

ΕΠΛ 602: Foundations of Internet Technologies

Internet Protocols (TCP), DNS, Request-Reply (HTTP)

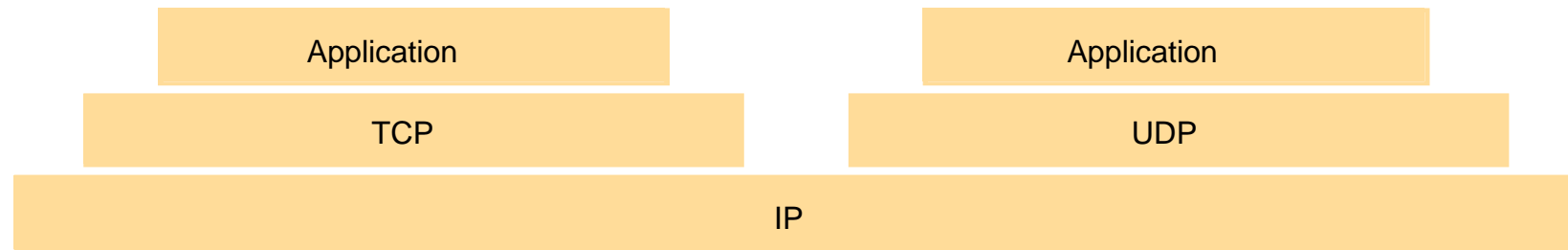
Lecture Outline

- ❖ Main points of Lecture 4
- ❖ The Internet Protocols: TCP
- ❖ DNS (name services)
- ❖ Request-Reply Protocols (HTTP)

Main Points

Internet Protocols

Internet does not follow the OSI



(inter) Network-layer packets consist of a header and a data field *HOST TO HOST*
(inter) Network Protocol: IP

Transport layer

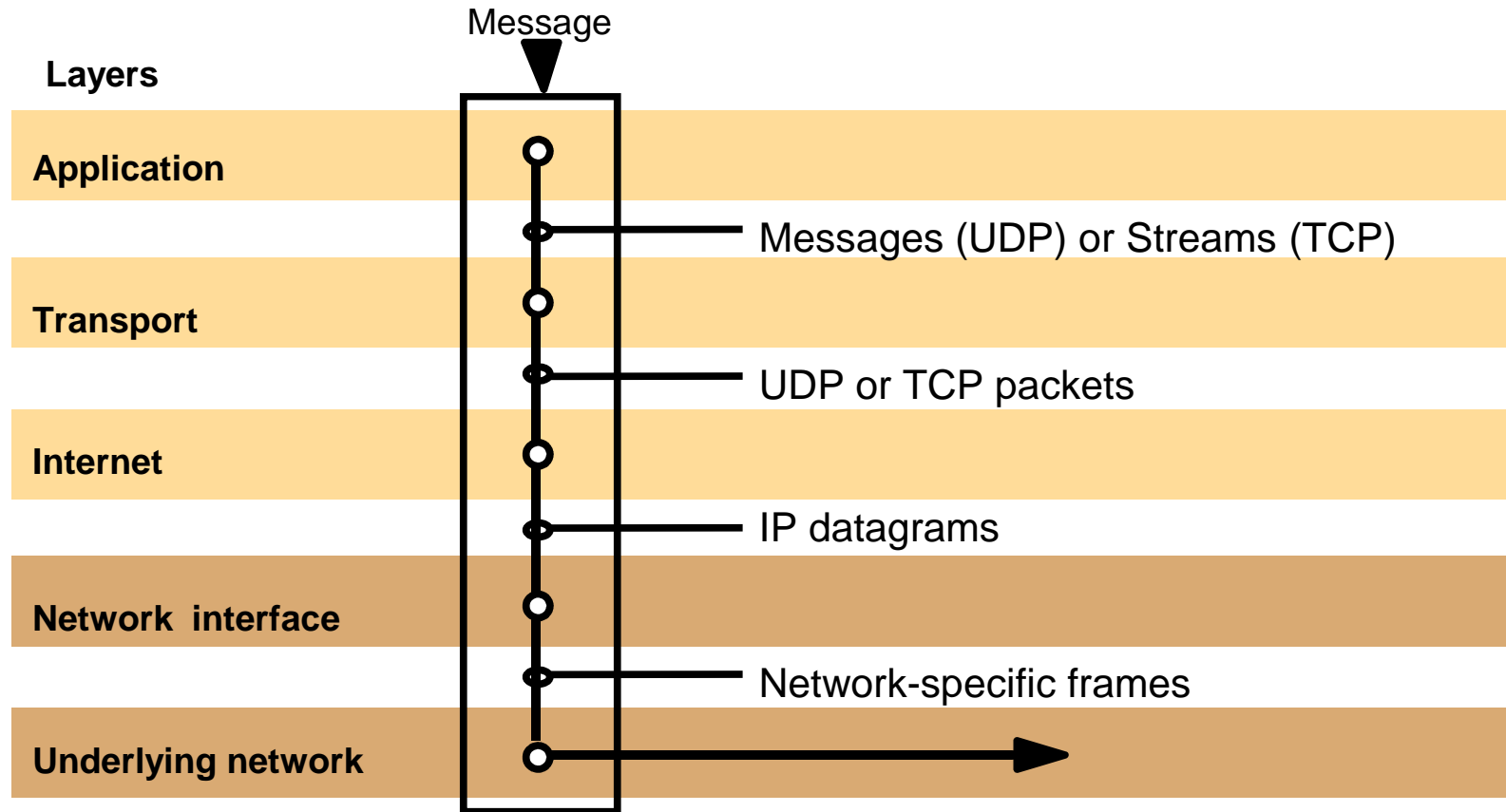
Deliver messages to destinations with transport addresses *PROCESS TO PROCESS*

Transport protocols: TCP (reliable connection-oriented) UDP (datagram protocol that does not guarantee reliable delivery)

Transport address: network address + a port number

– ports are software-defined destination points at a host computer attached to processes

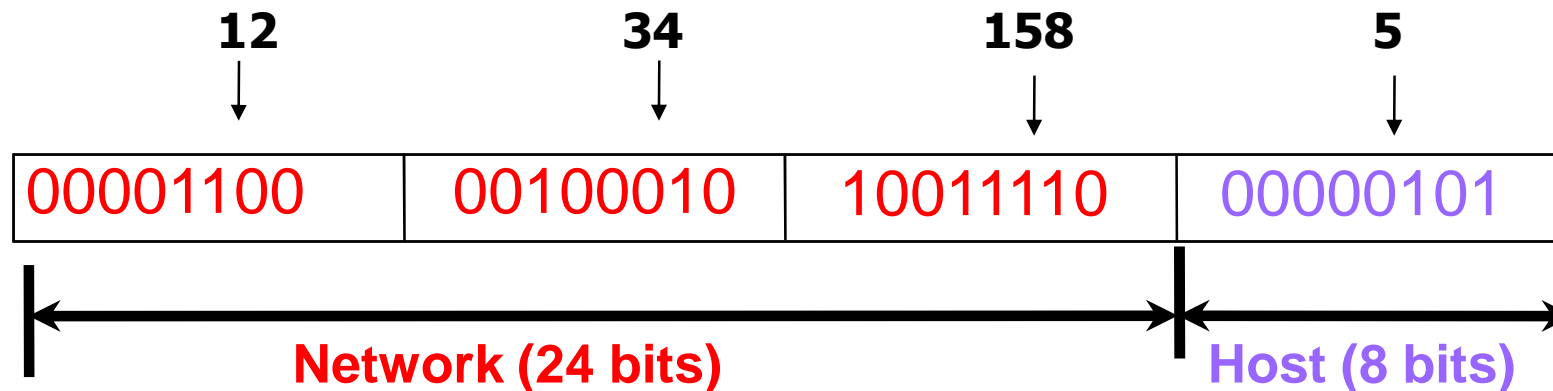
Internet Protocols



Internet Protocols: IP addressing

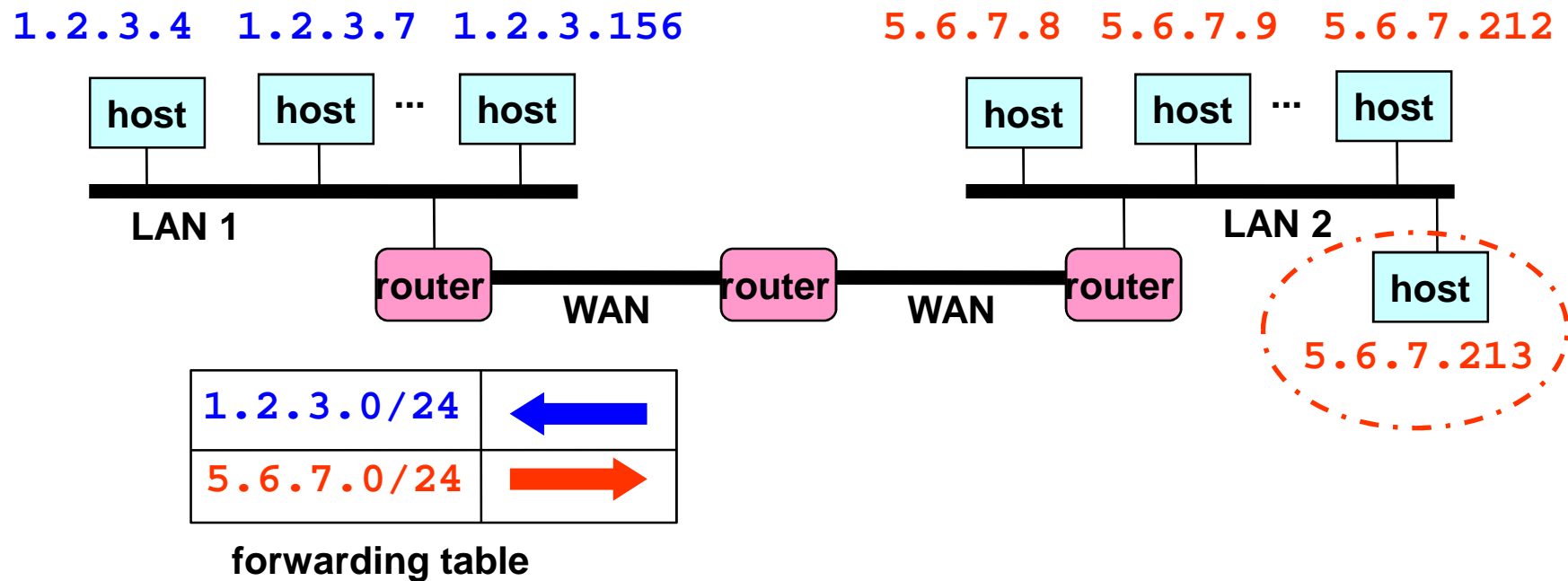
IP version 4

- A unique 32-bit number
 - Identifies an interface (on a host, on a router, ...)
 - Represented in dotted-quad notation
- ▶ Divided into **network** & **host portions** (left and right)
 - ▶ *CIDR (Classless Interdomain Routing)* – how many bits for network (prefix) specified through a mask to indicate bits for network portion that is compared with the routing table entry
 - ▶ 12.34.158.0/24 is a 24-bit prefix with 2^8 addresses



Internet Protocols: IP addressing

- ▶ Scalability
- ▶ Easy to add new hosts - no need to update the routers
 - ▶ E.g., adding a new host 5.6.7.213 on the right does not require adding a new forwarding entry



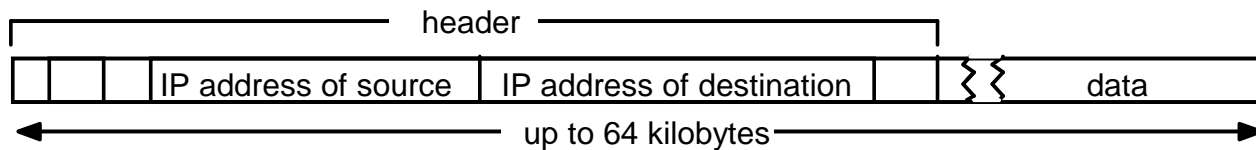
Internet Protocols: IP Addressing (NAT)

- *Not all computers and devices need to be assigned globally unique (i.e., registered) IP addresses*
- Computers attached to a *local network* -> access to **NAT** (Network Address Translation)-enable router that redirects incoming UDP/TCP packets for them
 - Router has a “global” IP address from ISP
 - Local hosts have no register address - Each machine has a “local” IP address via a Dynamic Host Configuration Protocol (DHCP)

NAT-enabled routers maintain an address translation table (ATT)

Internet Protocols: The IP Protocol

Transmits datagrams from one host to another, if necessary via intermediate routers



- Provides only header checksum
- Unreliable or best effort (packets may be lost, duplicated, delayed or delivered out of order)

If the IP datagram is longer than MTU, it is broken into network packets

Internet Protocols: The IP Protocol (ARP)

Address Resolution Protocol (ARP)

converts *Internet (IP) addresses* to *network addresses* for a specific underlying network (link-layer or physical addresses)

e.g., 32-bit Internet addresses to 48-bit Ethernet (MAC) addresses
(the actual hardware interface)

Network topology dependent

- If hosts connected directly to Internet packet switches (no translation)
- Some LANs allow network addresses to be assigned dynamically to hosts chosen to match the id portion of the IP address

Internet Protocols: The IP Protocol (ARP)

Translate *IP addresses* to *Ethernet addresses*.

- ▶ Translation done *only for outgoing IP packets*: this is when the IP header and the Ethernet header are created.
- ▶ Translation is performed with a **table look-up** in an ARP Table; stored (cached) in memory and contains a row for each computer:

IP address	Ethernet address
223.1.2.1	08-00-39-00-2F-C3
223.1.2.3	08-00-5A-21-A7-22
223.1.2.4	08-00-10-99-AC-54

Internet Protocols: The IP Protocol (ARP)

The sending host based on the IP address (and the subnet mask), decides if the destination IP is a local IP or not.

ARP operation for a local host

- Look the ARP table to find the MAC address. If not there,
 - *Broadcast an ARP request* to find out the MAC address for the destination IP.
- All machines on the LAN, receive it
 - If the IP address in the request is their own, reply.
- On receiving this information,
 - update ARP table to include the new information and
 - send out the frame (addressed with MAC address).

Internet Protocols: The IP Protocol (ARP)

ARP operation for a remote host

local host -> local gateway (router)

If the sending host knows the subnet mask and has a default gateway
use its MAC address

Else

The (local) gateway (router) will see the broadcast
and reply with its own MAC address.

The Router will un-encapsulate the data link frame and pass the data part up to the network layer.

Internet Protocols: The IP Protocol (ARP)

Router ->* Destination Router

At the *network layer*, the router will

- see that the destination IP address (in the header of the IP packet) does not match its own
- look in its *routing table* for the closest match to the destination IP
- create a new *data link* frame addressed to the next hop (and if the router does not know the hardware address for the next hop it will request it using the appropriate means for the technology in question).
- The data portion of this frame will contain the complete IP packet (where the destination IP address remains unchanged)

This process will continue at each router along the way until the information reaches a router connected to the destination network.

Destination Router -> Destination Host

The router send out an ARP request for MAC addr of the destination IP (if not in its table)

ARP broadcasts are needed only when a computer is newly connected to the local Ethernet

Internet Routing

Simplified Example

(subnet masks of 255.255.255.0)

1. Network 1 200.0.1.0/24 (ethernet)

Default gateway (Router A) 200.0.1.1 & also connected to Network 2 with 200.0.2

2. Network 2 200.0.1.1/24 (ethernet)

Default gateway (Router B) 200.0.2.2. & also connected to Network 3 with 200.0.3.1

3. Network 3 200.0.3.0/24 (ethernet)

Source IP 200.0.1.2 -> Destination IP 200.0.3.2

Assuming no information cached in ARP

1. **Source** creates an IP packet addressed to 200.0.3.2.
2. Packet sent to the data link layer, send an ARP request for the default MAC address of the gateway (MAC for 200.0.1.1?).
3. The source sends out the IP packet (still addressed to 200.0.3.2) encapsulated within a data link frame addressed to the MAC address of router A interface on Network 1

Internet Routing

Source IP 200.0.1.2 -> Destination IP 200.0.3.2

4. **Router A** receive this frame and send the data portion up to the network layer
5. At the network layer, Router A will see that the packet is not addressed to it and look in its routing table. The routing table shows that Network 3 (the closest match to 200.0.3.2) is reachable via Network 2 and the IP address 200.0.2.2 *for the next hop*
6. At the data link layer, Router A will send out an ARP request onto Network 2 asking for the MAC of Router B (well at least for the interface connected to Network 2).
7. Router A will send the IP packet (still addressed to 200.0.3.2) encapsulated in a data link frame addressed to router B MAC address.
8. **Router B** repeat the same and see that its is directly connected to Network 3
9. Router B will send out an ARP request to learn the MAC address for 200.0.3.2.
10. When received, router B will send out the IP packet (still addressed to 200.0.3.2) encapsulated within a data link frame addressed to the MAC address of the destination computer.
11. The **destination computer** will see that the data link frame is addressed to it and will pass the IP packet to the network layer.
12. At the network layer, the IP address will also match that of the computer and the data from the IP packet will be passed up to the transport layer.

Internet Routing

- ❖ Each layer examines the header and determines where to pass it up to until the data reaches the application running on the destination computer
- ❖ The IP address of the packet never changes (global)
- ❖ At the data link layer, the address used changes at each hop, it is always addressed to the next hop (local)
- ❖ Gateways and routers connected to two or more networks have one IP address for each

Internet Protocols: IP Routing

- ▶ The topological map of the Internet is divided into regions called Autonomous System (AS) subdivided in areas
- ▶ An *Autonomous System (AS)* is a collection of connected Internet Protocol (IP) routing prefixes (group of network IDs) under the control of one or more network operators that presents a common, clearly defined routing policy to the Internet.
- ❖ *This hierarchic structure is a conceptual one primarily for resource management and maintenance*

Internet Protocols: IP Routing

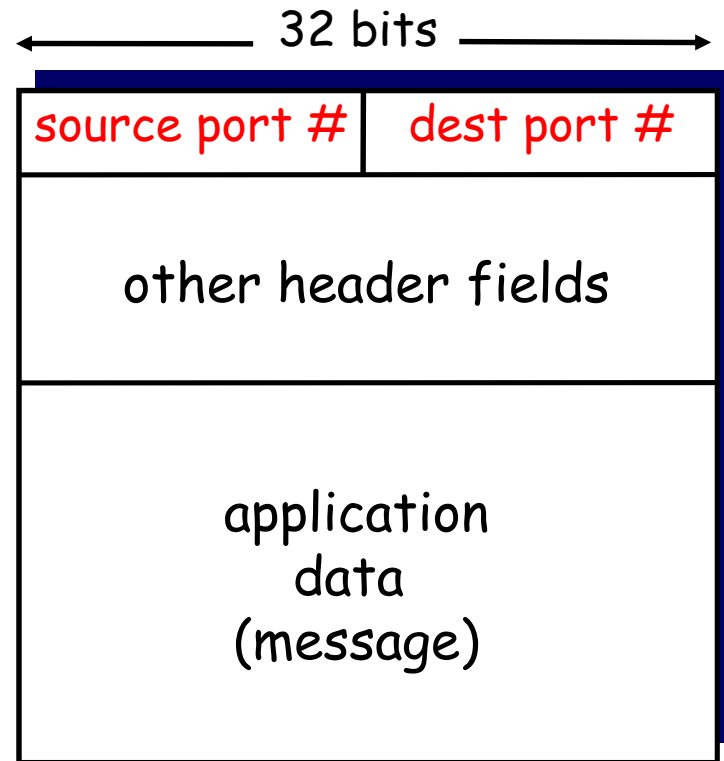
- ▶ Routing *inside an AS* is completely hidden from the rest of the Internet
- ▶ Routing *between ASs* done using the **Border Gateway Protocol (BGP)**
 - ▶ BGP maintains a table of IP networks or 'prefixes' which designate network reachability among ASs
 - ▶ Routes between ASs are computed in terms of **AS hops**: lists of intermediary ASs from the source AS to the destination AS.
 - ▶ All outside networks that belong to the same AS share the same route, expressed as the list of intermediary ASs.
 - ▶ To route to arbitrary ASs on the Internet, a router need only know the **next-hop router to every AS**, rather than to individual destinations.

Internet Protocols: IP Routing

- ▶ The **Internet backbone** refers to the principal data routes between large, strategically interconnected networks and core routers in the Internet.
- ▶ The links in the backbone are usually of high bandwidth and replicated for reliability

Internet protocols (Transport Layer)

- ▶ **Port numbers** are used for addressing messages to process within a particular computer and are valid only within that computer
- ▶ A port number is a 16 bit integer
- ▶ Once an IP packet has been delivered to the destination host, the TCP/UDP layer software dispatches it to a process via a specific port at that host

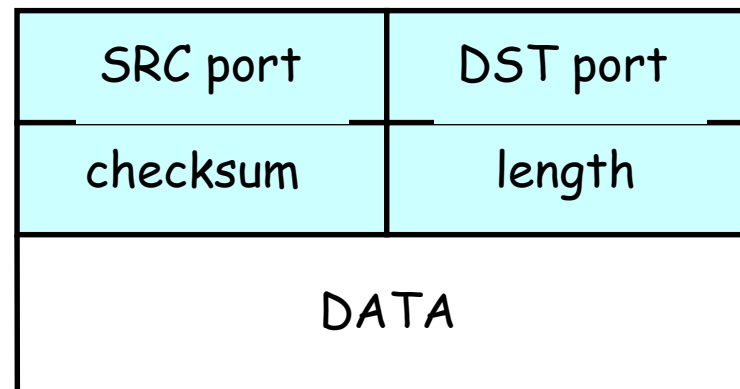


TCP/UDP segment format

Internet protocols: UDP

User Datagram Protocol (UDP)

- ▶ IP plus port numbers to support (de)multiplexing
- ▶ Optional error checking on the packet contents (if checksum field non-zero, receiver computes a check value on the packet content, if no match, the packet is dropped)



Lecture Outline

- ❖ Main points of Lecture 4
- ❖ The Internet Protocols: TCP
- ❖ DNS (name services)
- ❖ Request-Reply Protocols (HTTP)

Internet protocols: TCP

Transmission Control Protocol (TCP)

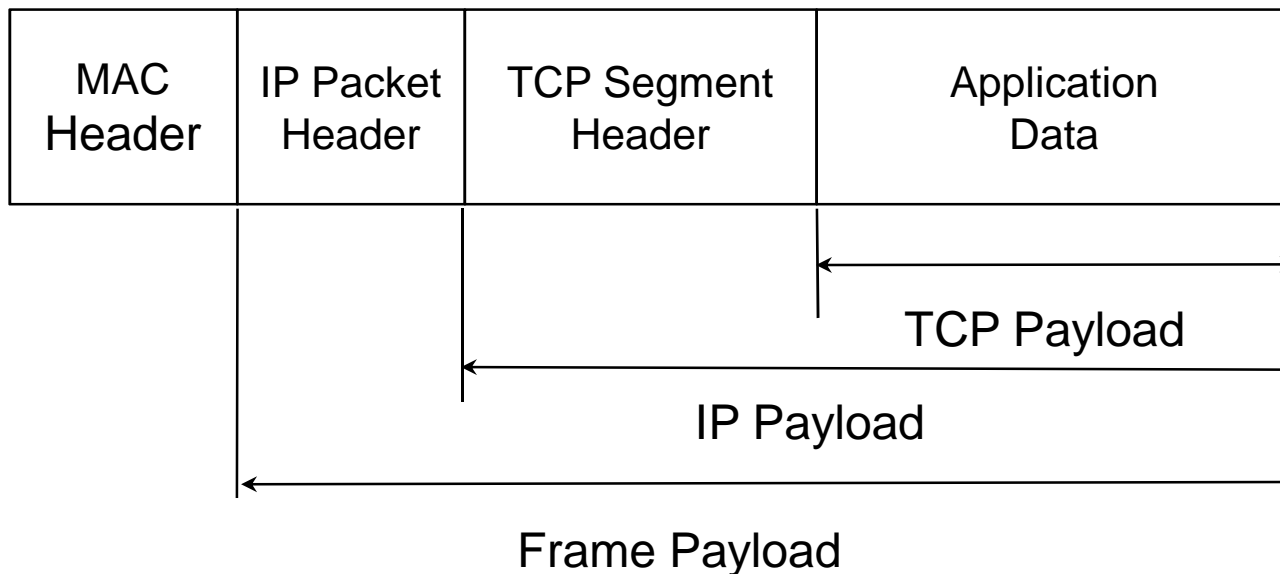
Reliable delivery of arbitrarily long sequences of bytes via a *stream-based* abstraction

Connection-oriented: before any data transfer, the sender and the receiver must cooperate in the establishment of a bidirectional communication channel

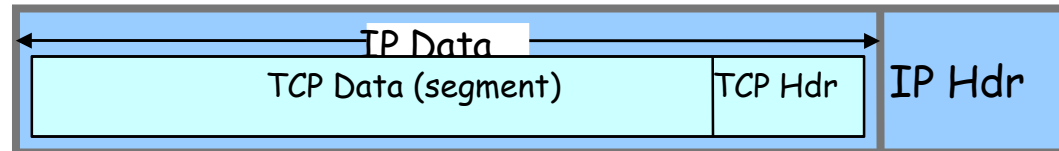
End-to-end agreement (intermediate nodes and routers have no knowledge)

TCP: Segments and sequencing

- ▶ TCP breaks the data stream into **segments**, which the network layer encapsulates into IP datagrams.



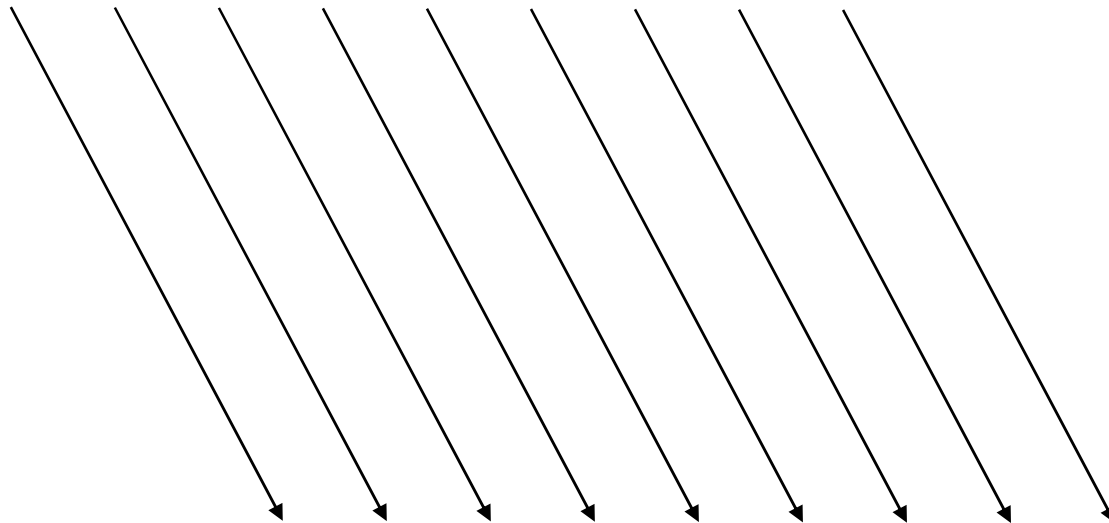
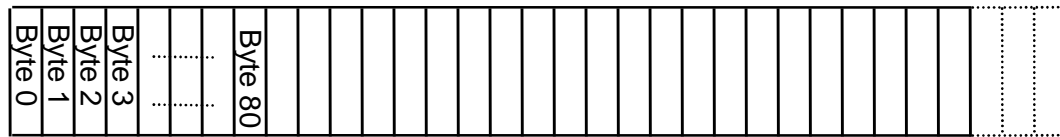
TCP: Segments and sequencing



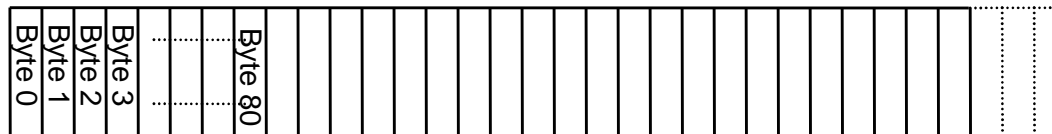
- ▶ IP packet
 - ▶ No bigger than Maximum Transmission Unit (MTU)
 - ▶ E.g., up to 1500 bytes on an Ethernet
- ▶ TCP packet
 - ▶ IP packet with a TCP header and data inside
 - ▶ TCP header is typically 20 bytes long
- ▶ TCP segment
 - ▶ No more than *Maximum Segment Size (MSS)* bytes
 - ▶ E.g., up to 1460 consecutive bytes from the stream

TCP: Segments and sequencing

Host A TCP "Stream of Bytes" Service



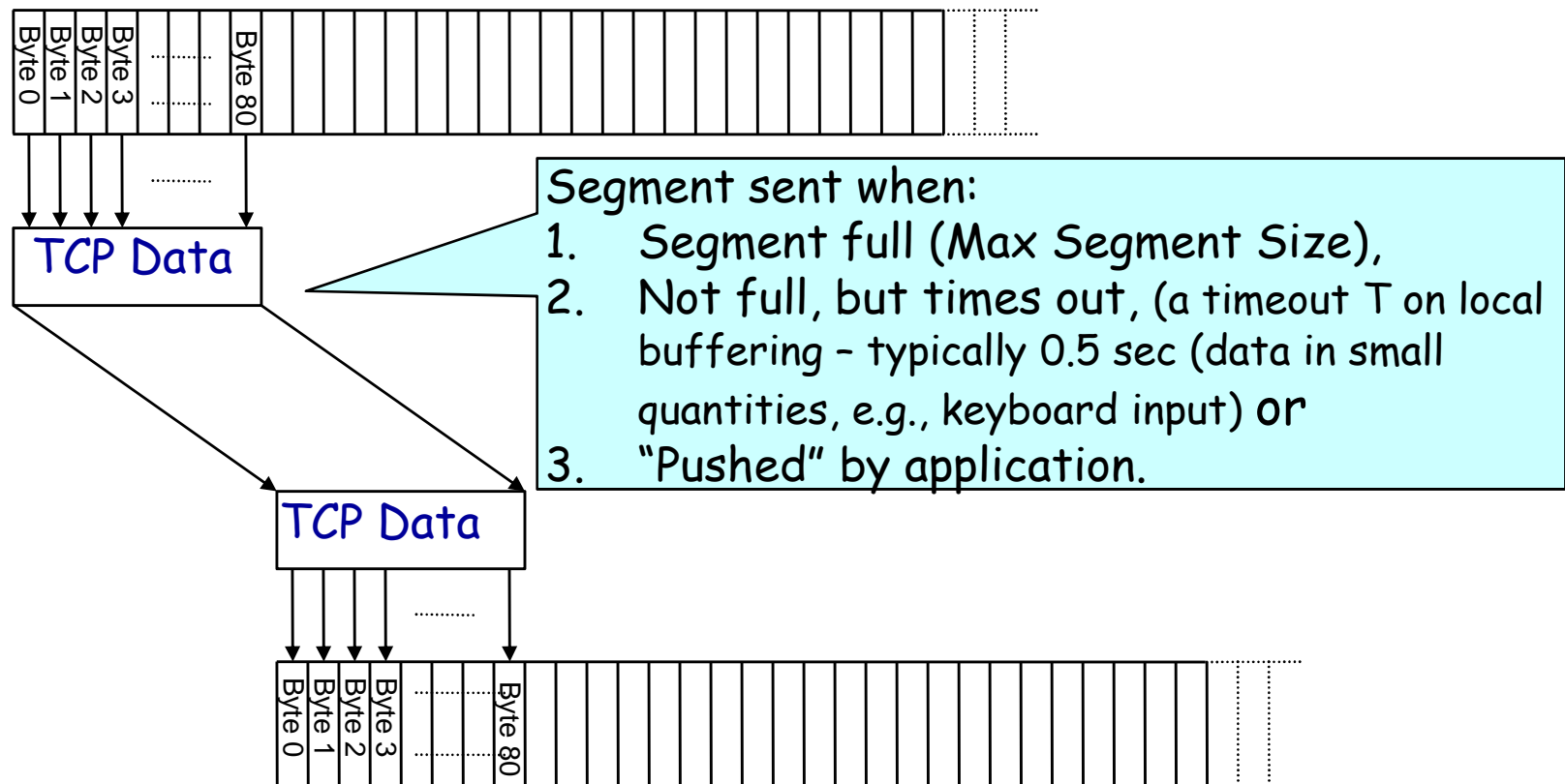
Host B



TCP: Segments and sequencing

...Emulated Using TCP "Segments"

Host A



Host B

TCP: Segments and sequencing

A **sequence number** is assigned to each TCP segment

- The *receiver* uses the sequence number to order the received segments before placing them in the input queue for the receiving process
 - No segments can be placed in the input stream until all lower-numbered segments have been received and placed in the stream
 - Out of order segments are held in buffer

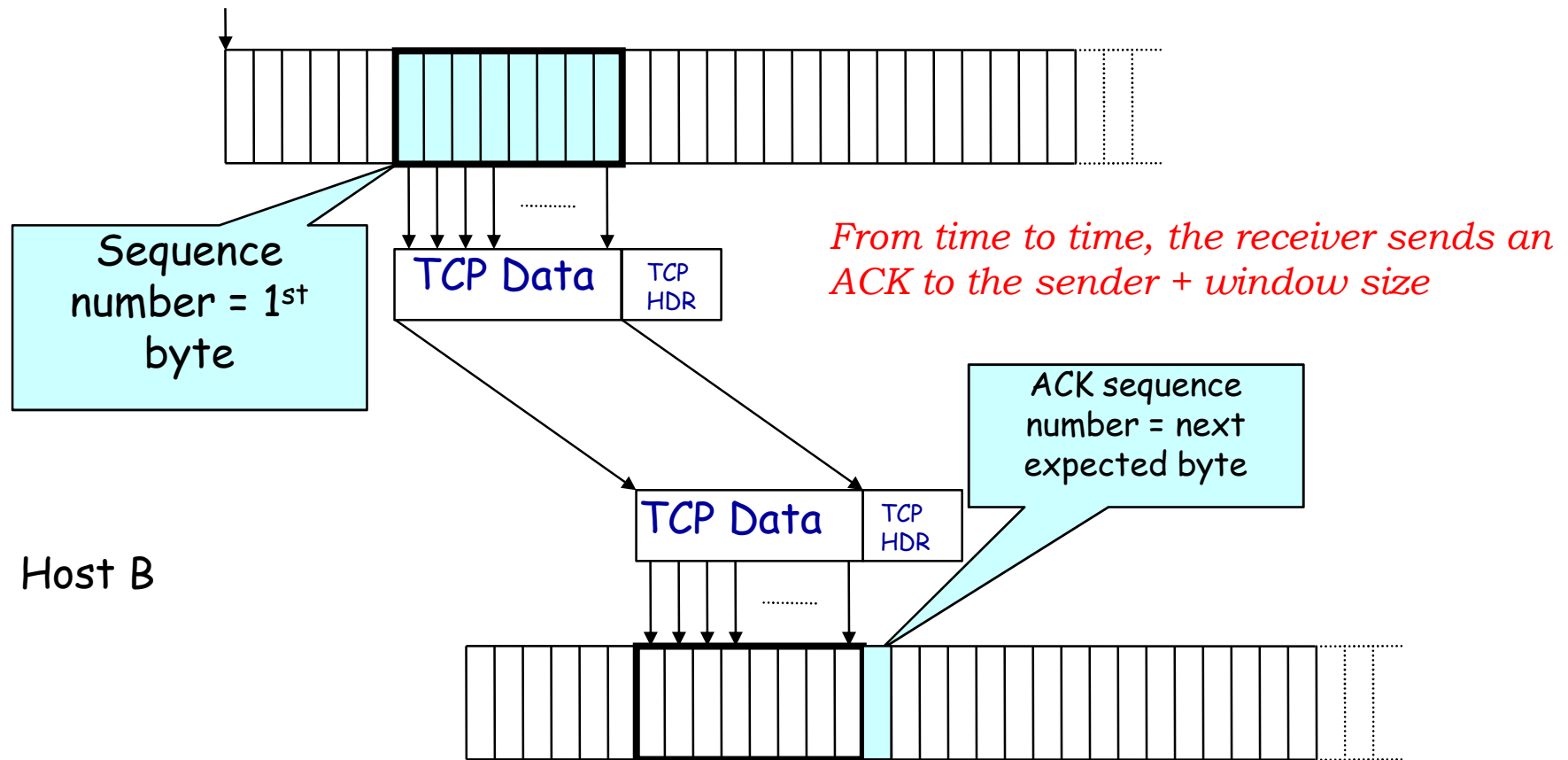
Sequence number = the byte number within the stream for the first byte in the segment

TCP: Segments and sequencing

Host A

ISN (initial sequence number)

sequence number = the byte number within the stream for the first byte in the segment
Used by the receiver



TCP: Segments and sequencing

How to choose the Initial Sequence Number (ISN)?

(i.e., sequence number for the very first byte)

Why not a de facto ISN of 1?

- It introduces the possibility of segments from different connections getting mixed up.
1. Suppose we established a TCP connection and sent a segment containing bytes 1 through 30.
 2. A problem with the internetwork caused this segment to be delayed, and eventually, the TCP connection itself to be terminated.
 3. We then started up a new connection (same IP and port) and again used a starting sequence number of 1.
 4. As soon as this new connection was started, however, the old segment with bytes labeled 1 to 30 showed up.
 5. The other device would erroneously think those bytes were part of the *new* connection.

Each TCP device, at the time a connection is initiated, chooses a 32-bit *ISN* for the connection. How?

TCP: Segments and sequencing

Traditionally, each device chose the ISN from **a timed counter**

- Counter initialized to 0 when TCP started up
- Then its value increased by 1 every 4 microseconds until it reached the largest 32-bit value possible (4,294,967,295) at which point it “wrapped around” to 0 and resumed incrementing.
- Any time a new connection is set up, the ISN was taken from the current value of this timer.

Since it takes over 4 hours to count from 0 to 4,294,967,295, this virtually assured that each connection will not conflict with any previous ones.

But, it makes ISNs predictable.

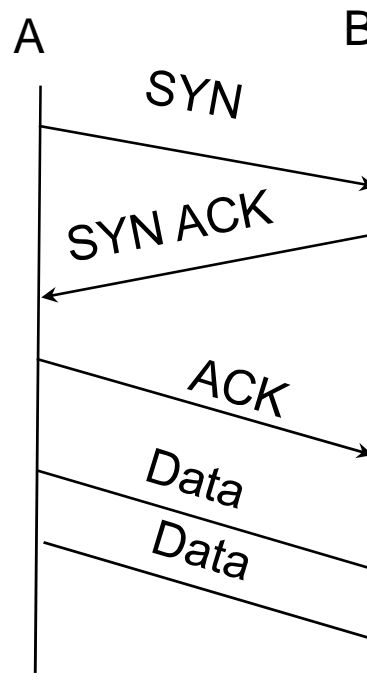
A malicious person write code to analyze ISNs and then predict the ISN of a subsequent TCP connection based on the ISNs used in earlier ones.

Implementations now use **a random number** in their ISN selection process.

TCP: 3-way handshake

Three-way handshake to establish connection

1. Host A sends a SYN (open) to the host B <A tells B it wants to open a connection>
2. Host B returns a SYN acknowledgment (SYN ACK) <B tells A it accepts, and is ready to hear the next byte. Upon receiving this packet, A can start sending data>
3. Host A sends an ACK to acknowledge the SYN ACK <A tells B it is okay to start sending. Upon receiving this packet, B can start sending data>



Each host tells its ISN to the other host.

TCP: 3-way handshake

What if the SYN Packet Gets Lost?

- Suppose the SYN packet gets lost
 - Packet is lost inside the network, or
 - Server rejects the packet (e.g., listen queue is full)
- Eventually, no SYN-ACK arrives
 - Sender sets a timer and wait for the SYN-ACK and retransmits the SYN-ACK if needed
- How should the TCP sender set the timer?
 - Sender has no idea how far away the receiver is
 - Hard to guess a reasonable length of time to wait
 - Some TCPs use a default of 3 or 6 seconds

TCP: 3-way handshake

SYN Loss and Web Downloads

- User clicks on a hypertext link
 - Browser creates a socket and does a “connect”
 - The “connect” triggers the OS to transmit a SYN
- If the SYN is lost...
 - The 3-6 seconds of delay may be very long
 - The user may get impatient and click the hyperlink again, or click “reload”
- User triggers an “abort” of the “connect”
 - Browser creates a new socket and does a “connect”
 - Essentially, forces a faster send of a new SYN packet!
 - Sometimes very effective, and the page comes fast

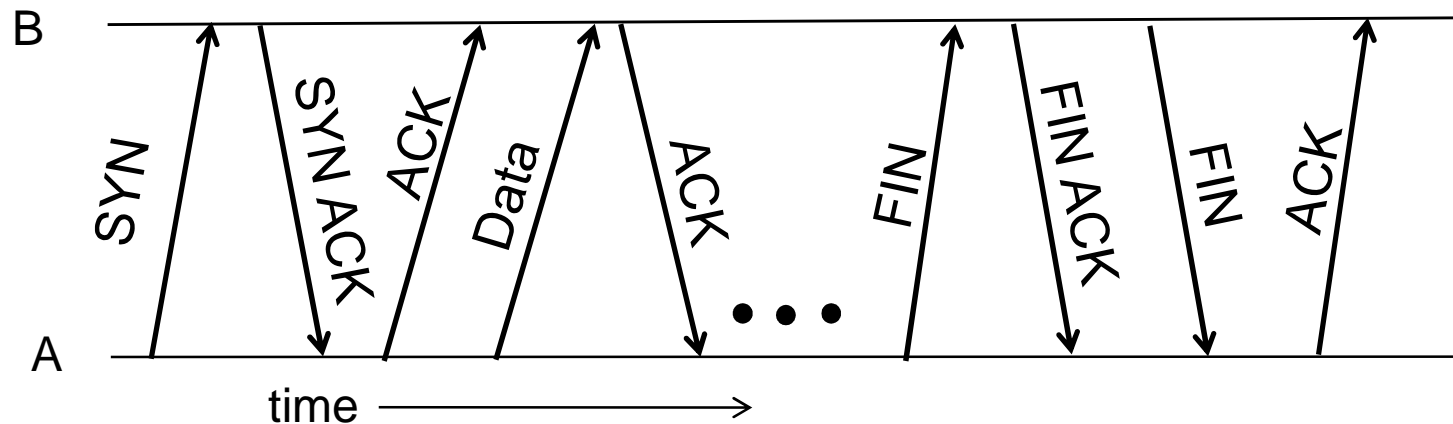
TCP: 3-way handshake

Performance implications

- ▶ TCP performance is suboptimal for transfers of small Web objects:
 - ▶ nearly all TCP implementations send the first data after the handshake is completed
 - ▶ this adds an additional *round-trip time (RTT)* to the application-layer data transfer

TCP: 3-way handshake

Tearing Down the Connection



- ▶ **Closing the connection**
 - ▶ Finish (FIN) to close and receive remaining bytes
 - ▶ Other host sends a FIN ACK to acknowledge
 - ▶ Reset (RST) to close and not receive remaining bytes

TCP: 3-way handshake

- ▶ Sending a FIN: close()
 - ▶ Process is done sending data via the socket
 - ▶ Process invokes “close()” to close the socket

- ▶ Once TCP has sent all of the outstanding bytes then TCP sends a FIN

Receiving a FIN: EOF

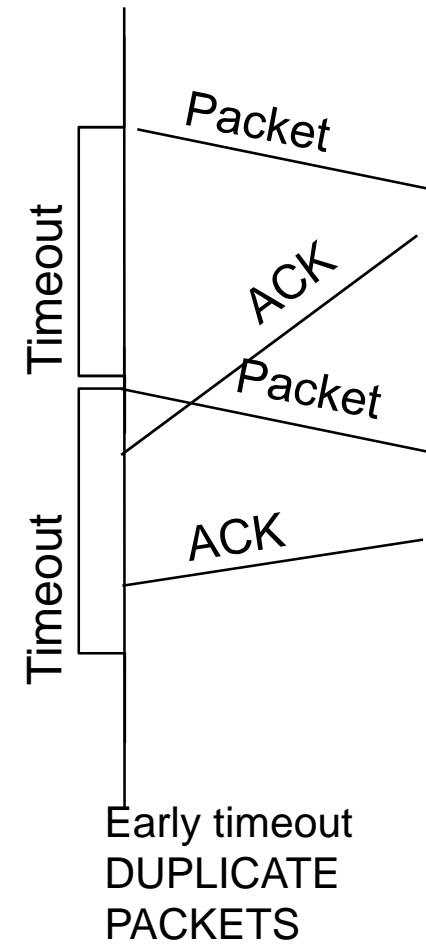
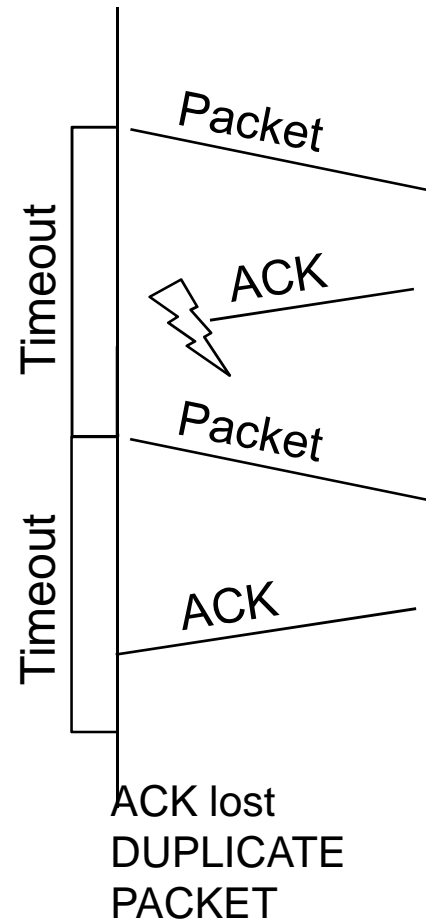
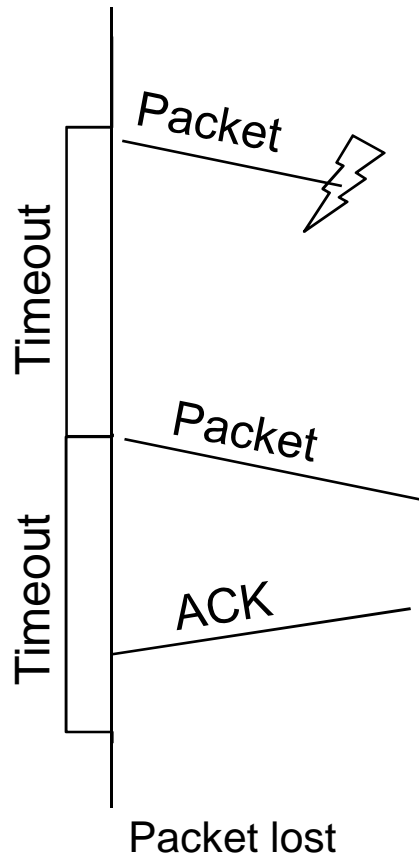
- Process is reading data from the socket
- Eventually, the attempt to read returns an EOF

TCP: Retransmissions

- ▶ A mechanism to deal with unreliability (loss of packets)
- ▶ TCP sender *retransmits segments that have not been acknowledged* by the receiver within a certain time after the initial transmission (**timeout**)
- ▶ The timeout value is adapted according to previously observed RTT (Round Trip Time) and RTT variance between two communicating hosts
- ▶ Initial timeout value set to a predefined value

TPC: Retransmissions

Reasons for Retransmission



TCP: Flow control

Flow control limits the transmission rate to a rate that the receiver can absorb the data

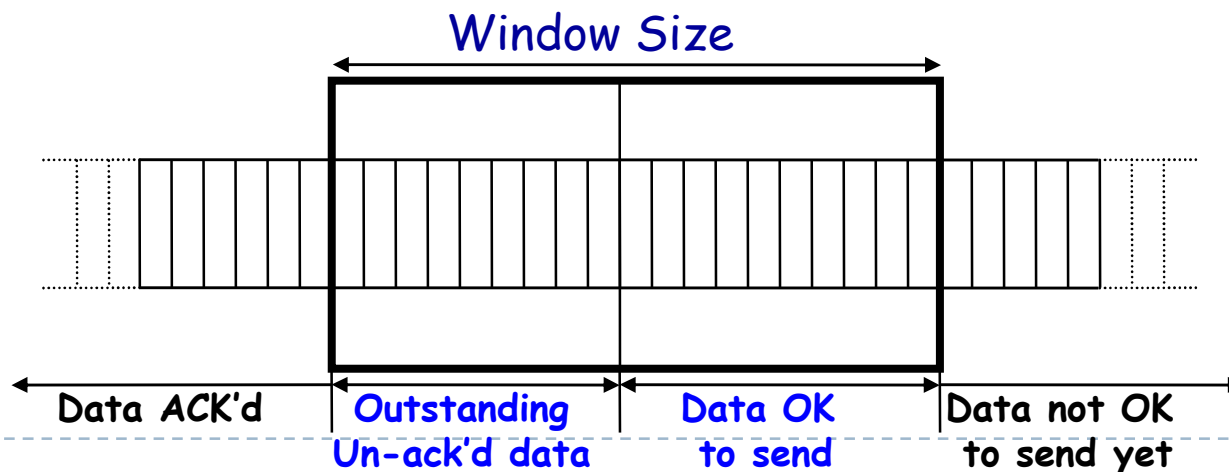
How?

1. TCP makes the sender wait for an ACK after transmitting a certain amount of data
 - ▶ To decide how much data to send before waiting, TCP uses the **sliding window** protocol
2. With every ACK segment, the receiver advertises a window in bytes, using the WINDOW header field
 - ▶ *this window size indicates how many more bytes the receiver is able to accept into its buffers*

TCP: Flow control

Receiver Buffering

- ▶ Window size
 - ▶ Amount that can be sent without acknowledgment
 - ▶ Receiver needs to be able to store this amount of data
- ▶ Receiver advertises the window to the sender
 - ▶ Tells the sender the amount of free space left and the sender agrees not to exceed this amount – Can Re-adjust this!!



TCP congestion control

- ▶ Applied in order to avoid packet loss and re-transmissions that further increase the congestion.
 - ▶ Timeout or dup ACKs
- ▶ TCP congestion control mechanism:
 - ▶ **Congestion window (cwnd)**: specifies the *number of unacknowledged segments* one host may send to the other
- ▶ Basic idea:
 - ▶ sender monitors the loss of packets; when this occurs, the sender reduces quickly (**multiplicatively**) its transmission rate.
 - ▶ if there are no lost packets, sender increases **slowly** its transmission rate

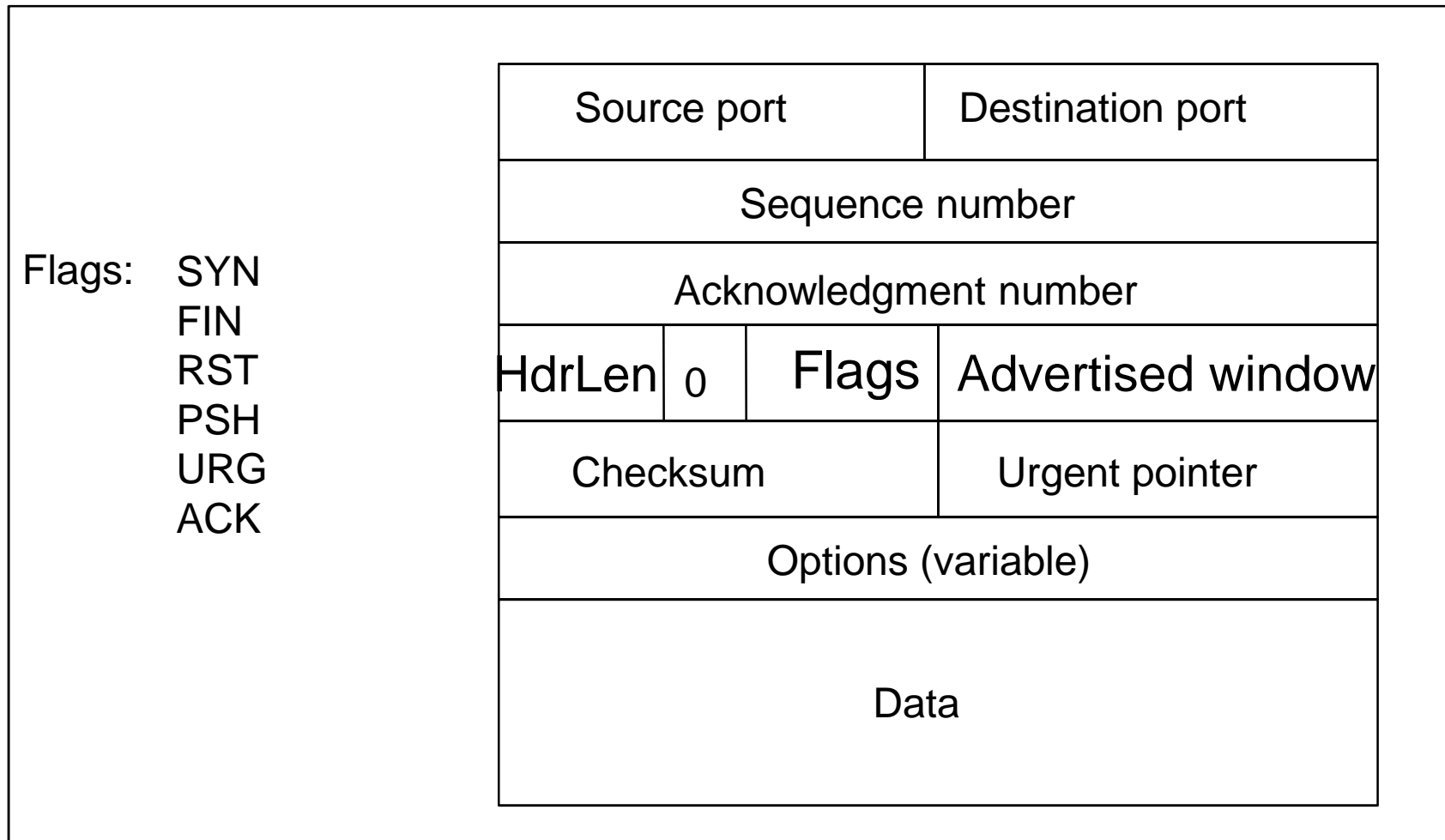
Slow start mechanism

- ▶ Slow start and Congestion avoidance are implemented **together**
- ▶ Start with a small transfer rate (allow only two unacknowledged segments)
- ▶ Initially, increase the congestion window by one segment for each acknowledged segment:
 - ▶ this leads to an **exponential** increase of transfer rate
- ▶ If a packet is lost, reduce drastically the transfer rate by *reducing the congestion window by half*
- ▶ When receiving ACKs, increase cwnd
 - ▶ If $cwnd < half\ the\ window\ of\ congestion$, do slow start else, do congestion avoidance

Performance implications

- ▶ The throughput of one large TCP transfer is usually higher than the throughput of many short ones, with the exception of very short connections that fit into the initial congestion window of two segments (~3KB)
- ▶ Short TCP transfers are affected mostly by the TCP handshake overhead, which cannot be amortized over the life of a connection

TCP Header



TCP Header Fields

- ▶ **Source** and **Destination** port numbers: for multiplexing/demultiplexing at the hosts
- ▶ **Sequence** number: the offset of a segment in a byte stream (the sequence number of the first segment byte in the byte stream)
- ▶ **Acknowledgment number**: confirms that the sender of the segment has received all bytes up to this number from the other host (valid when the ACK bit is set)
- ▶ **Code bits field** (flags): deal with connection management and urgency of the content of the segment (SYN, FIN, RST, ACK)
- ▶ **Window** field: used for flow control

TCP: Basic concepts

Connection oriented

- ▶ Explicit set-up and tear-down of TCP session
- ▶ State maintained at both hosts

Stream-of-bytes service

- ▶ Sends and receives a stream of bytes, not messages (packets)
- ▶ Division of data into datagrams, headers etc are invisible to the application above

Reliable, in-order delivery

- ▶ Checksums to detect corrupted data
- ▶ Acknowledgments & retransmissions for reliable delivery
- ▶ Sequence numbers to detect losses and reorder data

Full duplex transfer

- Allows data transfer between two applications in both directions at the same time

TCP: Basic concepts

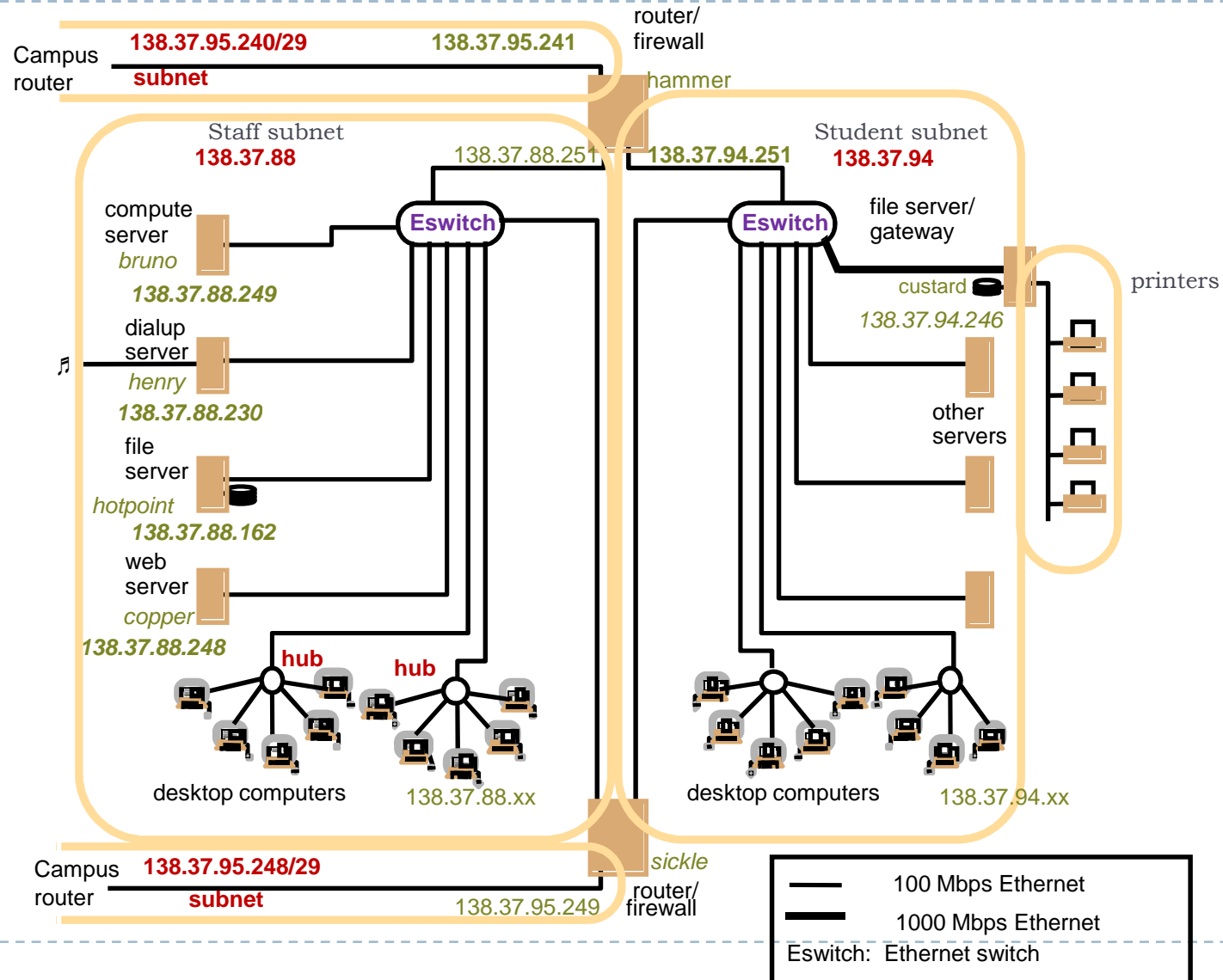
Flow control

- The sender takes care not to overwhelm the receiver or the intervening nodes (overflow their buffer space)

▶ Congestion control

- ▶ Regulates the rate at which the network can transfer the data

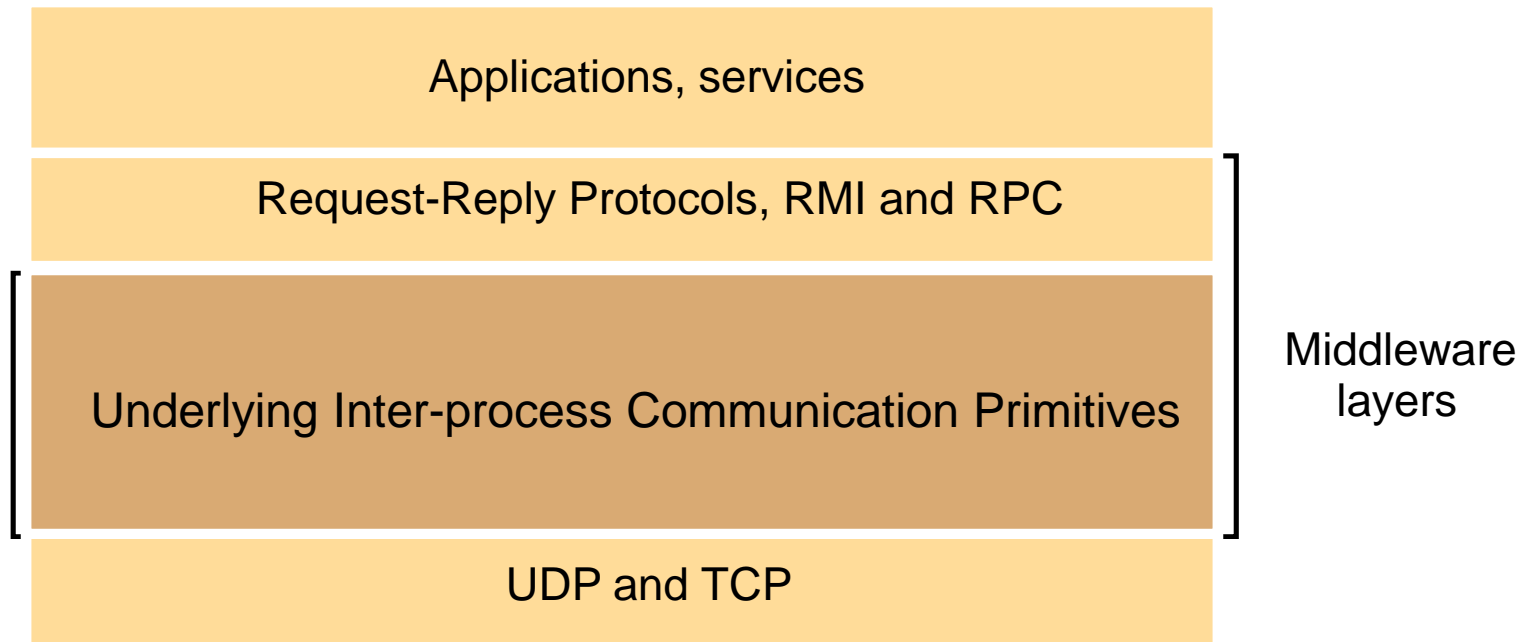
Internetworking: Routers



Lecture Outline

- ❖ Main points of Lecture 4
- ❖ The Internet Protocols: TCP
- ❖ DNS (name services)
- ❖ Request-Reply Protocols (HTTP)

Middleware layers



API for Internet Protocols

UDP and IP from a programmer's point of view

❖ The *application program interface to UDP* provides a **message passing** abstraction

A sending process transmits *a single message* to a receiving process


The independent packets containing the message are called datagrams

❖ The *application program interface to TCP* provides the abstraction of a **two-way stream** between pairs of processes

The information communicated consists of *a stream of data items* with no message boundaries

Producer-Consumer Communication

Data items are queued



API for Internet Protocols

Two message communication operations: send and receive

A queue is associated with each message destination

Synchronous and Asynchronous communication

Send

- ▶ blocking send: process (or thread) waits until the corresponding receive is issued
- ▶ non-blocking send: process (or thread) proceeds as soon as the message has been copied to a local buffer

Receive

- ▶ blocking receive: waits until the msg is received
- ▶ non-blocking receive: if the msg is not here, moves on (a buffer to be filled in the background, receives notifications by polling or interrupt)

- ▶ Synchronous Communication: blocking send and receive
- ▶ Asynchronous Communication non-blocking send and blocking or [non-blocking] receive



API for Internet Protocols

Message Destination

- ▶ IP address + port: (in general) each port one receiver, many senders
- ▶ Location transparency
 - ▶ name server or binder: translate service to location

Reliability

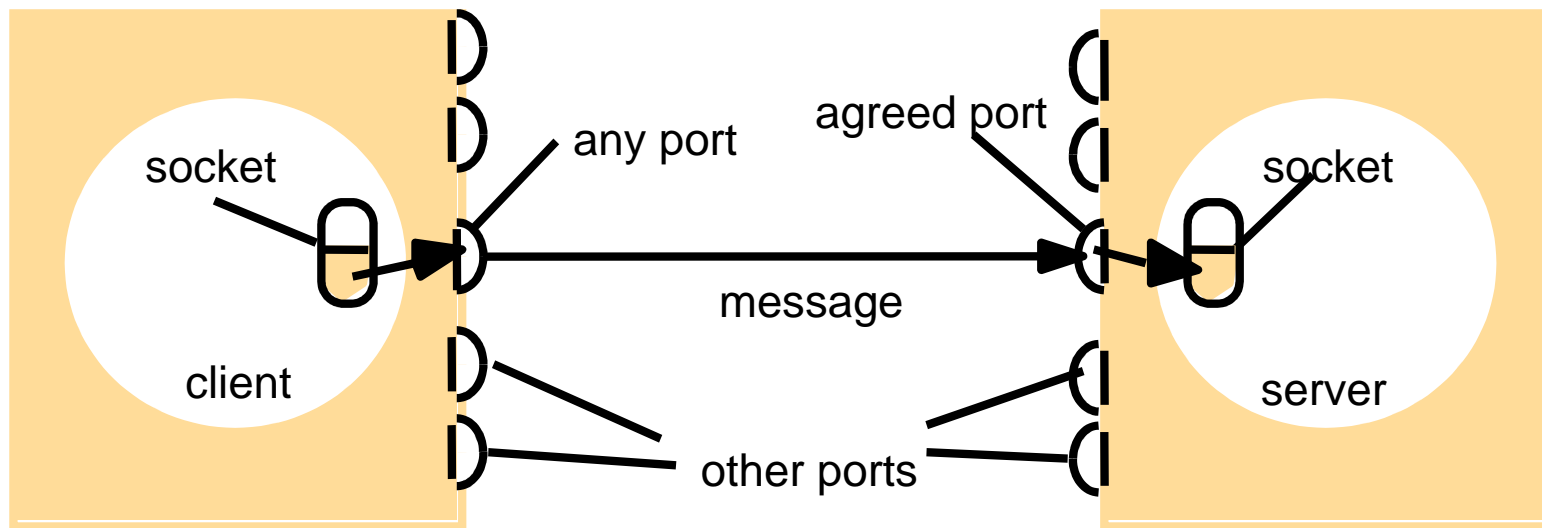
- ▶ Validity: any message in the outgoing message buffer is eventually delivered to the incoming message buffer
- ▶ Integrity: The message received is identical to the one sent (uncorrupted) and no messages are delivered twice.

Ordering



Sockets and ports

- ▶ programming abstraction for UDP/TCP – provides an endpoint for communication between processes
- ▶ Interprocess communication consists of transmitting a message between a socket in one process and a socket in another process
- ▶ To receive messages, a process must be bound to a local port and IP address
- ▶ 2^{16} ports per computer



Internet address = 138.37.94.248

Internet address = 138.37.88.249



UDP Datagram

Without acknowledgments or replies

A *datagram* is transmitted between processes when one sends it and another receives it

To send or receive, *first create a socket* bound to an IP of the local host and a local port

- ❖ A **server** binds its socket to *a server port*, one that it makes known to its clients
- ❖ A **client** binds its socket to *any free local port*
- ❖ The receive method returns the Internet address and port of the sender, to allow the recipient to reply



UDP Datagram

▶ Message Size

The receiving process must specify an array of bytes of a particular size in which to receive a message

IP up to 2^{16} , usually restrict to 8K

▶ Blocking

non-blocking send, blocking receive

Send returns when it has handed the message to the underlying UDP)

A message can be collected from the queue by an outstanding or future receive on that socket

Receive blocks until a datagram is received or timeout

Messages discarded if no process already a socket bound to the destination port



UDP Datagram

Receive from any:

Receive does not specify sender origin (possible to specify a remote host for send and receive)

Failure model:

- ▶ omission failures: messages may be dropped (checksum errors, no buffer space, etc)
- ▶ ordering: can be out of order
- ▶ **use of UDP**
 - ▶ DNS, Voice over IP
 - ▶ less overhead: no state information, extra messages, latency due to start up



TCP stream

Based on the abstraction of stream of bytes to which data is written and from which data may be read

- ▶ Message size:

Unlimited (applications, if necessary, force data to be send immediately)

- ▶ Lost messages: sequence #, ack, retransmit after timeout of no ack
- ▶ flow control: sender can be slowed down or blocked by the receiver
- ▶ message duplication and ordering: sequence #
- ▶ message destination: establish a connection, **one sender-one receiver**, high overhead for short communication



TCP stream

When a pair of processes establish a connection, one plays the role of the server and the other of the client

Then they could be peers

The client role involves **creating a stream socket** bound to any port and then making a **connect request** asking for a server to its server port

The server role involve **creating a listening socket** bound to a server port and waiting for clients to request connections

The listening socket maintains a queue of incoming connection requests

When the server accepts the connection, a **new stream socket** is created for the server to communicate with the client

The pair of sockets in the client and server are connected by a pair of streams. Each socket has an input and an output stream – Each process can send information to the other by writing to its output stream and the other process obtains information by reading from its input stream

Close a socket – no more writes to its output stream



TCP stream

- ▶ matching of data items: two processes need to agree on format and order (protocol)
- ▶ blocking: non-blocking send, blocking receive (send might be blocked due to flow control)
- ▶ concurrency: one receiver, multiple senders, one thread for each connection
- ▶ failure model
 - ▶ checksum to detect and reject corrupt packets
 - ▶ sequence # to deal with lost and out-of-order packets
 - ▶ connection broken if ack not received when timeout
 - ▶ could be traffic, could be lost ack, could be failed process..
 - ▶ can't tell if previous messages were received
- ▶ use of TCP: http, ftp, telnet, smtp



Lecture Outline

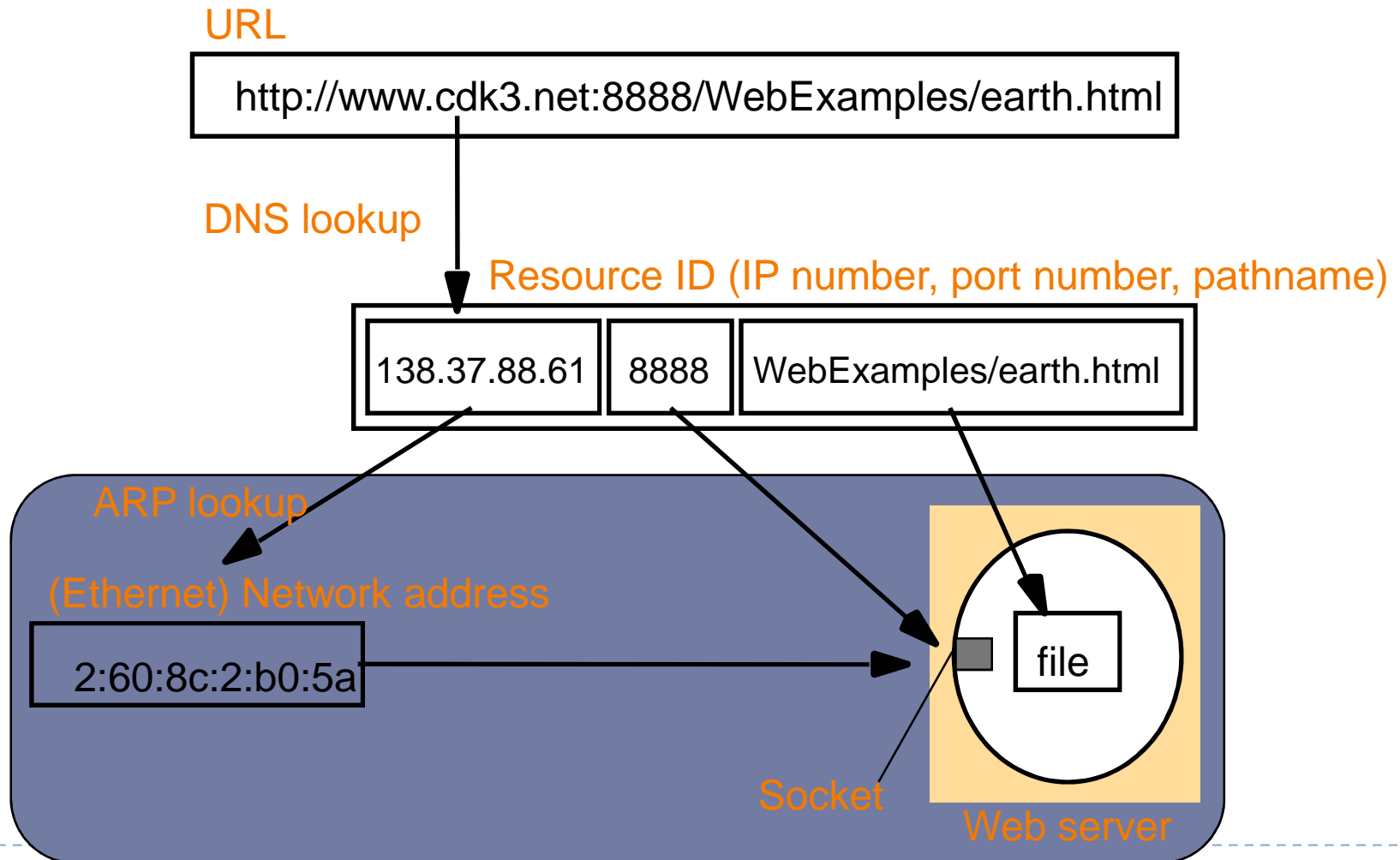
- ❖ Main points of Lecture 4
- ❖ The Internet Protocols: TCP
- ❖ DNS (name services)
- ❖ Request-Reply Protocols (HTTP)

Name Services: DNS

Names, addresses and other attributes

- ▶ Resources or objects (web pages, files, nodes, etc) are accessed using *identifier, names, or addresses*
 - ▶ The term *identifier* is sometimes used to refer to names that are interpreted only by programs
 - ▶ A *name* is human-readable value (usually a string) that can be resolved to an identifier or address
 - ▶ *Address* identifies the location of the resource rather than the resource itself
 - ▶ Other descriptive attributes: values of some property associated with the an object (address may consider a key such attribute)
- A name is **resolved** when it is translated into data about the named resource or object by a **name service**
- **Binding**: the association between an object and a resource

Names, addresses and other attributes



Names, addresses and other attributes

Currently, different name systems are used for each type of resource:

- ▶ resource name identifies
- ▶ File pathname file within a given file system
- ▶ Process process id process on a given computer
- ▶ Port port number IP port on a given computer

Names, addresses and other attributes

- ▶ *Uniform Resource Identifiers* (URIs) offer a general solution for any type of resource.
- ▶ There two main classes:
 - ▶ URL Uniform Resource Locator
 - typed by the protocol field (http, ftp, nfs, etc.)
 - part of the name is service-specific
 - resources cannot be moved between domains
 - ▶ URN Uniform Resource Name
 - requires a universal resource name lookup service - a DNS-like system for all resources

Name services

A *name service* stores information about a collection of textual names in the form of **binding between the names and the attributes** of the entities they denote, such as users, computers, services and objects

The collection is subdivided into one or more *naming contexts*: individual subsets of the binding that are managed as a unit

Main operation: *Name resolution* (look up attributes for a given name)

Name spaces

Name space: the collection of all valid names recognized by a particular service

Follow a syntax (valid vs invalid)

Hierarchical vs Flat name space

(+) more manageable

each part is resolved relative to a separate context of relatively small size and the same name may be used with different names in different contexts (e.g., filesystem, URLs, etc)

(+) different contexts may be managed by different people or organizations

(+) trust

(+) updates, allow restructure of subtrees

(+) potentially infinite, so a system may grow indefinitely

Name spaces

Alias: a name defined to denote the same information as another name

e.g., symbolic links in file systems

URL shorteners in Twitter posts, e.g., <http://bit.ly/ctqjvH>

Friday (Feb 10), 15:00-16:00

Demetris Antoniadis, "**We.b: The web of short URLs**"

Naming domain: a name space for which there exists a single administrative authority for assigning names within it

Name resolution

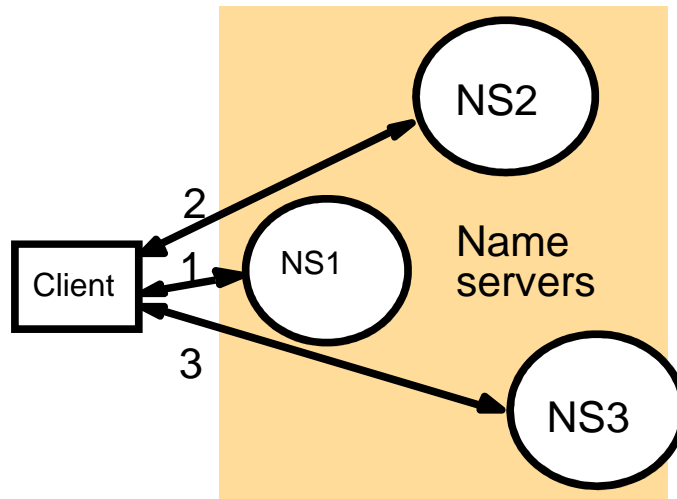
Data is partitioned into servers according to its domain

The process of locating naming data from more than one name server in order to resolve a name is called *navigation*

Iterative navigation vs recursive navigation

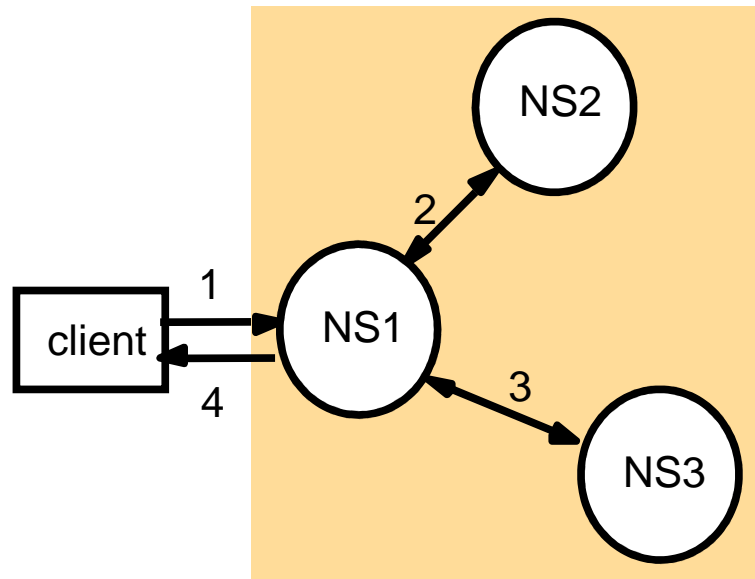
Multicast navigation

Name resolution

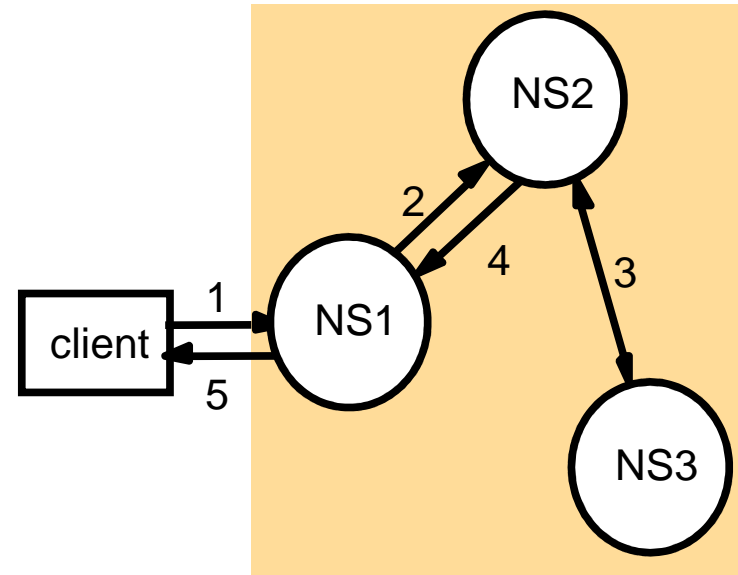


A client iteratively contacts name servers NS1–NS3 to resolve a name

Name resolution



Non-recursive
server-controlled



Recursive
server-controlled

A name server NS1 communicates with other name servers on behalf of a client

Name resolution

Caching

Client name resolution software and servers maintain a cache of the results of previous name resolutions

- Eliminate load from high-level name servers (root)
- Availability
- Communication cost

The Domain Name System (DNS)

Internet supports a scheme for the use of symbolic names for hosts and networks

Domain Name System: host names -> IP addresses

Organized into a naming hierarchy

Hierarchy reflect organizational structure

Independent of the physical layer

The Domain Name System (DNS)

▶ Host names

- ▶ Mnemonic name appreciated by humans
- ▶ Variable length, alpha-numeric characters
- ▶ Provide little (if any) information about location
- ▶ Examples: `www.cnn.com` and `ftp.eurocom.fr`

▶ IP addresses

- ▶ Numerical address appreciated by routers
- ▶ Fixed length, binary number
- ▶ Hierarchical, related to host location
- ▶ Examples: `64.236.16.20` and `193.30.227.161`

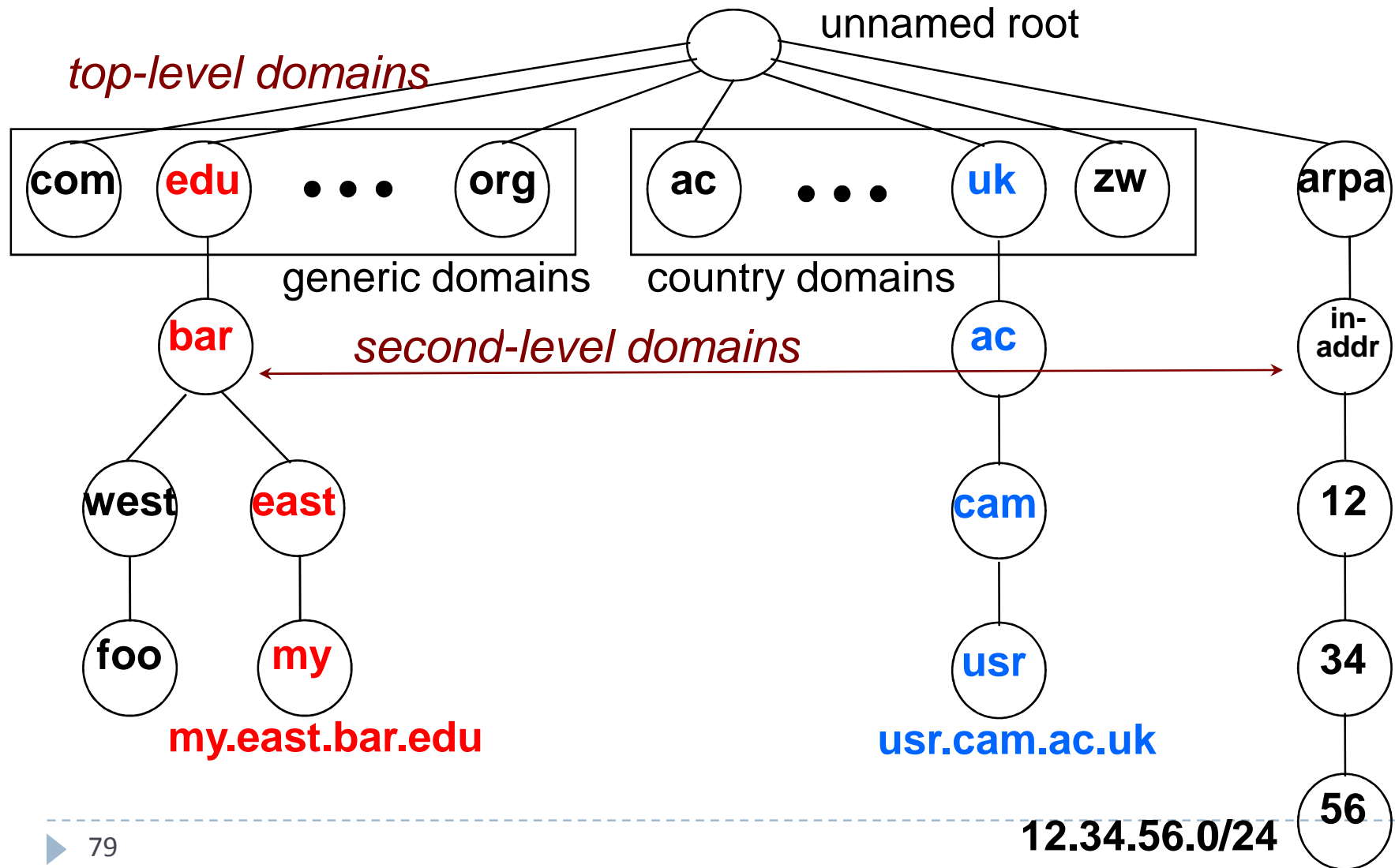
The Domain Name System (DNS)

Separating Naming and Addressing

- ▶ Names are easier to remember
 - ▶ www.cnn.com vs. 64.236.16.20
- ▶ Addresses can change underneath
 - ▶ Move www.cnn.com to 64.236.16.20
 - ▶ E.g., renumbering when changing providers
- ▶ Name could map to multiple IP addresses
 - ▶ www.cnn.com to multiple replicas of the Web site
- ▶ Map to different addresses in different places
 - ▶ Address of a nearby copy of the Web site
 - ▶ E.g., to reduce latency, or return different content
- ▶ Multiple names for the same address
 - ▶ E.g., aliases like ee.mit.edu and cs.mit.edu

The Domain Name System (DNS)

Distributed Hierarchical Database



The Domain Name System (DNS)

- ▶ **Central server**
 - ▶ One place where all mappings are stored
 - ▶ All queries go to the central server
- ▶ **Many practical problems**
 - ▶ Single point of failure
 - ▶ High traffic volume
 - ▶ Distant centralized database
 - ▶ Single point of update
 - ▶ Does not scale

Need a distributed, hierarchical collection of servers

DNS name servers

The DNS database is *distributed* across a **logical network of servers**
Each server holds **part** of the naming database (primarily data from its local domain)

Client-server with multiple servers

DNS client, called **resolver**

- Accepts queries
- Formats them to messages according to the DNS protocol
- Communicates with one or more name servers (list in order of preference)

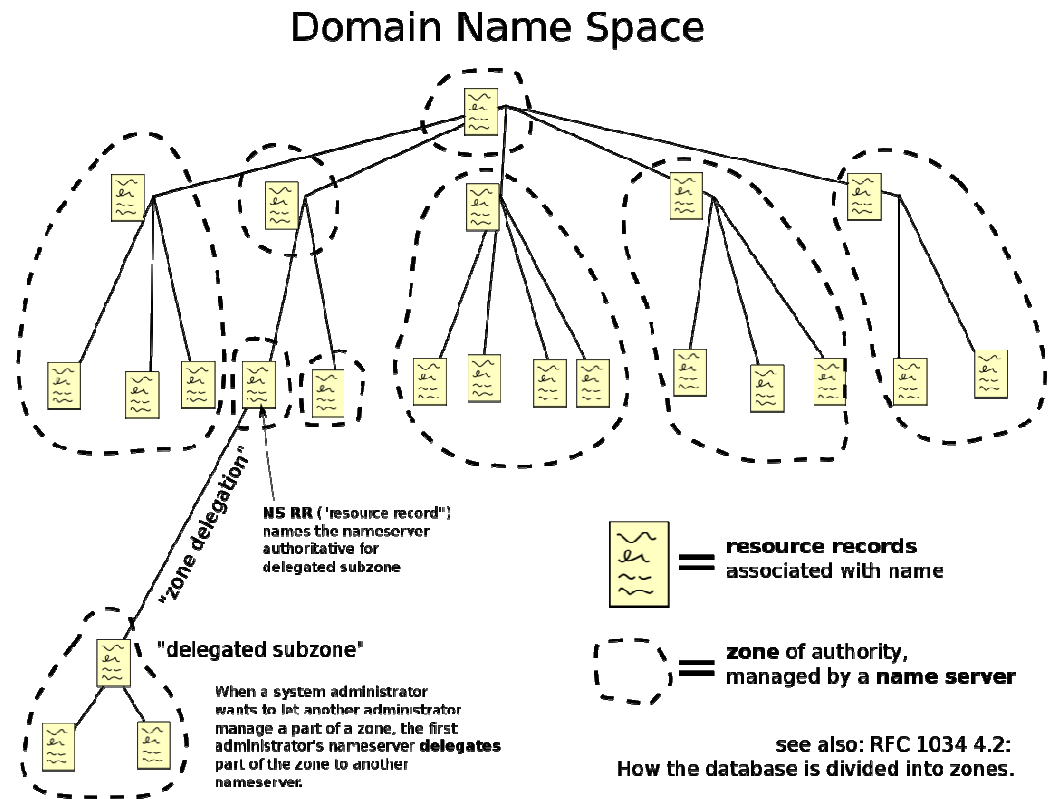
Request-reply protocol + UDP

- Queries for hosts in the local domain are satisfied by servers within that domain
- Each server also records the domain names and addresses of other name servers to satisfy queries outside the domain

DNS name servers: partition

DNS naming data are divided into *zones*

Name servers maintain data for one or more zones



DNS name servers: partition

Authoritative data: data that can be relied upon as being up-to-date

A zone contains

- Attribute data for names in its domain except any sub-domains administered by lower-level authorities
- Name and addresses of at least two name servers that provide *authoritative data* for the zone
- The names of name servers that hold authoritative data for delegated sub-domains
- Zone-management parameters (e.g., for caching)

Each zone must be replicated authoritatively in at least two servers
Primary (master) server and secondary servers

DNS name servers

Data are cached with a time-to-live value

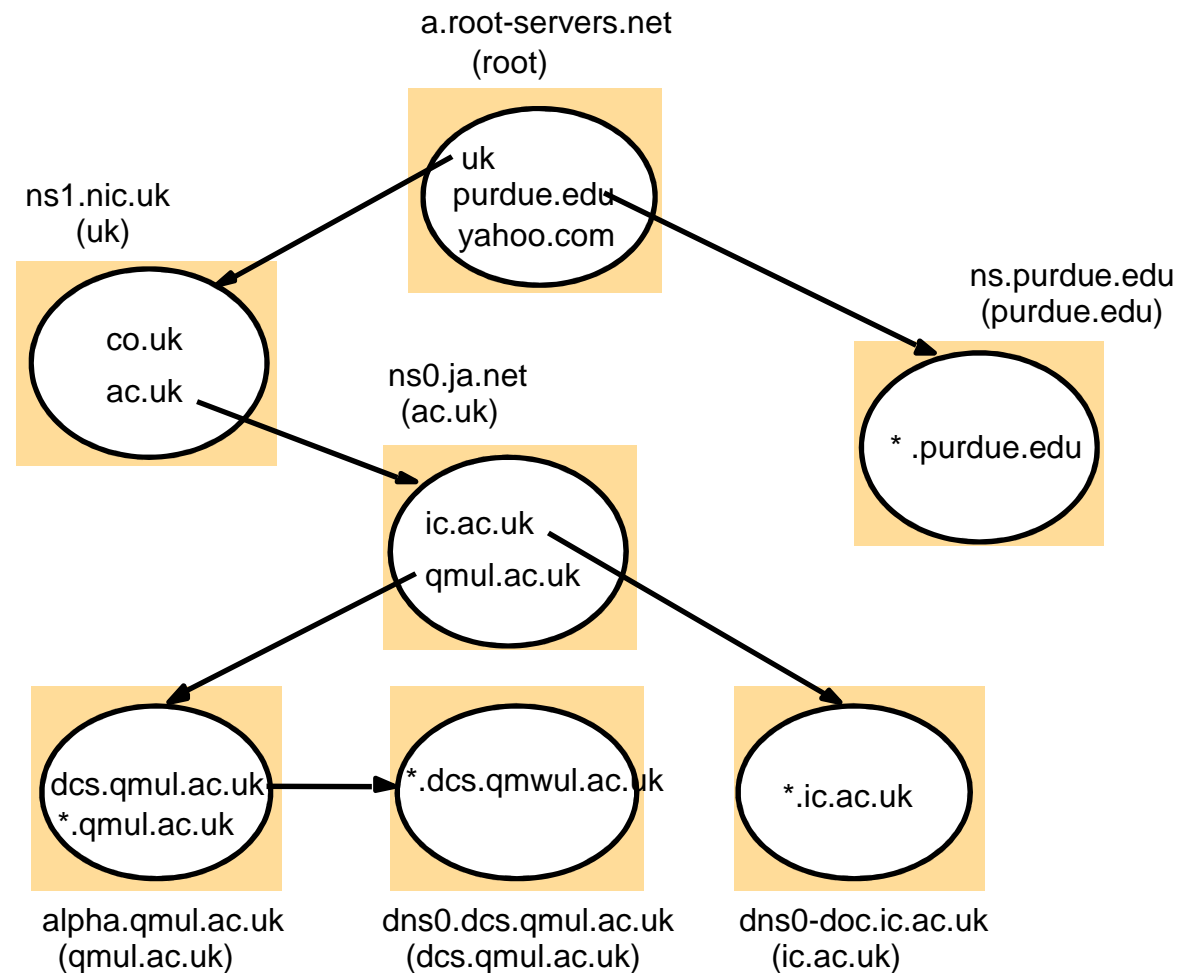
When it expires, contact the server

All DNS servers store the address of one or more root name servers

The Domain Name System (DNS)

Note: Name server names are in italics, and the corresponding domains are in parentheses.

Arrows denote name server entries



The Domain Name System (DNS)

- ▶ Properties of DNS
 - ▶ Hierarchical name space divided into zones
 - ▶ Distributed over a collection of DNS servers
- ▶ Hierarchy of DNS servers
 - ▶ Root servers
 - ▶ Top-level domain (TLD) servers
 - ▶ Authoritative DNS servers
- ▶ Performing the translations
 - ▶ Local DNS servers
 - ▶ Resolver software

The Domain Name System (DNS)

- ▶ **Top-level domain (TLD) servers**
 - ▶ Generic domains (e.g., com, org, edu)
 - ▶ Country domains (e.g., uk, fr, ca, jp)
 - ▶ Typically managed professionally
 - ▶ Network Solutions maintains servers for “com”
 - ▶ Educause maintains servers for “edu”
- ▶ **Authoritative DNS servers**
 - ▶ Provide public records for hosts at an organization
 - ▶ For the organization’s servers (e.g., Web and mail)
 - ▶ Can be maintained locally or by a service provider

The Domain Name System (DNS)

- ▶ 13 DNS root servers (see <http://www.root-servers.org/>)
- ▶ Labeled A through M



The Domain Name System (DNS)

- ▶ A distributed naming database
- ▶ Name structure reflects administrative structure of the Internet
- ▶ Rapidly resolves domain names to IP addresses
 - ▶ exploits caching heavily
 - ▶ typical query time ~100 milliseconds
- ▶ Scales to millions of computers
 - ▶ partitioned database
 - ▶ caching
- ▶ Resilient to failure of a server
 - ▶ replication

Using DNS

- ▶ **Local DNS server (“default name server”)**

- ▶ Usually near the end hosts who use it
- ▶ Local hosts configured with local server (e.g., `/etc/resolv.conf`) or learn the server via DHCP

- ▶ **Client application**

- ▶ Extract server name (e.g., from the URL)
- ▶ Do `gethostbyname()` to trigger resolver code

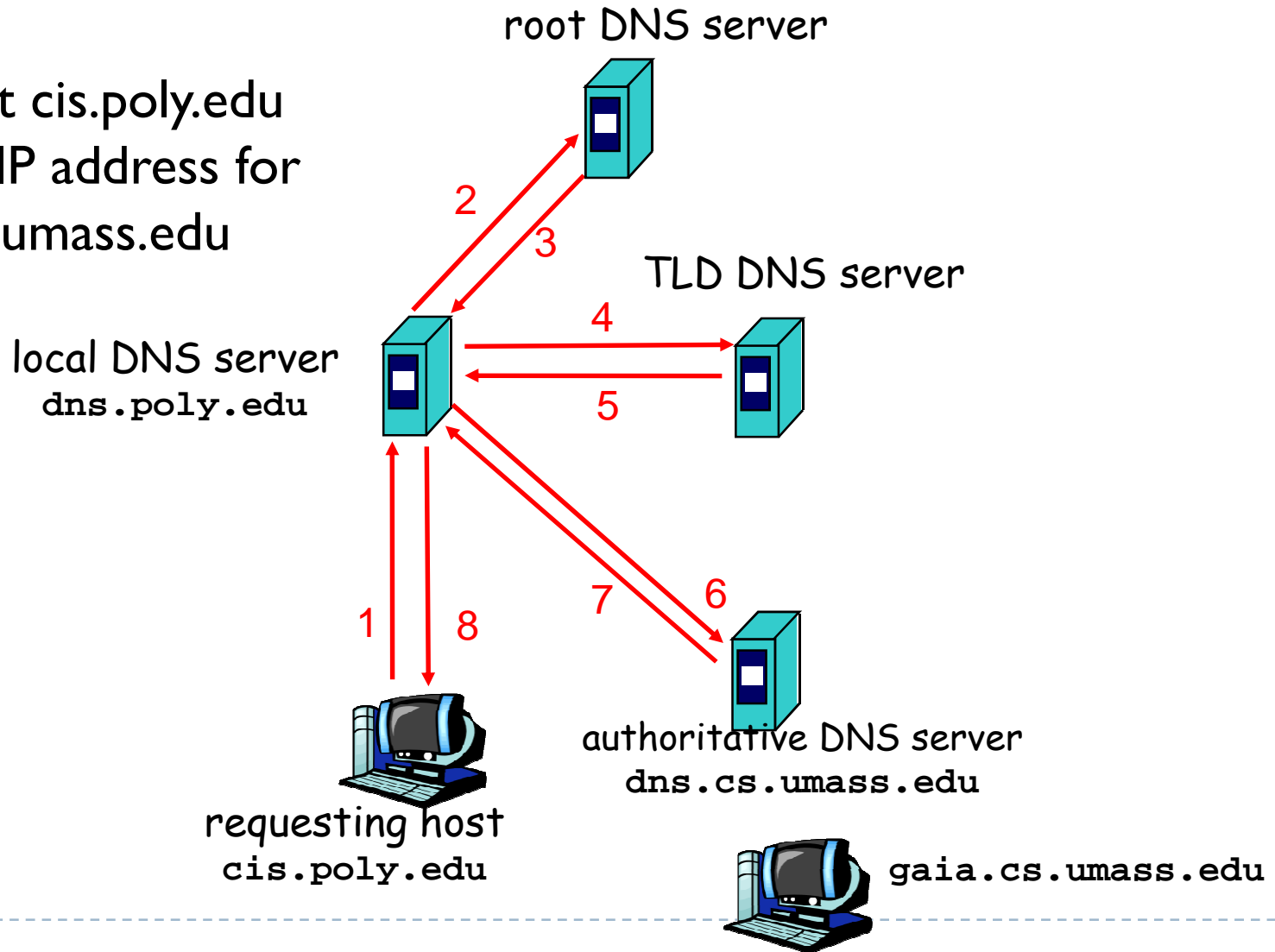
- ▶ **Server application**

- ▶ Extract client IP address from socket
- ▶ Optional `gethostbyaddr()` to translate into name



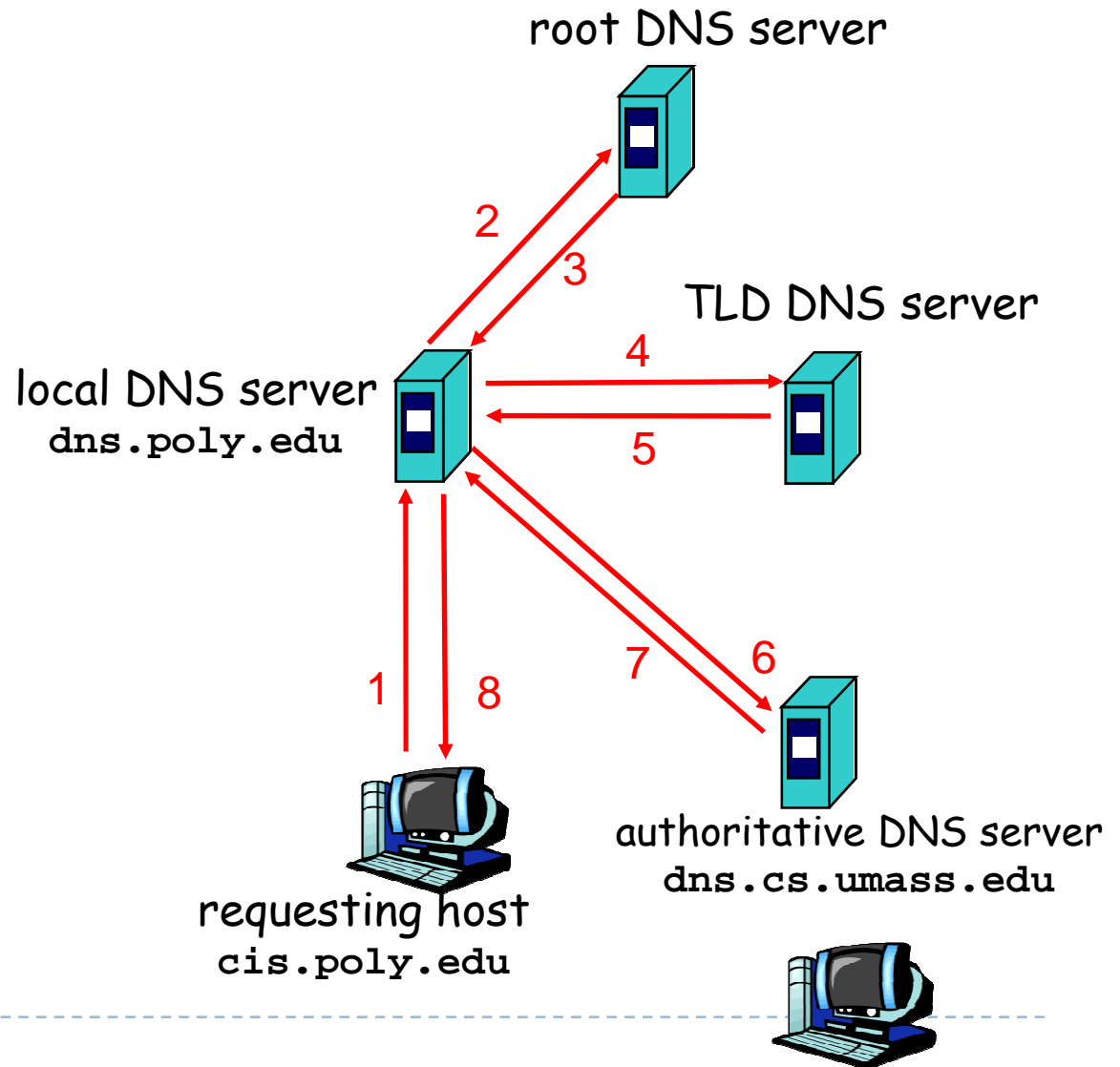
Example

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



Recursive vs. Iterative Queries

- ▶ **Recursive query**
 - ▶ Ask server to get answer for you
 - ▶ E.g., request 1 and response 8
- ▶ **Iterative query**
 - ▶ Ask server who to ask next
 - ▶ E.g., all other request-response pairs (2-7)



DNS Caching (more)

- ▶ **Performing all these queries takes time**
 - ▶ And all this before the actual communication takes place
 - ▶ E.g., 1-second latency before starting Web download
- ▶ **Caching can substantially reduce overhead**
 - ▶ The top-level servers very rarely change
 - ▶ Popular sites (e.g., www.cnn.com) visited often
 - ▶ Local DNS server often has the information cached
- ▶ **How DNS caching works**
 - ▶ DNS servers cache responses to queries
 - ▶ Responses include a “time to live” (TTL) field
 - ▶ Server deletes the cached entry after TTL expires

Negative Caching

- ▶ Remember things that don't work
 - ▶ Misspellings like `www.cnn.comm` and `www.cnnn.com`
 - ▶ These can take a long time to fail the first time
 - ▶ Good to remember that they don't work
 - ▶ ... so the failure takes less time the next time around

Reliability

- ▶ **DNS servers are replicated**
 - ▶ Name service available if at least one replica is up
 - ▶ Queries can be load balanced between replicas
- ▶ **UDP used for queries**
 - ▶ Need reliability: must implement this on top of UDP
- ▶ **Try alternate servers on timeout**
 - ▶ Exponential backoff when retrying same server
- ▶ **Same identifier for all queries**
 - ▶ Don't care which server responds



Inserting Resource Records into DNS

- ▶ **Example: just created startup “FooBar”**
- ▶ **Register foobar.com at Network Solutions**
 - ▶ Provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
 - ▶ Registrar inserts two RRs into the com TLD server:
 - ▶ (foobar.com, dns1.foobar.com, NS) – domain server
 - ▶ (dns1.foobar.com, 212.212.212.1, A) – IP address
- ▶ **Put in authoritative server dns1.foobar.com**
 - ▶ Type A record for www.foobar.com
 - ▶ Type MX record for foobar.com



DNS Query in Web Download

- ▶ **User types or clicks on a URL**
 - ▶ E.g., `http://www.cnn.com/2006/leadstory.html`
- ▶ **Browser extracts the site name**
 - ▶ E.g., `www.cnn.com`
- ▶ **Browser calls `gethostbyname()` to learn IP address**
 - ▶ Triggers resolver code to query the local DNS server
- ▶ **Eventually, the resolver gets a reply**
 - ▶ Resolver returns the IP address to the browser
- ▶ **Then, the browser contacts the Web server**
 - ▶ Creates and connects socket, and sends HTTP request



Multiple DNS Queries

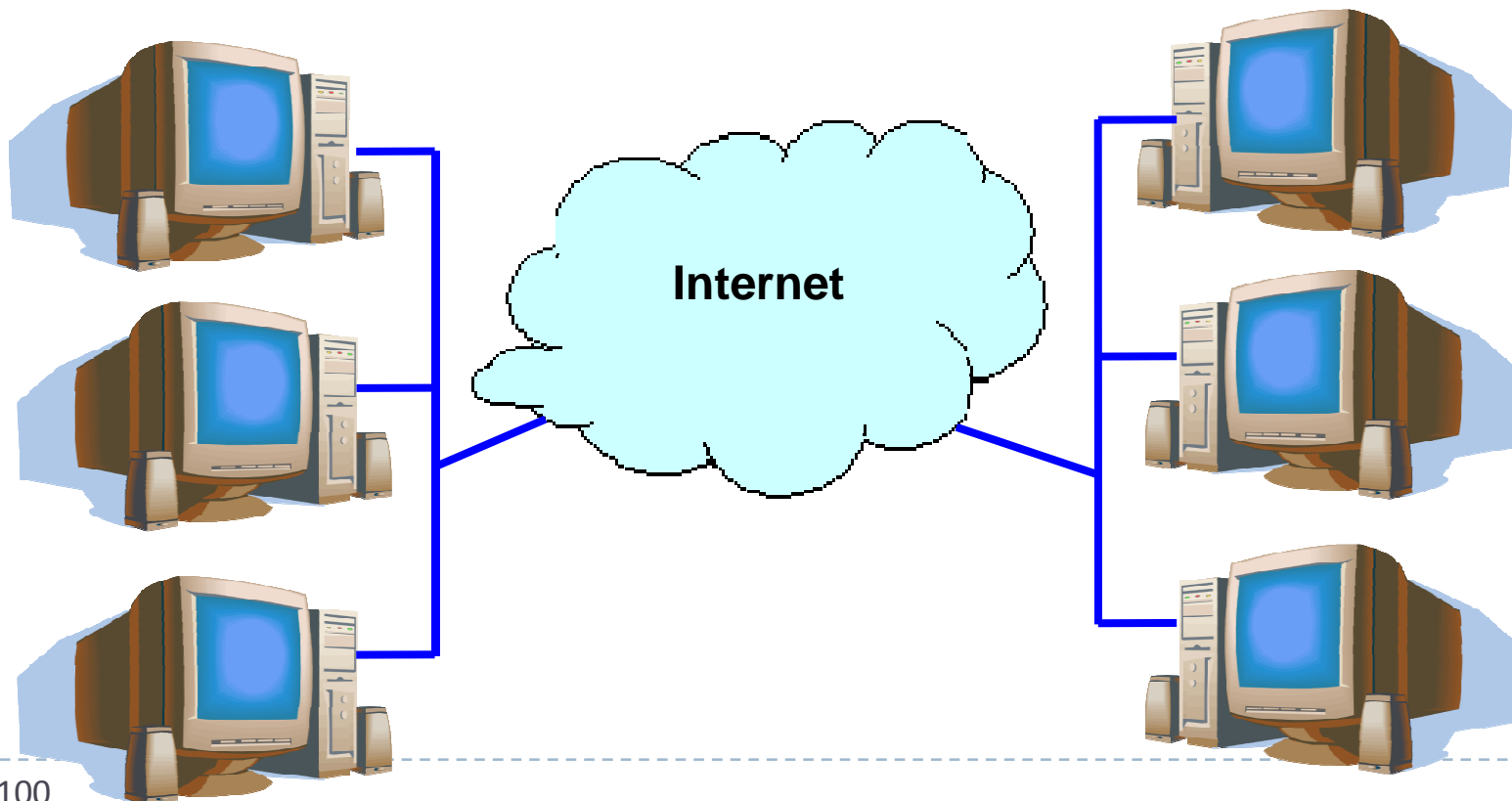
- ▶ **Often a Web page has embedded objects**
 - ▶ E.g., HTML file with embedded images
- ▶ **Each embedded object has its own URL**
 - ▶ and potentially lives on a different Web server
 - ▶ E.g., <http://www.myimages.com/image1.jpg>
- ▶ **Browser downloads embedded objects**
 - ▶ Usually done automatically, unless configured otherwise
 - ▶ Requires learning the address for www.myimages.com

When are DNS Queries Unnecessary?

- ▶ **Browser is configured to use a proxy**
 - ▶ E.g., browser sends all HTTP requests through a proxy
 - ▶ Then, the proxy takes care of issuing the DNS request
- ▶ **Requested Web resource is locally cached**
 - ▶ E.g., cache has `http://www.cnn.com/2006/leadstory.html`
 - ▶ No need to fetch the resource, so no need to query
- ▶ **Browser recently queried for this host name**
 - ▶ E.g., user recently visited `http://www.cnn.com/`
 - ▶ So, the browser already called `gethostbyname()` and may be locally caching the resulting IP address

Web Server Replicas

- ▶ Popular Web sites can be easily overloaded
 - ▶ Web site often runs on multiple server machines



Directing Web Clients to Replicas

- ▶ **Simple approach: different names**
 - ▶ www1.cnn.com, www2.cnn.com, www3.cnn.com
 - ▶ But, this requires users to select specific replicas
- ▶ **More elegant approach: different IP addresses**
 - ▶ Single name (e.g., www.cnn.com), multiple addresses
 - ▶ E.g., 64.236.16.20, 64.236.16.52, 64.236.16.84, ...
- ▶ **Authoritative DNS server returns many addresses**
 - ▶ And the local DNS server selects one address
 - ▶ Authoritative server may vary the order of addresses

Clever Load Balancing Schemes

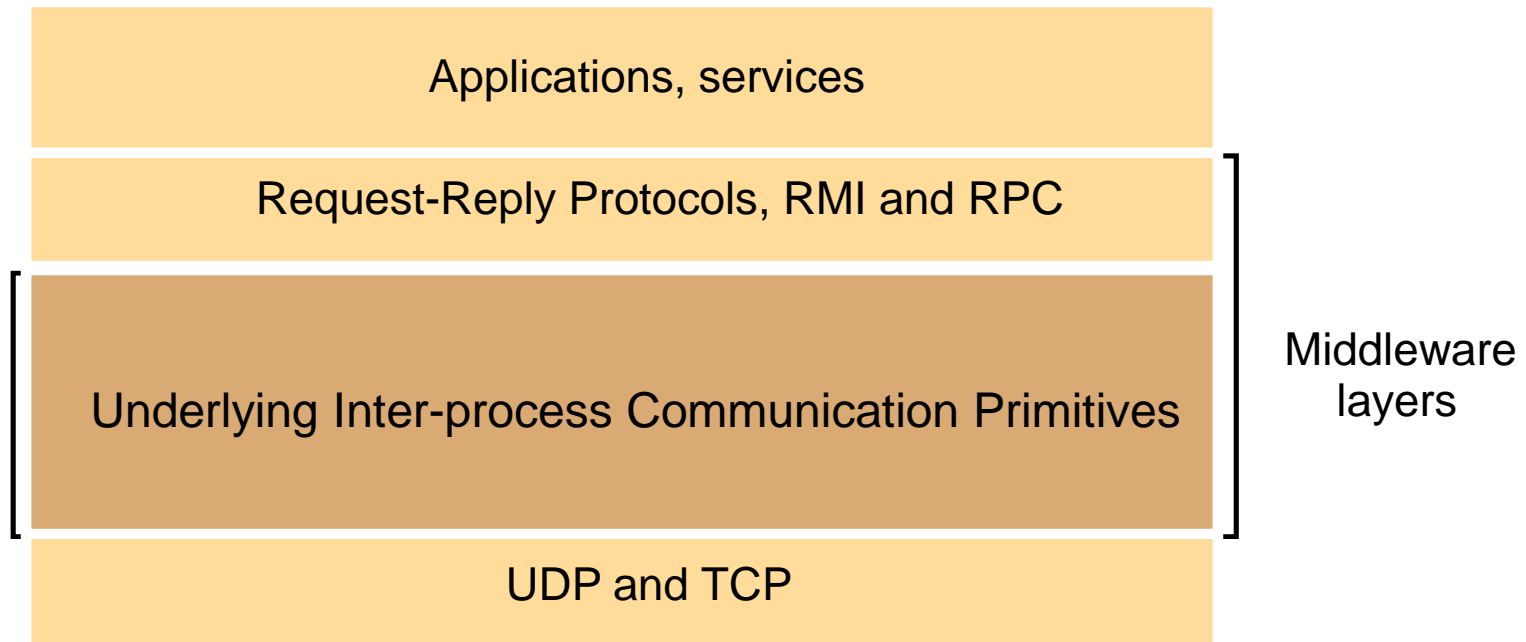
- ▶ **Selecting the “best” IP address to return**
 - ▶ Based on server performance
 - ▶ Based on geographic proximity
 - ▶ Based on network load
 - ▶ ...
- ▶ **Example policies**
 - ▶ Round-robin scheduling to balance server load
 - ▶ U.S. queries get one address, Europe another
 - ▶ Tracking the current load on each of the replicas

Challenge: What About DNS Caching?

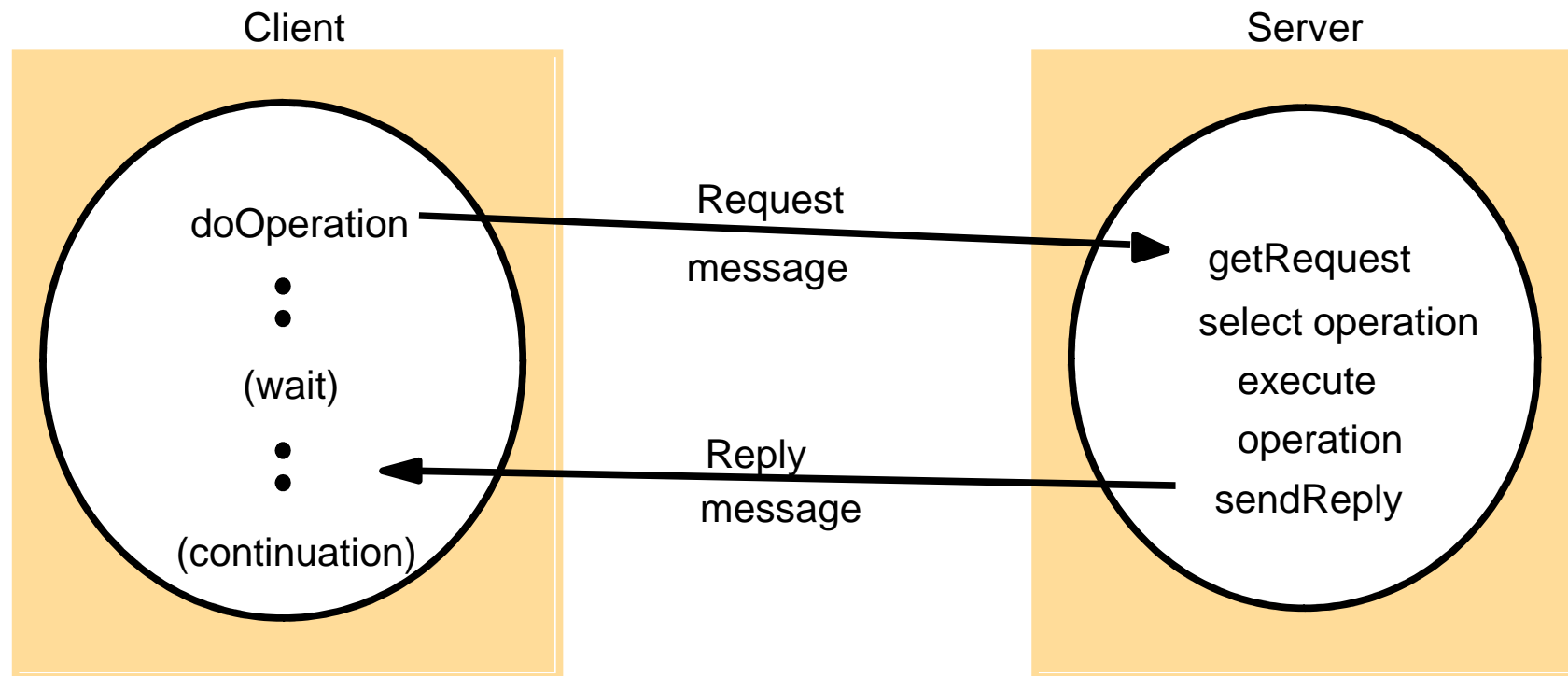
- ▶ **Problem: DNS caching**
 - ▶ What if performance properties change?
 - ▶ Web clients still learning old “best” Web server until the cached information expires
- ▶ **Solution: Small Time-to-Live values**
 - ▶ Setting artificially small TTL values so replicas picked based on fresh information
- ▶ **Disadvantages: abuse of DNS?**
 - ▶ Many more DNS request/response messages
 - ▶ Longer latency in initiating the Web requests

Request-Reply Protocols (HTTP)

Middleware layers



Client-server communication



Synchronous: client waits for a reply

Asynchronous: client doesnot wait for a reply



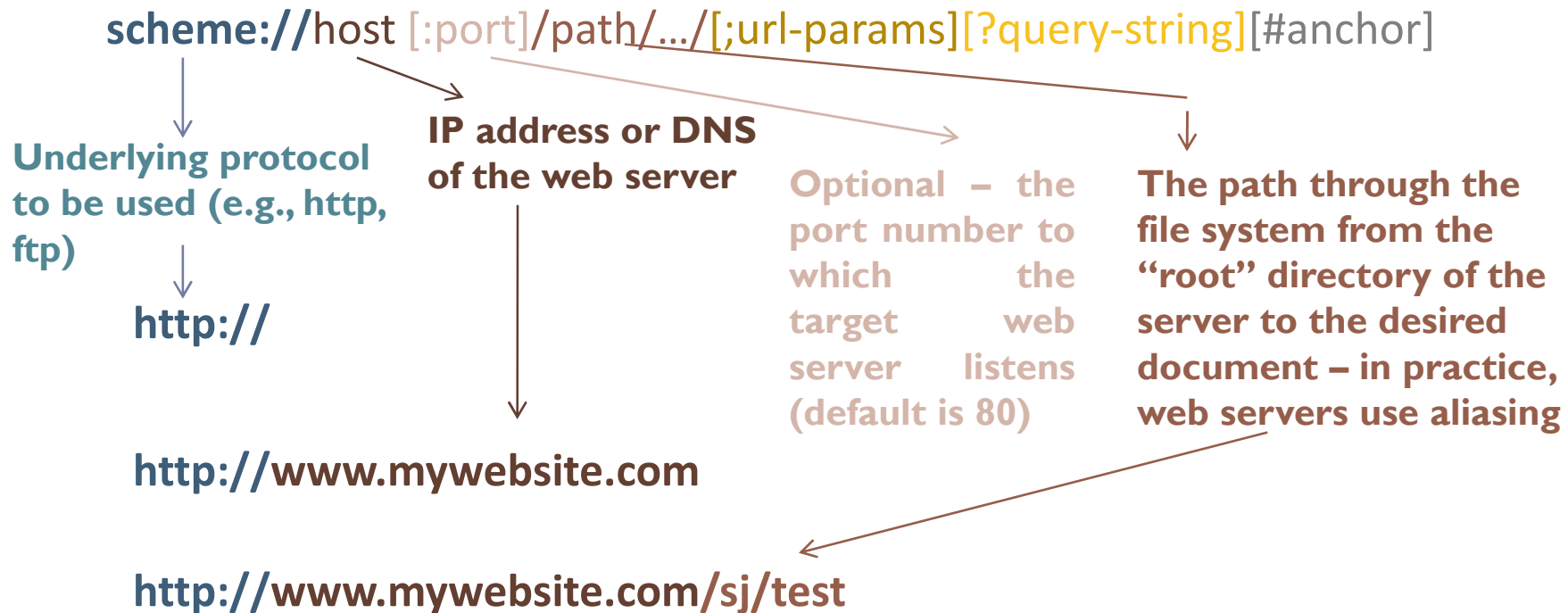
Client-server communication

- ▶ Failure model
 - ▶ UDP: could be out of order, lost...
 - ▶ process can fail...
- ▶ not getting a reply
 - ▶ timeout and retry
- ▶ duplicate request messages on the server
 - ▶ How does the server find out?
- ▶ *idempotent* operation: can be performed repeatedly with the same effect as performing once.
 - ▶ idempotent examples?
 - ▶ non-idempotent examples?
- ▶ history of replies
 - ▶ retransmission without re-execution
 - ▶ how far back if we assume the client only makes one request at a time?



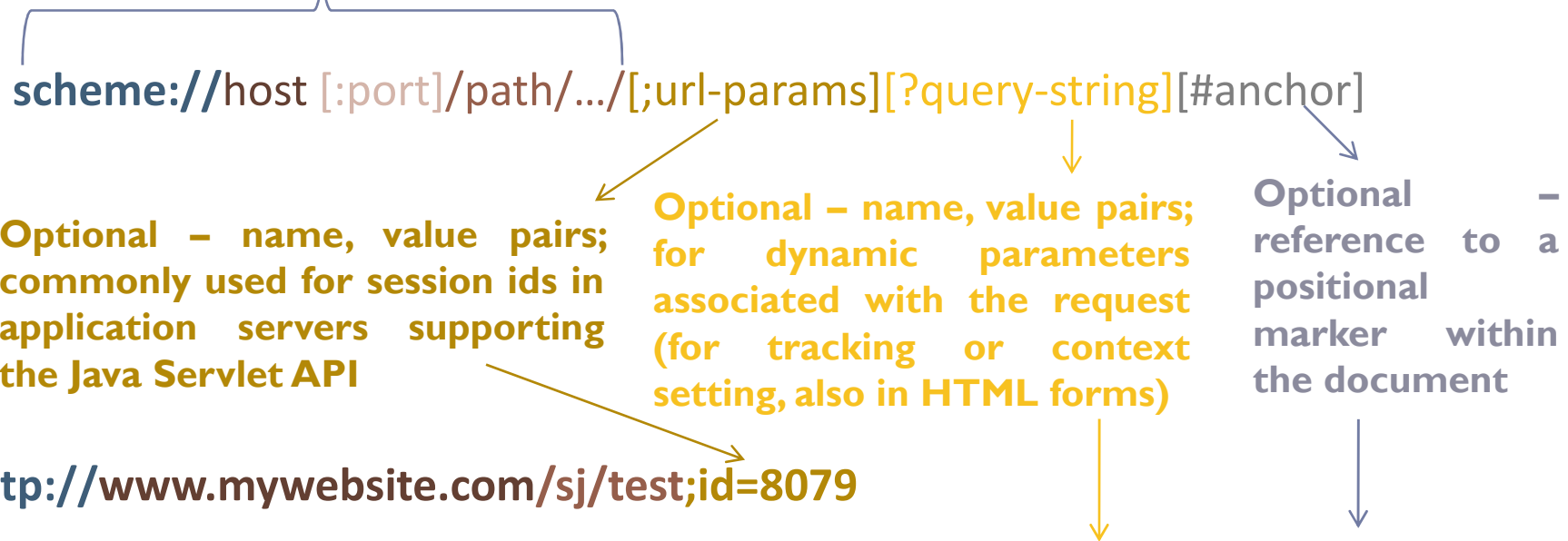
<http://www.webappbuilders.com/>

Uniform Resource Locator (URL)



Uniform Resource Locator (URL)

http://www.mywebsite.com/sj/test



http://www.mywebsite.com/sj/test;id=8079

http://www.mywebsite.com/sj/test;id=8079 ?name=bob&x=true#label

application-protocol://IP-address[:port]path-from-the-root[/;par][?dyn-par][#anchor]

HTTP

- Application-level protocol in the TCP/IP protocol suite
- Uses TCP
- Client-Server model
- Follows the request-response communication paradigm
- Stateless (more later)
 - vs stateful: sequences of related commands are treated as a a single interaction, often called a *session*
 - session are within a persistent connection
- Through Proxies
 - Firewalls
 - Support for caching
 - Filtering
- Connection defined as a virtual circuit (browser, server, proxies)

HTTP message

[message header]

[message body]

Simple example request

Method /path-to-resource HTTP/version-number

Header-Name-1: value

Header-Name-2: value

[optional request body]

GET /sj/index.html HTTP/1.1

Host: www.mywebsite.com

HTTP message

[message header]

[message body]

Simple example reply

HTTP/version-number status-code explanation

Header-Name-1: value

Header-Name-2: value

[response body]

GET /sj/index.html HTTP/1.1

Host: www.mywebsite.com

Request Methods

**GET HEAD POST
PUT DELETE TRACE OPTIONS CONNECT**

```
Method /path-to-resource HTTP/version-number
Header-Name-1: value
Header-Name-2: value

[optional request body]
```

GET

Most common (type a URL, click on a link, etc), if the UR: refers to data, the web server replies by returning the data, of refers to a program, then the web server runs the program and returns its output

POST

POST has a body where the URL parameters are placed, GET appends them to the path – usually used when the action may change data on the server

Web application dependent: e.g., display a form when GET request and process it when POST

Request Methods

```
Method /path-to-resource HTTP/version-number  
Header-Name-1: value  
Header-Name-2: value  
  
[optional request body]
```

```
GET /q?s=YHOO HTTP/1.1  
Host: filename.yahoo.com  
User-Agent: Mozilla/5.0 (Windows; U; Windows XP; en-US; rv:1.8.0.1)
```

```
POST /q HTTP/1.1  
Host: filename.yahoo.com  
User-Agent: Mozilla/5.0 [en] (WinNT; U)  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 6
```

```
s = YHOO
```

Request Methods

HEAD

Requests that use the HEAD method are processed similarly to request that use the GET method but the server sends back *only headers* (not the body) in the response

Used to support caching

Still useful for implementing change-tracking systems, testing and debugging new applications and discovering server capabilities

Status Codes

1 Informational

100 (notify clients that they can continue) Expect: 100-continue header

2 Successful responses

200

3 *Tell the client to perform additional actions (redirection)*

4 Client requests errors or special conditions

400 Bad Request 401 Not Authorized 403 Forbidden 404 Not found

5 Server errors

500 Internal Server Error 501 Not Implemented

Status Codes: Redirection (3xx)

- ❖ Redirection: the browser is instructed to resubmit the request to another URL
 - 301 moved permanently
 - 302 temporarily
- ❖ Browsers respond “silently” to redirection status codes
- ❖ (not supported or disabled) Web servers include a message body that explicitly references a link to the new location -> follow the link manually
- ❖ Web servers treat *a URL ending in a slash* as a request for a directory (depending on the server configuration return either a file with a default name (e.g., index.html or the contents of the directory)
- ❖ If the user forgets the trailing “/”, the server a redirection response
- ❖ Proxies react to 301 status by updating internal relocation tables (*cache 301 redirections*) e.g., redirecting users to the login page when trying to access a protected URL

Headers

General Headers

Apply to both request and response messages
Do not describe the body of the message

Example:

Date (time and date of the message creation),
Connection (keep-alive default setting for HTTP/1.1)

Request Headers

Allows clients to pass additional information

Example

User-Agent (type of software)
Host (may look redundant, but it is there for HTTP1.0 and virtual hosting)
Referee (context information about the request, e.g., if because of a click on a link in a page, the header is set to the URL of that page)
Authorization Browsers include this header in all follow-up requests [after being notified of an authorization challenge (401) and prompting the user for credentials, once credential accepted included (expiration is browser-specific)]

Headers

Response Headers

Help the server to pass additional information about the response that cannot be inferred from the status code

Examples

Location for redirecting (used with 301, 302)

WWW-Authenticate (used with 401) Basic realm = “KremlinFiles”, if browser, users are prompted for credentials

Realm: which resources require what type of authorization – web masters can administrate web servers to define realms, associate them with files and directories and establish userid and passwords that limit access to these resources

Server server software

Entity Headers

Either message bodies or (in the case of no body) target resources

Examples

Content-Type the MIME type of the message body

Content-Length to help the browser in rendering

Last-modified critical for caching

Support for content types

HTTP borrows its content typing system from Multipurpose Internet Mail Extensions (MIME)

A two layer ordered encoding model
Content-Encoding (gzip, compress, deflate)

Content type

`type */* subtype [“;” parameter-string]`

Examples

Content type: text/html

Content type: text/plain; charset='us-ascii'

Content type: application/pdf

Content type: video/quicktime

Browsers use types and sub-types either to select a proper content-rendering module or to invoke a third-party tool

Server-side applications use type information to process requests

Support for content types

Multipart message

A MIME multipart message contains a **boundary** in the "Content-Type: " header; this boundary, is placed between the parts, and at the beginning and end of the body of the message

```
Content-Type: multipart/mixed; boundary="frontier"
```

```
This is a message with multiple parts in MIME format.
```

```
--frontier
```

```
Content-Type: text/plain
```

```
This is the body of the message.
```

```
--frontier
```

```
Content-Type: application/octet-stream
```

```
Content-Transfer-Encoding: base64
```

```
PGh0bWw+CiAgPGhIYWQ+CiAgPC9oZWFKPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB0aGUg
```

```
Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg=
```

```
--frontier--
```

- Each part consists of its own content header (zero or more *Content-* header fields) and a body.
- The sending client must choose a boundary string that doesn't clash with the body text. Typically this is done by inserting a long random string.

Support for content types

Multipart message

The content type multipart/x-mixed-replace developed as part of a technology to emulate server push and streaming over HTTP.

All parts of a mixed-replace message have the same semantic meaning. However, each part invalidates - "replaces" - the previous parts as soon as it is received completely.

Clients should process the individual parts as soon as they arrive and should not wait for the whole message to finish.

```
Content-Type: multipart/x-mixed-replace; boundary=ThisRandomString
```

```
Connection: close
```

```
--ThisRandomString
```

```
Content-Type: image/gif
```

```
...
```

```
--ThisRandomString
```

```
Content-Type: image/gif
```

```
...
```

Caching

- Browser-side
- Proxy-side
- *Server-side*

HTTP1.1 provides a mechanism for enforcing caching rules based on the Cache-Content header

- **public** setting authorizes both shared and user-localized caching
- **private** setting indicates that the response is directed to a single user and should not be stored in a shared cache (e.g., a secure request about their private accounts)
- **no-cache** setting indicates that neither browser nor proxies are allowed to cache, but there are options

HTTP1.0 browsers and proxies are not guarantee to obey such instructions

Header with GET

If-Modified-Since: (If-Unmodified-Since)

304 Not Modified or the body

Security

- authentication vs
- authorization

Built-in support for basic authentication, where user credentials (userid and password) are transmitted via the Authorization header as a single encoded (not encrypted) string
Safe only if performed over a secure connection (e.g., https)

Many web applications implement their own authentication and authorization schemes

401

After attempts 403

Session support

Server applications can use the Set-Cookie header

```
Set-Cookie: <name>=<value>  
[; Comment=<value>] [; Max-Age=<value>]  
[; Expires=<date>] [; Path =<path>]  
[; Domain=<domain name>] [; Secure]  
[; Version=<version>]
```

An attribute-value pair `<name> = <value>` is sent back by the browser in qualifying subsequent requests

Max-Age the lifetime of the cookie in secs (Expires)

The Path and Domain attributes delimit which request qualify, by specifying the server domains and URL paths to which this cookie applies

Domains: suffixes of the originating server's host name

Path attribute default to the path of the URL associated with the server application

For subsequent requests directed at URLs where the domain and path match, the browser must include a Cookie header with the appropriate attribute-value pair

Secure tells the browser to submit corresponding Cookie headers only over secure connections -- Version

Session support

```
Set-Cookie2: <name>=<value>  
...  
[; Expires=<date>] [; Path =<path>]  
[; Domain=<domain name>] [; Port=<portlist>]  
...
```

Persistent connection

HTTP/1.1 connections are persistent, except when explicitly closed by a participating program via the Connection: close header

Pipelining: browsers can queue requests messages without waiting for resources

Servers are responsible for submitting responses to browsers in the order of their arrival

HTTP

- ▶ using TCP increase reliability and also cost
- ▶ HTTP uses TCP
 - ▶ one connection per request-reply
 - ▶ HTTP 1.1 uses "persistent connection"
 - ▶ multiple request-reply
 - ▶ closed by the server or client at any time
 - ▶ closed by the server after timeout on idle time
 - ▶ Marshal messages into ASCII text strings
 - ▶ resources are tagged with MIME (Multipurpose Internet Mail Extensions) types: test/plain, image/gif...
 - ▶ content-encoding specifies compression alg



HTTP methods

- ▶ GET: return the file, results of a cgi program, ...
- ▶ HEAD: same as GET, but no data returned, modification time, size are returned
- ▶ POST: transmit data from client to the program at url
- ▶ PUT: store (replace) data at url
- ▶ DELETE: delete resource at url
- ▶ OPTIONS: server provides a list of valid methods
- ▶ TRACE: server sends back the request



Questions?