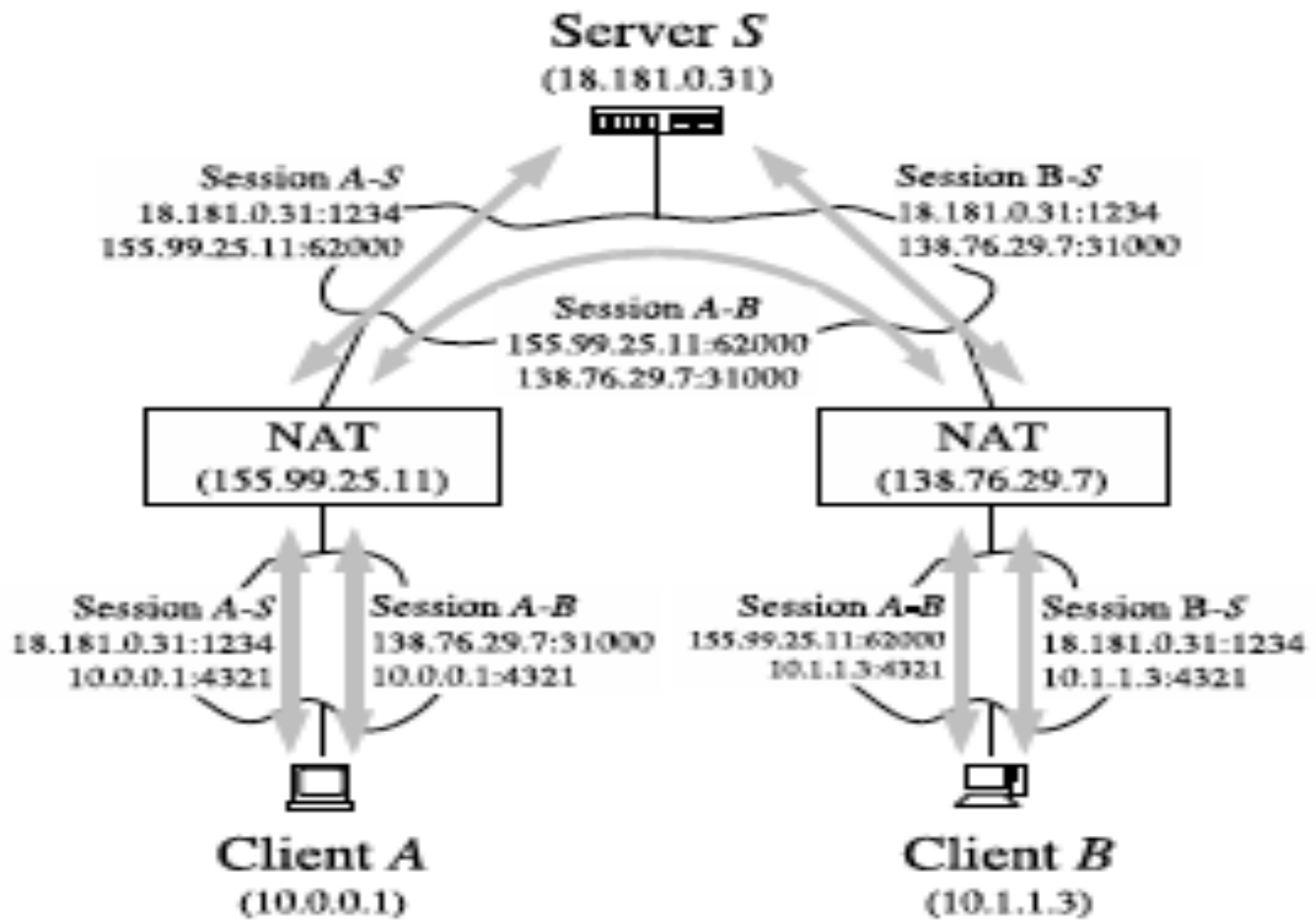
More complicated examples



Before Hole Punching



The Hole Punching Process

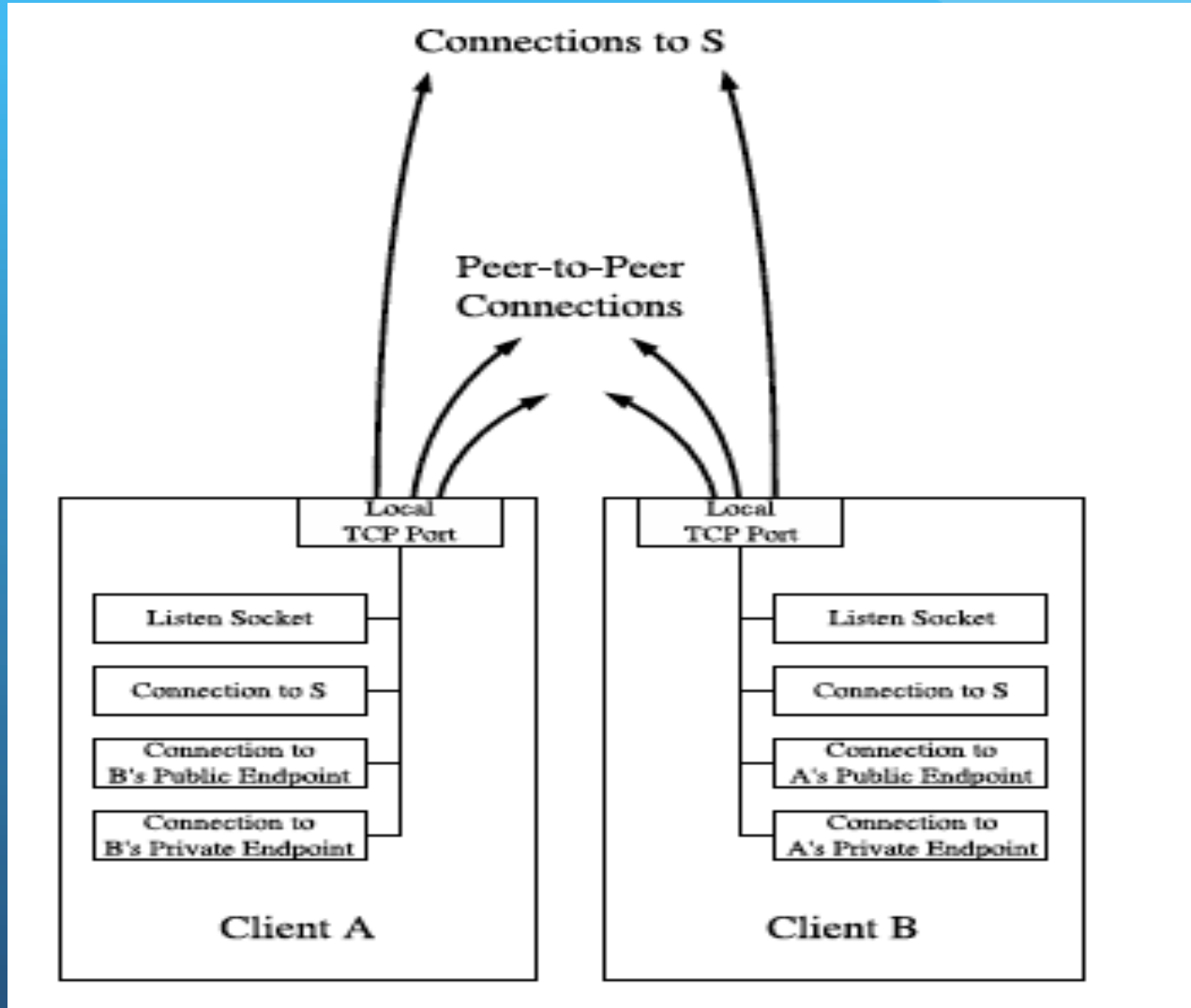


After Hole Punching

TCP Hole Punching

- **Sockets and TCP Port Re-use**: The main practical challenge to applications wishing to implement TCP hole punching is not a protocol issue but an application programming interface (API) issue.
 - the API allows a TCP stream socket to be used to initiate an outgoing connection via `connect()`, or to listen for incoming connections via `listen()` and `accept()`, *but not both*.
 - TCP **sockets** usually have a **one-to-one correspondence** to TCP **port numbers** on the local host: after the application binds one socket to a particular local TCP port, attempts to bind a second socket to the same TCP port fail.
- For TCP hole punching to work, however, we need to use a **single local TCP port to listen for incoming** TCP connections and to **initiate multiple outgoing** TCP connections concurrently, i.e, **DO BOTH**.

Sockets versus Ports for Hole Punching



Opening P2P TCP Streams

- Client **A** wishes to set up a TCP connection with client **B**. We assume as usual that both **A** and **B** already have active TCP connections with a well-known “rendezvous” server **S**. The server records each **registered** client’s **public and private endpoints**, just as for UDP. At the protocol level, TCP hole punching works almost exactly as for UDP:
 1. Client **A** uses its active TCP session with **S** to ask **S** for help connecting to **B**.
 2. **S** replies to **A** with **B**’s public and private TCP endpoints, and at the same time sends **A**’s public and private endpoints to **B**.
 3. From *the same local TCP ports* that **A** and **B** used to register with **S**, **A** and **B** each asynchronously make outgoing connection attempts to the other’s public and private endpoints as reported by **S**, while simultaneously listening for incoming connections on their respective local TCP ports.

Contd.

4. A and B wait for **outgoing connection attempts to succeed**, and/or for incoming connections to appear. If one of the outgoing connection **attempts fails** due to a network error such as “connection reset” or “host unreachable,” **the host simply re-tries** that connection attempt after a short delay (e.g., one second), up to an application-defined maximum timeout period.
5. When a TCP connection is made, the hosts **authenticate** each other to verify that they connected to the intended host. If **authentication fails**, the **clients close that connection** and continue waiting for others to succeed. The clients use the first successfully authenticated TCP stream resulting from this process.

Properties of P2P Friendly NATs

1. Consistent Endpoint Translation

- NAT “focuses” all sessions originating from a single private endpoint through the same public endpoint on the NAT. E.g., When client **A** initially contacted the well-known server **S**, NAT **A** chose to use port **62000** at its own public IP address, **155.99.25.11**, as a temporary public endpoint to representing **A**’s private endpoint **10.0.0.1:4321**.
- When **A** later attempts to establish a peer-to-peer session with **B** by sending a message from the same local private endpoint to **B**’s public endpoint, **A** depends on NAT **A** preserving the identity of this private endpoint, and re-using the existing public endpoint of **155.99.25.11:62000**, because that is the public endpoint for **A** to which **B** will be sending its corresponding messages.
- A NAT that is only designed to support client/server protocols will not necessarily preserve the identities of private endpoints in this way. E.g., after the NAT assigns the public endpoint **155.99.25.11:62000** to client **A**’s session with server **S**, the NAT might assign a different public endpoint, such as **155.99.25.11:62001**, to the P2P session that **A** tries to initiate with **B**. In this case, the hole punching process fails to provide connectivity, because the subsequent incoming messages from **B** reach NAT **A** at the **wrong port number**.

Contd.

2. Handling Unsolicited TCP Connections

- When a NAT receives a SYN packet on its public side for what appears to be an unsolicited incoming connection attempt, it is important that the NAT just silently drop the SYN packet.
- Some NATs instead actively reject such incoming connections by sending back a **TCP RST** packet or even an **ICMP error** report, which interferes with the TCP hole punching process.

3. Leaving Payloads Alone

- A few existing NATs are known to scan “blindly” through **packet payloads for 4-byte values** that look like IP addresses, and translate them as they would the IP address in the packet header, without knowing anything about the application protocol in use. This bad behavior fortunately appears to be uncommon, and applications can easily protect themselves against it by **obfuscating** IP addresses they send in messages.

Concerns about NAT

- **Performance:**

- Modifying the IP header by changing the IP address requires that NAT boxes recalculate the IP header checksum
- Modifying port number and IP address requires that NAT boxes recalculate TCP and UDP checksum (pseudo header)

- **Fragmentation**

- Care must be taken that a datagram that is fragmented **before** it reaches the NAT device, is not assigned a different IP address or different port numbers for each of the fragments.

Concerns about NAT

- **End-to-end connectivity:**

- NAT destroys universal end-to-end reachability of hosts on the Internet.
- A host in the public Internet often cannot initiate communication to a host in a private network.
- The problem is worse, when two hosts that are in a private network need to communicate with each other.

Concerns about NAT

- **IP address in application data:**

- Applications that carry IP addresses in the payload of the application data generally do not work across a private-public network boundary.
- Some NAT devices inspect the payload of widely used application layer protocols and, if an IP address is detected in the application-layer header or the application payload, translate the address according to the address translation table.

Dynamic Host Control Protocol - DHCP

Mostly to facilitate the management of mobile devices.

Allows an active device to obtain a temporary IP address that is valid on the local IP subnet.

IP address is released when device moves to another subnet.

Dynamic Assignment of IP addresses

- Dynamic assignment of IP addresses is desirable for several reasons:
 - IP addresses are assigned on-demand
 - Avoid manual IP configuration
 - Support mobility of laptops

DHCP Message Types

Value	Message Type
1	DHCPDISCOVER
2	DHCPOFFER
3	DHCPREQUEST
4	DHCPDECLINE
5	DHCPACK
6	DHCPNAK
7	DHCPRELEASE
8	DHCPINFORM

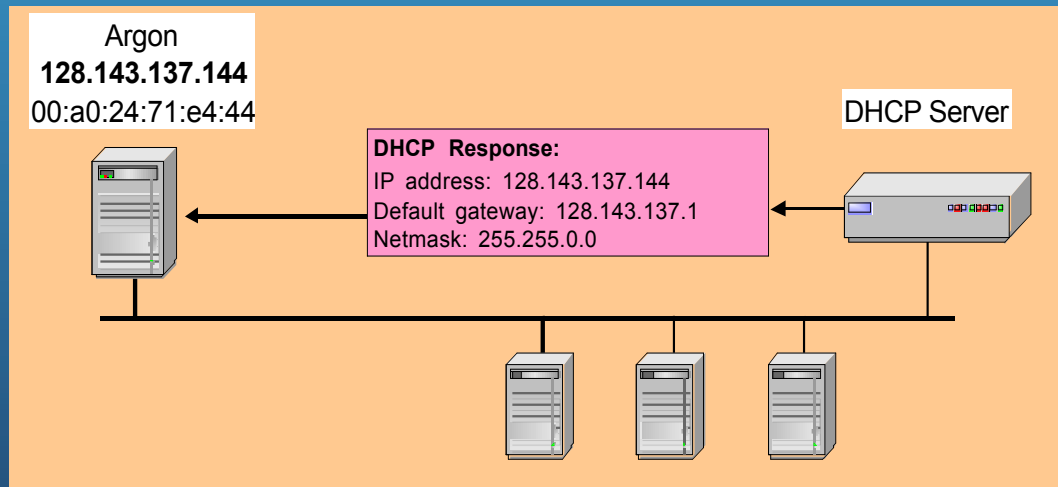
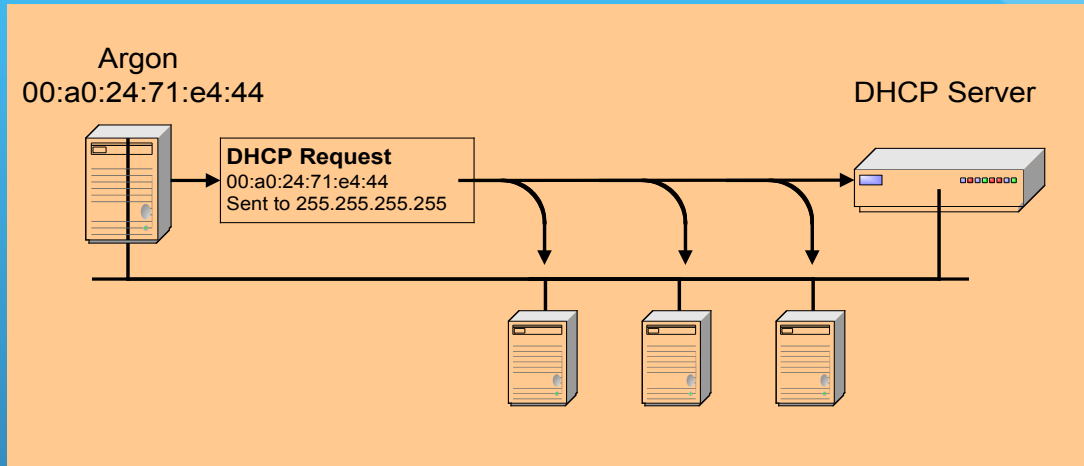
Message Types

- **DHCPDISCOVER**: Broadcast by a client to find available DHCP servers.
- **DHCPOFFER**: Response from a server to a DHCPDISCOVER and offering IP address and other parameters.
- **DHCPREQUEST**: Message from a client to servers that does one of the following:
 - Requests the parameters offered by one of the servers and declines all other offers.
 - Verifies a previously allocated address after a system or network change (a reboot for example).
 - Requests the extension of a lease on a particular address.

Contd.

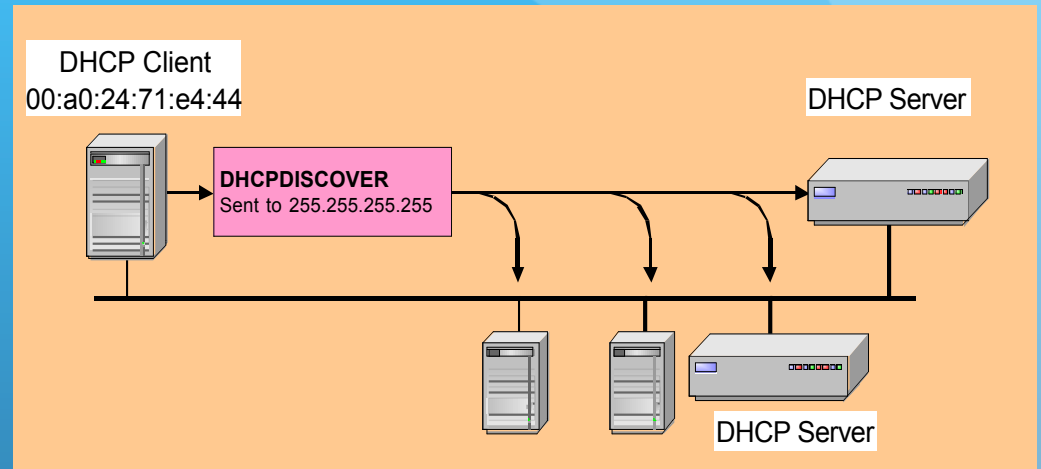
- **DHCPACK**: Acknowledgement from server to client with parameters, including IP address.
- **DHCPNACK**: Negative acknowledgement from server to client, indicating that the client's lease has expired or that a requested IP address is incorrect.
- **DHCPDECLINE**: Message from client to server indicating that the offered address is already in use.
- **DHCPRELEASE**: Message from client to server canceling remainder of a lease and relinquishing network address.
- **DHCPINFORM**: Message from a client that already has an IP address (manually configured for example), requesting further configuration parameters from the DHCP server.

DHCP Interaction (simplified)

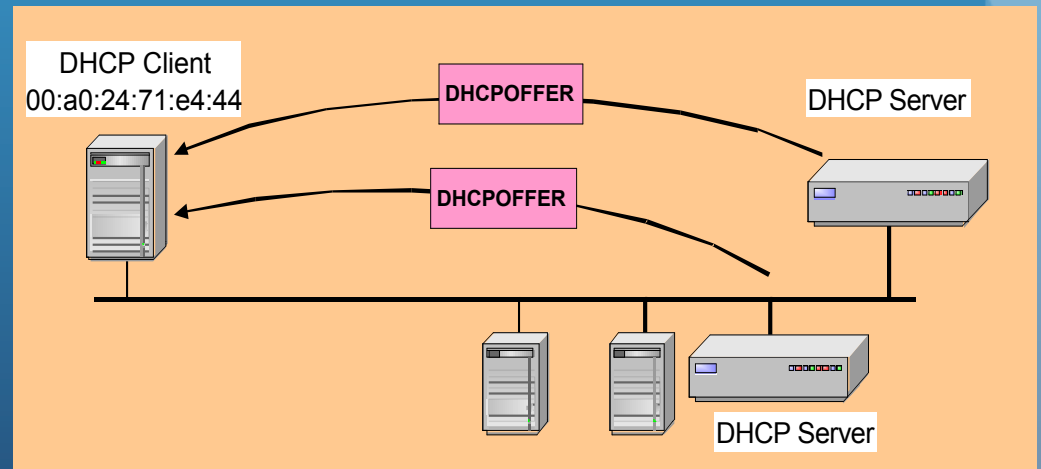


DHCP Operation – First search for DHCP servers

- DCHP DISCOVER



- DCHP OFFER



Client-Server Interactions

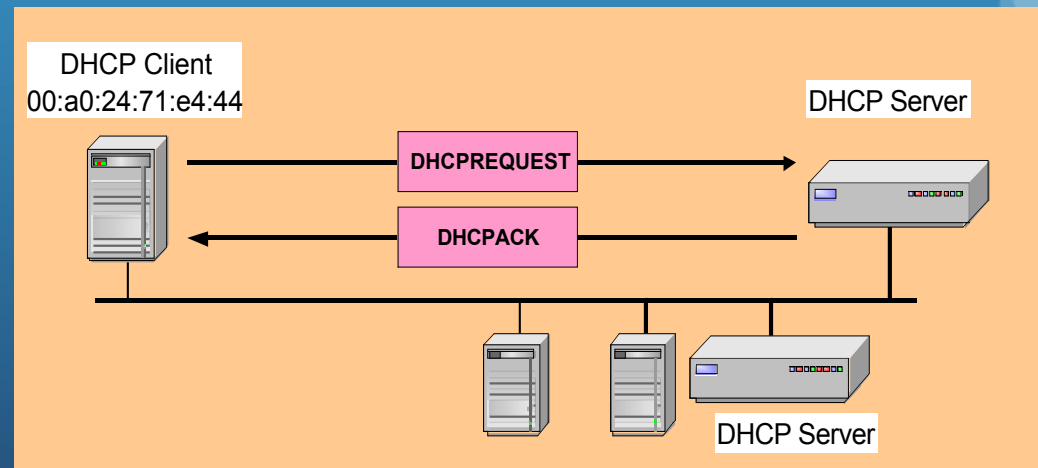
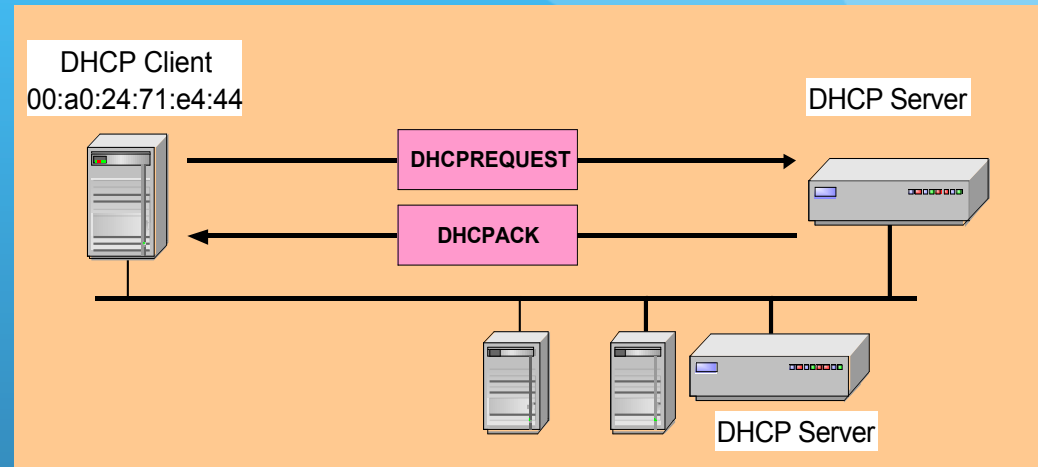
- The client broadcasts a DHCPDISCOVER message on its local physical subnet.
 - The DHCPDISCOVER message **may include some options** such as network address suggestion or lease duration.
- Each server may respond with a DHCPOFFER message that includes an available network address (your IP address) and other configuration options.
 - The servers record the address as offered to the client to prevent the same address being offered to other clients in the event of further DHCPDISCOVER messages being received before the first client has completed its configuration.

DHCP Operation - accepts offer from one server

- DHCP REQUEST
Accepts one offer

At this time, the DHCP client can start to use the IP address

- Renewing a Lease (sent when 50% of lease has expired)
- If DHCP server sends DHCPNACK, then address is released when timer expires



Contd.

- If the client receives one or more DHCPOFFER messages from one or more servers.
 - The client chooses one based on the configuration parameters offered and broadcasts a DHCPREQUEST message that includes the server identifier option to indicate which message it has selected and the requested IP address option, taken from your IP address in the selected offer.
 - In the event that no offers are received, if the client has knowledge of a previous network address, the client may reuse that address if its lease is still valid, until the lease expires.

Contd.

- The **servers receive** the **DHCPREQUEST** broadcast from the client.
- Those **servers not selected** by the **DHCPREQUEST** message use the message as notification that the client has **declined** that server's offer.
- The **server selected** in the **DHCPREQUEST** message **commits the binding** for the client to persistent storage and responds with a **DHCPACK** message containing the configuration parameters for the requesting client.

Contd.

- The combination of **client hardware** and **assigned network address** constitute a **unique identifier** for the client's lease and are used by both the client and server to identify a **lease** referred to in any DHCP messages.
- The *your IP* address field in the **DHCPACK** messages contains/confirms the selected network address.

Contd.

- The **client receives** the **DHCPACK** message with configuration parameters.
 - The client performs a **final check** on the parameters, for example **with ARP** for allocated network address, and notes the duration of the lease and the lease identification cookie specified in the DHCPACK message. At this point, the client is configured.
 - If the client **detects a problem** with the parameters in the **DHCPACK** message (the address is already in use on the network, for example), the **client sends** a **DHCPDECLINE** message to the server and restarts the configuration process.

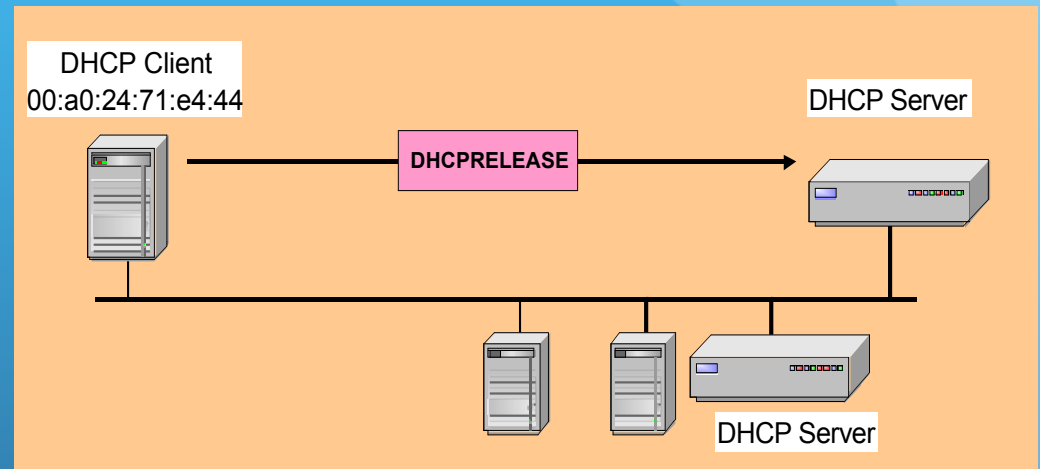
Contd.

- The client should wait a minimum of ten seconds before restarting the configuration process to avoid excessive network traffic in case of looping.
- On receipt of a **DHCPDECLINE**, the server must mark the offered **address** as **unavailable** (and possibly inform the system administrator that there is a configuration problem).
- If the **client receives** a **DHCPNAK** message, the client restarts the configuration process.

DHCP Operation - Release

- DHCP RELEASE

At this time, the DHCP client has released the IP address



Contd.

- The **client** may choose to **relinquish** its lease on a network address by sending a **DHCPRELEASE** message to the server.
- The client **identifies** the **lease** to be released by including its **network address** and its **hardware address**.

Lease Renewal

- When a server sends the DHCPACK to a client with IP address and configuration parameters, it also registers the start of the lease time for that address.
- This lease time is passed to the client as one of the options in the DHCPACK message, together with two timer values, T1 and T2.
- The client is rightfully entitled to use the given address for the duration of the lease time.

Contd.

- On applying the receive configuration, the **client** also **starts** the **timers T1** and **T2**. At this time, the client is in the BOUND state.
- Times T1 and T2 are options configurable by the server but T1 must be less than T2, and T2 must be less than the lease time.
- According to RFC 2132, T1 defaults to $(0.5 * \text{lease time})$ and T2 defaults to $(0.875 * \text{lease time})$.

Contd.

- When timer **T1 expires**, the client will send a **DHCPREQUEST** (unicast) to the server that offered the address, asking to extend the lease for the given configuration. The client is now in the RENEWING state
- The **server** would usually **respond** with a **DHCPACK** message indicating the new lease time, and timers T1 and T2 are reset at the client accordingly.
- The server also resets its record of the lease time.
- Under normal circumstances, an active client would continually renew its lease in this way indefinitely, without the lease ever expiring.

Contd.

- If no DHCPACK is received until timer T2 expires, the client enters the REBINDING state.
- Client now **broadcasts** a **DHCPREQUEST** message to extend its lease.
- This request can be confirmed by a DHCPACK message from **any DHCP server** on the network.

Contd.

- If the client does not receive a DHCPACK message after its lease has expired, it has to stop using its current TCP/IP configuration.
- The client may then return to the INIT state, issuing a DHCPDISCOVER broadcast to try and obtain any valid address.

DHCP Pros

- It relieves the network administrator of a great deal of manual configuration work.
- The ability for a device to be moved from network to network and to automatically obtain valid configuration parameters for the current network can be of great benefit to mobile users.
- Because IP addresses are only allocated when clients are actually active, it is possible, by the use of reasonably short lease times and the fact that mobile clients do not need to be allocated more than one address, to reduce the total number of addresses in use in an organization.

DHCP Cons

- Uses UDP, an unreliable and insecure protocol.
- DNS cannot be used for DHCP configured hosts.

Domain Name Service - DNS

Mostly to facilitate the management and accessibility of hosts on the Internet.

Allows users to access a host using a name instead of an IP address.

Names are mapped onto a valid IP address for routing through the Internet.

What is DNS?

- DNS is a host name to IP address translation service
- DNS is
 - a distributed database implemented in a hierarchy of name servers
 - an application level protocol for message exchange between clients and servers

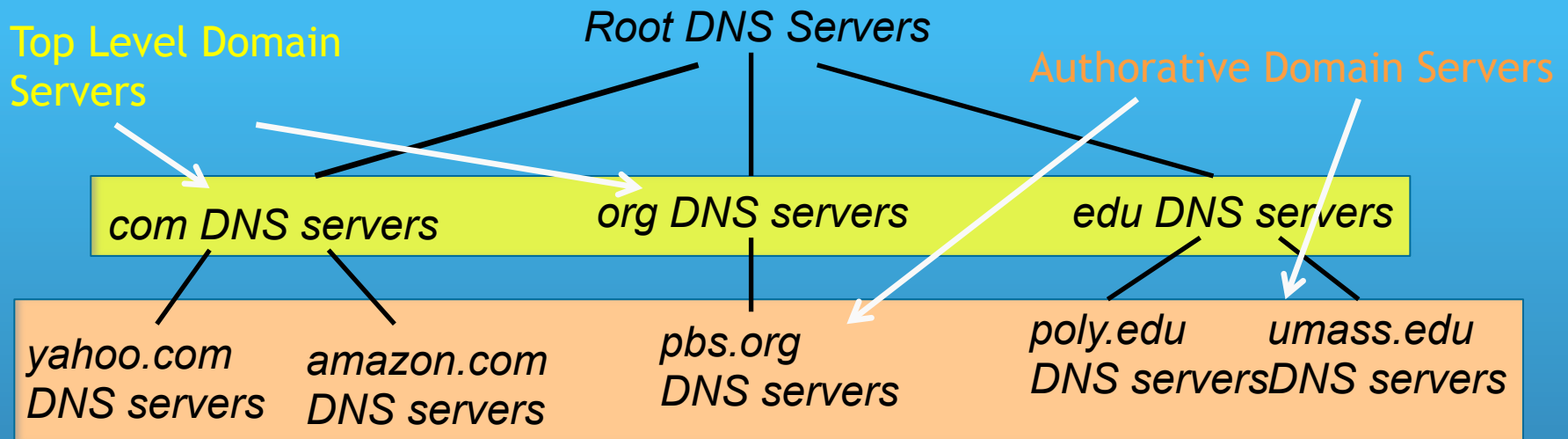
Why DNS?

- It is easier to remember a host name than it is to remember an IP address.
- A name has more meaning to a user than a 4 byte number.
- Applications such as FTP, HTTP, email, etc., all require the user to input a destination.
- The user generally enters a host name.
- The application takes the host name supplied by the user and forwards it to DNS for translation to an IP address.

How does it work?

- DNS works by exchanging messages between client and server machines.
- A client application will pass the destination host name to the DNS process (in Unix referred to as the `gethostbyname()` routine) to get the IP address.
- The application then sits and waits for the response to return.

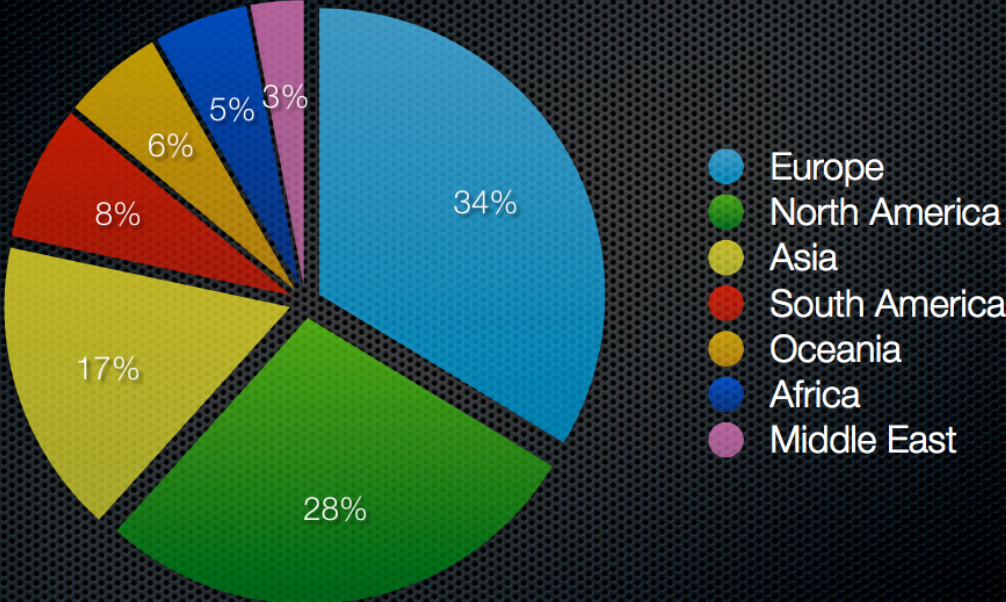
Distributed, Hierarchical Database



Client wants IP for www.amazon.com; 1st approx:

- client queries a root server to find “com” DNS server
- client queries “com” DNS server to get “amazon.com” DNS server
- client queries “amazon.com” DNS server to get IP address for “www.amazon.com”

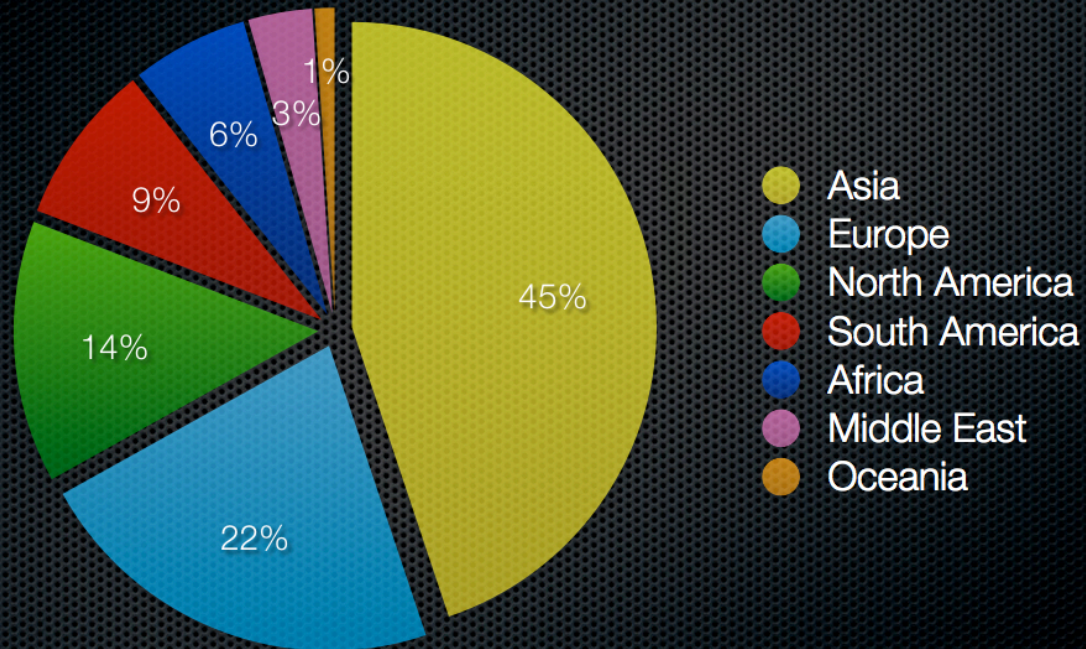
Geographical distribution of DNS root server sites



Data source: Root-servers.org

www.Pingdom.com

Geographical distribution of Internet users



Data source: Internet World Stats

www.Pingdom.com

DNS: Root name servers

- contacted by local name server that cannot resolve name
- root name server:
 - contacts authoritative name server if name mapping not known
 - gets mapping
 - returns mapping to local name server

*Verisign Dulles, VA
Cogent Comm. Herndon, VA
U Maryland College Park, MD
US DoD Vienna, VA
ARL Aberdeen, MD
Verisign*

*NASA Mt View, CA
Internet Systems Consortium. Palo Alto, CA*

*USC-ISI Marina del Rey, CA
ICANN Los Angeles, CA*

RIPE London

Autonomica, Stockholm

WIDE Tokyo



**limited number of
root name server
operators
worldwide**

TLD and Authoritative Servers

- **Top-level domain (TLD) servers:**
 - responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.
 - Network Solutions maintains servers for com TLD
 - Educause for edu TLD
- **Authoritative DNS servers:**
 - organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
 - can be maintained by organization or service provider

Local Name Server

- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
 - also called “default name server”
- when host makes DNS query, query is sent to its local DNS server
 - acts as proxy, forwards query into hierarchy

DNS Queries

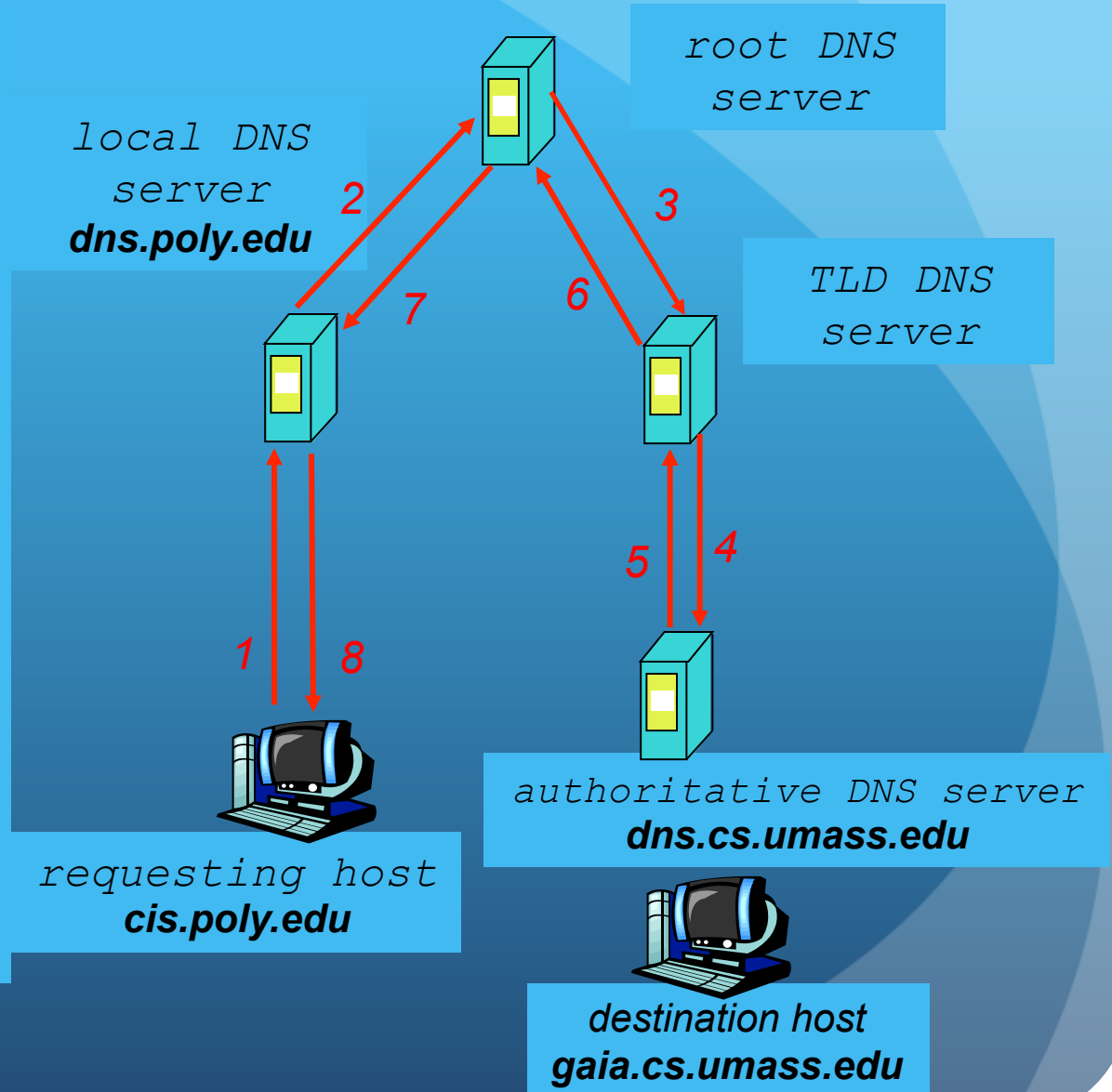
- Recursive:
 - The client machine sends a request to the local name server, which, if it does not find the address in its database, sends a request to the root name server, which, in turn, will route the query to an intermediate or authoritative name server. Note that the root name server can contain some hostname to IP address mappings. The intermediate or Top Level name server always knows who the authoritative name server is.

DNS name resolution example

- Host at cis.poly.edu wants IP address for: gaia.cs.umass.edu

recursive query:

- puts burden of name resolution on contacted root name server
- heavy load



DNS Queries (cont' d)

- Iterative:
 - The local server queries the root server. If address not in its database, will have the name/address of an intermediate or authoritative name server and forward that information to the local name server so that it can directly communicate with the intermediate or authoritative name server. This is to prevent the overloading of the root servers that handle millions of requests.

DNS name resolution example

- Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

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

local DNS server
dns.poly.edu

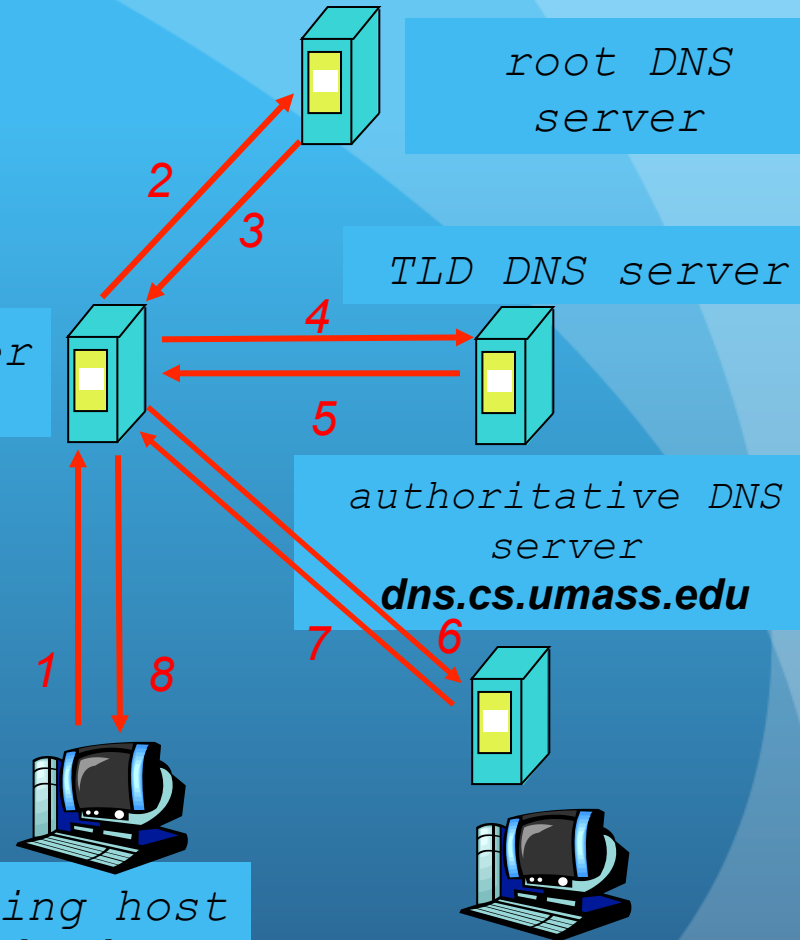
root DNS
server

TLD DNS server

authoritative DNS
server
dns.cs.umass.edu

requesting host
cis.poly.edu

destination
gaia.cs.umass.edu



DNS: caching and updating records

- once (any) name server learns a mapping, it *caches* mapping
 - cache entries timeout (disappear) after some time
 - TLD servers typically cached in local name servers
 - Thus root name servers not often visited

Operation of DNS

- The DNS data is stored in the database in the form of resource records (RR). The RRs are directly inserted in the DNS messages.
- The RRs are a 4 tuple that consist of: {name, value, type, TTL}.

RRs

- TTL: time to live, used to indicate when an RR can be removed from the DNS cache.
- Type =
 - A – then NAME is a hostname and Value its IP address
 - NS – then NAME is a domain name and Value is the IP address of an authoritative name server
 - CNAME – then NAME is an alias for a host and Value is the canonical (actual) name for the host
 - MX – then NAME is an alias for an email host and Value is the canonical (actual) name for the email server

DNS records

DNS: distributed db storing resource records (RR)

RR format: (name, value, type, ttl)

○ Type=A

- **name** is hostname
- **value** is IP address

○ Type=NS

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

○ Type=CNAME

- **name** is alias name for some “canonical” (the real) **name**, eg., **www.ibm.com** is really **servereast.backup2.ibm.com**
- **value** is canonical name

○ Type=MX

- **value** is canonical name of mailserver associated with **name**

Summary

- DNS provides a mechanism for maintaining the user friendliness of the Internet by hiding some of the operational details.
- DNS servers have to be created manually. Recently an update protocol was introduced that allows DNS to exchange data for additions and deletions.

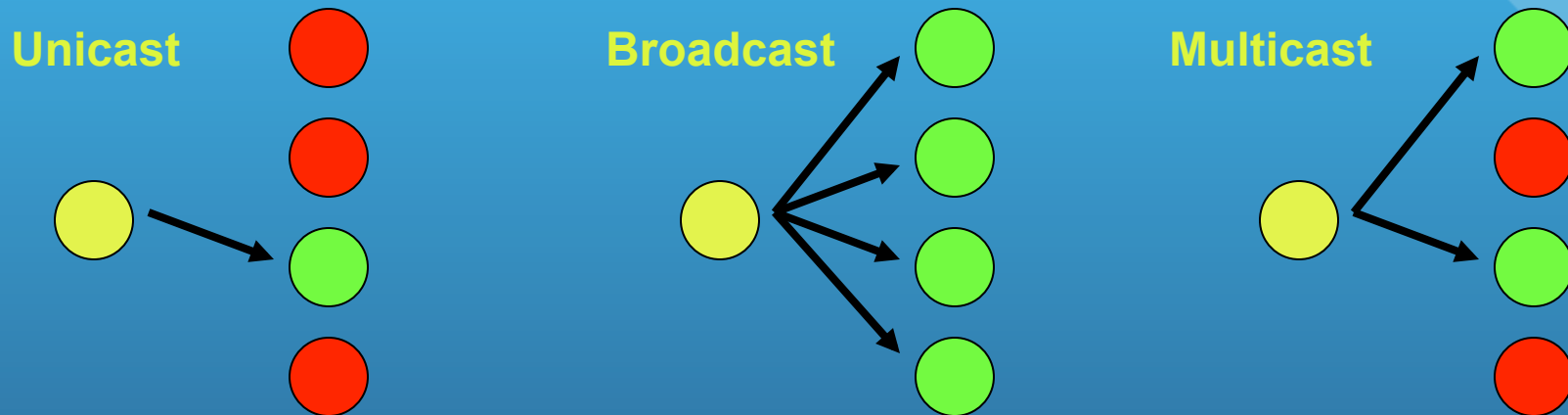
IP Multicasting

The ability to send a message from one device to multiple (N) devices without having to send to each device individually.

The Internet manages the duplication at splitting points in the network

Multicasting

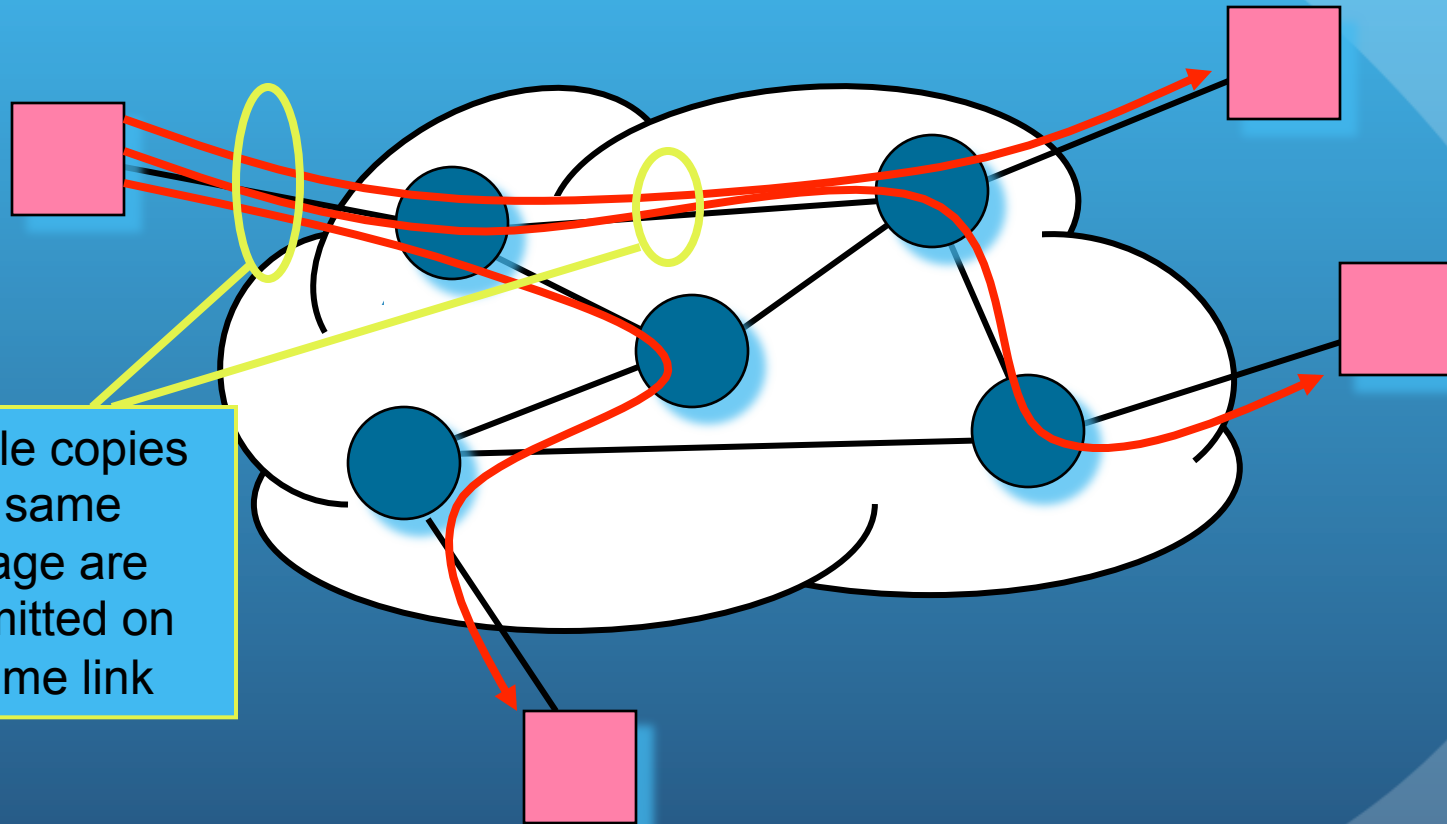
- Multicast communications refers to one-to-many or many-to-many communications.



IP Multicasting refers to the implementation of multicast communication in the Internet

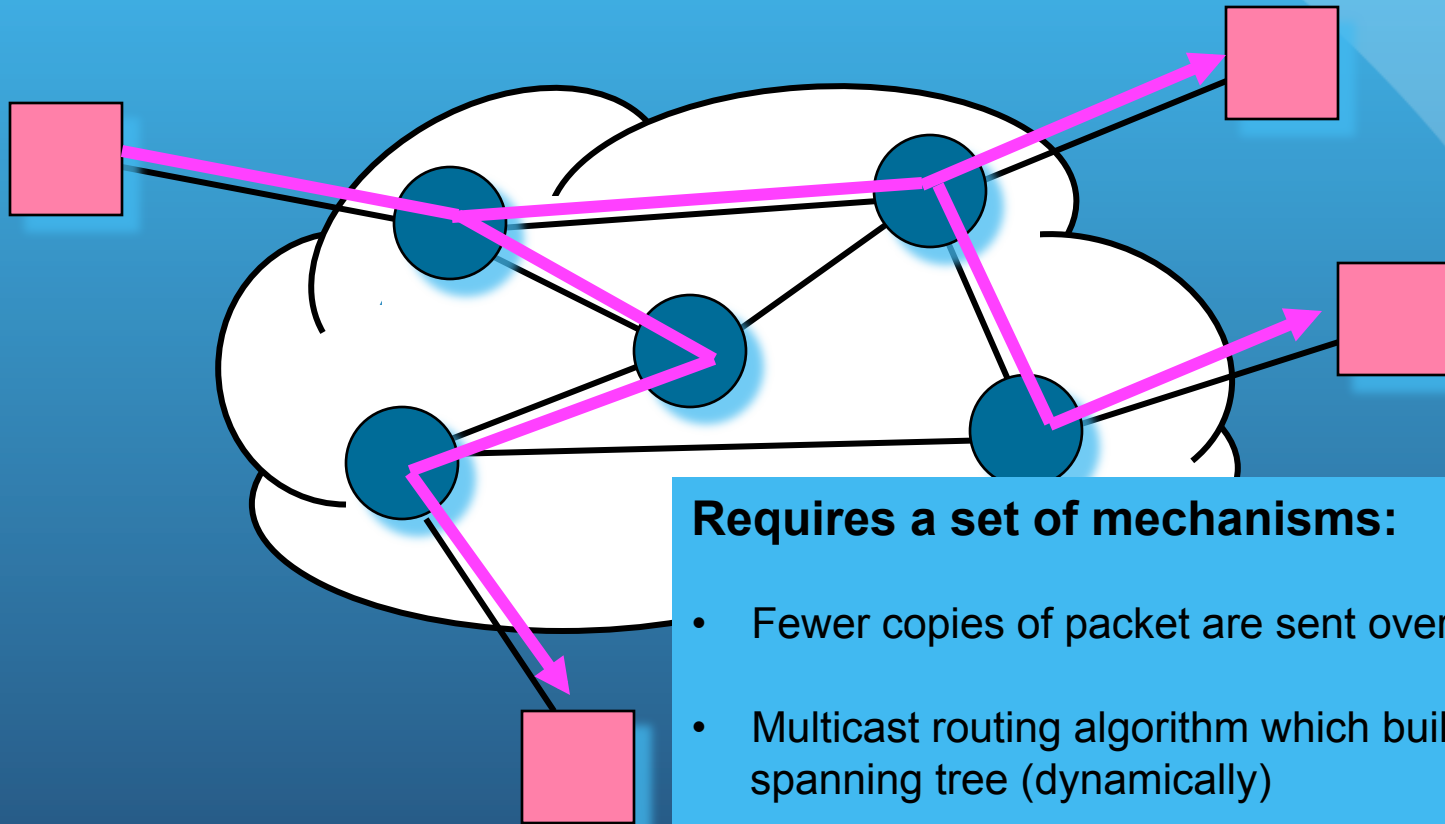
Multicasting over a Packet Network

- Without support for multicast at the network layer:



Multicasting over a Packet Network

- With support for multicast at the network layer:



Requires a set of mechanisms:

- Fewer copies of packet are sent over the links
- Multicast routing algorithm which builds a spanning tree (dynamically)
- Packet forwarding can send multiple copies of same packet where necessary

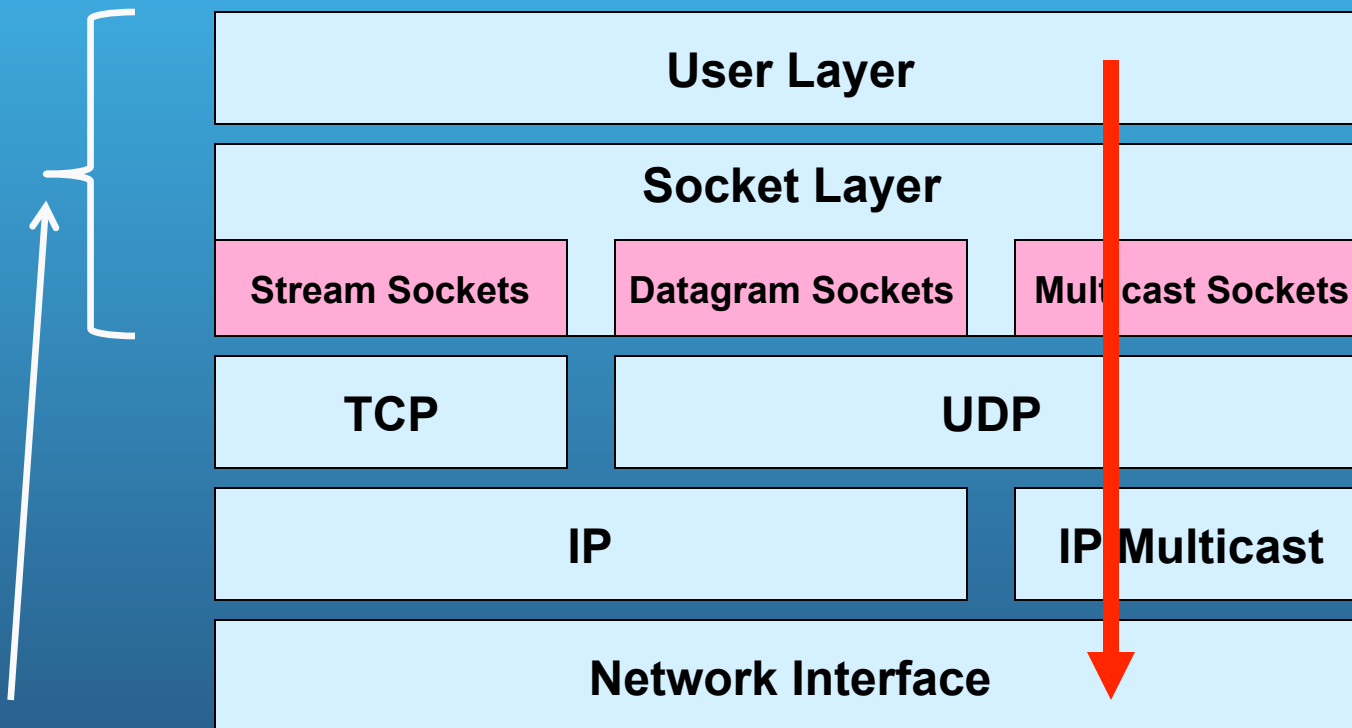
Semantics of IP Multicast

IP multicast works as follows:

- Multicast groups are identified by IP addresses in the range 224.0.0.0 - 239.255.255.255 (OLD class D address)
- Every host (*more precisely*: interface) can join and leave a multicast group dynamically
 - no access control
- Every IP datagram sent to a multicast group is transmitted to all members of the group
- The IP Multicast service is **unreliable**

The IP Protocol Stack

- IP Multicasting only supports UDP as higher layer
- There is no multicast TCP !



Application Layer

Multicast Addressing



	From	To
	224.0.0.0	239.255.255.255

- Multicast addresses are dynamically assigned.
- An IP datagram sent to a multicast address is forwarded to everyone who has joined the multicast group
- If an application is terminated, the multicast address is (implicitly) released.

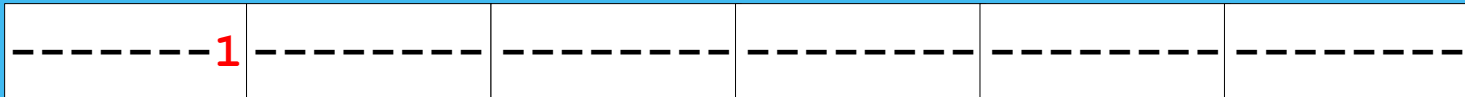
Types of Multicast addresses

- The range of addresses between 224.0.0.0 and 224.0.0.255, inclusive, is reserved for the use of routing protocols and other low-level topology discovery or maintenance protocols
- Multicast routers should not forward any multicast datagram with destination addresses in this range.
- Examples of special and reserved multicast addresses:

224.0.0.1	All systems on this subnet
224.0.0.2	All routers on this subnet
224.0.1.1	NTP (Network Time Protocol)
224.0.0.9	RIP-2 (a routing protocol)

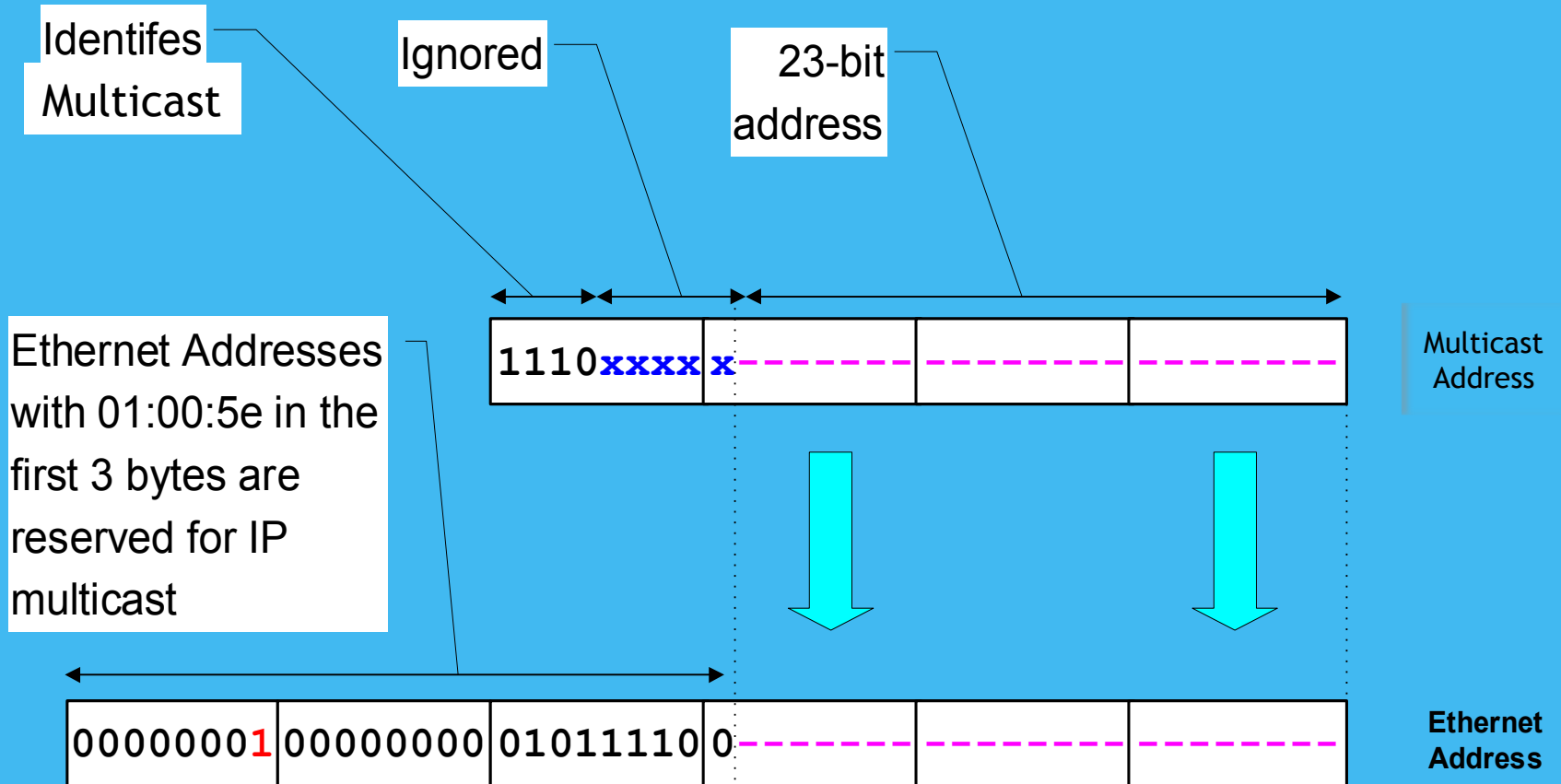
Multicast Address Translation

- In Ethernet MAC addresses, a multicast address is identified by setting the lowest bit of the “left most byte”



Not all Ethernet cards can filter multicast addresses in hardware
Then: **Filtering is done in software by device driver.**

Multicast Address Mapping



IGMP

- The **Internet Group Management Protocol (IGMP)** is a simple protocol for the support of IP multicast.
- IGMP is defined in RFC 1112.
- IGMP operates on a physical network (e.g., single Ethernet Segment).
- IGMP is used by multicast routers to keep track of membership in a multicast group.
- Support for:
 - Joining a multicast group
 - Query membership
 - Send membership reports

Network Time Protocol - NTP

- An application protocol designed to allow network devices to synchronize their clocks
- Critical for delay sensitive applications

To Synchronize or not to synchronize

- Criticality of the situation - how does timing affect the outcome of an action or sequence of actions
- Synchronization of events calls for a common time reference
- The ordering of events is done using a common clock
- Network Time Protocol (NTP) allows for timing exchange to synchronize clocks.

NTP

- NTP is a protocol
- NTP uses a set of time servers
- The time servers are organized in a hierarchy (stratums).
 - Stratum “0” being the top and they are atomic clocks
 - Stratum “1” are time servers connected to stratum “0”
 - And Stratum “2” are connected to stratum “1” etc.
 - Clients get information from their local time server at stratum “N”

NTP Daemon - ntpd

- On most systems, there is an **ntpd** daemon that synchronizes the local clock to a time server in the area. Often a person will pick which time server they want to have their machine use, e.g.,
 - apple.time.com
- NTP operates by getting the time from the local time server and estimating a clock offset to adjust its own clock.

NTP calculation

- Client A sends a packet to time server at time t_0 .
- Client A receives a response from time server at time t_3
- Server receives request at time t_1
- Server sends response at time t_2
- Network latency estimate = $((t_3 - t_0) - (t_2 - t_1))/2$
- Clock offset estimate = $(t_1 - t_0) - \text{Network latency estimate}$
- Packet from Server to Client contains t_0 , t_1 , t_2

NTP Packet Exchange

- UDP port 123
- Packets sent:
 - Client request with timestamp - t_0
 - Server response with 3 timestamps
 - Echo of client timestamp - t_0
 - Receipt of client request - t_1
 - Time of response to client - t_2
- Client can have several servers
- Client chooses one to sync with
- Uses feedback loop to keep running estimate of RTT and offset.