COMPUTER NETWORKS

Compiled by **Kundhavai K R**

Department of Computer Science and Engineering

DOMAIN NAME SYSTEM

Unit – 2 Application Layer

2.1 Principles of Network Applications
2.2 Web, HTTP and HTTPS
2.3 The Domain Name System
2.4 P2P Applications
2.5 Socket Programming with TCP & UDP
2.6 Other Application Layer Protocols

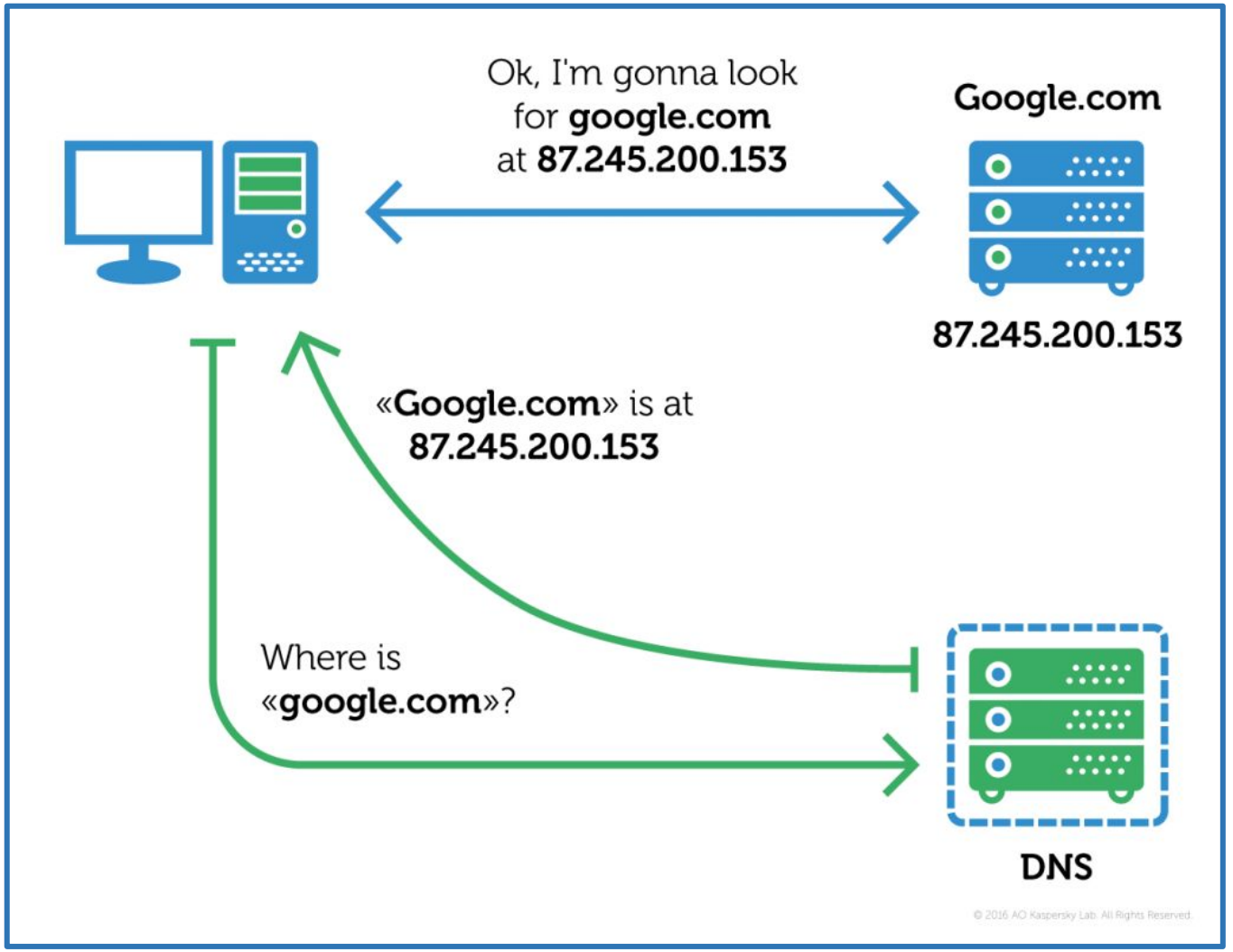
DNS: Domain Name System

DNS—The Internet's Directory Service

- Human beings can be identified - Names that appear on our birth certificates, Aadhar Card Number, driver's license numbers etc.,
- Hosts / End systems are identified by **IP addresses.**
- IP address consists of **four bytes** and has a rigid hierarchical structure.
  - An IP address looks like **121.7.106.83**, where each period separates one of the bytes expressed in decimal notation from **0 to 255**.
  - An IP address is **hierarchical** because as we scan the address from left to right.
- People prefer the more mnemonic **hostname** identifier, while the routers **prefer fixed–length, hierarchically structured IP addresses.**
- In order to reconcile these different preferences, a directory service is required that **translates hostnames to IP addresses**.
- This is the main task of the Internet's **Domain Name System (DNS).**

COMPUTER NETWORKS

DNS: Domain Name System



COMPUTER NETWORKS

DNS: Domain Name System

- DNS is
  1. a distributed database implemented in **a hierarchy of name servers** &
  2. an application–layer protocol that allows hosts & name servers to communicate in order to provide the **translation service**.
- The DNS protocol runs over **UDP & uses port 53.**
- DNS is commonly employed by other application–layer protocols – including HTTP, SMTP, & FTP – to translate user–supplied host names to IP addresses.
- The DNS is specified in **RFC 1034 & RFC 1035**, & updated in several additional RFCs.

COMPUTER NETWORKS

DNS: Domain Name System

- As an example, user's host, requests the URL **www.pesuacademy.edu/index.html.**
- Steps are as follows:

1. The user machine runs the client side of the DNS application
2. The browser extracts the hostname from the URL & passes the hostname to the client side of the DNS application
3. The DNS client sends a query containing the hostname to a DNS server
4. The DNS client eventually receives a reply, which includes the IP address for the hostname
5. Once the browser receives the IP address from DNS, it can initiate a TCP connection to the HTTP server process located at port 80 at that IP address

COMPUTER NETWORKS

DNS: Domain Name System

relay1.west-coast .enterprise.com -> Canonical Host Name
www.enterprise.com , enterprise.com -> Alias Name

- DNS provides a few other important services in addition to translating hostnames to IP addresses:

1. **Host aliasing**: a host with a complicated hostname can have one or more alias names.
   - Ex: A hostname such as **relay1.west-coast .enterprise.com** could have, two aliases such as **enterprise.com** and **www.enterprise.com**.
   - In this case, the hostname relay1 .west-coast.enterprise.com is said to be a **canonical hostname.**

relay1.west-coast.yahoo.com-> Canonical Host Name
bob@yahoo.com -> Alias Name

2. **Mail server aliasing**:
   - For example, if Bob has an account with Yahoo Mail, Bob's e-mail address might be as simple as **bob@yahoo.com.**
   - However, the hostname of the Yahoo mail server is more complicated and much less than simply yahoo.com (for example, the canonical hostname might be something like **relay1.west-coast.yahoo.com**).

DNS: Domain Name System

3. Load distribution: DNS is also used to perform load distribution among replicated servers, such as replicated Web servers.

- Busy sites, such as **cnn.com**, are replicated over **multiple servers**, with each server running on a **different end system** and each having a different IP address.
- For replicated Web servers, a **set of IP addresses** is thus associated with one canonical hostname.
- The **DNS database contains this set of IP addresses.**
- When clients make a DNS query for a name mapped to a set of addresses, the server responds with **the entire set of IP addresses.**
- **DNS rotation distributes the traffic** among the replicated servers

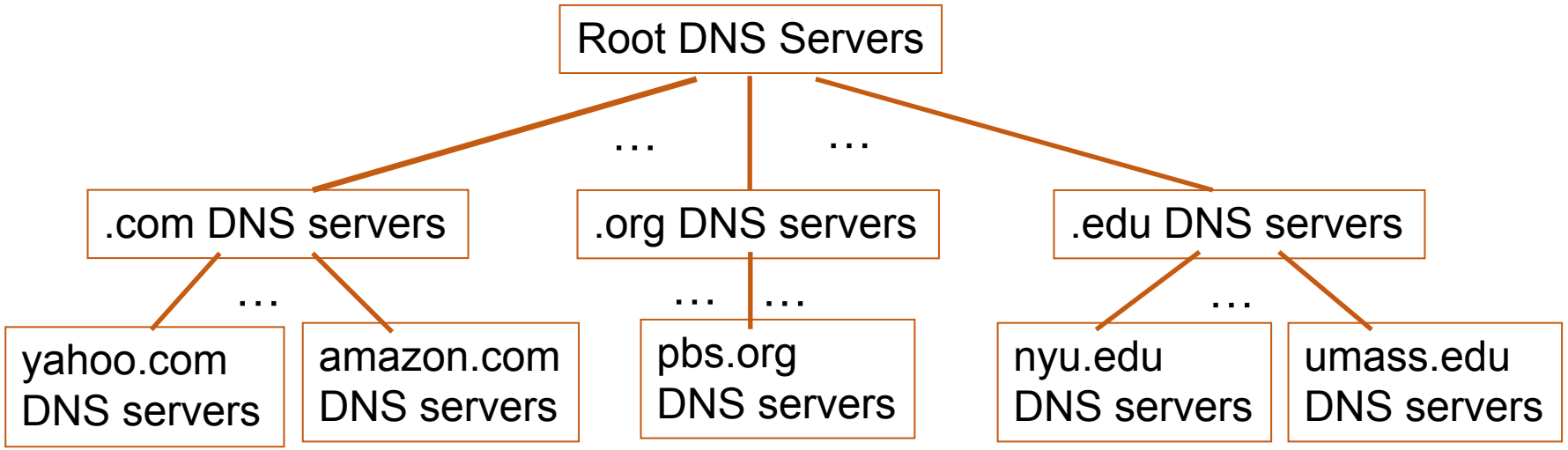
DNS: Domain Name System

Why not centralize DNS?

The problems with a centralized design include:

- **A single point of failure.** If the DNS server crashes, so does the entire Internet!
- **Traffic volume.** A single DNS server would have to handle all DNS queries (for all the HTTP requests and e-mail messages generated from hundreds of millions of hosts).
- **Distant centralized database.** A single DNS server cannot be "close to" all the querying clients. If we put the single DNS server in New York City, then all queries from Australia must travel to the other side of the globe, perhaps over slow and congested links. This can lead to significant delays.
- **Maintenance.** The single DNS server would have to keep records for all Internet hosts. Not only would this centralized database be huge, but it would have to be updated frequently to account for every new host.

DNS: a distributed, hierarchical database



Root

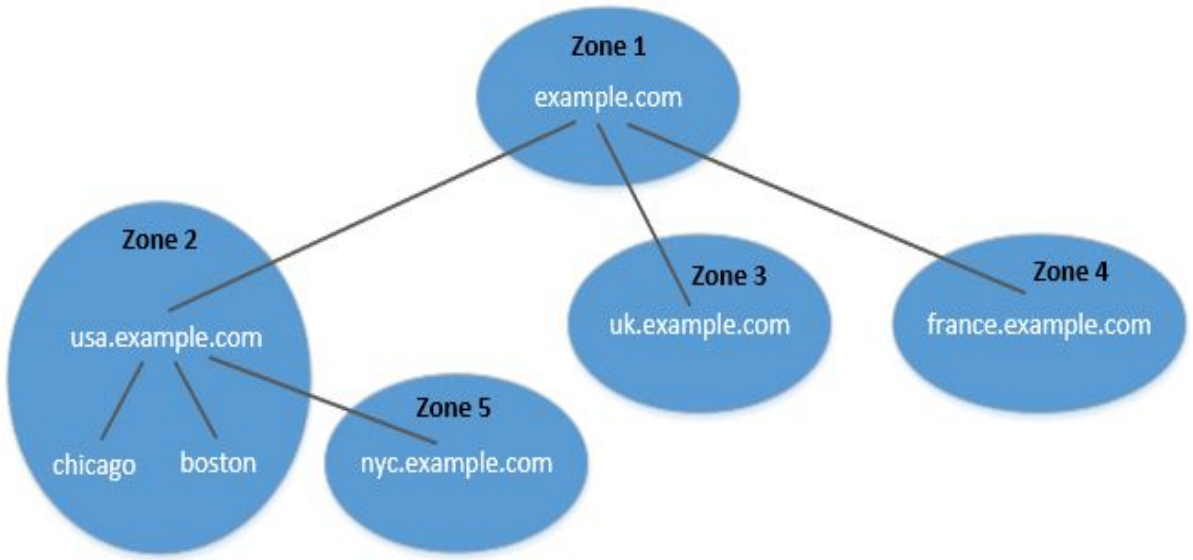
Top Level Domain

Authoritative

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

- client queries 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

DNS Zone vs Domain



- DNS is organized according to zones.
- A zone groups contiguous domains and subdomains on the domain tree.
- Assign management authority to an entity.

- The tree structure depicts subdomains within example.com domain.
- Multiple DNS zones one for each country. The zone keeps records of who the authority is for each of its subdomains.
- The zone for example.com contains only the DNS records for the hostnames that do not belong to any subdomain like mail.example.com

COMPUTER NETWORKS

DNS: root name servers

- There are more than 1000 root servers instances scattered all over the world.
- These root servers are copies of 13 different root servers, managed by 12 different organizations
- coordinated through the Internet Assigned Numbers Authority [IANA 2020].
- DNSSEC – provides security (authentication and message integrity)

- Root name servers provide the IP addresses of the TLD servers.

TLD: authoritative servers

Top–Level Domain (TLD) servers:

- responsible for **.com, .org, .net, .edu, .aero, .jobs, .museums**
- top–level country domains, e.g.: **.in**, **.us, .uk, .fr, .ca, .jp**
  - Verisign Global Registry Services : .com TLD
  - Educause: .edu TLD
- TLD servers provide the IP addresses for authoritative DNS servers.

Authoritative DNS servers:

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

Local DNS Name Servers/DNS Resolver

- 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
  - has local cache of recent name–to–address translation pairs (but may be out of date!)
  - acts as proxy, forwards query into hierarchy

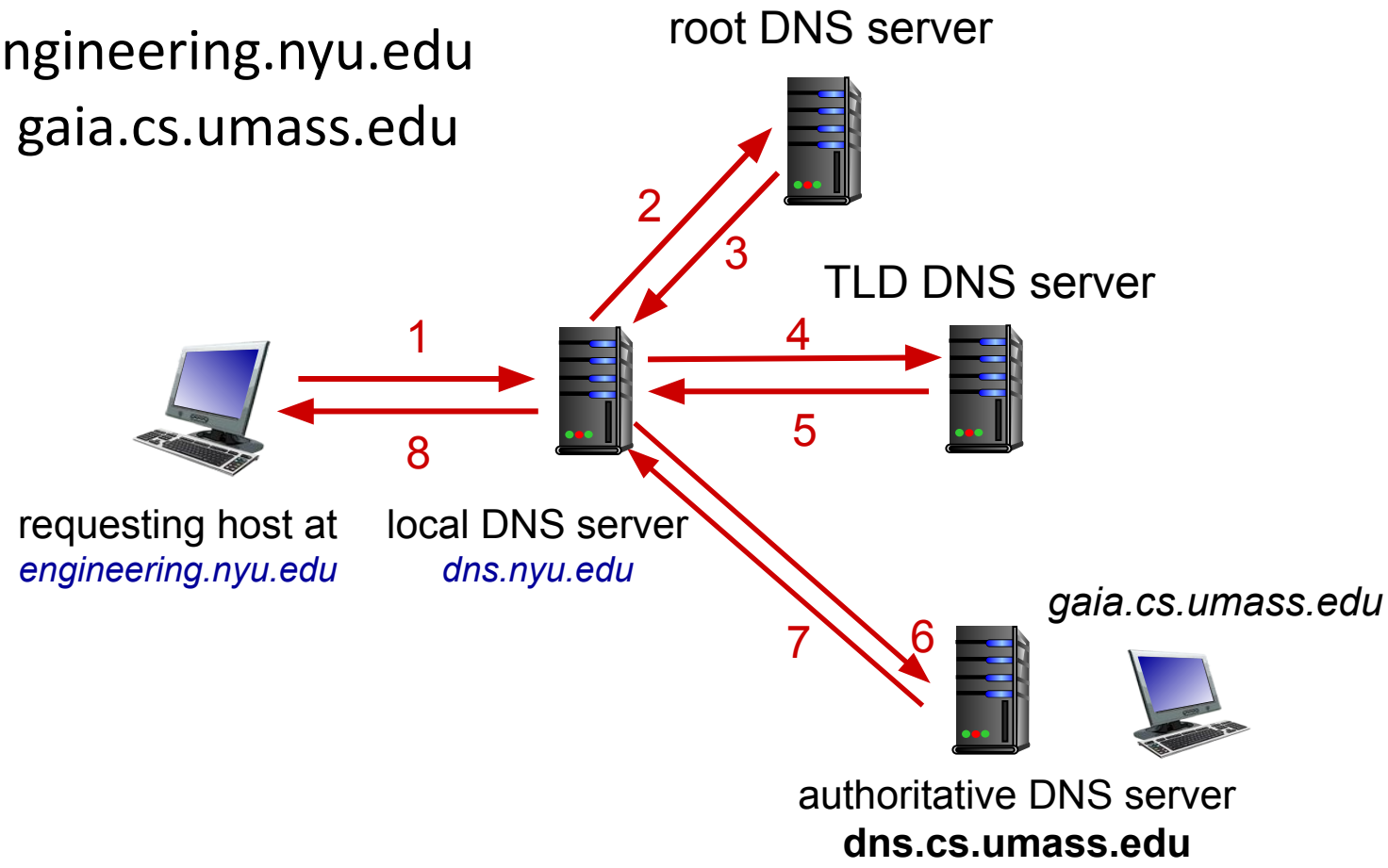
DNS: Domain Name System

DNS resolution can be solved in two ways:

1. **Iterated query**
2. **Recursive query**

DNS name resolution: iterated query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu



Iterated query:

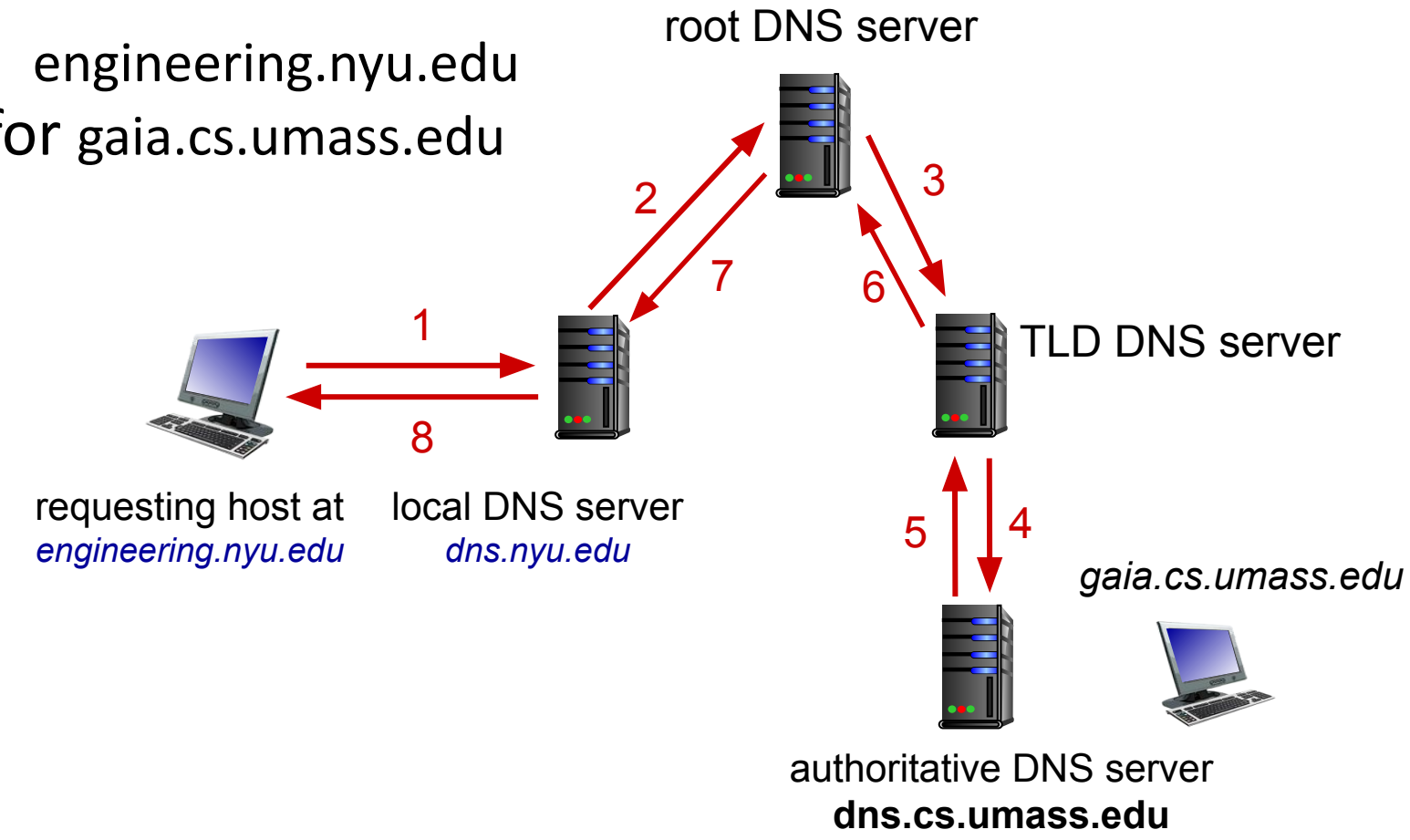
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

DNS name resolution: recursive query

Example: host at engineering.nyu.edu wants IP address for gaia.cs.umass.edu

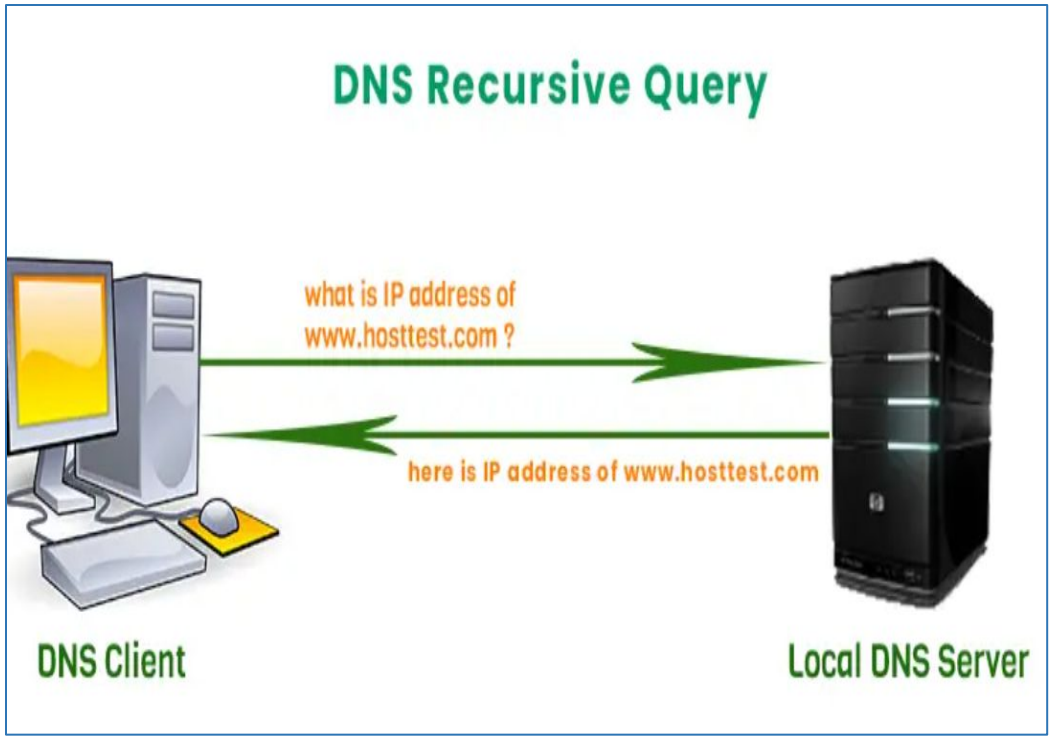
Recursive query:

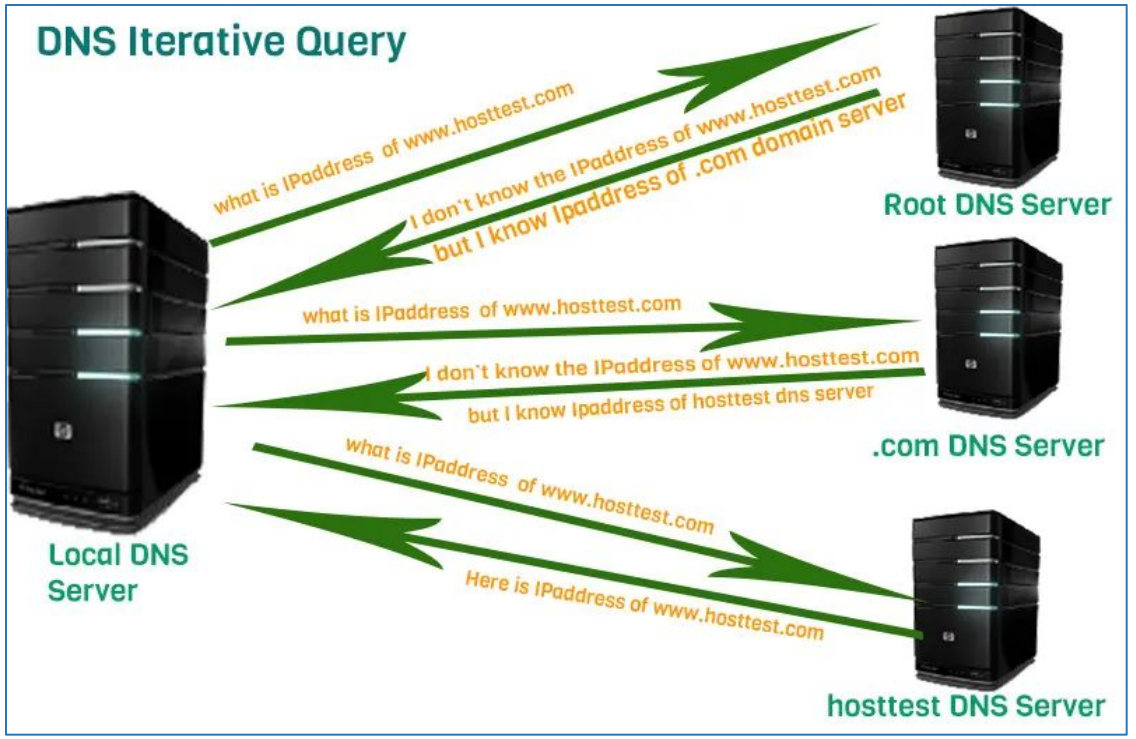
- puts burden of name resolution on contacted name server
- heavy load at upper levels of hierarchy?



COMPUTER NETWORKS

DNS: Domain Name System





Caching and Updating DNS Records

- Suppose that a host **apricot.nyu.edu** queries **dns.nyu.edu** for the IP address for the hostname **cnn.com**. After an hour later, another NYU host, say, **kiwi.nyu.edu**, also queries **dns.nyu.edu.**
- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
    - thus root name servers not often visited
- cached entries may be *out-of-date* (best-effort name-to-address translation!)
  - if name host changes IP address, may not be known Internet-wide until all TTLs expire!
- update/notify mechanisms proposed IETF standard
  - RFC 2136

DNS – Records and Messages

DNS Records

DNS: distributed database storing resource records **(RR)**

- The DNS servers that together implement the DNS distributed database store **resource records** (**RRs**), including RRs that provide hostname–to–IP address mappings.
- Each DNS reply message carries one or more resource records.

DNS Records

- A resource record is a four–tuple, that contains the following fields:

Name, Value, Type, TTL

- **TTL** is the time to live of the resource record; it determines when a resource should be removed from a cache.
- **Type A** RR:
  - ✔ If Type=A, then **Name is a hostname and Value is the IP address** for the hostname.
  - ✔ Thus, a Type A record provides the standard hostname–to–IP address mapping.
  - ✔ **As an example, (relay1.bar.foo.com, 145.37.93.126, A) is a Type A record.**

DNS Records

- **Type NS** RR:

✔ If Type=NS, then **Name is a domain (such as foo.com) and Value is the hostname of an authoritative DNS server** that knows how to obtain the IP addresses for hosts in the domain.

✔ This record is used to route DNS queries further along in the query chain.

✔ As an example, (foo.com, dns.foo.com, NS) is a Type NS record.

DNS Records

- **Type CNAME** RR:

✔ If Type=CNAME, then **Value is a canonical hostname for the alias hostname Name.**

✔ This record can provide querying hosts the canonical name for a hostname.

✔ **As an example, (foo.com, relay1.bar.foo.com, CNAME) is a CNAME record.**

DNS Records

- **Type MX** RR:

✔ If Type=MX, then Value is the canonical name of a mail server that has an alias hostname Name.

✔ **As an example, (foo.com, mail.bar.foo.com, MX) is an MX record.**

✔ MX records allow the hostnames of mail servers to have simple aliases.

✔ Note that by using the MX record, a company can have the same aliased name for its mail server and for one of its other servers (such as its Web server).

✔ To obtain the canonical name for the mail server, a DNS client would query for an MX record; to obtain the canonical name for the other server, the DNS client would query for the CNAME record.

DNS Records

DNS: distributed database storing resource records (RR)

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

type=A

- `name` is hostname
- `value` is IP address

relayl.bar.foo.com, 145.37.93.126, A

type=CNAME

- `name` is alias name for some "canonical" (the real) name
- www.ibm.com is really servereast.backup2.ibm.com
- `value` is canonical name

ibm.com, servereast.backup2.ibm.com, CNAME

type=NS

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

foo.com, dns.foo.com, NS

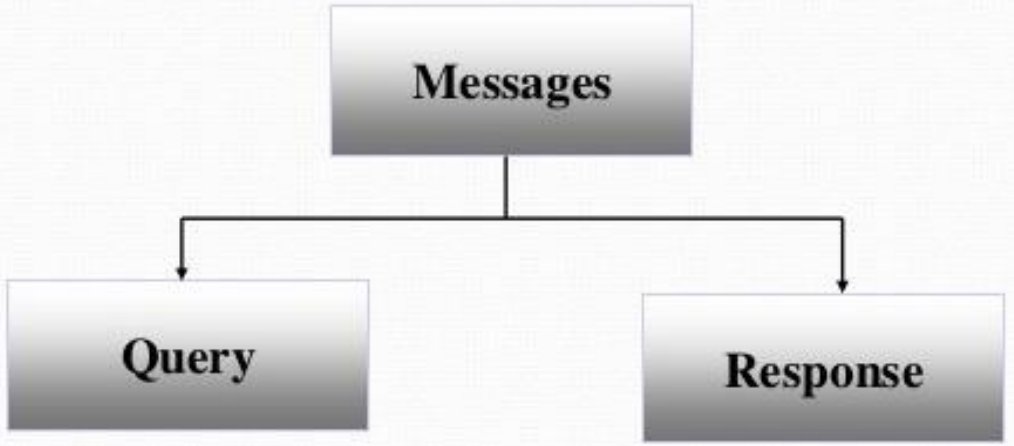
type=MX

- `value` is canonical name of a mailserver associated with alias hostname `name`

example.com, mail.example.com, MX

DNS – MESSAGES

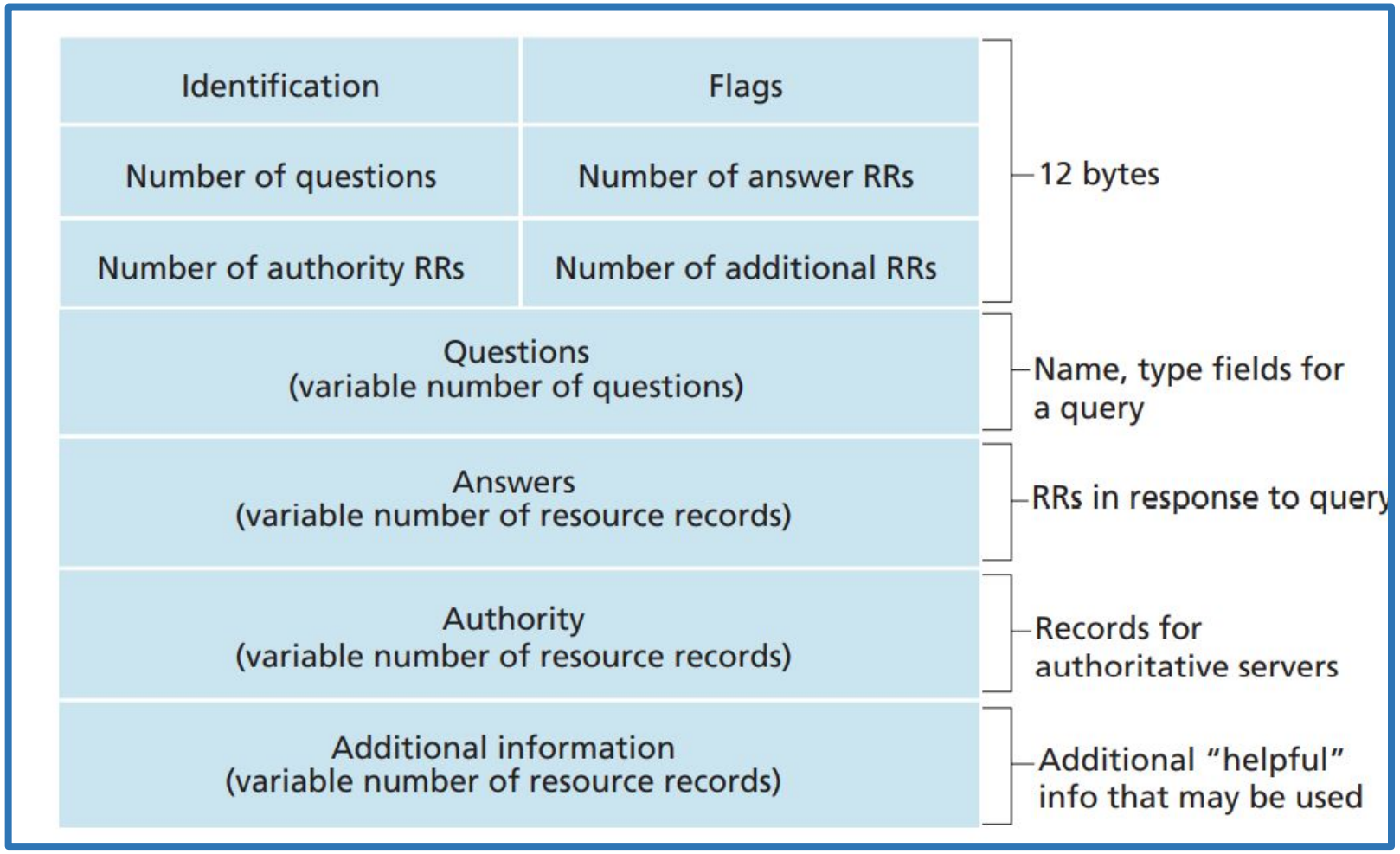
DNS Protocol Messages



DNS has two types of messages: query and response. Both types have the same format. The query message consists of a header and question records; the response message consists of a header, question records, answer records, authoritative records, and additional records

COMPUTER NETWORKS

DNS Messages Format



DNS Protocol Messages

- The semantics of the various fields in a DNS message are as follows:

✔ The first 12–bytes is the header section, which has a number of fields.

✔ The first field is a 16–bit number that identifies the query.

✔ This identifier is copied into the reply message to a query, allowing the client to match received replies with sent queries.

DNS Protocol Messages

- There are a number of flags in the flag field.
- A 1–bit query/reply flag indicates whether the message is a **query (0) or a reply (1).**
- A 1–bit authoritative flag is set in a reply message when a **DNS server is an authoritative server** for a queried name.
- A 1–bit recursion–desired flag is set when a client (host or DNS server) desires that the **DNS server perform recursion** when it doesn't have the record.
- A 1–bit recursion–available field is set in a reply if the **DNS server supports recursion.**

DNS Protocol Messages

- ✔ In the header, there are also four number–of fields.
- ✔ These fields indicate the number of occurrences of the four types of data sections that follow the header.

DNS Protocol Messages

- ✔ The question section contains information about the query that is being made.
- ✔ This section includes (1) a name field that contains the name that is being queried, and (2) a type field that indicates the type of question being asked about the name—for example, a host address associated with a name (Type A) or the mail server for a name (Type MX).

DNS Protocol Messages

- ✔ In a reply from a DNS server, the answer section contains the resource records for the name that was originally queried.
- ✔ Recall that in each resource record there is the Type (for example, A, NS, CNAME, and MX), the Value, and the TTL.
- ✔ A reply can return multiple RRs in the answer, since a hostname can have multiple IP addresses.

DNS Protocol Messages

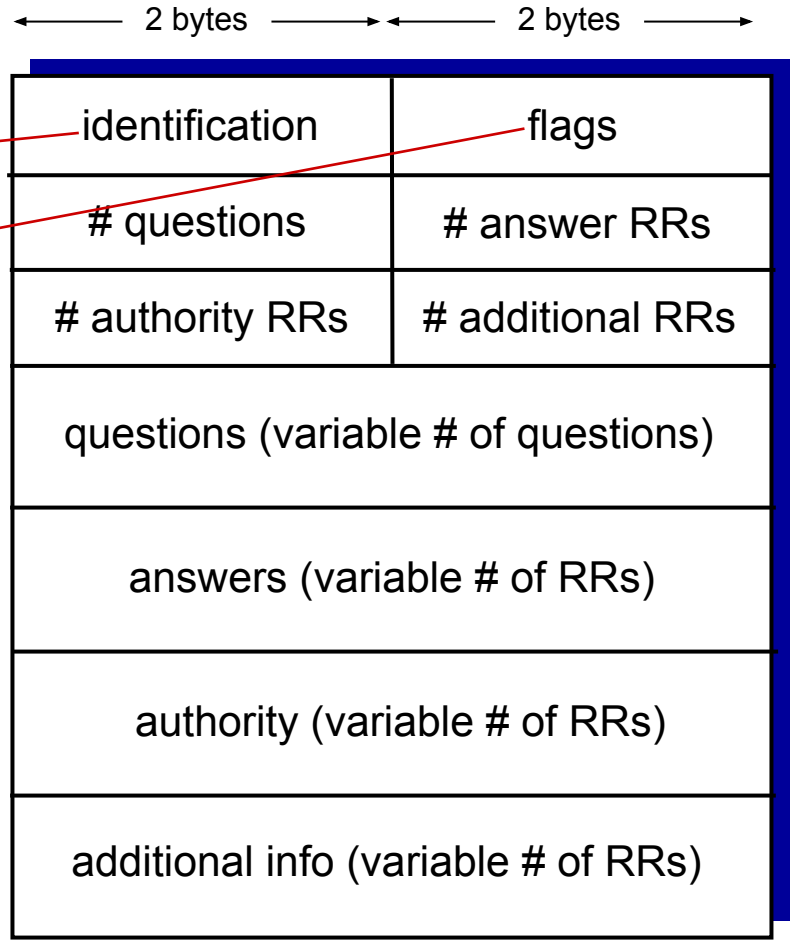
- ✔ The authority section contains records of other authoritative servers.
- ✔ The additional section contains other helpful records.
- ✔ For example, the answer field in a reply to an MX query contains a resource record providing the canonical hostname of a mail server.
- ✔ The additional section contains a Type A record providing the IP address for the canonical hostname of the mail server.

DNS Protocol Messages

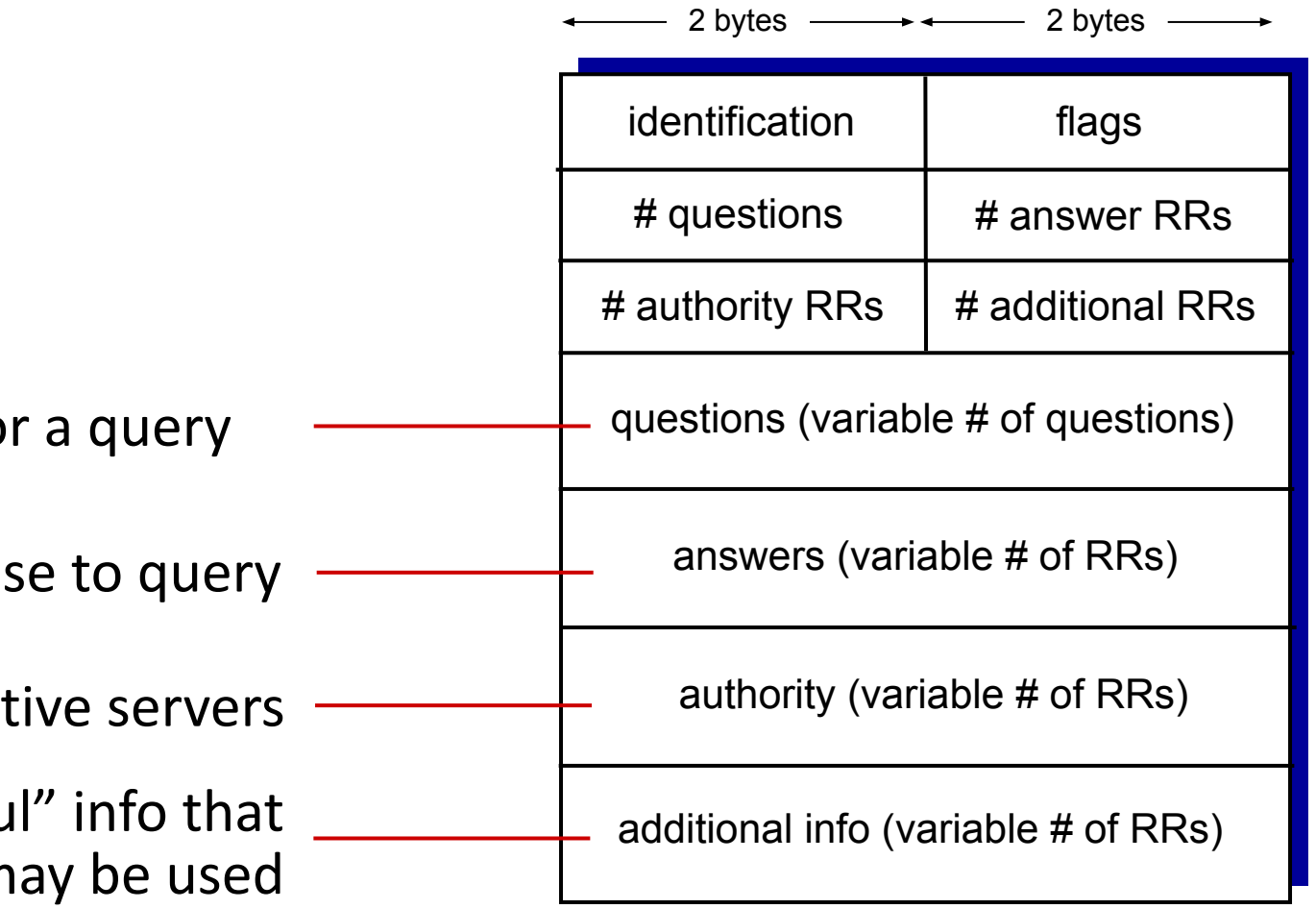
DNS *query* and *reply* messages, both have same *format:*

message header:

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



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



name, type fields for a query

RRs in response to query

records for authoritative servers

additional " helpful" info that may be used

Type (for example, A, NS, CNAME, and MX), the Value, and the TTL.

Emulating Local DNS Server (Step 1: Ask Root)

Directly send the query to this server.

```
seed@ubuntu:~$ dig @a.root-servers.net www.example.net

(Only a portion of the reply is shown here)
;; QUESTION SECTION:
;www.example.net.               IN      A

;; AUTHORITY SECTION:
net.                    172800  IN      NS      m.gtld-servers.net.
net.                    172800  IN      NS      l.gtld-servers.net.
net.                    172800  IN      NS      k.gtld-servers.net.

;; ADDITIONAL SECTION:
m.gtld-servers.net.     172800  IN      A       192.55.83.30
l.gtld-servers.net.     172800  IN      A       192.41.162.30
k.gtld-servers.net.     172800  IN      A       192.52.178.30
```

No answer (the root does not know the answer)

Go ask them!

Steps 2–3: Ask .net & example.net servers

```
seed@ubuntu:~$ dig @m.gtld-servers.net www.example.net

;; QUESTION SECTION:
;www.example.net.               IN      A

;; AUTHORITY SECTION:
example.net.            172800  IN      NS      a.iana-servers.net.
example.net.            172800  IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     172800  IN      A       199.43.132.53
b.iana-servers.net.     172800  IN      A       199.43.133.53
```

- Ask a .net nameservers.

Go ask them!

```
seed@ubuntu:$ dig @a.iana-servers.net www.example.net

;; QUESTION SECTION:
;www.example.net.               IN      A

;; ANSWER SECTION:
www.example.net.        86400   IN      A       93.184.216.34
```

- Ask an example.net nameservers.

Finally got the answer

COMPUTER NETWORKS

Inserting records into DNS

Example: new startup "Network Utopia"

- register name **networkuptopia.com** at *DNS registrar* (e.g., Network Solutions)
  - provide names, IP addresses of authoritative name server (primary and secondary)
  - registrar inserts NS, A RRs into .com TLD server:

  (networkutopia.com, dns1.networkutopia.com, NS)

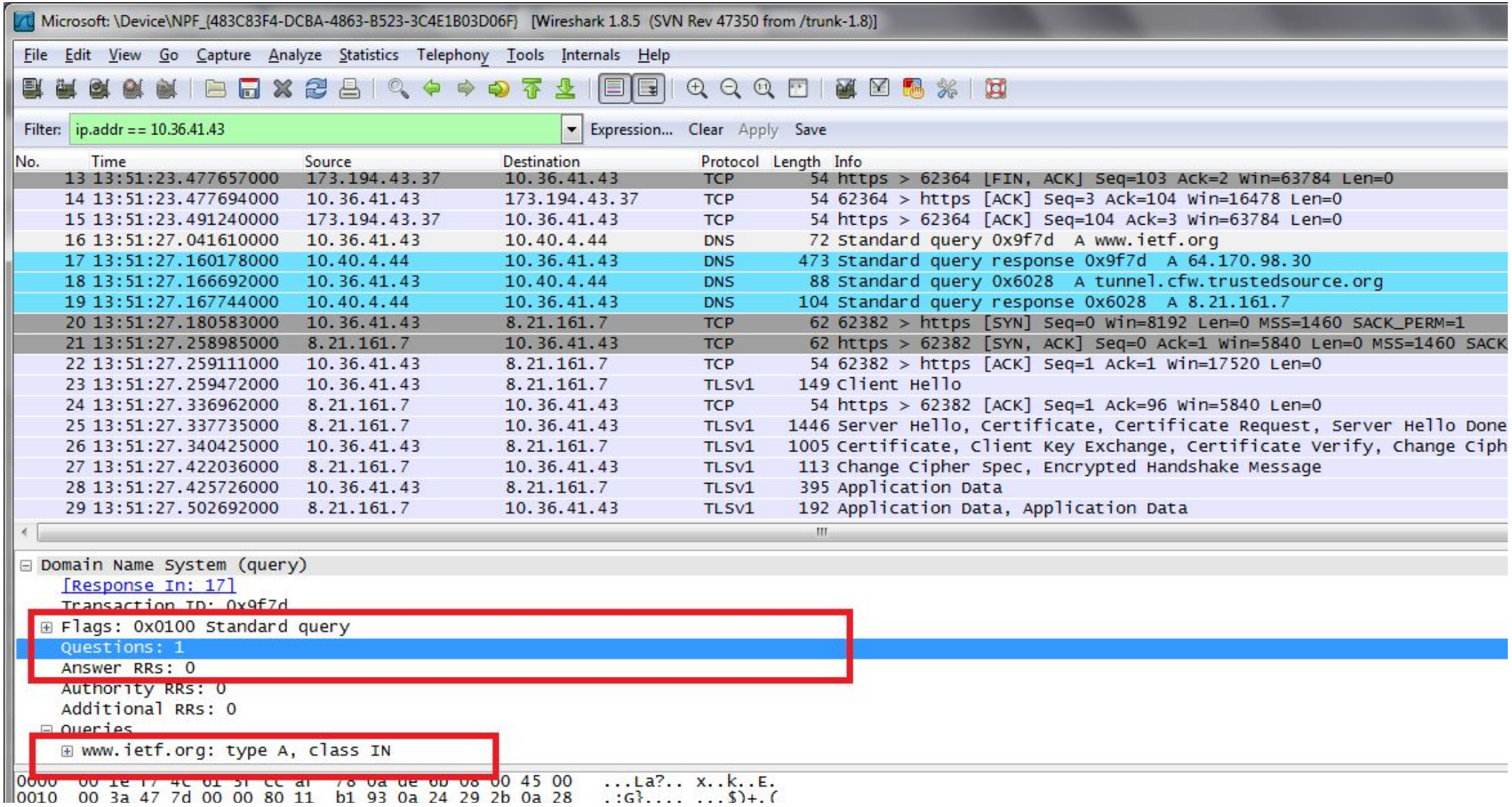
  (dns1.networkutopia.com, 212.212.212.1, A)
- create authoritative server locally with IP address 212.212.212.1
  - type A record for www.networkuptopia.com
  - type MX record for networkutopia.com

A registrar is a commercial entity that verifies the uniqueness of the domain name, enters the domain name into the DNS database and collects a small fee from you for its services.

Internet Corporation for Assigned Names and Numbers **(ICANN)** accredits the various registrars.
A complete list of accredited registrars is available at **http:// www.internic.net.**

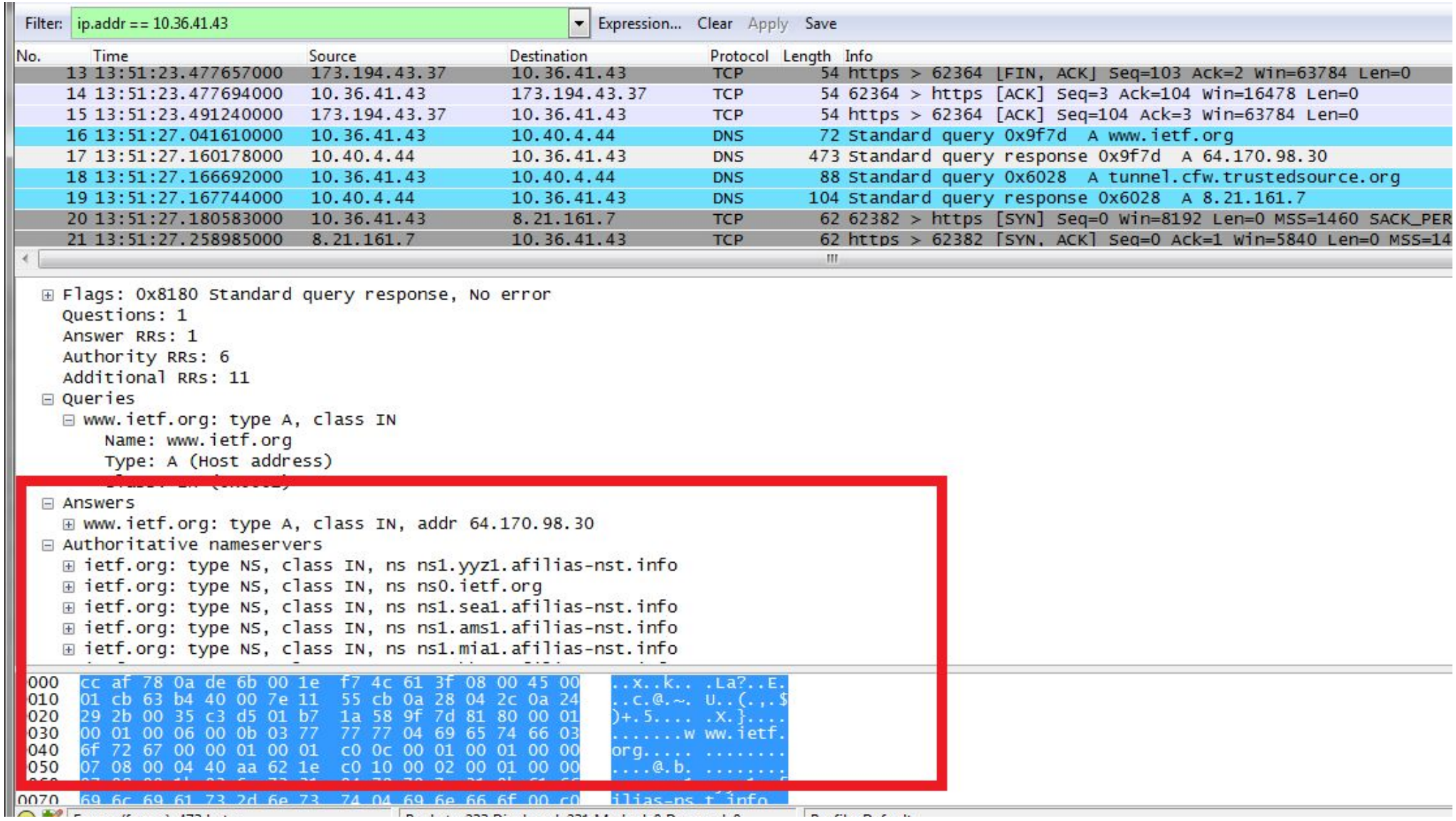
COMPUTER NETWORKS

DNS Request - Wireshark Packet Capture



COMPUTER NETWORKS

DNS Response - Wireshark Packet Capture



Suggested Readings

- DNS (Domain Name System) – Explained – https://youtu.be/JkEYOt08-rU
- How a DNS Server (Domain Name System) works – https://youtu.be/rdVPflECed8
- Wireshark Lab: DNS v7.0 – http://www-net.cs.umass.edu/wireshark-labs/Wireshark_DNS_v7.0.pdf

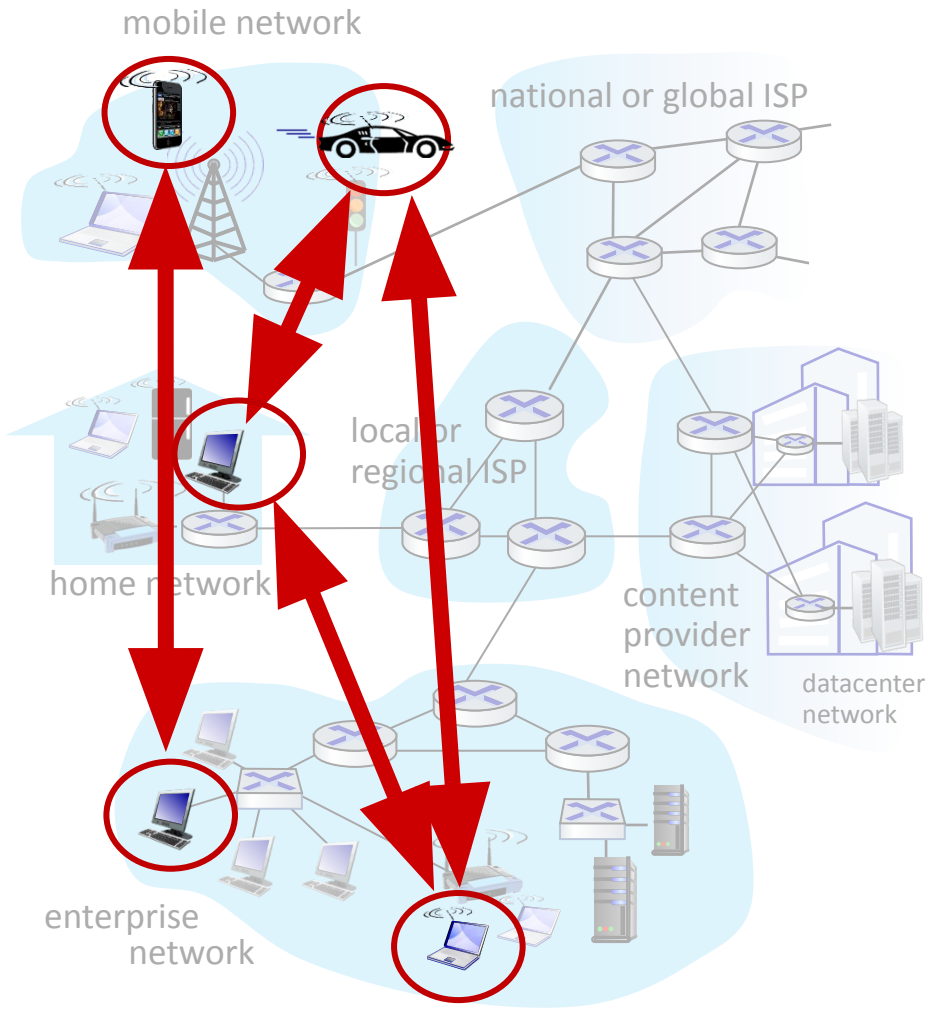
Unit – 2 Application Layer

2.1 Principles of Network Applications
2.2 Web, HTTP and HTTPS
2.3 The Domain Name System
2.4 P2P Applications
2.5 Socket Programming with TCP & UDP
2.6 Other Application Layer Protocols

COMPUTER NETWORKS

Peer–to–peer (P2P) architecture

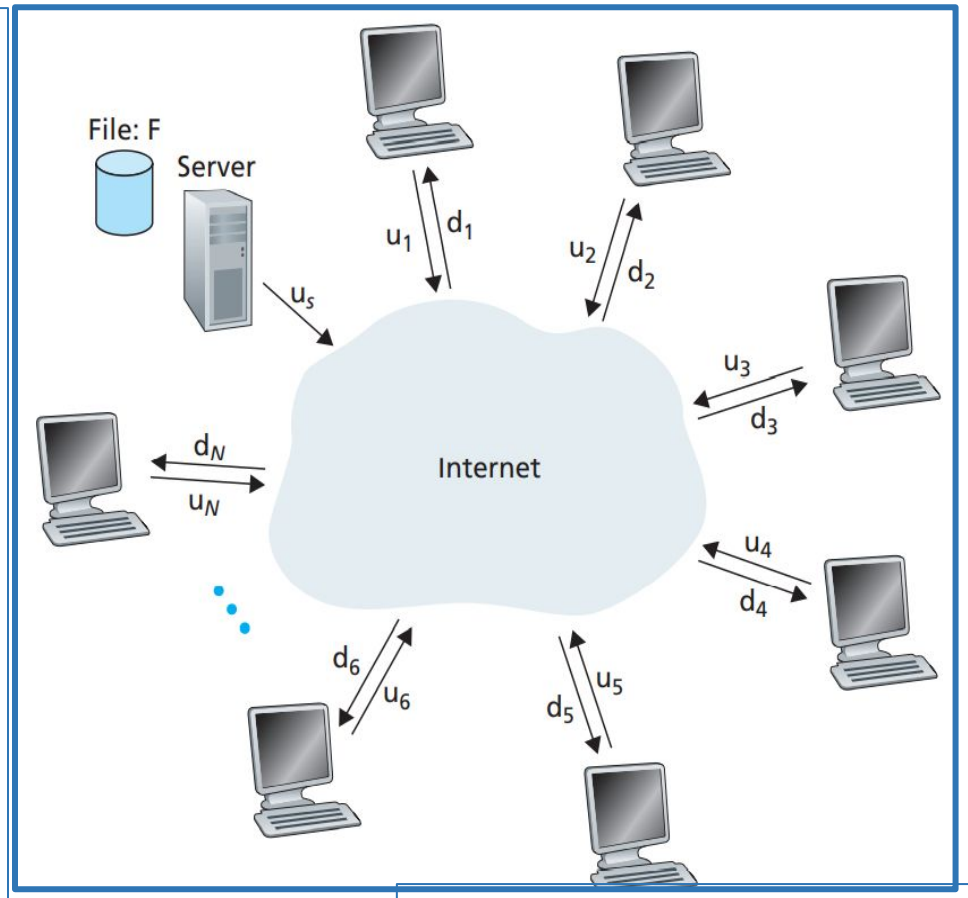
- *not* always–on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, and new service demands
- peers are intermittently connected and change IP addresses
  - complex management
- examples: P2P file sharing (BitTorrent), media streaming (Spotify), VoIP (Skype)



COMPUTER NETWORKS

Scalability of P2P Architectures

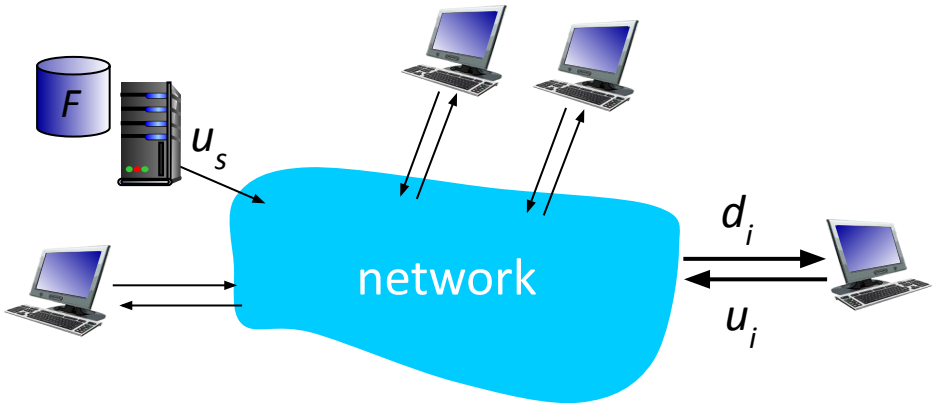
- Compare **client-server architectures with peer-to-peer architectures**
- Consider a simple quantitative model for distributing a file to a **fixed set of peers** for both architecture types.
- The server and the peers are connected to the Internet with access links.
- Denote the
  - **Upload rate of the server's access link by u_s**
  - **The upload rate of the ith peer's access link by u_i**
  - **The download rate of the ith peer's access link by d_i**
  - **Denote the size of the file to be distributed (in bits) by F**
  - **The number of peers that want to obtain a copy of the file by N**



The distribution time is the time it takes to get a copy of the file to all *N* peers.

File distribution time: Client–Server

- *server transmission:* must sequentially send (upload) *N* file copies:
  - time to send one copy: F/u_s
  - time to send *N* copies: NF/u_s
- *client:* each client must download file copy
  - d_{min} = min client download rate
  - min client download time: F/d_{min}



$d_{min} = \min\{d_1, d_2, \ldots, d_N\}$.

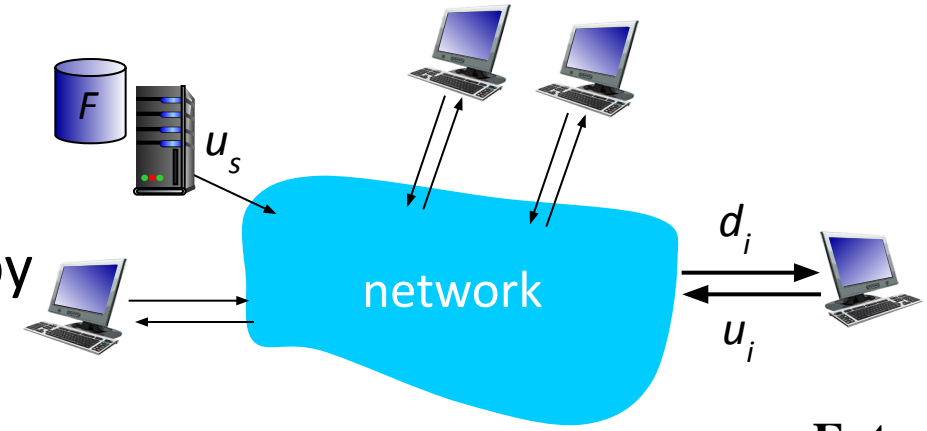
time to distribute F to N clients using client-server approach

$$D_{c\text{-}s} = max\{NF/u_s, F/d_{min}\}$$

increases linearly in N

File distribution time: P2P

- *server transmission:* must upload at least one copy:
  - time to send one copy: F/u_s
- *client:* each client must download file copy
  - min client download time: F/d_{min}
- *clients:* as aggregate must download NF bits
  - max upload rate (limiting max download rate) is $u_s + \Sigma u_i$



Total upload capacity of the system as a whole

Eqtn - provides a lower bound for the minimum distribution time for the P2P architecture.

time to distribute F to N clients using P2P approach

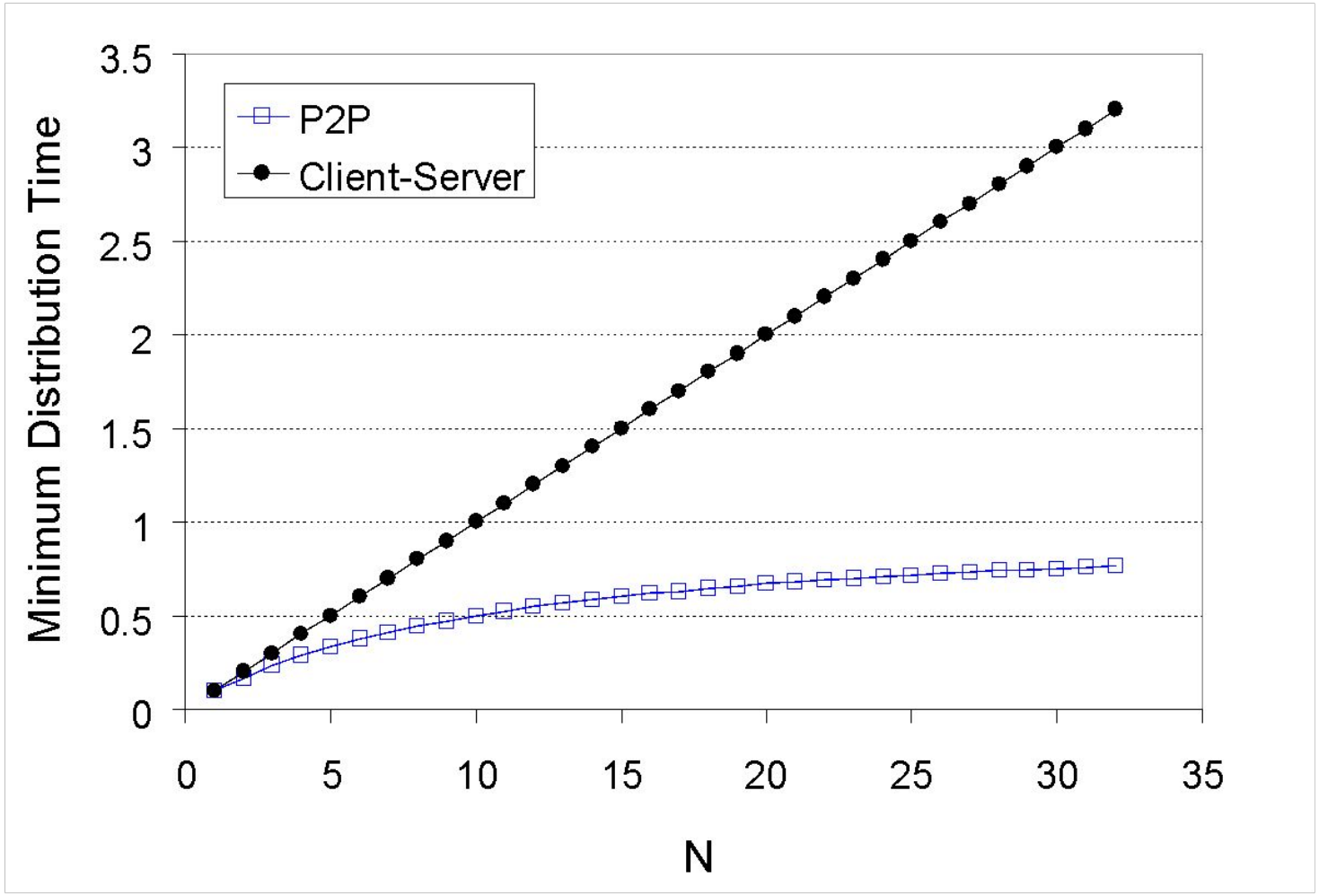
$$D_{P2P} = max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

increases linearly in N

… but so does this, as each peer brings service capacity

Client–Server vs P2P: example

Client (all peers) upload rate = u, F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$



- A peer can transmit the entire file in one hour.
- The server transmission rate is 10 times the peer upload rate.
- Peer download rates are set large enough so as not to have an effect.

P2P file distribution: BitTorrent

- ***BitTorrent*** is a popular P2P protocol for file distribution.
- In BitTorrent lingo, the collection of all peers participating in the distribution of a particular file is called a ***torrent***.
- Peers in a torrent download equal–size chunks of the file from one another, with a typical chunk size of 256 KBytes.
- When a peer first joins a torrent, it has no chunks.

P2P file distribution: BitTorrent

- Over time it accumulates more and more chunks.
- While it downloads chunks, it also uploads chunks to other peers.
- Once a peer has acquired the entire file, it may (selfishly) leave the torrent, or remain in the torrent and continue to upload chunks to other peers.
- Also, any peer may leave the torrent at any time with only a subset of chunks, and later rejoin the torrent.

P2P file distribution: BitTorrent

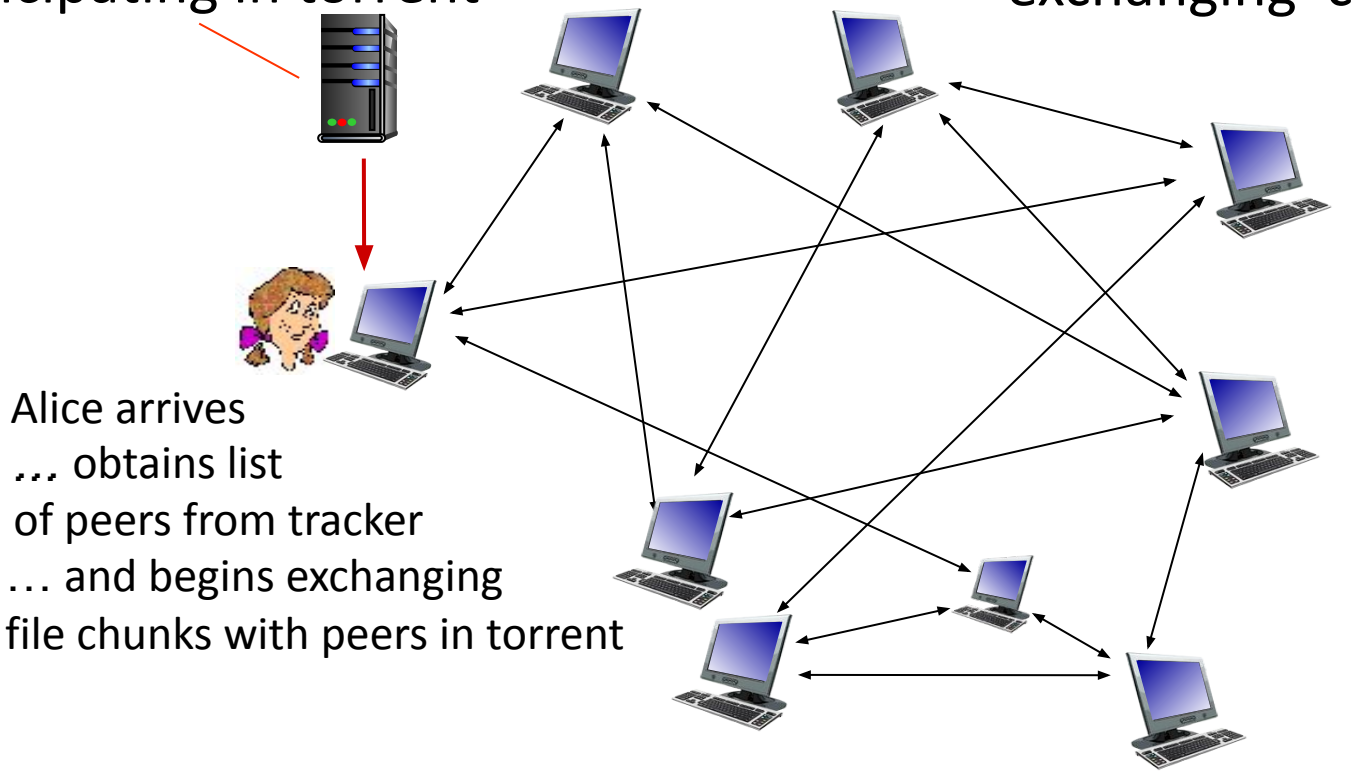
- Each torrent has an infrastructure node called a ***tracker***.
- When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent.
- In this manner, the tracker keeps track of the peers that are participating in the torrent.
- A given torrent may have fewer than ten or more than a thousand peers participating at any instant of time.

P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

tracker: tracks peers participating in torrent

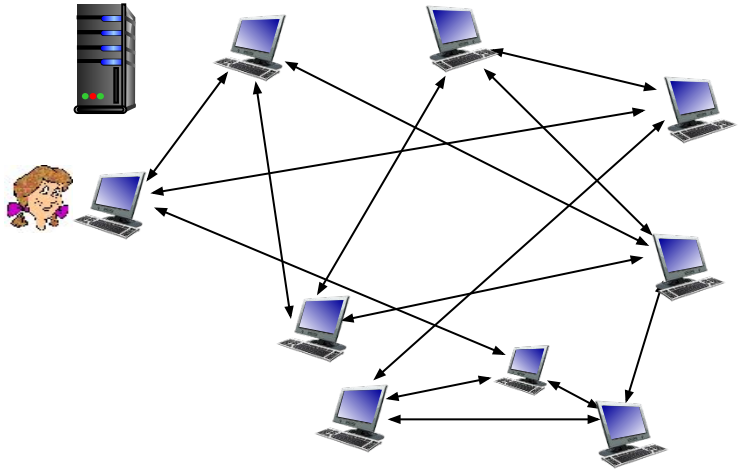
torrent: group of peers exchanging chunks of a file



Alice arrives
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent

P2P file distribution: BitTorrent

- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")



- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: requesting, sending file chunks

Requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, **rarest first**

Sending chunks: tit-for-tat

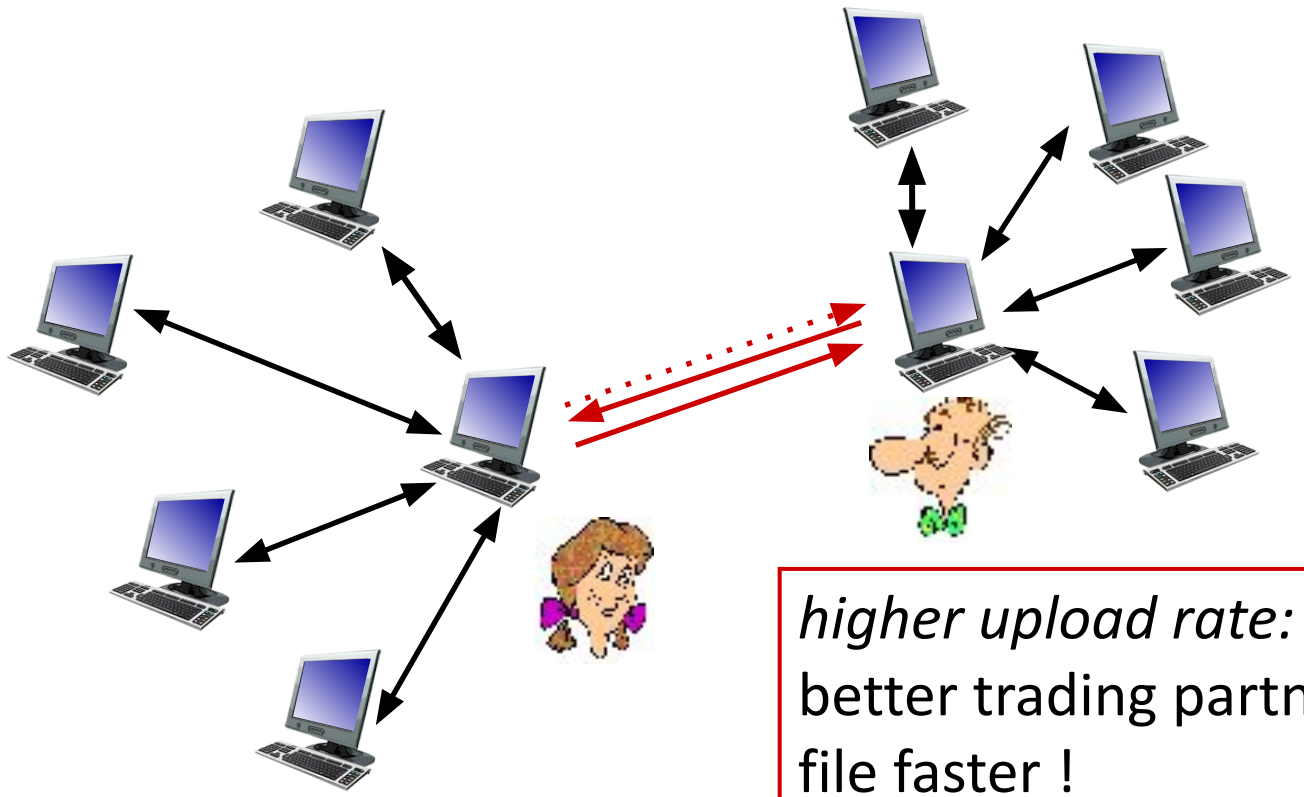
- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "**optimistically unchoke**" this peer
  - newly chosen peer may join top 4

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

(3) Bob becomes one of Alice's top-four providers



Pieces (mini-chunks), pipelining, random first selection, endgame mode, and anti-snubbing

higher upload rate: find better trading partners, get file faster !

P2P file distribution: BitTorrent

- As shown in figure, when a new peer, ***Alice***, joins the torrent, the tracker randomly selects a subset of peers (for concreteness, say 50) from the set of participating peers, and sends the IP addresses of these 50 peers to Alice.
- Possessing this list of peers, Alice attempts to establish concurrent TCP connections with all the peers on this list.
- Let's call all the peers with which Alice succeeds in establishing a TCP connection, "***neighbouring peers***."
- As time evolves, some of these peers may leave and other peers (outside the initial 50) may attempt to establish TCP connections with Alice.
- So a peer's neighbouring peers will fluctuate over time.

P2P file distribution: BitTorrent

- At any given time, each peer will have a subset of chunks from the file, with different peers having different subsets.
- Periodically, Alice will ask each of her neighbouring peers (over the TCP connections) for the list of the chunks they have.
- If Alice has L different neighbours, she will obtain L lists of chunks.
- With this knowledge, Alice will issue requests (again over the TCP connections) for chunks she currently does not have.

P2P file distribution: BitTorrent

- So at any given instant of time, Alice will have a subset of chunks and will know which chunks her neighbours have.
- With this information, Alice will have two important decisions to make.
- First, which chunks should she request first from her neighbours?
- And second, to which of her neighbours should she send requested chunks?
- In deciding which chunks to request, Alice uses a technique called ***rarest first***.

P2P file distribution: BitTorrent

- The idea is to determine, from among the chunks she does not have, the chunks that are the rarest among her neighbours (that is, the chunks that have the fewest repeated copies among her neighbours) and then request those rarest chunks first.
- In this manner, the rarest chunks get more quickly redistributed, aiming to (roughly) equalize the numbers of copies of each chunk in the torrent.

P2P file distribution: BitTorrent

- To determine which requests she responds to, BitTorrent uses a clever trading algorithm.
- The basic idea is that Alice gives priority to the neighbours that are currently supplying her data at the highest rate.
- Specifically, for each of her neighbours, Alice continually measures the rate at which she receives bits and determines the four peers that are feeding her bits at the highest rate.

P2P file distribution: BitTorrent

- She then reciprocates by sending chunks to these same four peers.
- Every 10 seconds, she recalculates the rates and possibly modifies the set of four peers.
- In BitTorrent lingo, these four peers are said to be ***unchoked***.
- Importantly, every 30 seconds, she also picks one additional neighbour at random and sends it chunks.
- Let's call the randomly chosen peer ***Bob***.

P2P file distribution: BitTorrent

- In BitTorrent lingo, Bob is said to be ***optimistically unchoked***.
- Because Alice is sending data to Bob, she may become one of Bob's top four uploaders, in which case Bob would start to send data to Alice.
- If the rate at which Bob sends data to Alice is high enough, Bob could then, in turn, become one of Alice's top four uploaders.
- In other words, every 30 seconds, Alice will randomly choose a new trading partner and initiate trading with that partner.

P2P file distribution: BitTorrent

- If the two peers are satisfied with the trading, they will put each other in their top four lists and continue trading with each other until one of the peers finds a better partner.
- The effect is that peers capable of uploading at compatible rates tend to find each other.

P2P file distribution: BitTorrent

- The random neighbour selection also allows new peers to get chunks, so that they can have something to trade.
- All other neighbouring peers besides these five peers (four "top" peers and one probing peer) are "***choked***," that is, they do not receive any chunks from Alice.

Suggested Readings

- BitTorrent (BTT) White Paper – https://www.bittorrent.com/btt/btt-docs/BitTorrent_(BTT)_White_Paper_v0.8.7_Feb_2019.pdf
- Peer-to-peer networking with BitTorrent – http://web.cs.ucla.edu/classes/cs217/05BitTorrent.pdf
- Torrents Explained: How BitTorrent Works – https://youtu.be/urzQeD7ftbI

Unit – 2 Application Layer

2.1 Principles of Network Applications

2.2 Web, HTTP and HTTPS

2.3 The Domain Name System

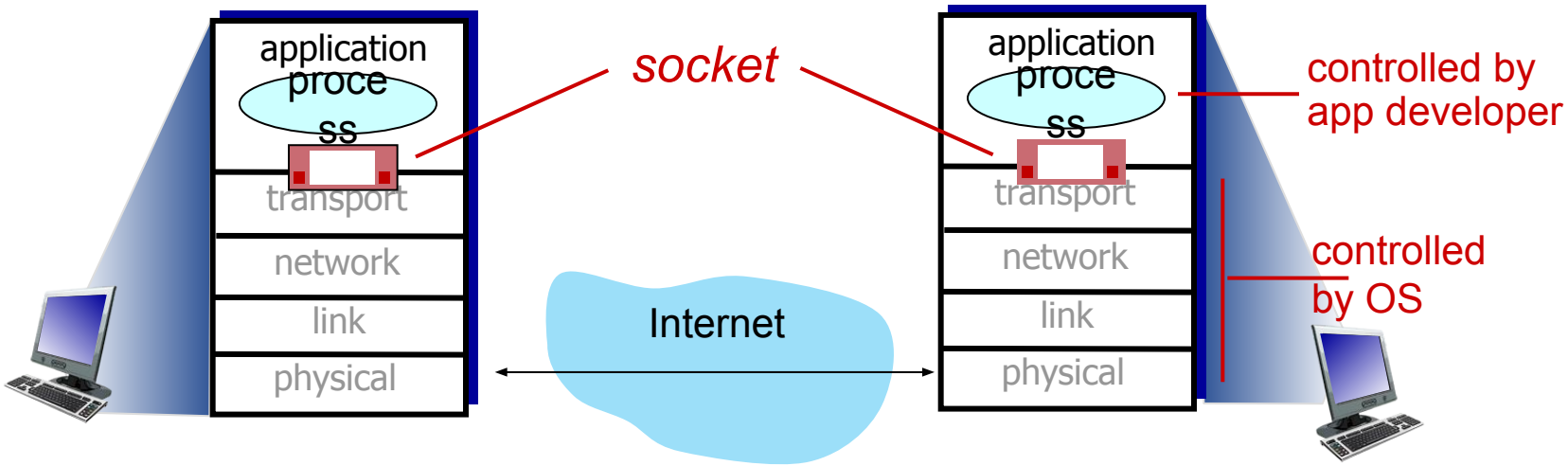
2.4 P2P Applications

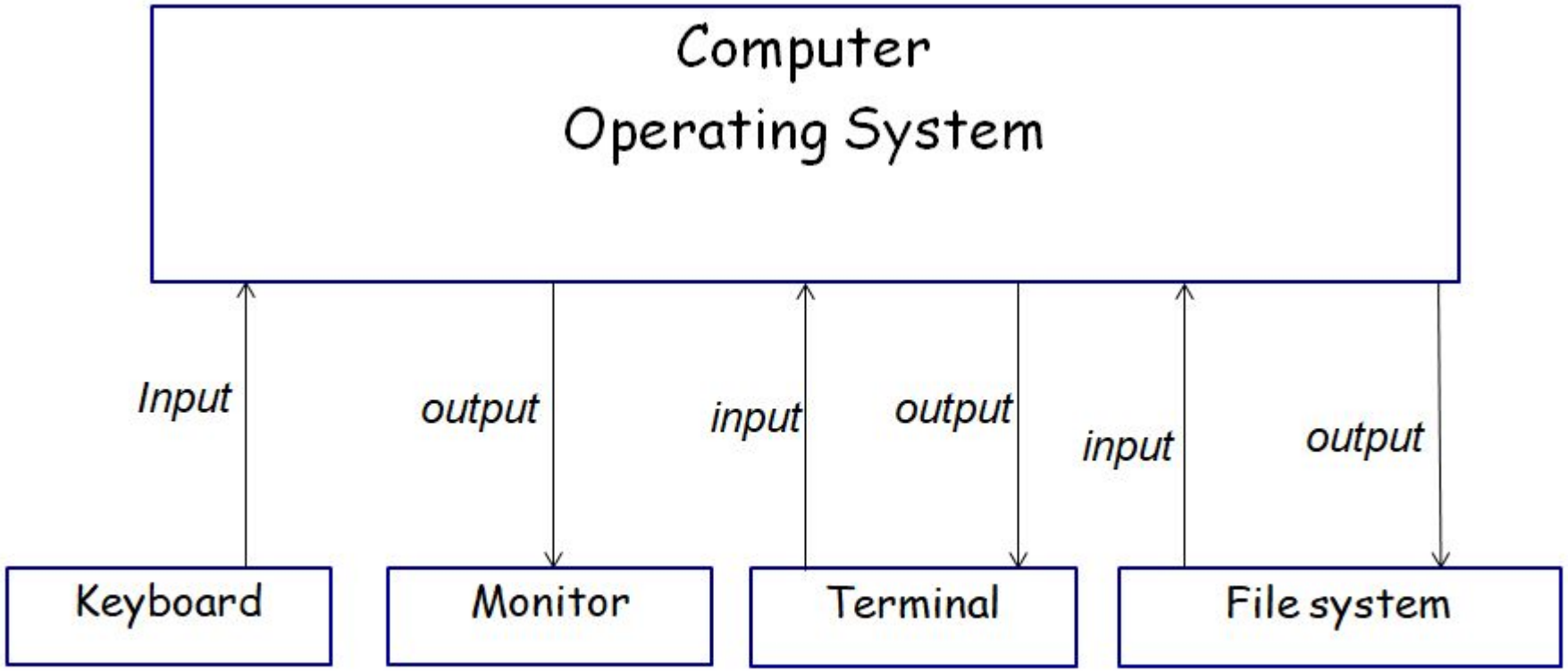
2.5 Socket Programming with TCP & UDP

2.6 Other Application Layer Protocols

goal: learn how to build client/server applications that communicate using sockets

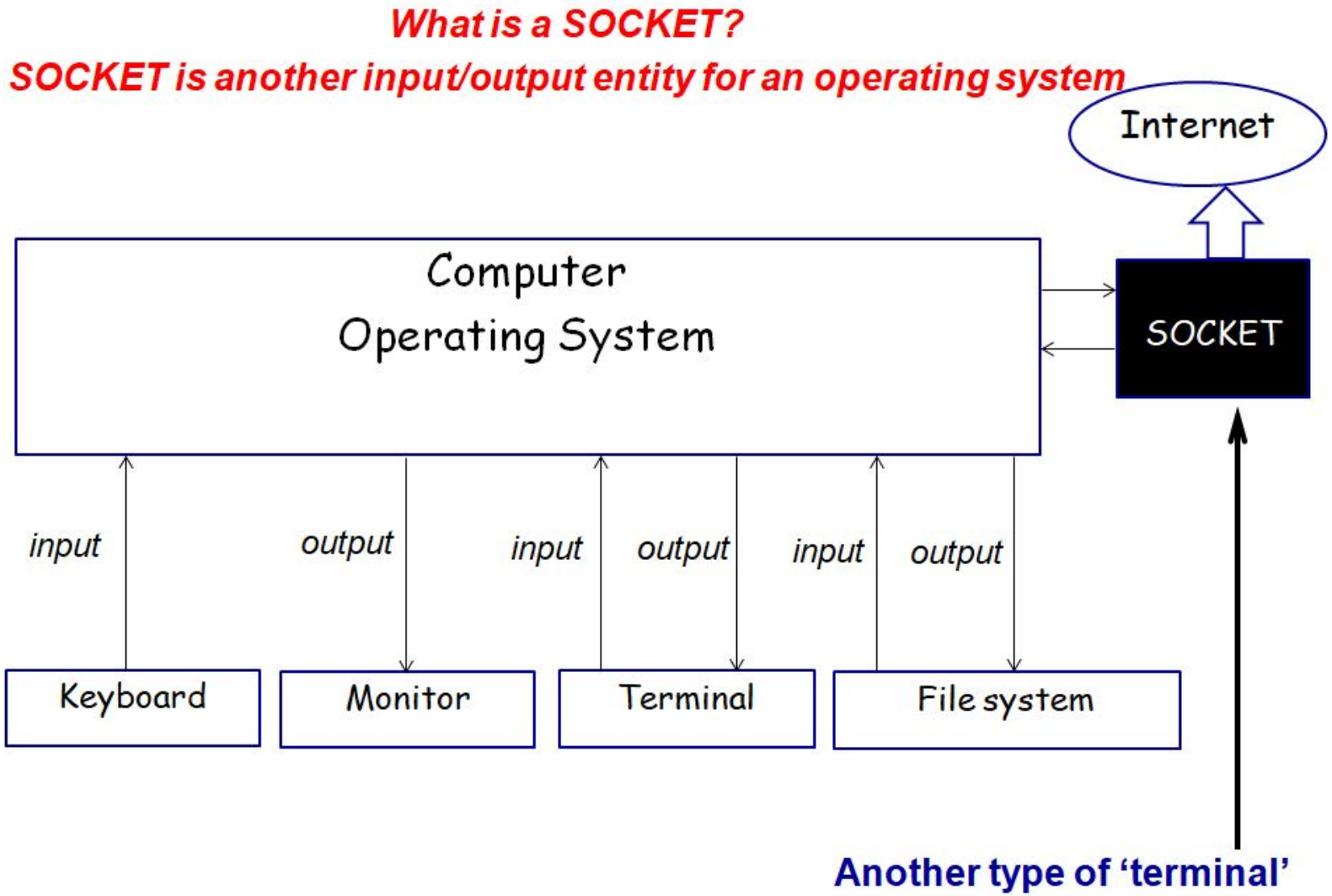
socket: door between application process and end–end–transport protocol





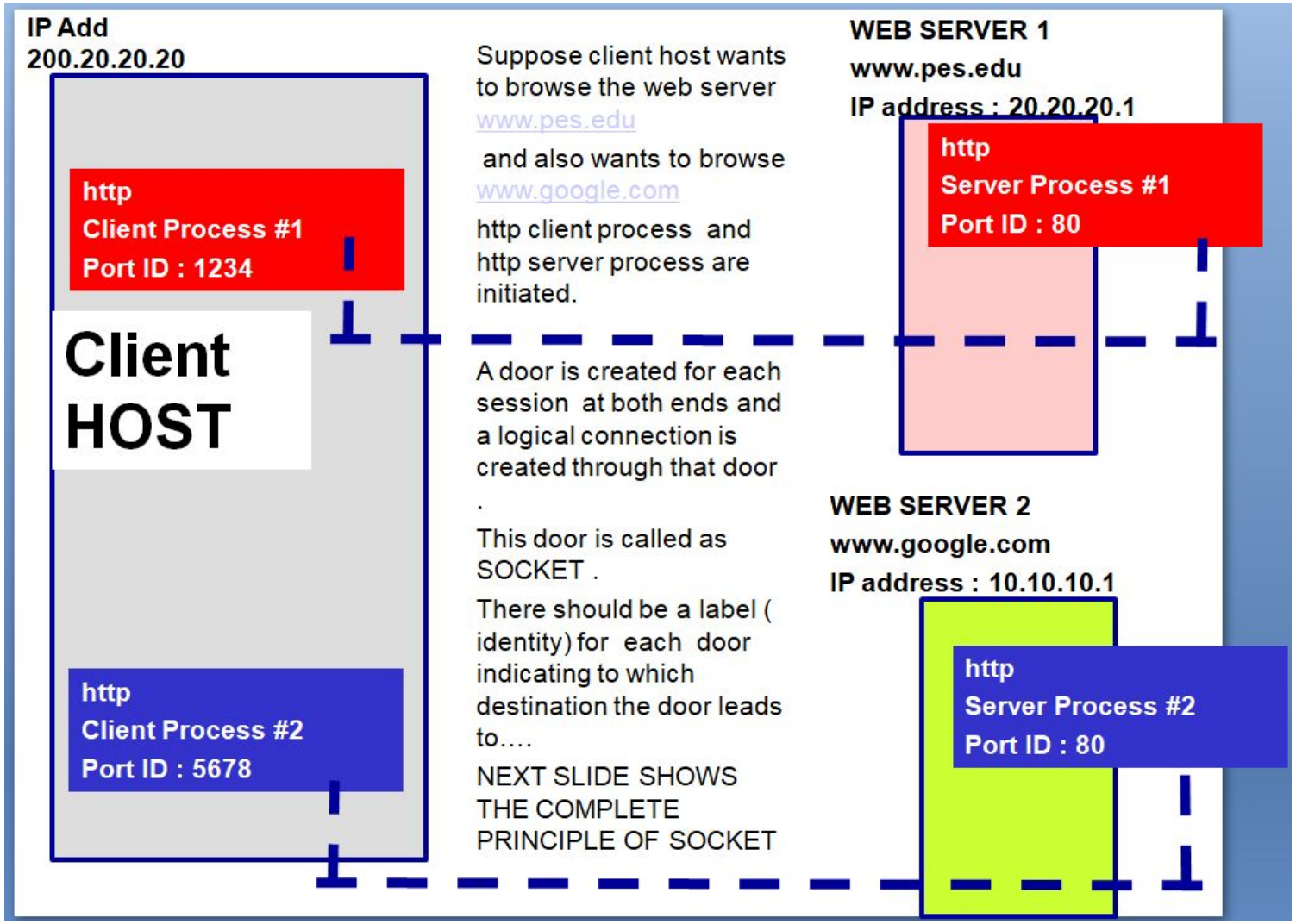
Typical entities connected to an Operating System

Socket Programming



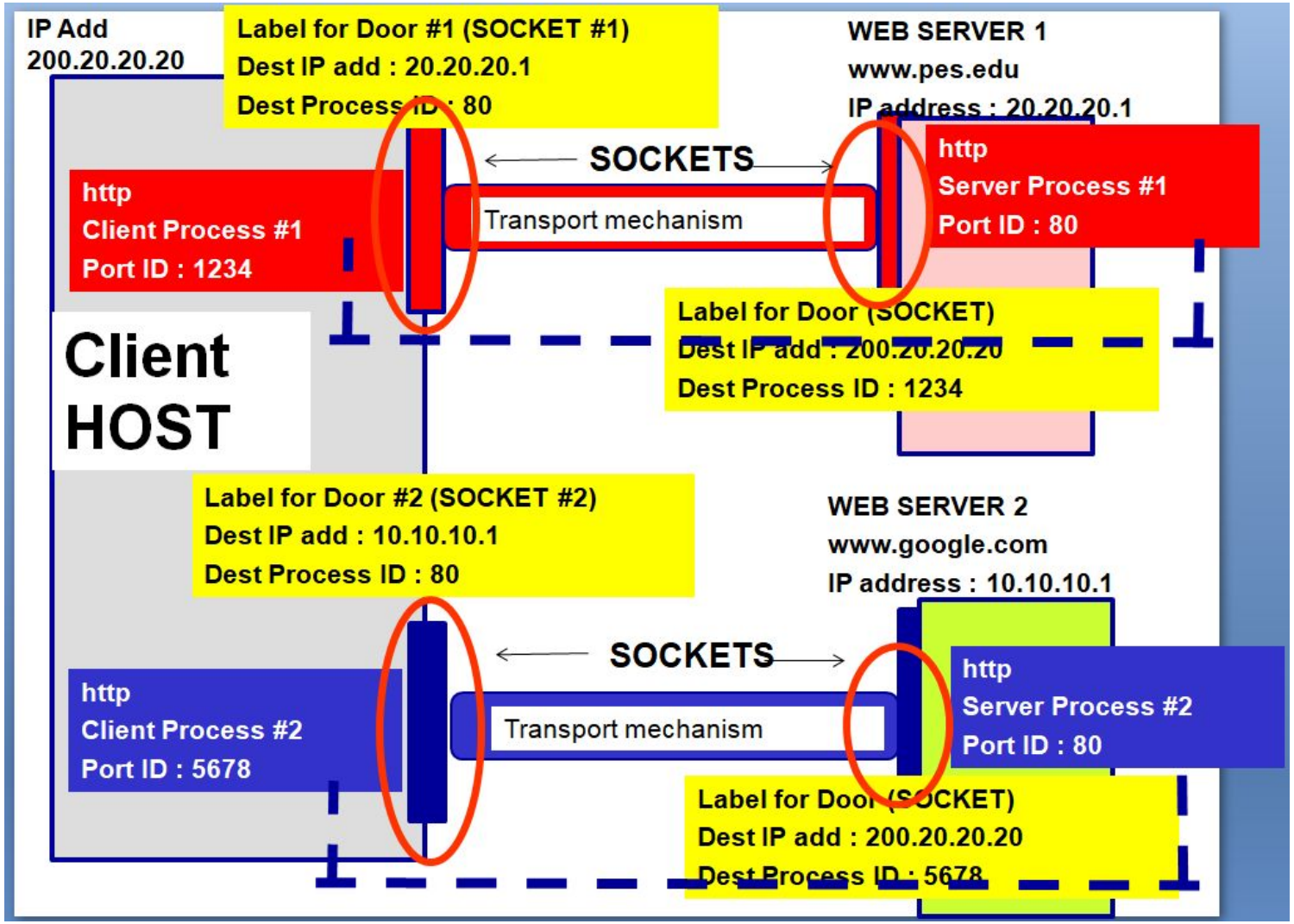
COMPUTER NETWORKS

Socket Programming



COMPUTER NETWORKS

Socket Programming



PES UNIVERSITY ONLINE

What is a socket?

- Though socket is supposed to behave like a **terminal**
- it is not a physical entity

- ***then what?***

- It is a DATA STRUCTURE
- that is CREATED and USED
- by
- **application program**

What is a SOCKET?

Data structure of a SOCKET

Family	Type	Protocol
Local Socket Address		
Remote Socket Address		

Data structure of a SOCKET address

Length	Family
Port number	
IP address	


```
int socket(int family, int type, int proto);
```

`family` specifies the protocol family
`type` specifies the type of service
`protocol` specifies the specific protocol (usually 0, which means *the default*).

PES UNIVERSITY ONLINE

socket( )

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing sockets
AF_KEY	Key socket

Figure 4.2 Protocol *family* constants for `socket` function.

type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_SEQPACKET	sequenced packet socket
SOCK_RAW	raw socket

Figure 4.3 *type* of socket for `socket` function.

Two socket types for two transport services:

- *UDP:* unreliable datagram
- *TCP:* reliable, byte stream–oriented

Application Example:

2. Client reads a line of characters (data) from its keyboard and sends the data to the server.
3. The server receives the data and converts characters to uppercase.
4. The server sends the modified data to the client.
5. The client receives the modified data and displays the line on its screen.

Socket programming with TCP

Client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

Client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

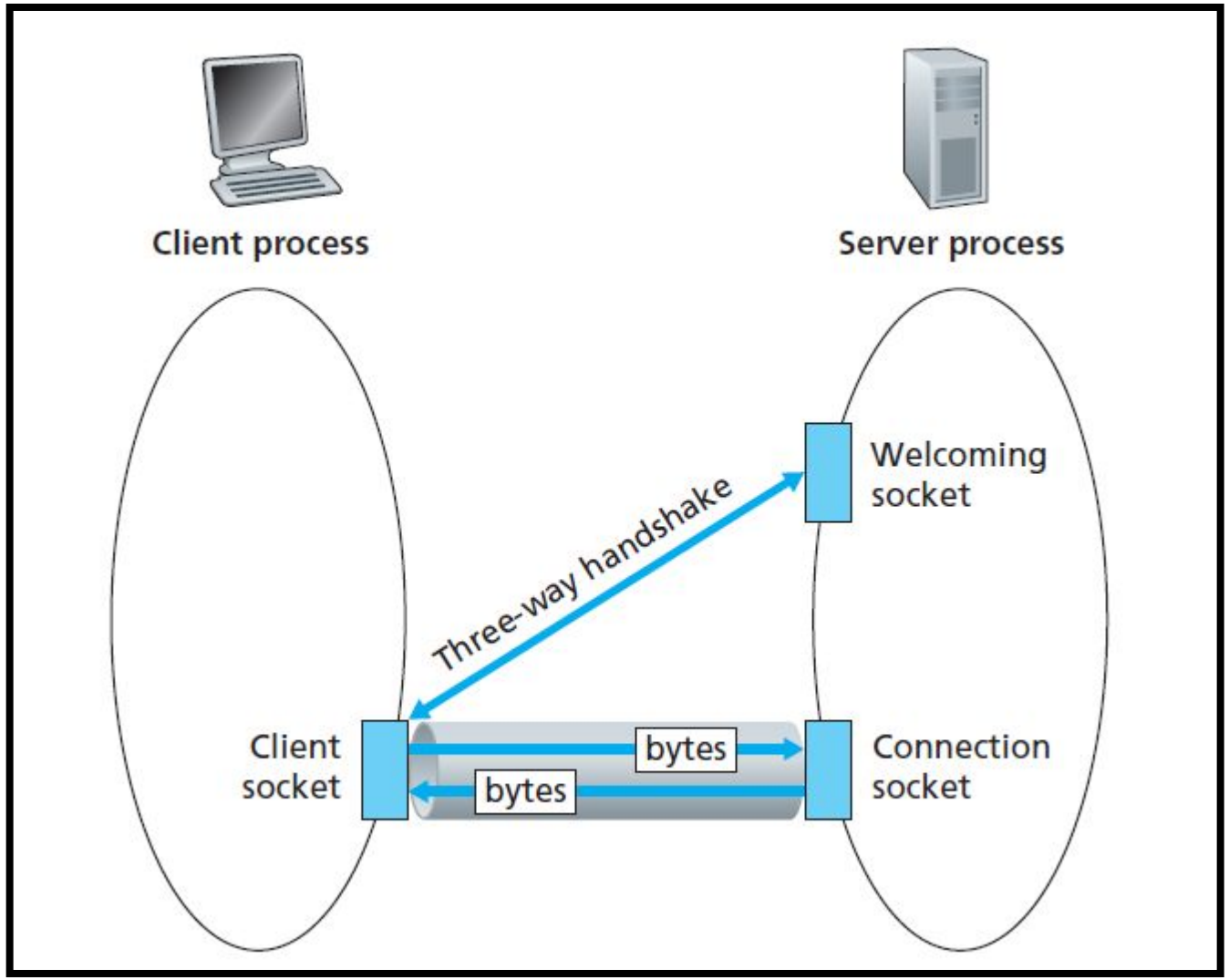
- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients

Application viewpoint

TCP provides reliable, in–order byte–stream transfer ("pipe") between client and server

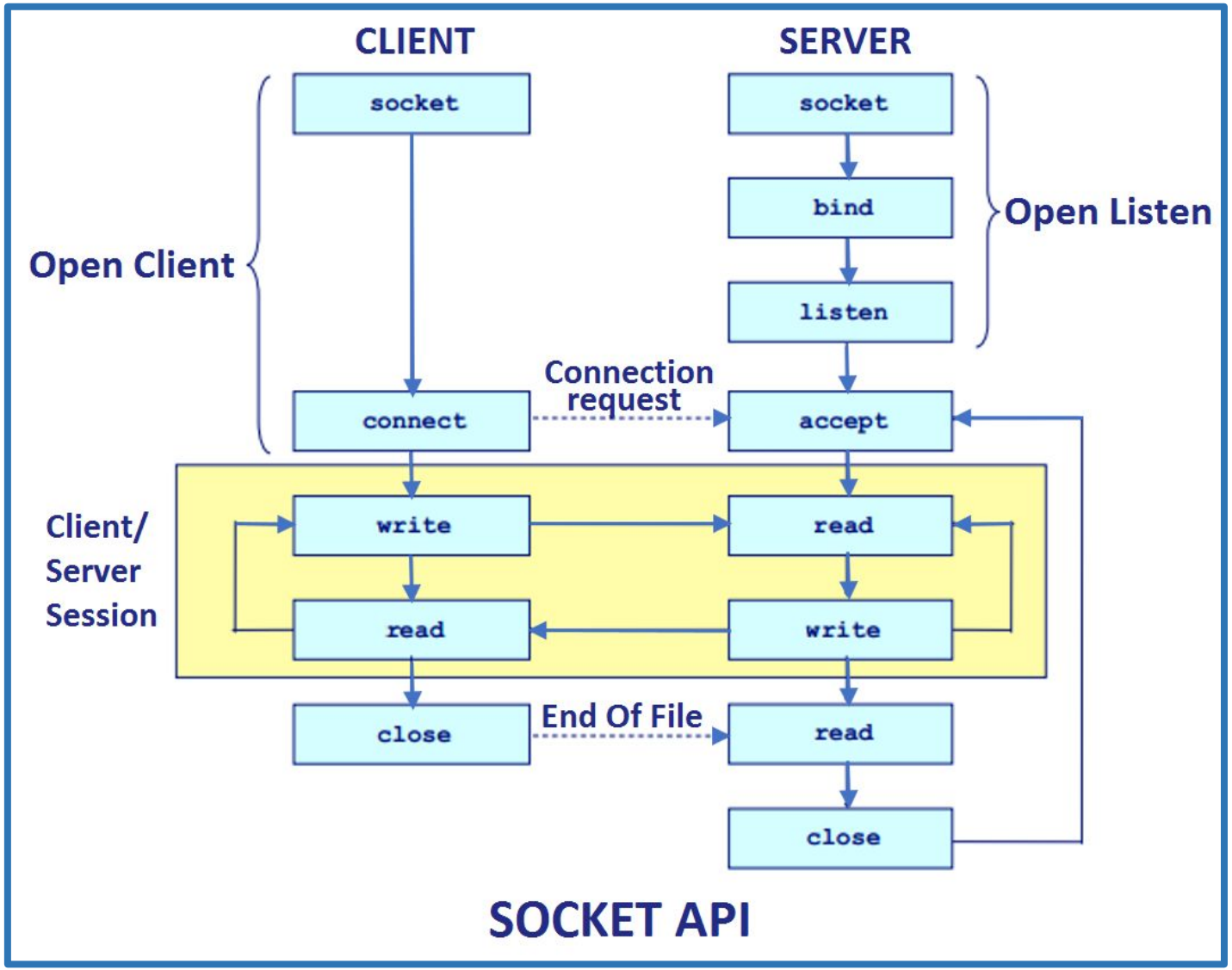
COMPUTER NETWORKS

The TCPServer Process has Two Sockets



COMPUTER NETWORKS

The TCPServer Process



Example app: TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

No need to attach server name, port

Example app: TCP server

Python TCPServer

```
 from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind((‘’,serverPort))
serverSocket.listen(1)
print ‘The server is ready to receive’
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                        encode())
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

Socket Programming with UDP

UDP: no "connection" between client & server

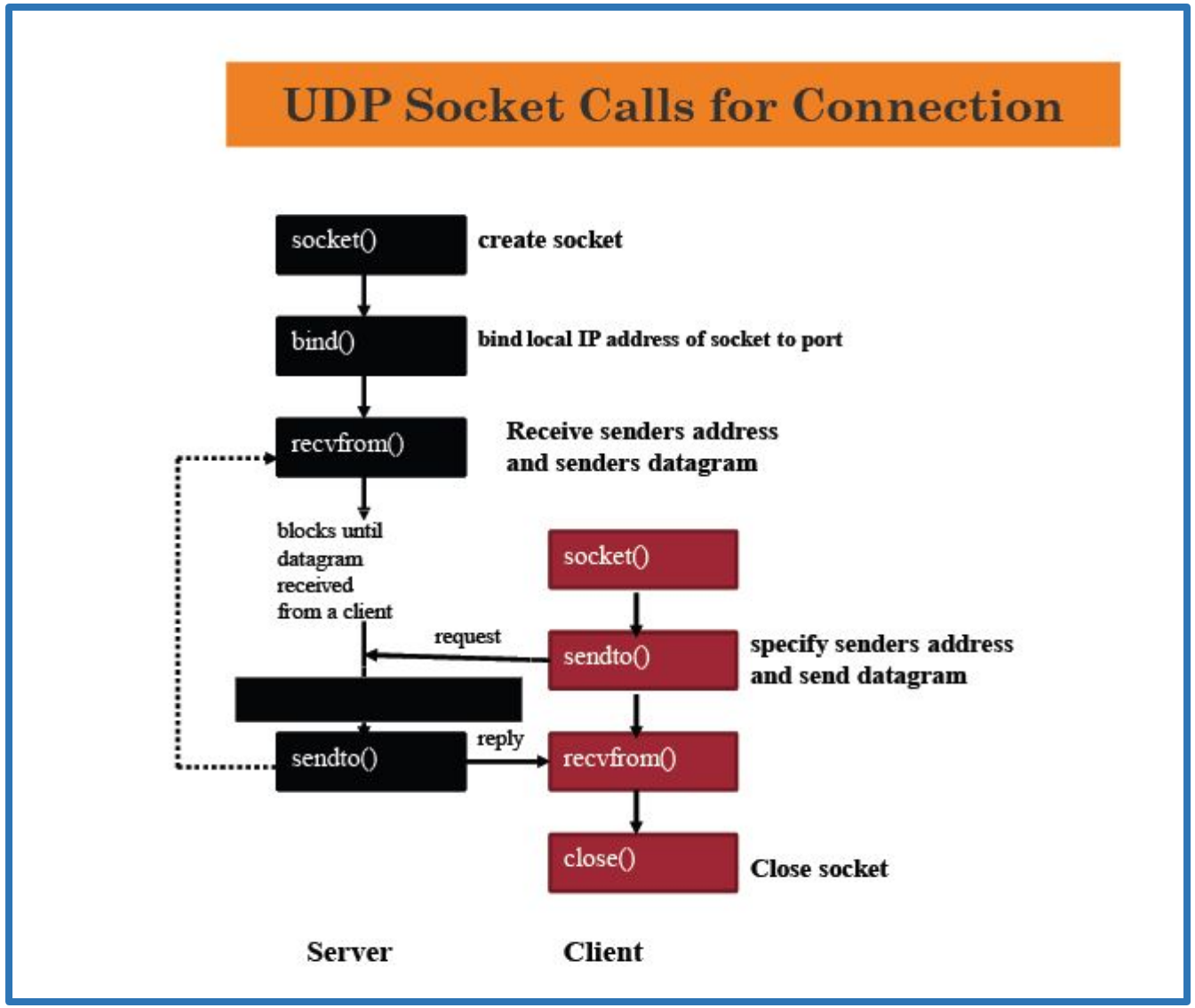
- no handshaking before sending data
- sender explicitly attaches IP destination address and port # to each packet
- receiver extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- UDP provides *unreliable* transfer of groups of bytes ("datagrams") between client and server

Socket Programming with UDP



Example app: UDP client

Python UDPClient

include Python's socket library →

```
from socket import *
serverName = 'hostname'
serverPort = 12000
```

create UDP socket for server →

```
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
```

get user keyboard input →

attach server name, port to message; send into socket →

```
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
```

read reply characters from socket into string →

print out received string and close socket →

```
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
print modifiedMessage.decode()
clientSocket.close()
```

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.decode().upper()
    serverSocket.sendto(modifiedMessage.encode(),
                        clientAddress)
```

create UDP socket →

bind socket to local port number 12000 →

loop forever →

Read from UDP socket into message, getting client's address (client IP and port) →

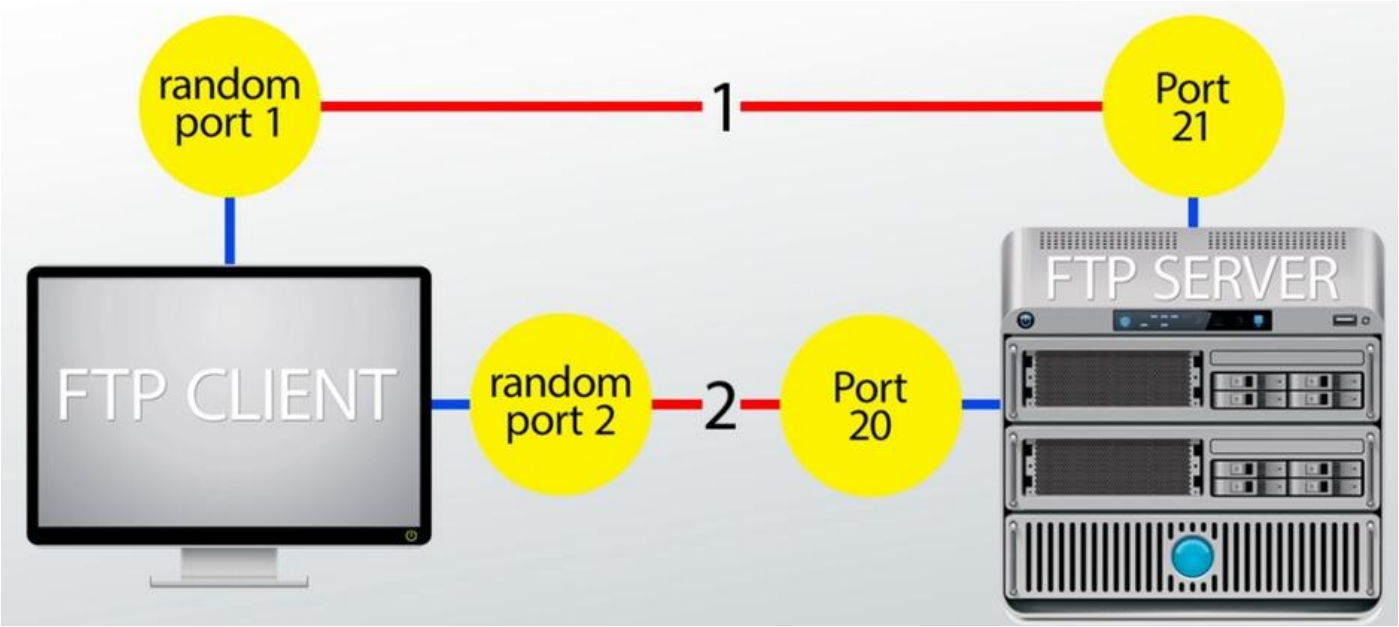
send upper case string back to this client →

Unit – 2 Application Layer

2.1 Principles of Network Applications
2.2 Web, HTTP and HTTPS
2.3 The Domain Name System
2.4 P2P Applications
2.5 Socket Programming with TCP & UDP
2.6 Other Application Layer Protocols

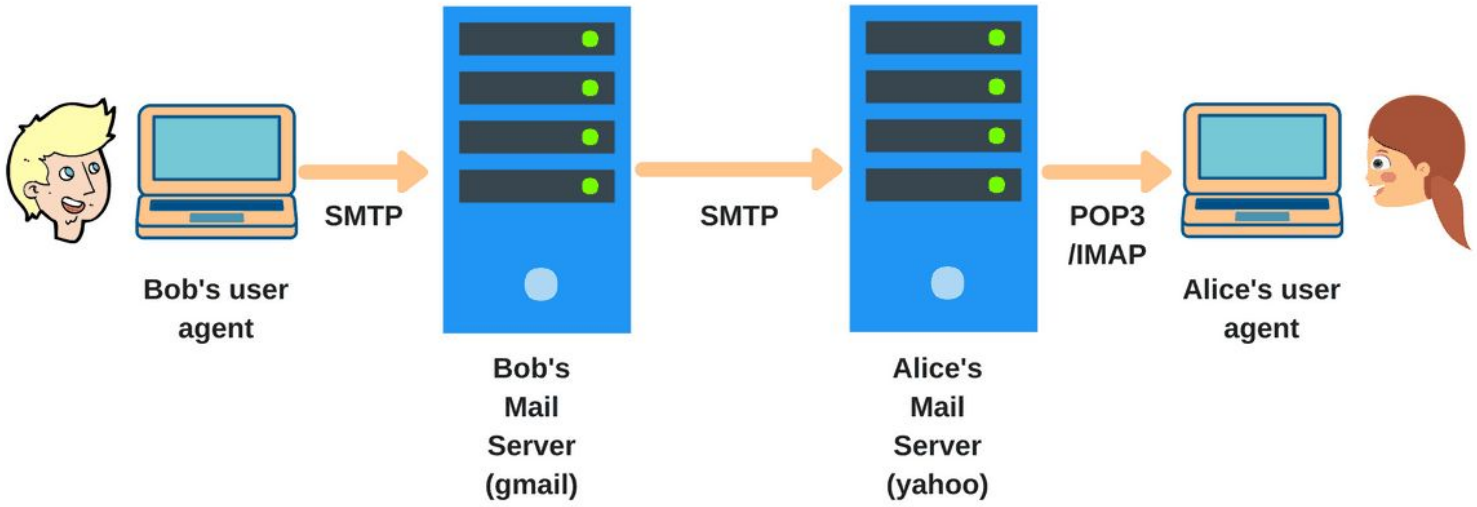
Other Application Layer Protocols - FTP

- **File Transfer Protocol** (**FTP**) – used to exchange large files on the internet TCP
- Invoked from the command prompt or some GUI.
- Allows to update (delete, rename, move, and copy) files at a server.
- Data connection (Port No. 20) & Control connection (Port No. 21)



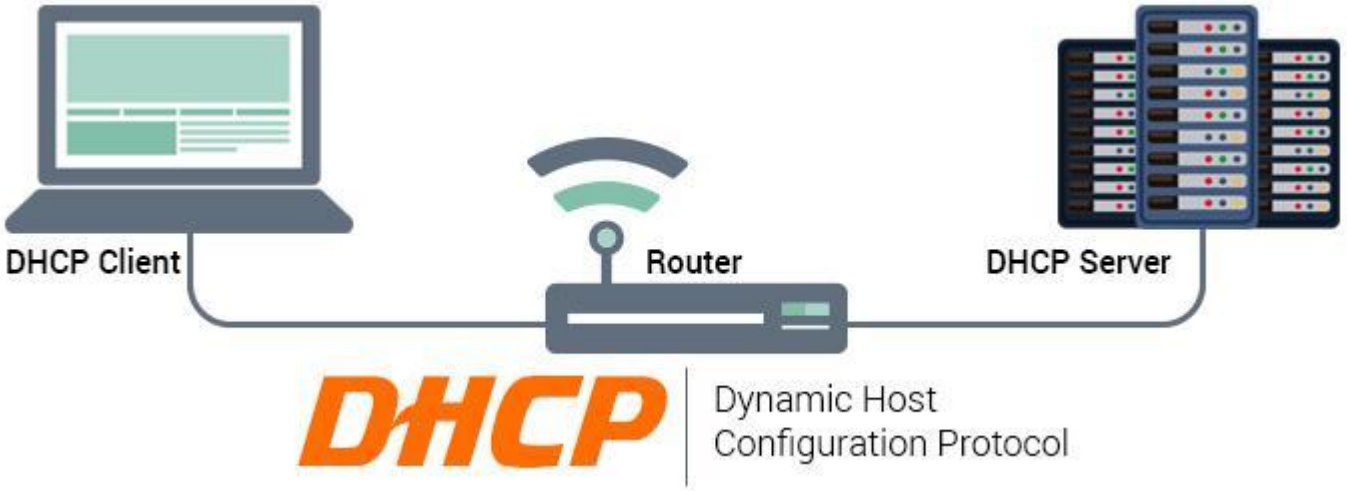
Other Application Layer Protocols - SMTP

- **Simple Mail Transfer Protocol** (**SMTP**) – an internet standard for e–mail Transmission.
- Connections are secured with SSL (Secure Socket Layer).
- Messages are stored and then forwarded to the destination (relay).
- SMTP uses a port number 25 of TCP.



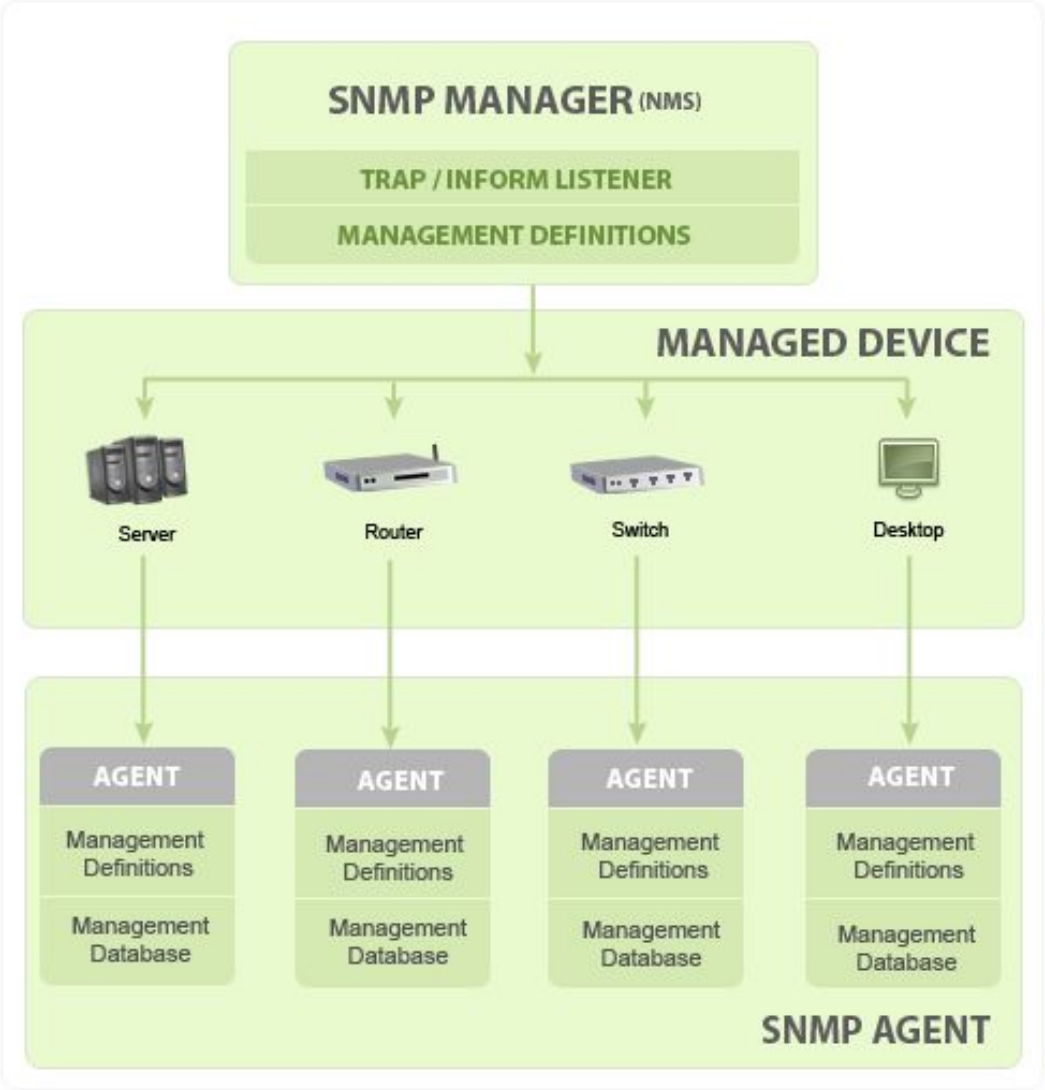
Other Application Layer Protocols - DHCP

- **Dynamic Host Configuration Protocol** (**DHCP**) – assign IP addresses to computers in a network dynamically.
- IP addresses may change even when computer is in network (DHCP leases).
- DHCP port number for server is 67 and for the client is 68.
- A client–server model, based on **discovery**, **offer**, **request**, and **ACK**.
- Includes subnet mask, DNS server address, default gateway.



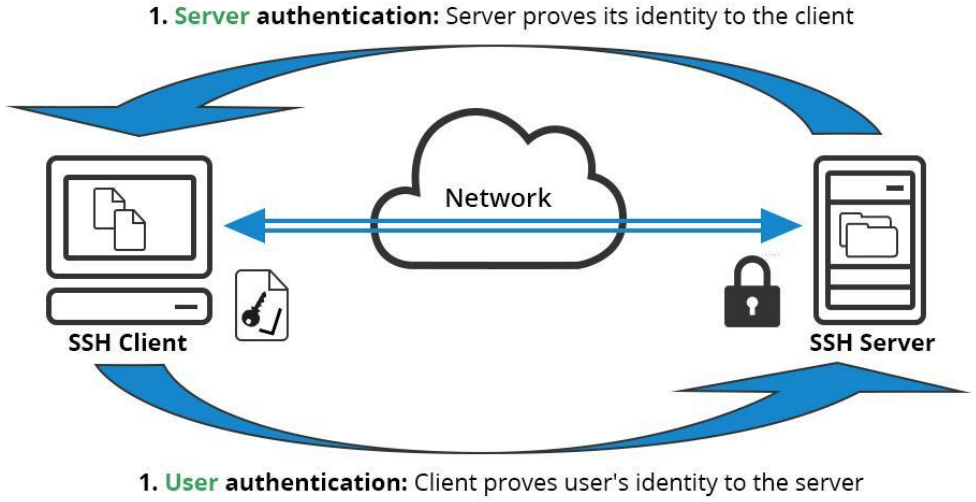
Other Application Layer Protocols - SNMP

- **Simple Network Management Protocol (SNMP)** – exchange management information between network devices.
- Basic components & functionalities
  - SNMP Manager
  - Managed Devices
  - SNMP Agents
  - MIB (Management Information Base)



Other Application Layer Protocols – Telnet & SSH

- Allows a user to communicate with a remote device.
- Used mostly by network admin to remotely access and manage devices.
- Telnet client and server installed – uses TCP port no. 23
- SSH – uses public key **encryption** & TCP port 22 by default.



Summary of Application Layer Protocols

Port #	Application Layer Protocol	Type	Description
20	FTP	TCP	File Transfer Protocol - data
21	FTP	TCP	File Transfer Protocol - control
22	SSH	TCP/UDP	Secure Shell for secure login
23	Telnet	TCP	Unencrypted login
25	SMTP	TCP	Simple Mail Transfer Protocol
53	DNS	TCP/UDP	Domain Name Server
67/68	DHCP	UDP	Dynamic Host
80	HTTP	TCP	HyperText Transfer Protocol
123	NTP	UDP	Network Time Protocol
161,162	SNMP	TCP/UDP	Simple Network Management Protocol
389	LDAP	TCP/UDP	Lightweight Directory Authentication Protocol
443	HTTPS	TCP/UDP	HTTP with Secure Socket Layer

Summary

our study of network application layer is now complete!

- application architectures
  - client–server
  - P2P
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection–oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - DNS
  - P2P: BitTorrent
- socket programming: TCP, UDP sockets

Summary (more)

Most importantly: learned about *protocols*!

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - *headers*: fields giving info about data
  - *data:* info (payload) being communicated

important themes:

- centralized vs. decentralized
- stateless vs. stateful
- scalability
- reliable vs. unreliable message transfer
- "complexity at network edge"

COMPUTER NETWORKS

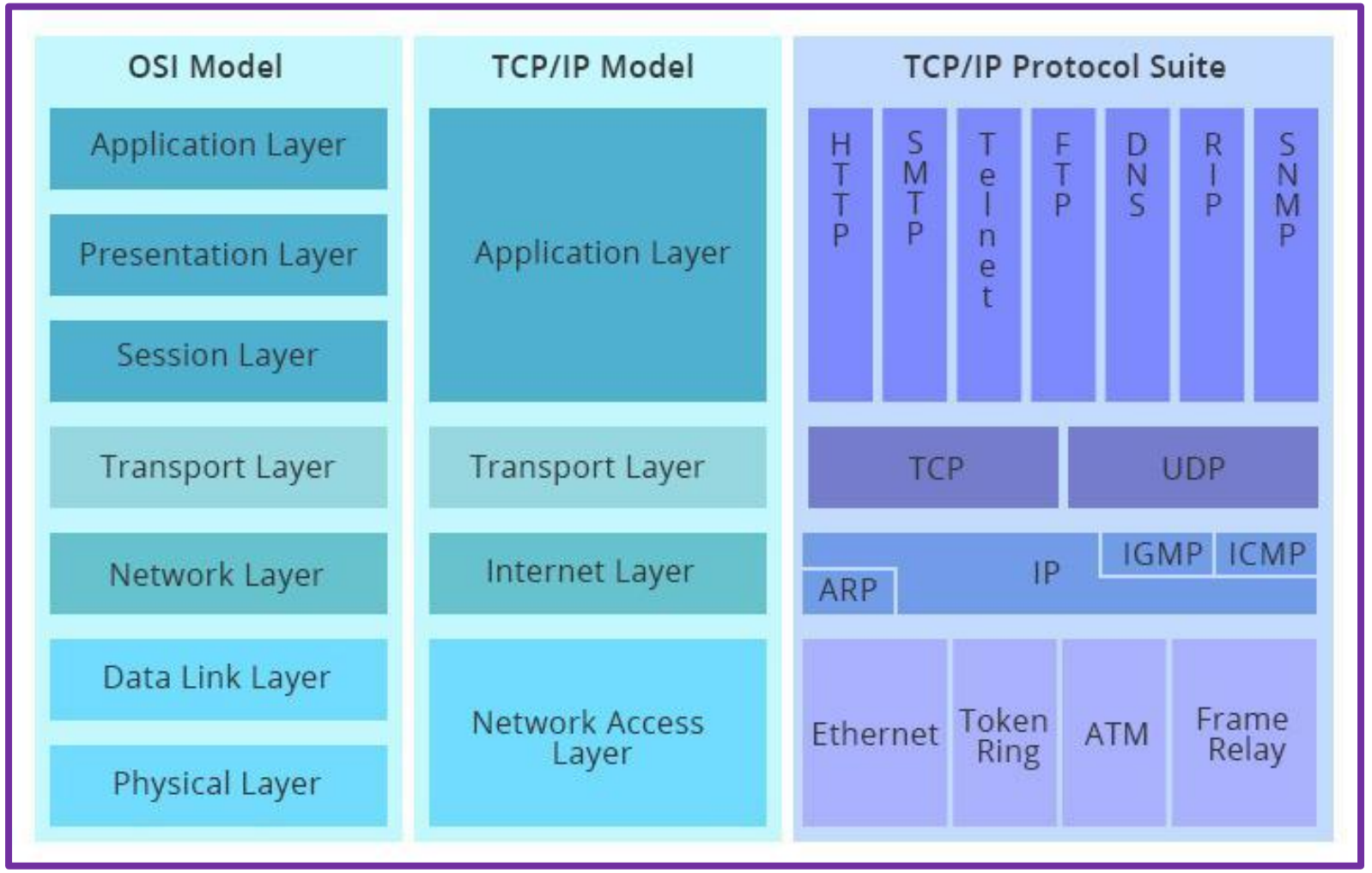
Transport Layer

Kundhavai K R
Department of Computer Science & Engineering

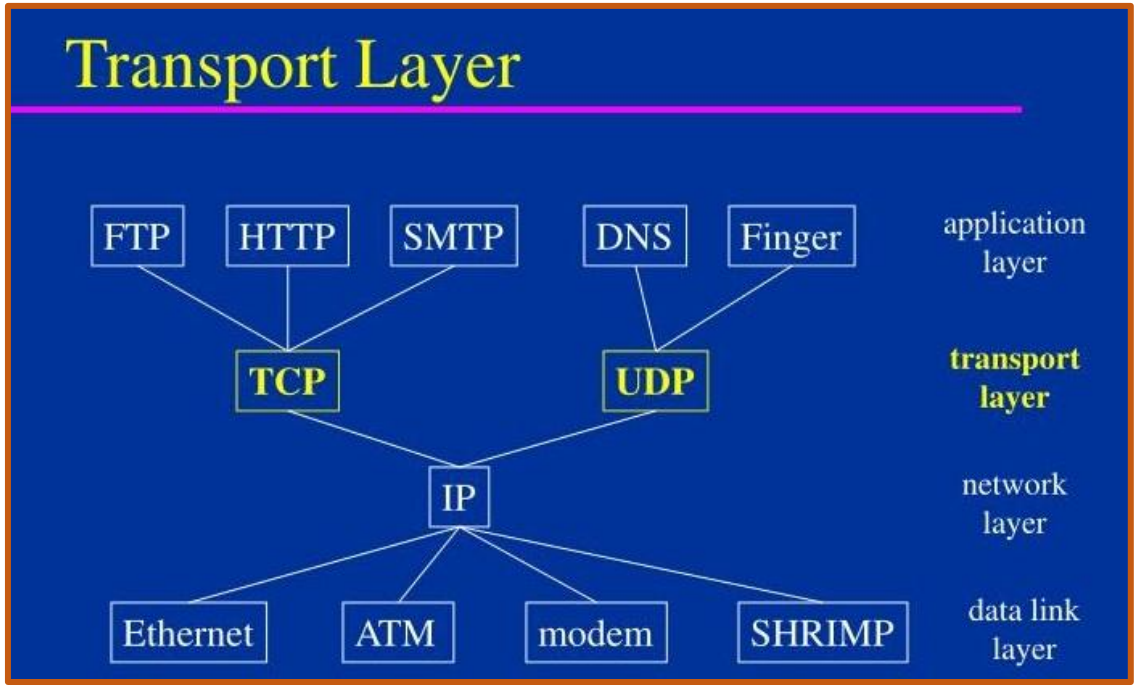
In this segment

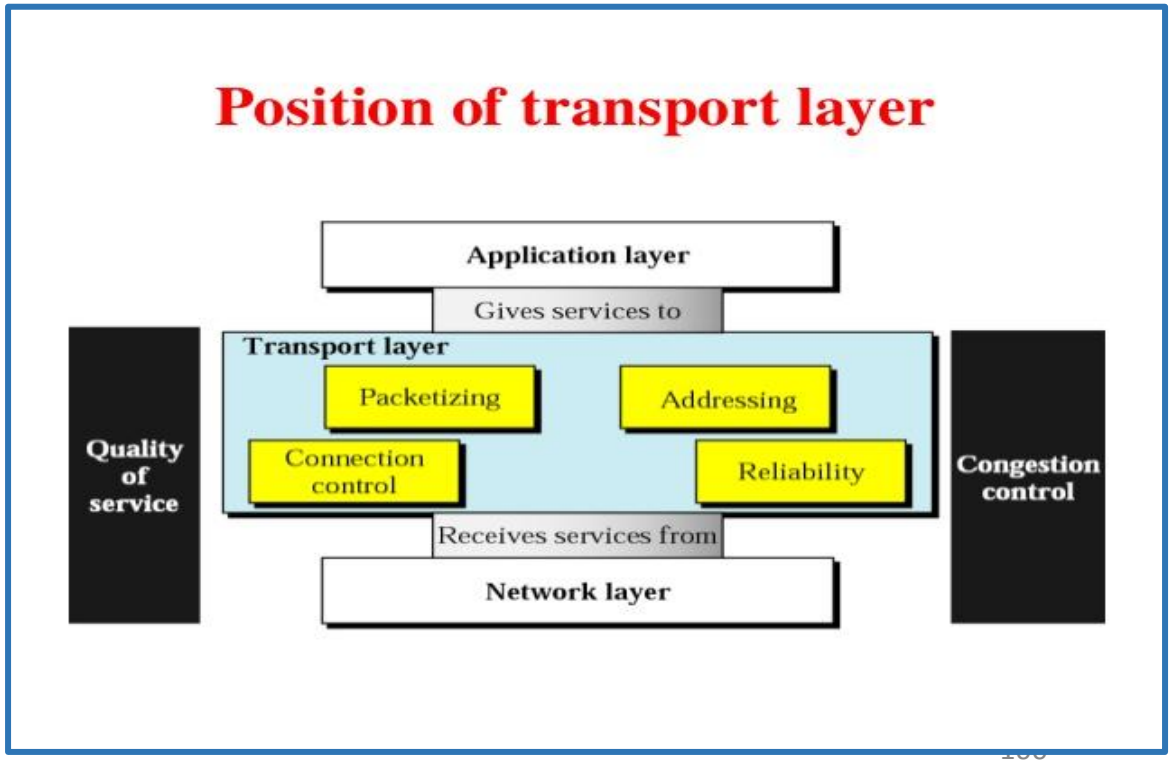
- Transport layer goals
- Transport layer services
- Transport services & protocols
- Transport vs Network layer
- Transport layer actions
- Internet transport layer protocols

Transport Layer in the OSI



Transport Layer





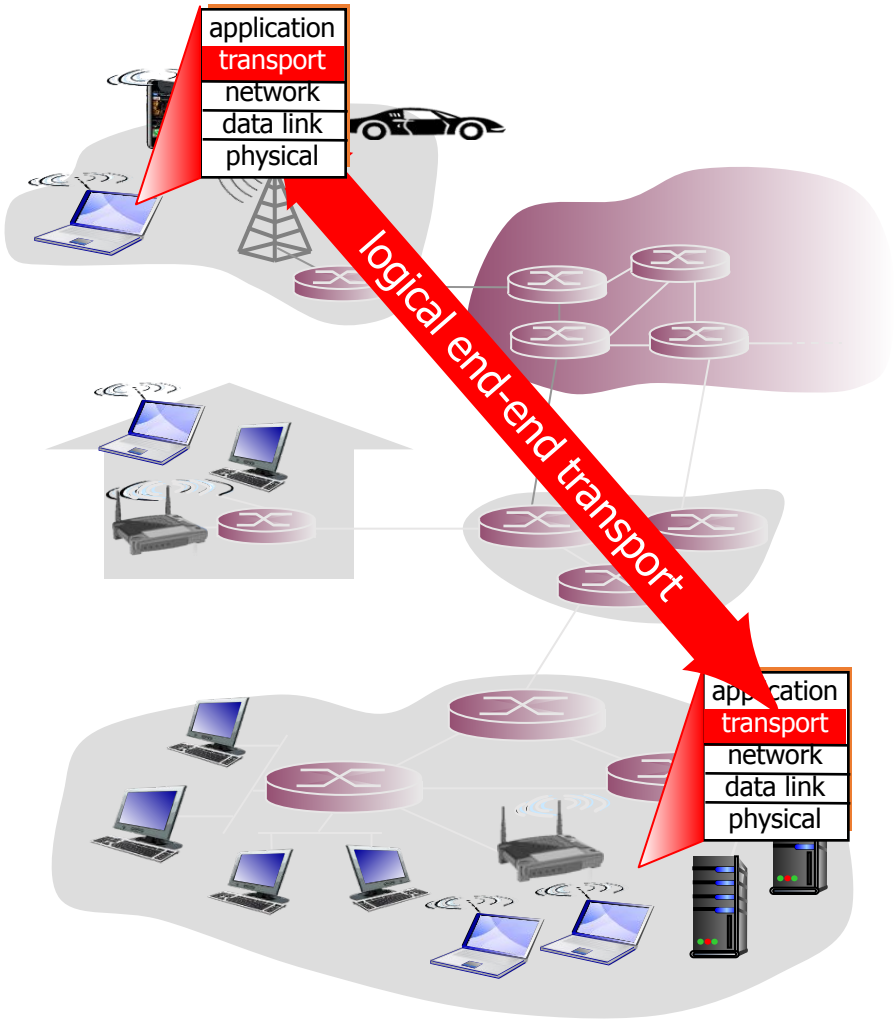
COMPUTER NETWORKS

Transport Layer Services

Compiled by **Kundhavai K R**
Department of Computer Science & Engineering

Transport Services & protocols

- provide *logical communication* between **app processes** running on different hosts.
- transport protocols run in end systems
  - **send side:** breaks app messages into *segments*, passes to network layer
  - **rcv side:** reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport vs. Network Layer

- *Network layer:* logical communication between **hosts**
- *Transport layer:* logical communication between **processes**
  - relies on, enhances, network layer services

Transport: Determines how data is to be sent: Reliably or unreliably. Defines well known services (ports.)

Network: Provides logical addressing, finds best path to a destination.

household analogy:

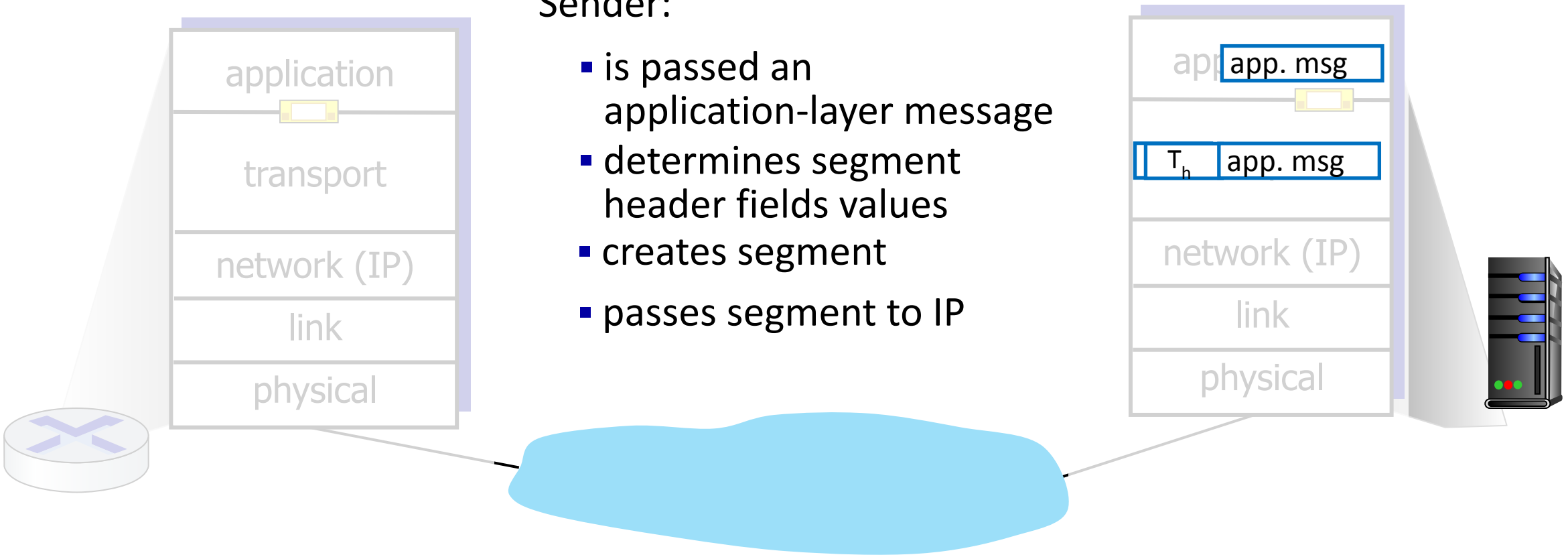
*3 kids in **Ann's house**(East coast) sending letters to 3 kids in **Bill's house**(West Coast):*

- **hosts = houses**
- **processes = kids**
- **app messages = letters in envelopes**
- **transport protocol = Ann and Bill who demux to in-house siblings**
- **network-layer protocol = postal service**

Transport-layer Actions

Sender:

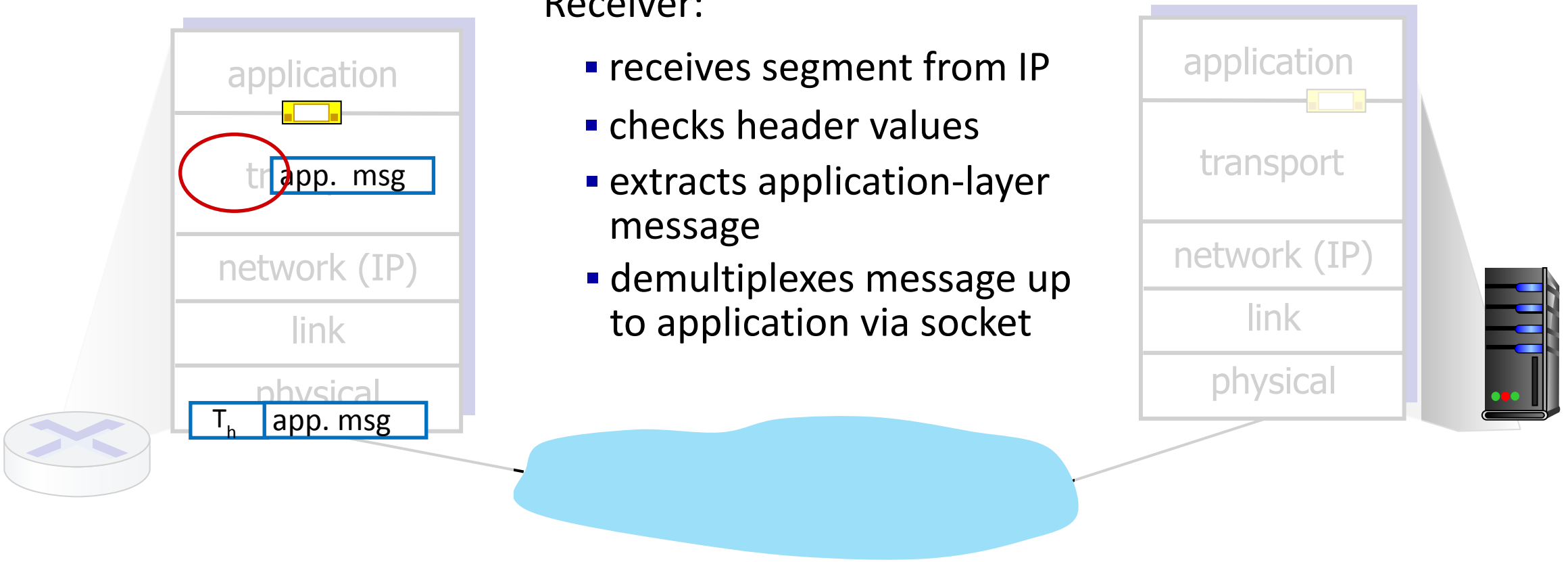
- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to IP



Transport-layer Actions

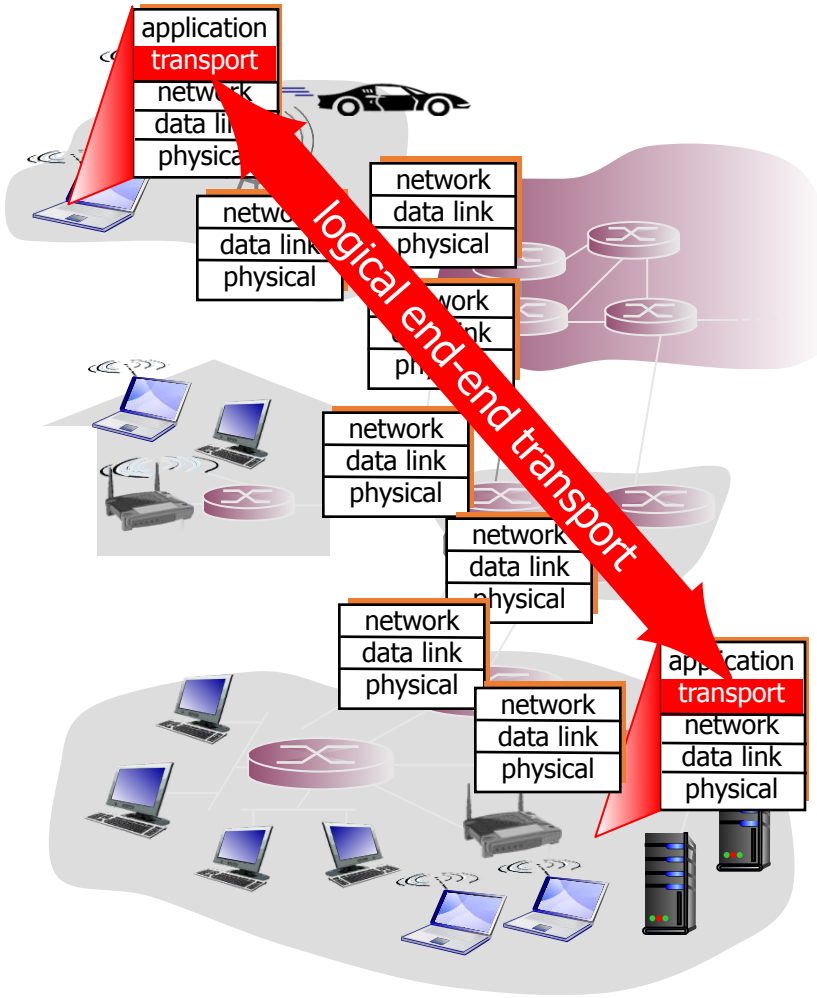
Receiver:

- receives segment from IP
- checks header values
- extracts application-layer message
- demultiplexes message up to application via socket



Internet Transport-layer protocols

- **TCP:** Transmission Control Protocol
  - reliable, connection oriented
  - in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, connectionless
  - unordered delivery
  - extension of “best-effort”
- services not available:
  - delay guarantees
  - bandwidth guarantees



COMPUTER NETWORKS

Transport Layer – Multiplexing and Demultiplexing

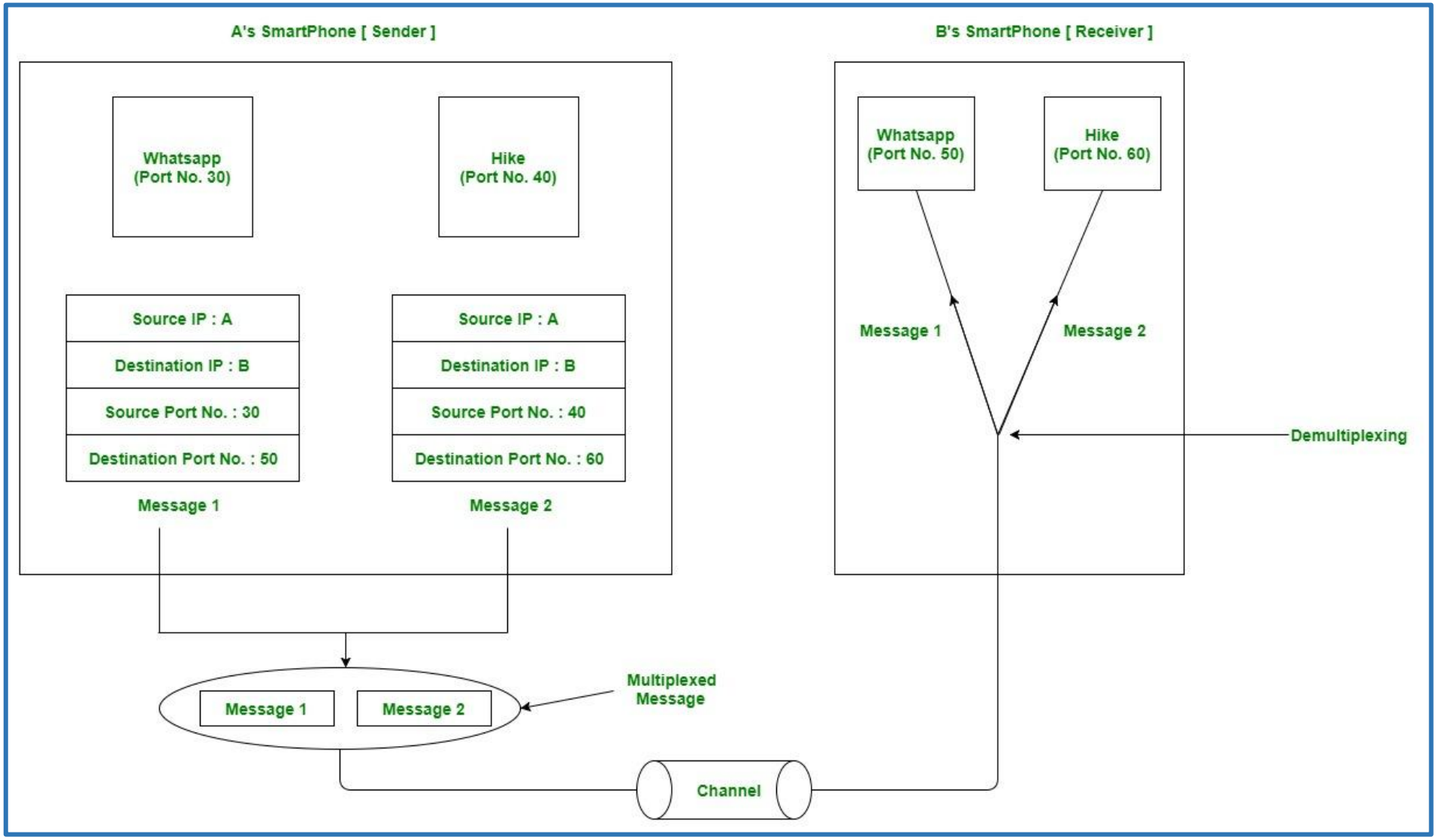
Compiled by **Kundhavai K R**
Department of Computer Science & Engineering

Transport Layer – Mux and Demux Intro

- **Multiplexing -** At the sender site, there may be many processes that want to send their packets.
- TCP at transport layer accepts all these messages from different sockets differentiated by their port numbers and after adding headers, passes the segment to the network layer.
- This process is termed as Multiplexing.

- **Demultiplexing -** At the receiver site, the TCP at transport layer accepts datagrams from network layer, removes IP header, checks their port numbers and then delivers the datagrams to the respective sockets and ultimately to the attached processes.
- This process is referred to as Demultiplexing.

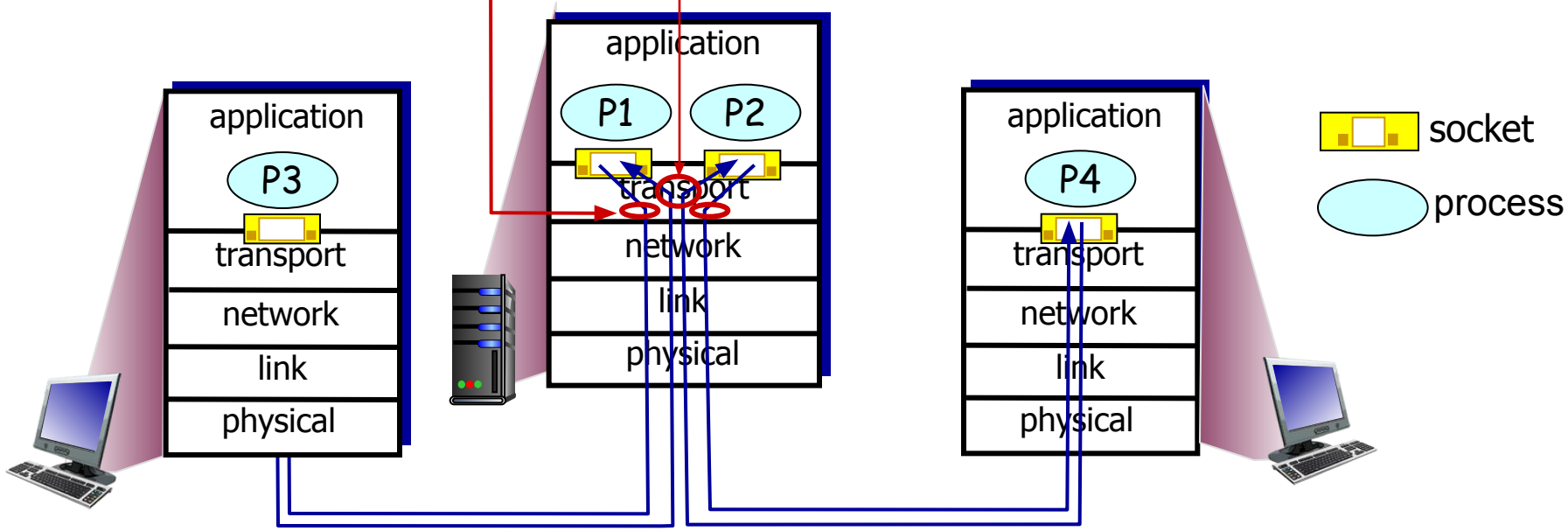
Transport Layer – Mux and Demux



Multiplexing / demultiplexing

multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:
use header info to deliver received segments to correct socket

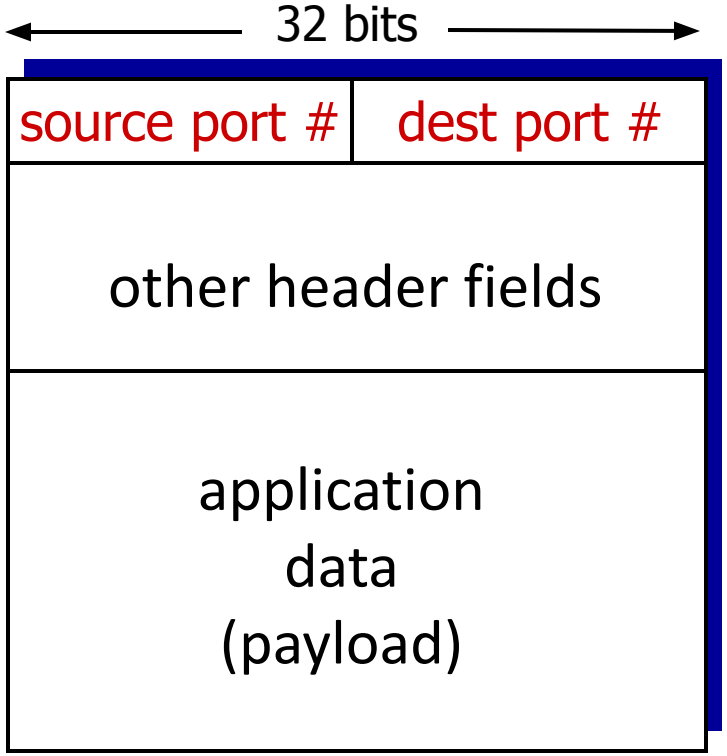


Transport Layer – Mux and Demux

- Transport-layer multiplexing requires
  - (1) that sockets have unique identifiers, and
  - (2) that each segment have special fields that indicate the socket to which the segment is to be delivered.
- These special fields are the **source port number field** and the **destination port number field.**
- Each port number is a **16-bit number**, ranging from **0 to 65535.**
- The port numbers ranging from **0 to 1023** are called **well-known port numbers and are restricted,** which means that they are reserved for use by well-known application protocols such as HTTP (which uses port number 80) and FTP (which uses port number 21).

How demultiplexing works

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



TCP/UDP segment format

In demultiplexing there are **2** types:

Connectionless Demultiplexing: requires the **" Destination IP Address"** and **"destination port #"** . It requires only that two -tuple to identify where to send the datagram.

Connection-Oriented Demultiplexing: is only TCP viable and requires a four-tuple, meaning in order for the datagram to know where to be sent it needs (**source IP address, source port number, destination IP address, destination port number**).

Connectionless demultiplexing

- *recall:* created socket has host-local port #:

```
DatagramSocket mySocket1
= new DatagramSocket(12534);
```

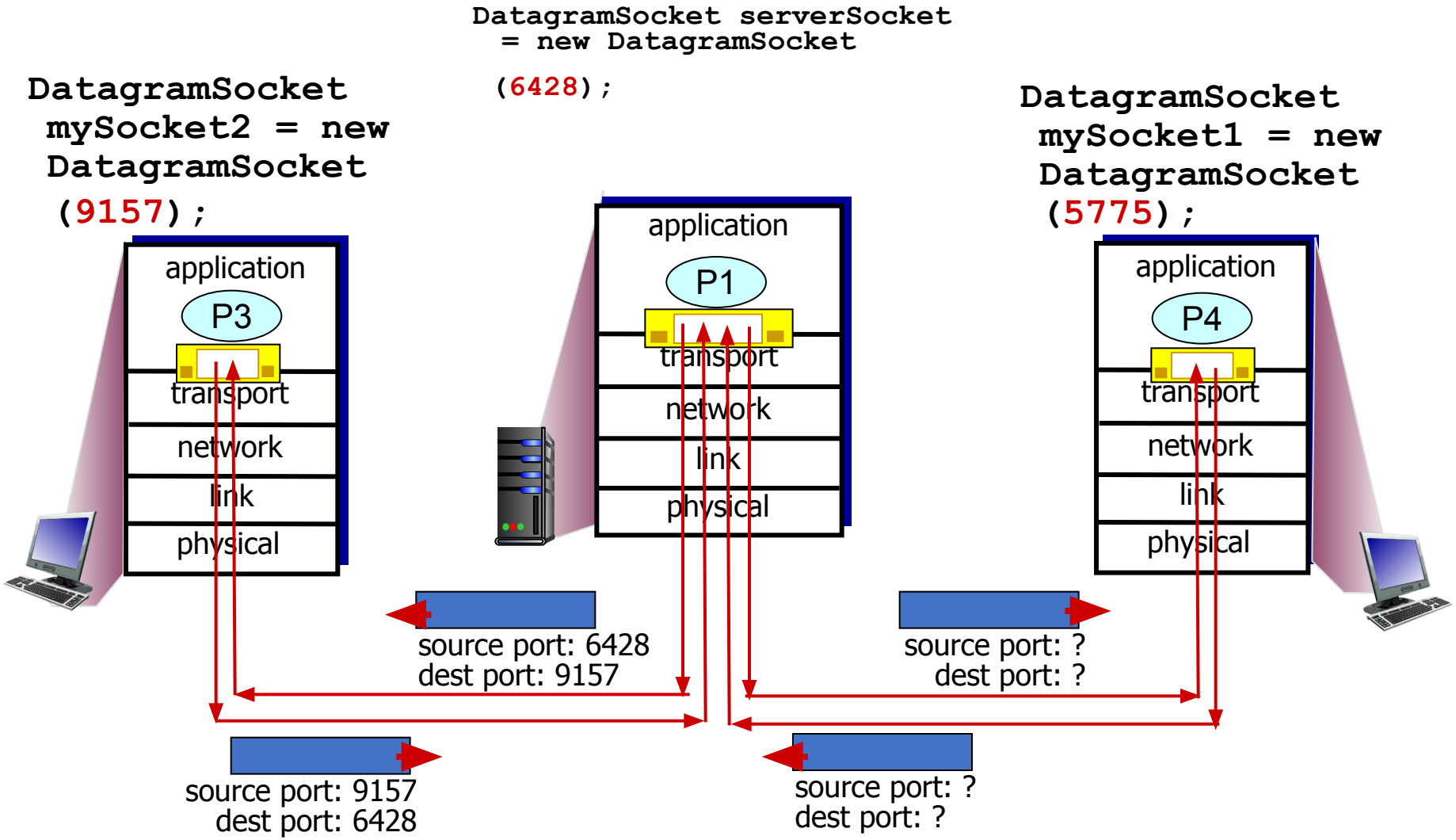
- *recall:* when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

- when host receives UDP segment:
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

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

COMPUTER NETWORKS

Connectionless demux: example

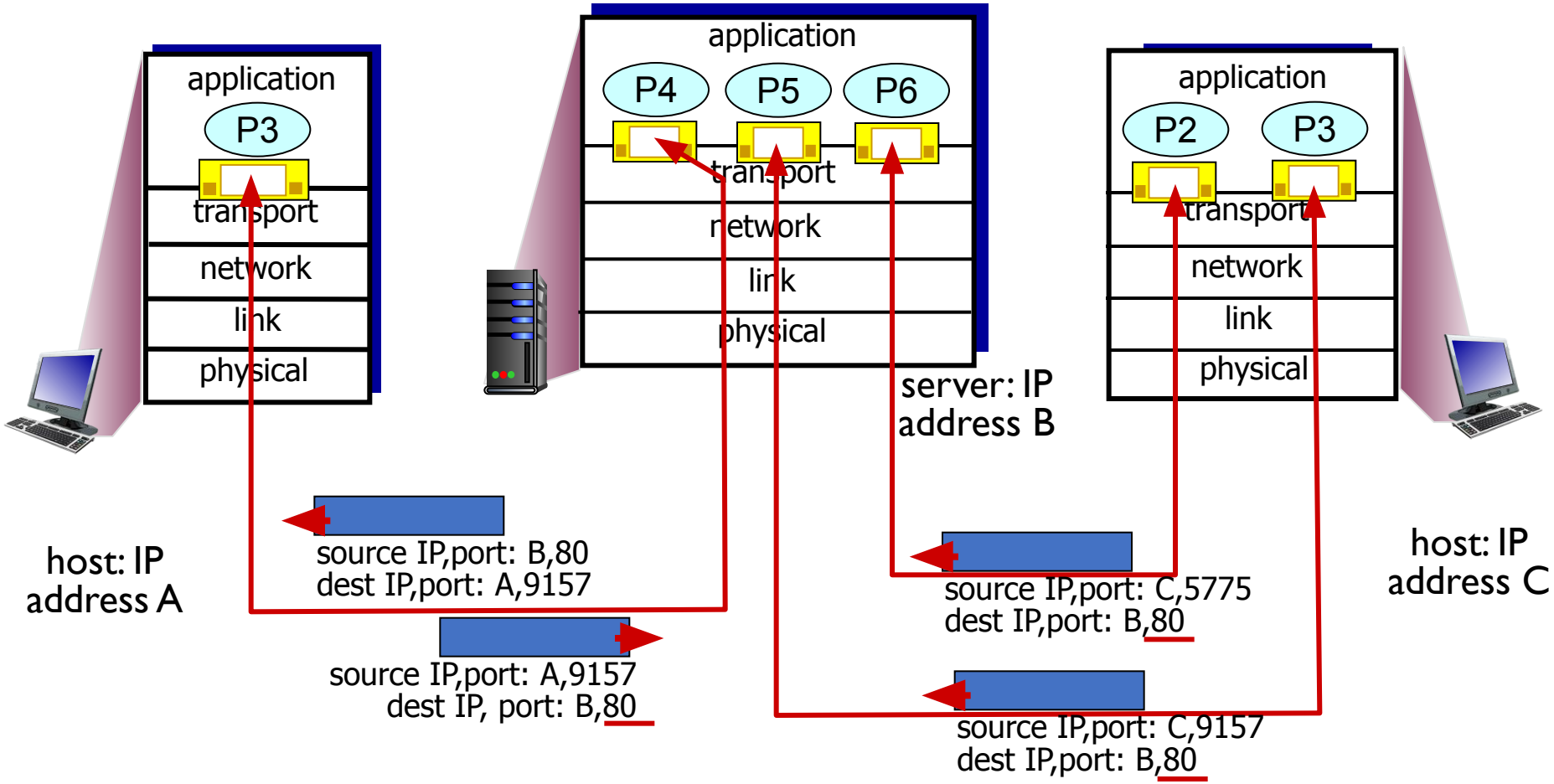


Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses all four values (4-tuple) to direct segment to appropriate socket

- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Connection-oriented demux : example



three segments, all destined to IP address: B, dest port: 80 are demultiplexed to *different* sockets

COMPUTER NETWORKS

Connectionless Transport: UDP

Compiled by **Kundhavai K R**
Department of Computer Science & Engineering

In this segment

- UDP: User Datagram Protocol [RFC 768]
- UDP: segment header
- UDP Checksum
  - Internet Checksum: example
  - Internet Checksum: weak protection!

UDP : INTRODUCTION

- **UDP (User Datagram Protocol)** is a communications protocol that is primarily used for establishing low-latency and loss-tolerating connections between applications on the internet.
- It **speeds up transmissions** by enabling the transfer of data before an agreement is provided by the receiving party.
- As a result, UDP is beneficial in time-sensitive communications, including **voice over Internet Protocol (VoIP), domain name system (DNS) lookup, and video or audio playback.**

- UDP sends messages, called ***datagrams***, and is considered a **best-effort mode of communications** -- meaning the service does not provide any guarantees that the data will be delivered or offer special features to retransmit lost or corrupted messages.

UDP: User Datagram Protocol [RFC 768]

- "best effort" service
- UDP segments may be:
  - lost
  - delivered out-of-order to app
- ***connectionless:***
  - no handshaking between UDP sender, receiver

Why UDP rather than TCP?

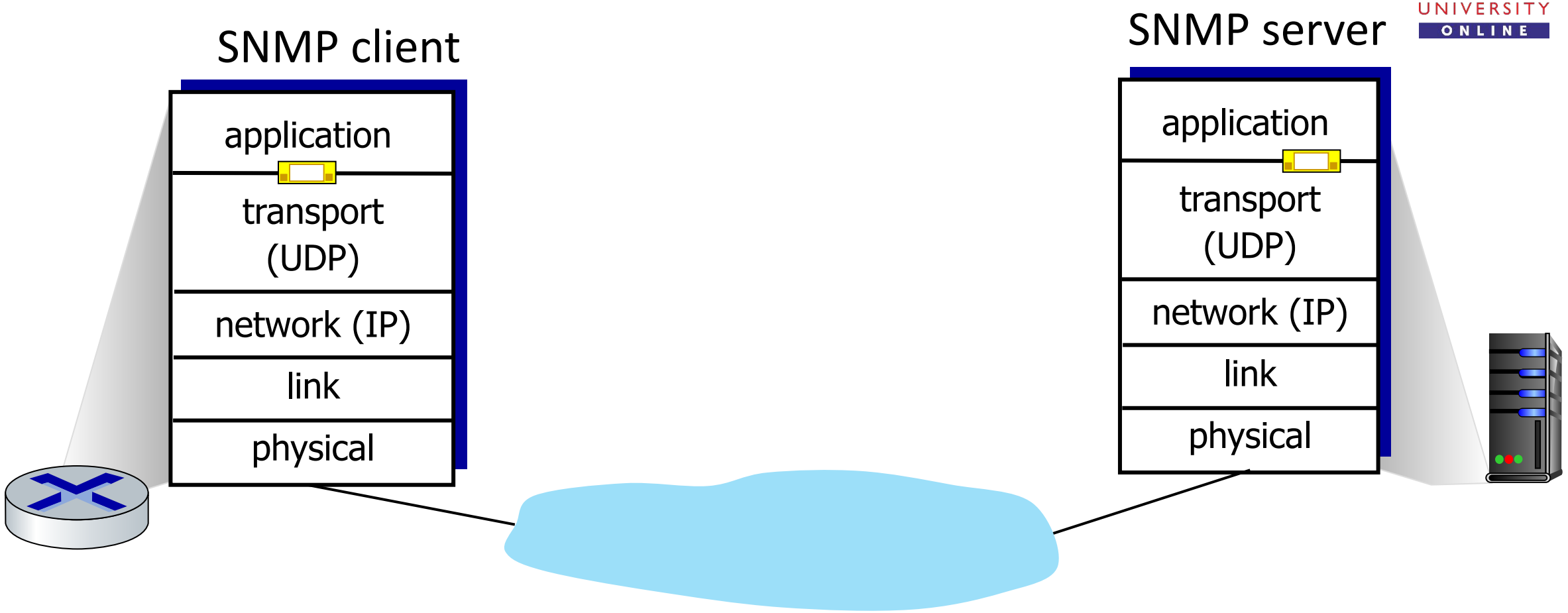
- **No connection establishment** (which can add delay)
- **Simple:** no connection state at sender, receiver
- **Small header size**
  - **UDP – 8 bytes.**
  - TCP – 20 bytes.
- **No congestion control**
  - UDP can blast away as fast as desired
  - can function in the face of congestion

UDP: User Datagram Protocol [RFC 768]

- UDP use:
  - **It can be used where a large number of clients are connected and where real-time error correction isn't necessary, such as gaming, voice or video conferencing, and streaming media.**
  - **DNS (Domain Name Server)**
  - **SNMP (Simple Network Management Protocol)**
  - **Trivial File Transfer Protocol (TFTP)**
  - **Real Time Streaming Protocol (RTSP)**

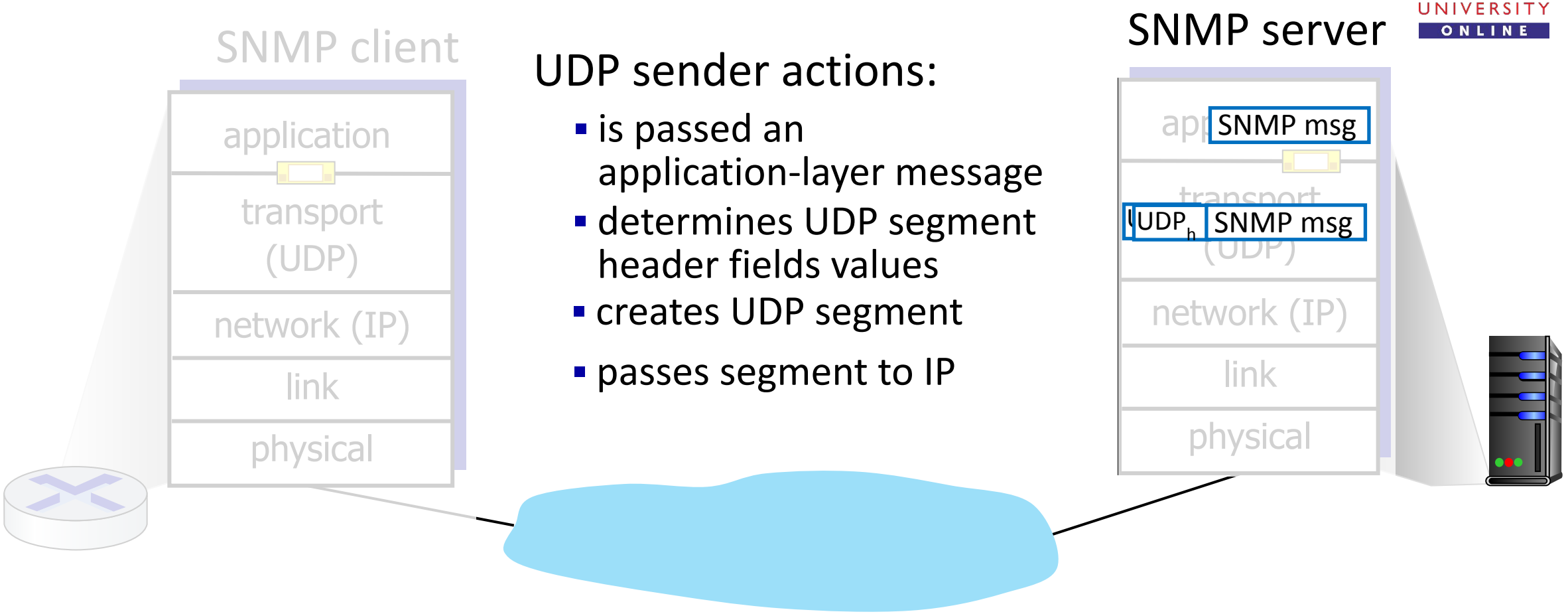
- If reliable transfer needed over UDP:
  - **add needed reliability at application layer**
  - **application-specific error recovery!**
  - **add congestion control at application layer**

UDP: Transport Layer Actions



COMPUTER NETWORKS

UDP: Transport Layer Actions



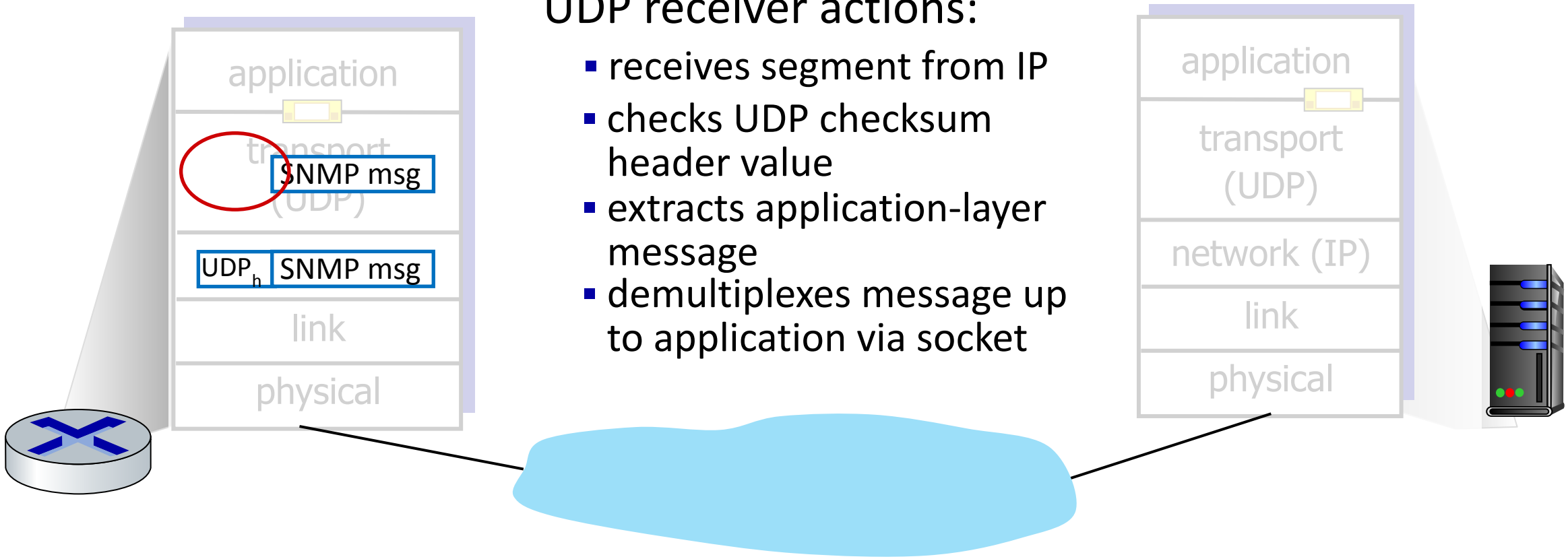
UDP sender actions:

- is passed an application-layer message
- determines UDP segment header fields values
- creates UDP segment
- passes segment to IP

UDP: Transport Layer Actions

SNMP client

SNMP server



UDP receiver actions:

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

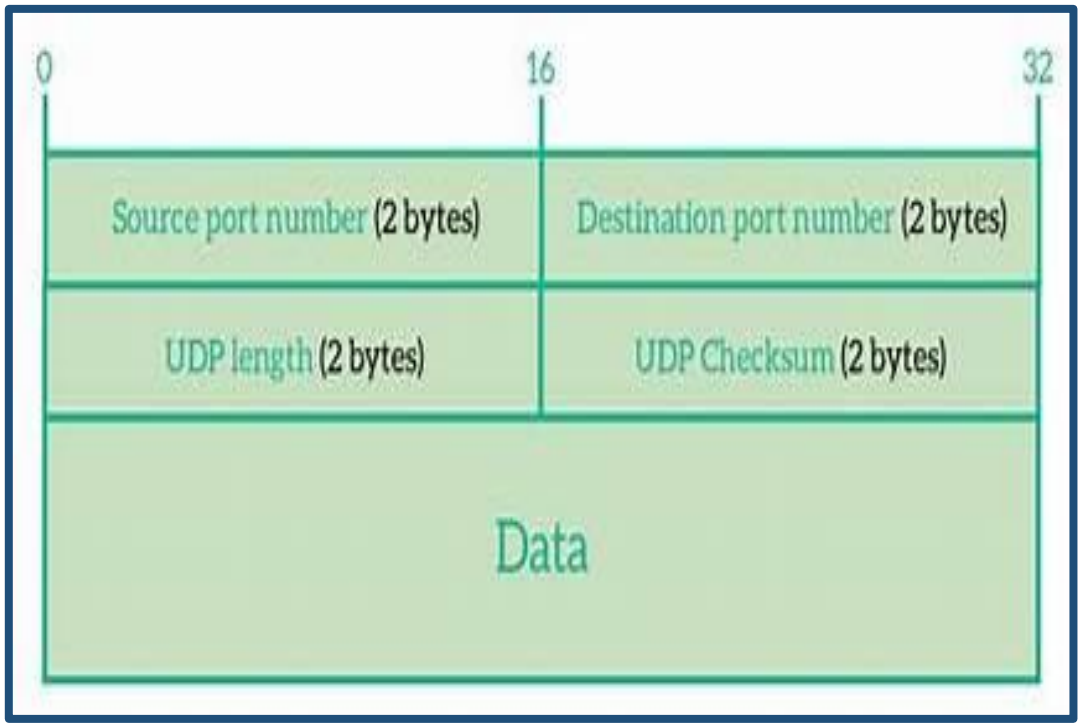
UDP SEGMENT HEADER

UDP: Segment Header

- UDP headers contain a set of parameters -- called ***fields*** -- defined by the technical specifications of the protocol.
- The User Datagram Protocol header has four fields, each of which is **2 bytes.**

They are the following:

- **Source port number:** which is the port number of the sender;
- **Destination port number**: the port the datagram is addressed to;
- **Length**: the length in bytes of the UDP header and any encapsulated data; and
- **Checksum**: which is used in error checking



UDP: In a Nutshell!!!!

- It is a connectionless protocol.
- No handshaking
- It is used for VoIP, video streaming, gaming and live broadcasts.
- It is faster and needs fewer resources.
- The packets don't necessarily arrive in order.
- It allows missing packets -- the sender is unable to know whether a packet has been received.
- It is better suited for applications that need fast, efficient transmission, such as games.

UDP CHECKSUM

UDP: Checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted segment

SENDER:

1. Breaks the original message into “K” blocks with “N” bits in each blocks. (sequence of 16-bit integers)
2. Sum all the “K” data blocks.
3. Add the carry to the sum, if any.
4. Do 1’s complement to the sum ---- > Checksum => “N” bits.
5. Sender puts checksum value into UDP checksum field

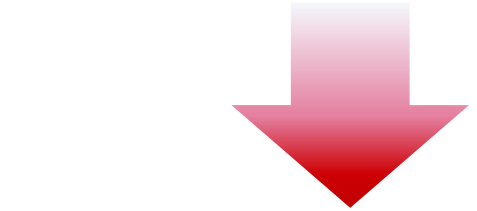
RECEIVER:

1. compute checksum of received segment
2. check if computed checksum equals checksum field value:
   - NO - error detected
   - YES - no error detected. *But maybe errors nonetheless?*

UDP Checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

	1st number	2nd number	sum
Transmitted:	5	6	11
Received:	4	6	11



receiver-computed checksum ≠ sender-computed checksum (as received)

Internet Checksum: example

example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	(1)	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

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

Internet Checksum: Weak protection!

example: add two 16-bit integers