

# Discussion Session #4

EE450: Computer Networks

Topic: Network Applications

# Some network apps

- ❖ e-mail
- ❖ web
- ❖ instant messaging
- ❖ remote login
- ❖ P2P file sharing
- ❖ multi-user network games
- ❖ streaming stored video (YouTube)
- ❖ voice over IP
- ❖ real-time video conferencing
- ❖ cloud computing
- ❖ ...
- ❖ ...
- ❖

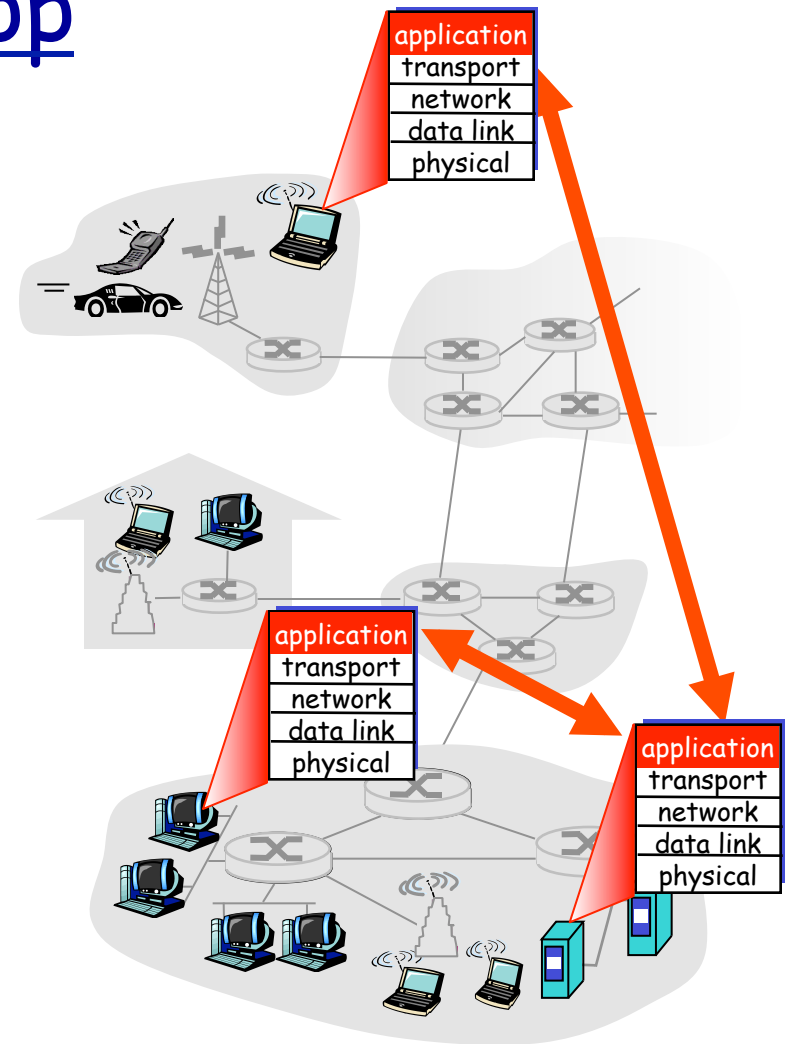
# Creating a network app

## write programs that

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

## No need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



# Processes communicating

**process:** program running within a host.

- ❖ within same host, two processes communicate using **inter-process communication** (defined by OS).
- ❖ processes in different hosts communicate by exchanging **messages**

**client process:** process that initiates communication

**server process:** process that waits to be contacted

- ❖ aside: applications with P2P architectures have client processes & server processes

# Web and HTTP

## First, a review...

- ❖ **web page** consists of **objects**
- ❖ object can be HTML file, JPEG image, Java applet, audio file,...
- ❖ web page consists of **base HTML-file** which includes several referenced objects
- ❖ each object is addressable by a **URL**
- ❖ example URL:

`www.someschool.edu/someDept/pic.gif`

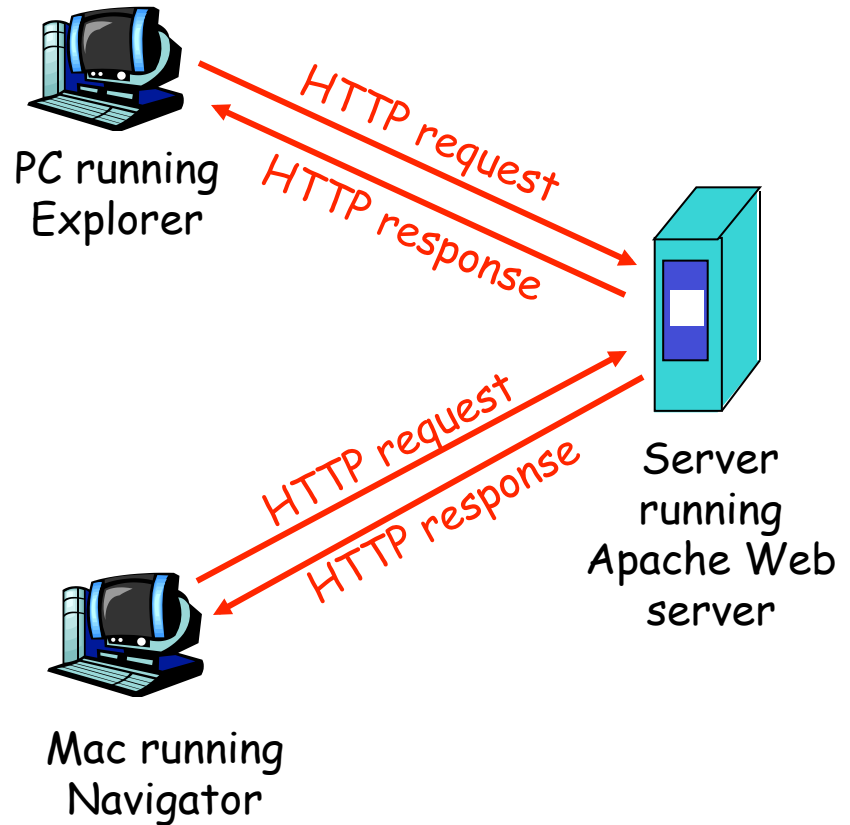
host name

path name

# HTTP overview

## HTTP: hypertext transfer protocol

- ❖ Web's application layer protocol
- ❖ client/server model
  - *client*: browser that requests, receives, "displays" Web objects
  - *server*: Web server sends objects in response to requests



# HTTP overview (continued)

## Uses TCP:

- ❖ client initiates TCP connection (creates socket) to server, port 80
- ❖ server accepts TCP connection from client
- ❖ HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- ❖ TCP connection closed

## HTTP is “stateless”

- ❖ server maintains no information about past client requests

aside  
protocols that maintain  
“state” are complex!

- ❖ past history (state) must be maintained
- ❖ if server/client crashes, their views of “state” may be inconsistent, must be reconciled

# HTTP connections

## non-persistent HTTP

- ❖ at most one object sent over TCP connection.

## persistent HTTP

- ❖ multiple objects can be sent over single TCP connection between client, server.



# Nonpersistent HTTP

suppose user enters URL:

`www.someSchool.edu/someDepartment/home.index`

(contains text,  
references to 10  
jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at `www.someSchool.edu` on port 80

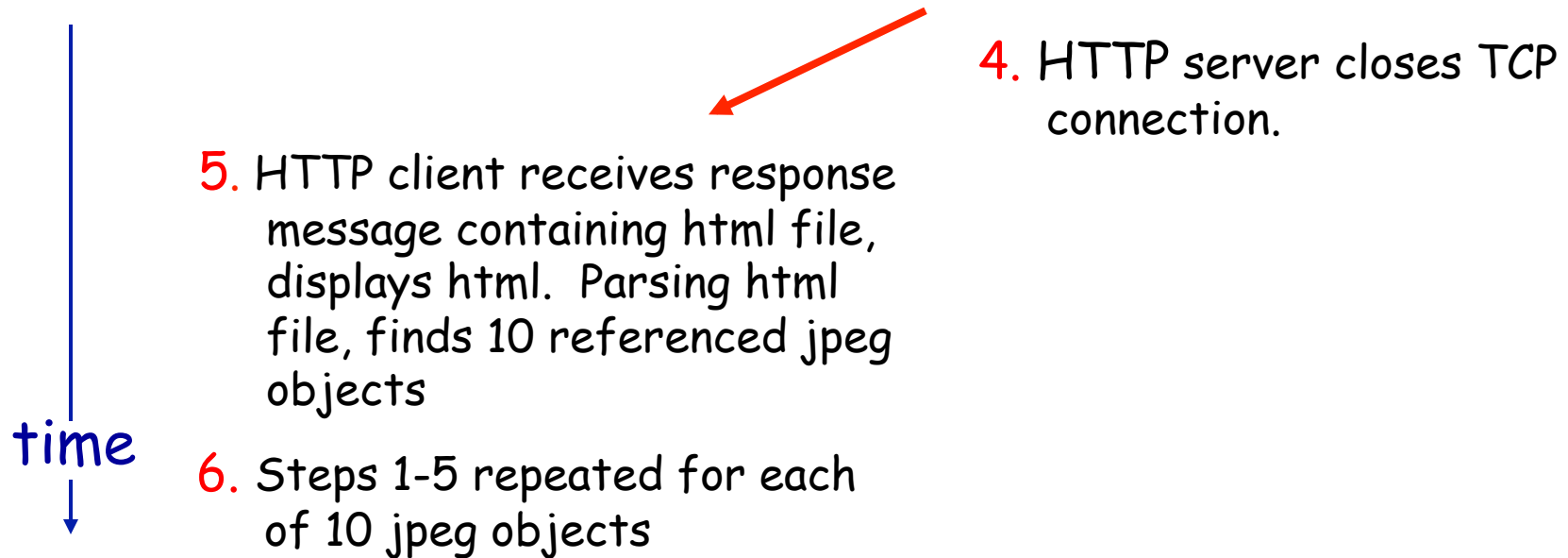
1b. HTTP server at host `www.someSchool.edu` waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object `someDepartment/home.index`

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time  
↓

# Nonpersistent HTTP (cont.)



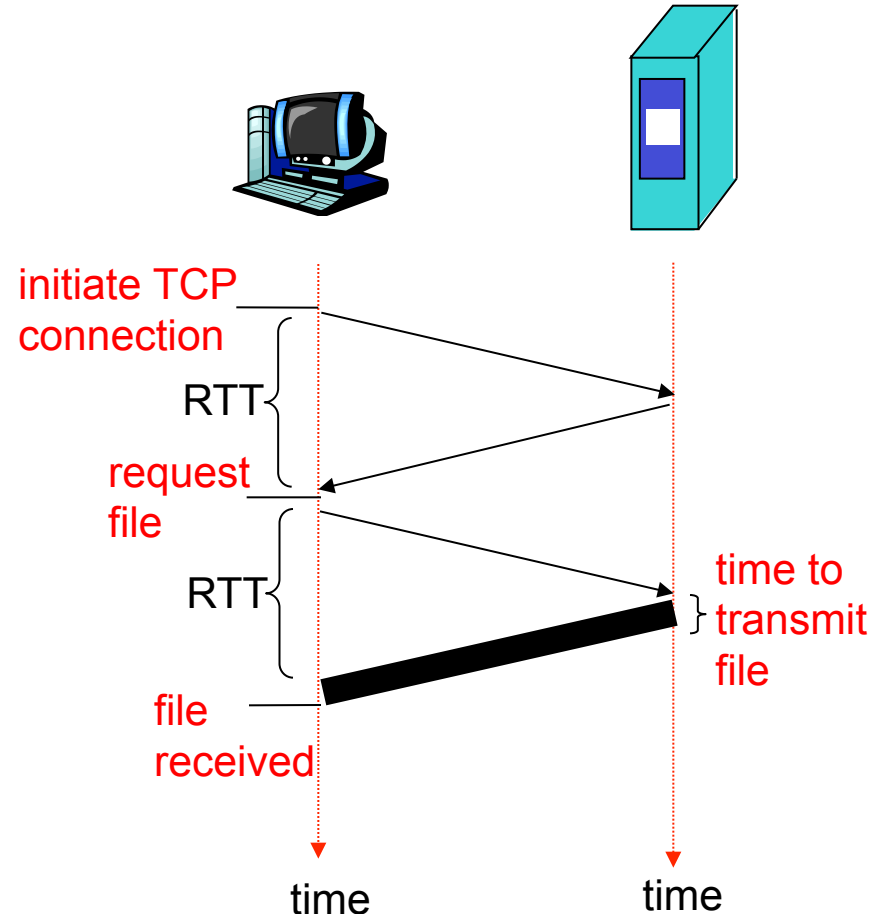
# Non-Persistent HTTP: Response time

**definition of RTT:** time for a small packet to travel from client to server and back.

**response time:**

- ❖ one RTT to initiate TCP connection
- ❖ one RTT for HTTP request and first few bytes of HTTP response to return
- ❖ file transmission time

**total =  $2RTT + \text{transmit time}$**



# Persistent HTTP

## non-persistent HTTP issues:

- ❖ requires 2 RTTs per object
- ❖ OS overhead for *each* TCP connection
- ❖ browsers often open parallel TCP connections to fetch referenced objects

## persistent HTTP

- ❖ server leaves connection open after sending response
- ❖ subsequent HTTP messages between same client/server sent over open connection
- ❖ client sends requests as soon as it encounters a referenced object
- ❖ as little as one RTT for all the referenced objects

# HTTP request message

- ❖ two types of HTTP messages: *request, response*
- ❖ **HTTP request message:**
  - ASCII (human-readable format)

request line  
(GET, POST,  
HEAD commands)

header  
lines

carriage return,  
line feed at start  
of line indicates  
end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

carriage return character  
line-feed character

# HTTP response message

status line  
(protocol  
status code  
status phrase)

header  
lines

data, e.g.,  
requested  
HTML file

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html;\r\n
    charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

# HTTP response status codes

❖ status code appears in 1st line in server->client response message.

❖ some sample codes:

## **200 OK**

- request succeeded, requested object later in this msg

## **301 Moved Permanently**

- requested object moved, new location specified later in this msg (Location:)

## **400 Bad Request**

- request msg not understood by server

## **404 Not Found**

- requested document not found on this server

## **505 HTTP Version Not Supported**

# Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

opens TCP connection to port 80  
(default HTTP server port) at cis.poly.edu.  
anything typed in sent  
to port 80 at cis.poly.edu

2. type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1  
Host: cis.poly.edu
```

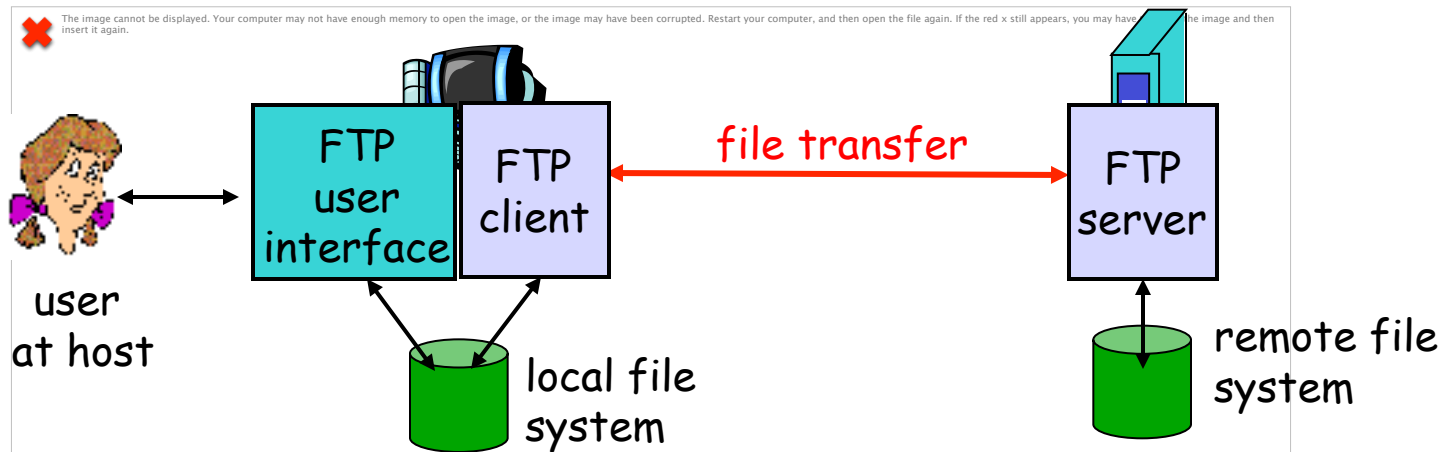
by typing this in (hit carriage  
return twice), you send  
this minimal (but complete)  
GET request to HTTP server

3. look at response message sent by HTTP server!

(or use Wireshark!)



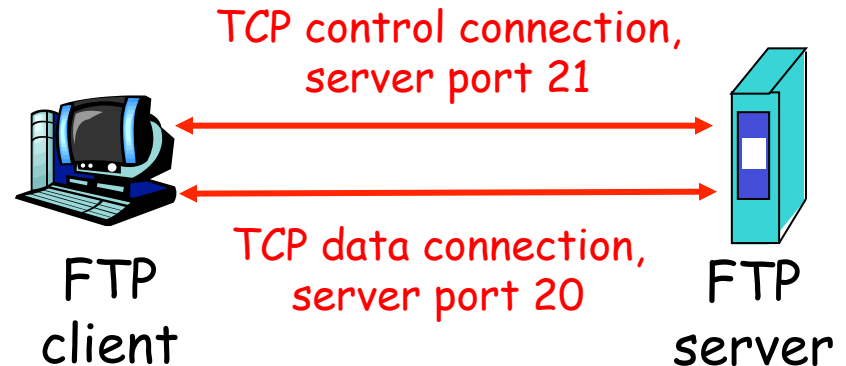
# FTP: the file transfer protocol



- ❖ transfer file to/from remote host
- ❖ client/server model
  - *client*: side that initiates transfer (either to/from remote)
  - *server*: remote host
- ❖ ftp: RFC 959
- ❖ ftp server: port 21

# FTP: separate control, data connections

- ❖ FTP client contacts FTP server at port 21, TCP is transport protocol
- ❖ client authorized over control connection
- ❖ client browses remote directory by sending commands over control connection.
- ❖ when server receives file transfer command, server opens 2<sup>nd</sup> TCP connection (for file) to client
- ❖ after transferring one file, server closes data connection.



- ❖ server opens another TCP data connection to transfer another file.
- ❖ control connection: “out of band”
- ❖ FTP server maintains “state”: current directory, earlier authentication

# FTP commands, responses

## sample commands:

- ❖ sent as ASCII text over control channel
- ❖ USER *username*
- ❖ PASS *password*
- ❖ LIST return list of file in current directory
- ❖ RETR *filename* retrieves (gets) file
- ❖ STOR *filename* stores (puts) file onto remote host

## sample return codes

- ❖ status code and phrase (as in HTTP)
- ❖ 331 Username OK, password required
- ❖ 125 data connection already open; transfer starting
- ❖ 425 Can't open data connection
- ❖ 452 Error writing file

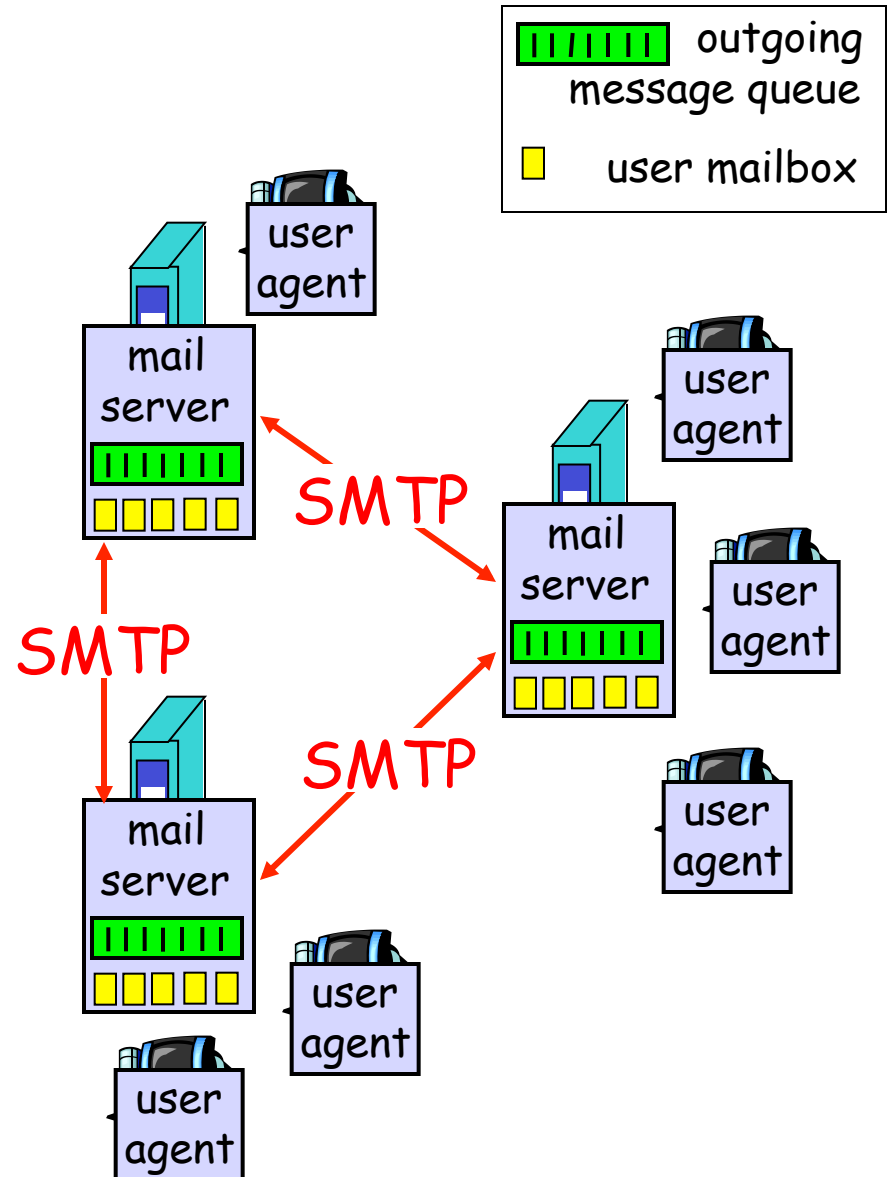
# Electronic Mail

## Three major components:

- ❖ user agents
- ❖ mail servers
- ❖ simple mail transfer protocol: SMTP

## User Agent

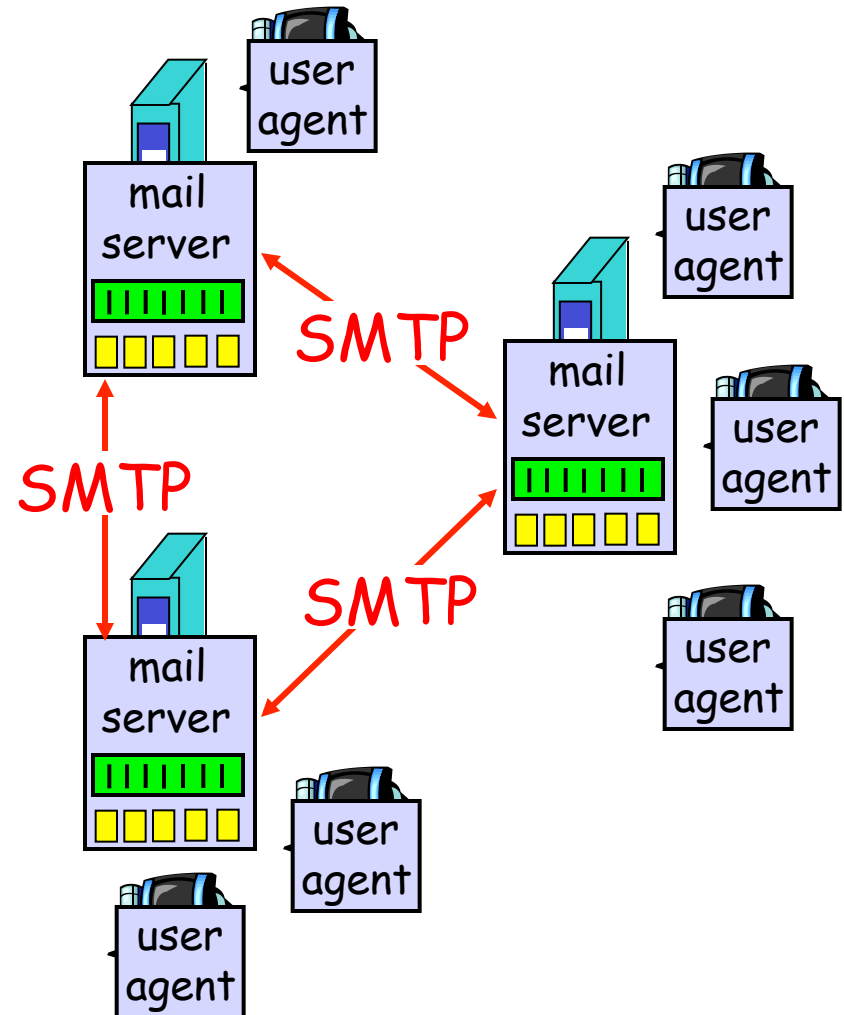
- ❖ a.k.a. “mail reader”
- ❖ composing, editing, reading mail messages
- ❖ e.g., Outlook, elm, Mozilla Thunderbird, iPhone mail client
- ❖ outgoing, incoming messages stored on server



# Electronic Mail: mail servers

## Mail Servers

- ❖ **mailbox** contains incoming messages for user
- ❖ **message queue** of outgoing (to be sent) mail messages
- ❖ **SMTP protocol** between mail servers to send email messages
  - client: sending mail server
  - “server”: receiving mail server

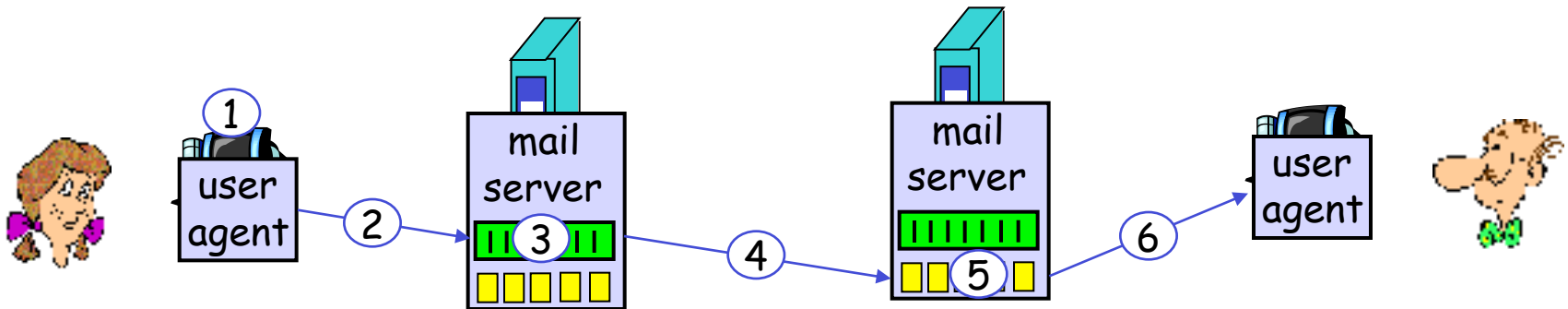


# Electronic Mail: SMTP [RFC 2821]

- ❖ uses TCP to reliably transfer email message from client to server, port 25
- ❖ direct transfer: sending server to receiving server
- ❖ three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- ❖ command/response interaction
  - **commands:** ASCII text
  - **response:** status code and phrase
- ❖ messages must be in 7-bit ASCII

# Scenario: Alice sends message to Bob

- 1) Alice uses UA to compose message and "to"  
`bob@someschool.edu`
- 2) Alice's UA sends message to her mail server; message placed in message queue
- 3) Client side of SMTP opens TCP connection with Bob's mail server
- 4) SMTP client sends Alice's message over the TCP connection
- 5) Bob's mail server places the message in Bob's mailbox
- 6) Bob invokes his user agent to read message



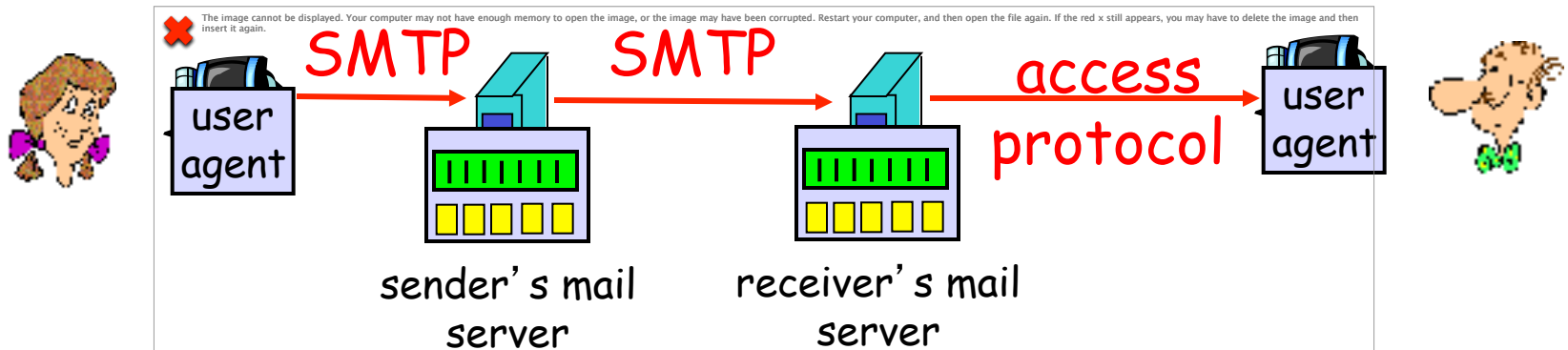
## Try SMTP interaction for yourself:

- ❖ `telnet servername 25`
- ❖ see 220 reply from server
- ❖ enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)



# Mail access protocols



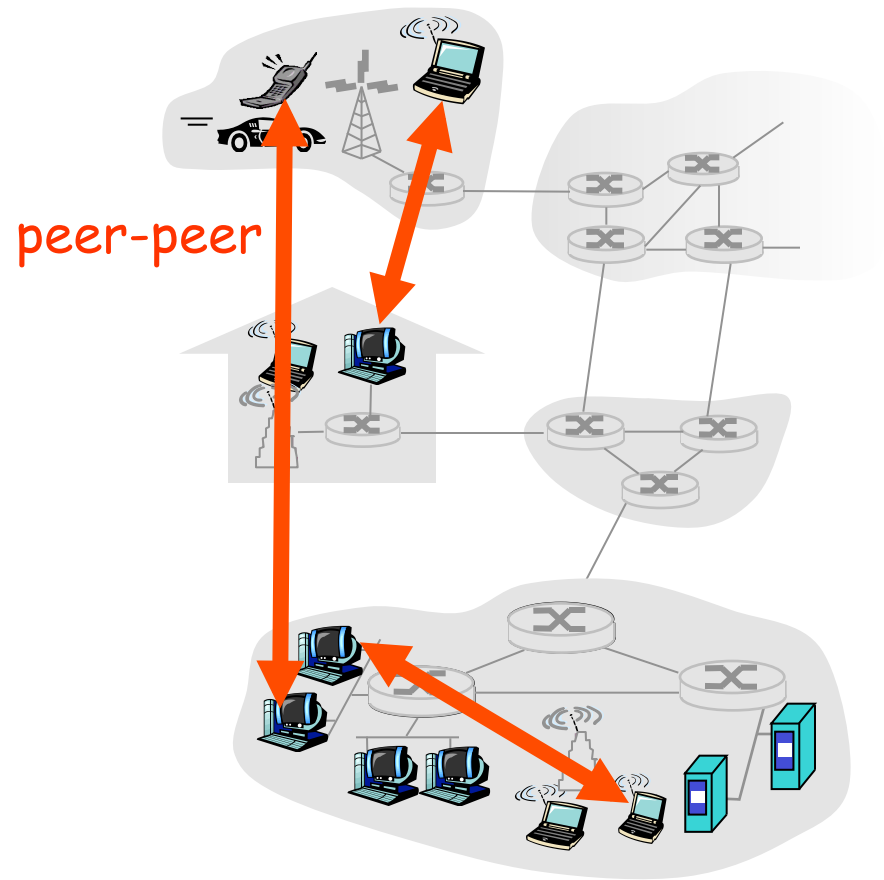
- ❖ SMTP: delivery/storage to receiver's server
- ❖ mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]
    - authorization (agent <-->server) and download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

# Pure P2P architecture

- ❖ *no always-on server*
- ❖ arbitrary end systems directly communicate
- ❖ peers are intermittently connected and change IP addresses

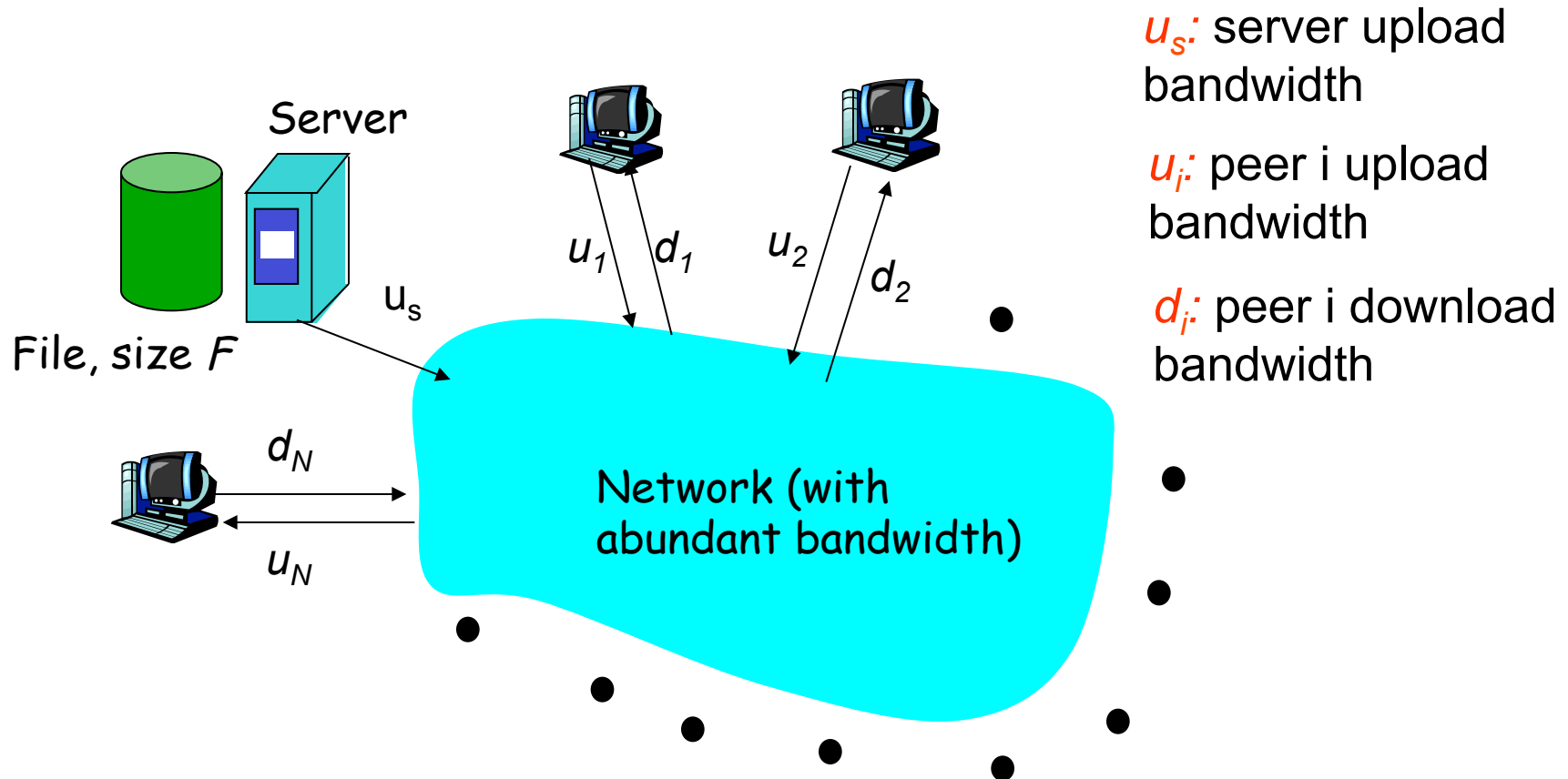
## Three topics:

- file distribution
- searching for information
- case Study: Skype



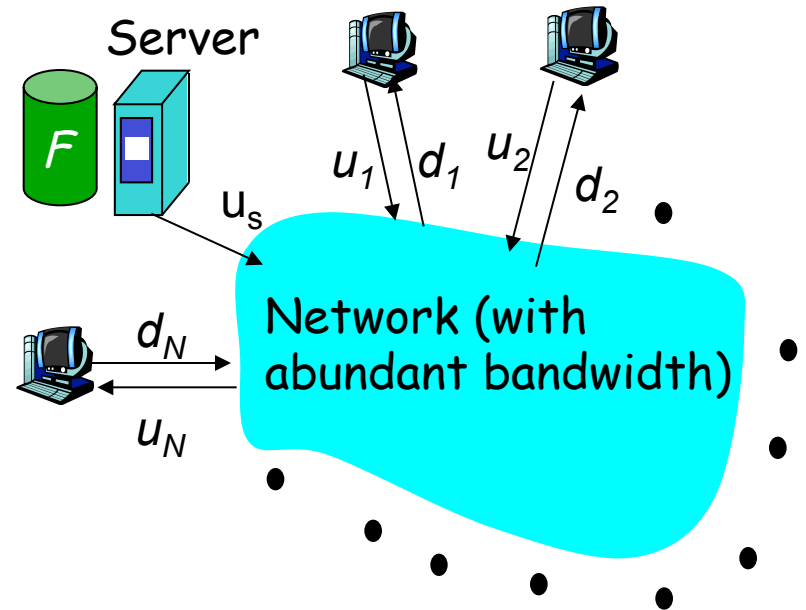
# File Distribution: Server-Client vs P2P

Question: How much time to distribute file from one server to  $N$  peers?



# File distribution time: server-client

- ❖ server sequentially sends  $N$  copies:
  - $NF/u_s$  time
- ❖ client  $i$  takes  $F/d_i$  time to download

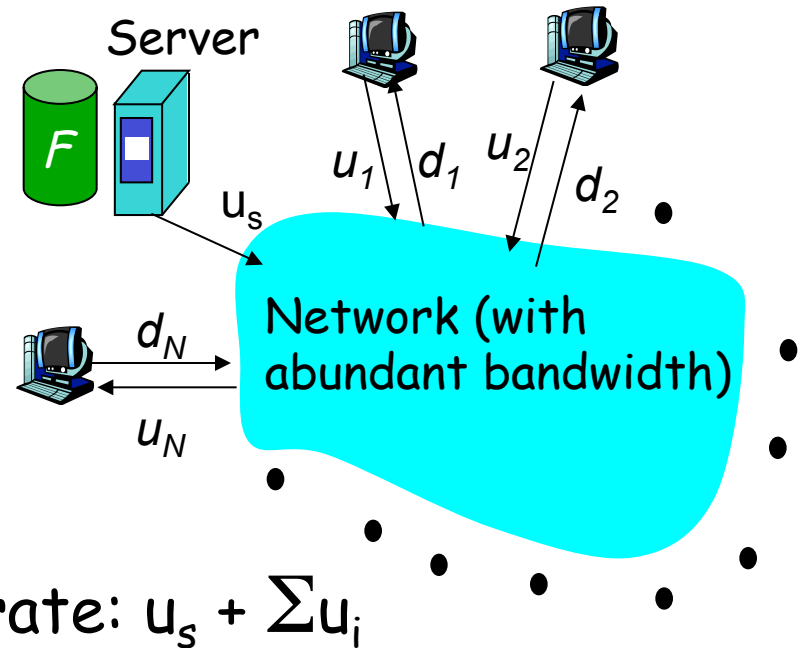


Time to distribute  $F$   
to  $N$  clients using client/server approach  
 $= d_{cs} = \max \{ NF/u_s, F/\min_i(d_i) \}$

increases linearly in  $N$   
(for large  $N$ )

# File distribution time: P2P

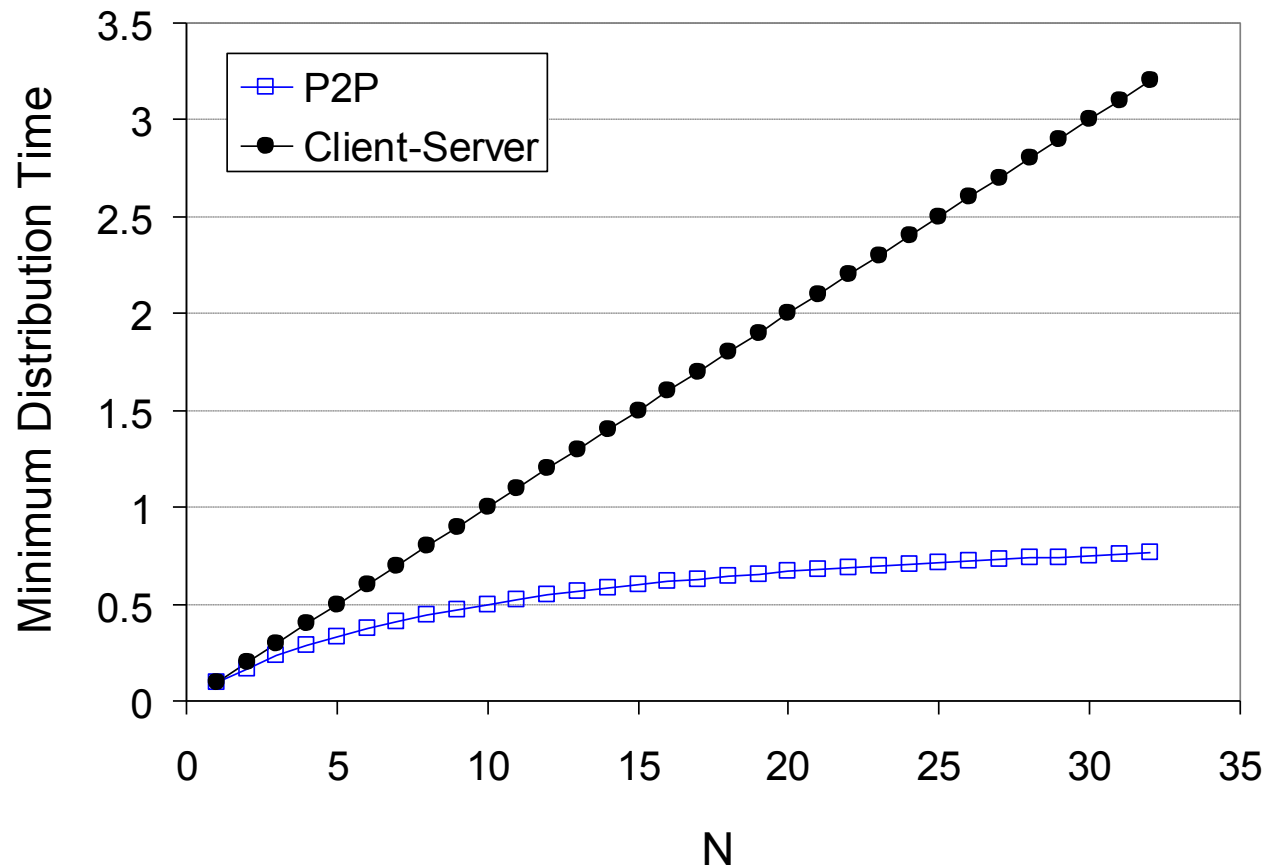
- ❖ server must send one copy:  $F/u_s$  time
- ❖ client  $i$  takes  $F/d_i$  time to download
- ❖  $NF$  bits must be downloaded (aggregate)
  - fastest possible upload rate:  $u_s + \sum u_i$



$$d_{P2P} = \max \left\{ F/u_s, F/\min(d_i)_i, NF/(u_s + \sum u_i) \right\}$$

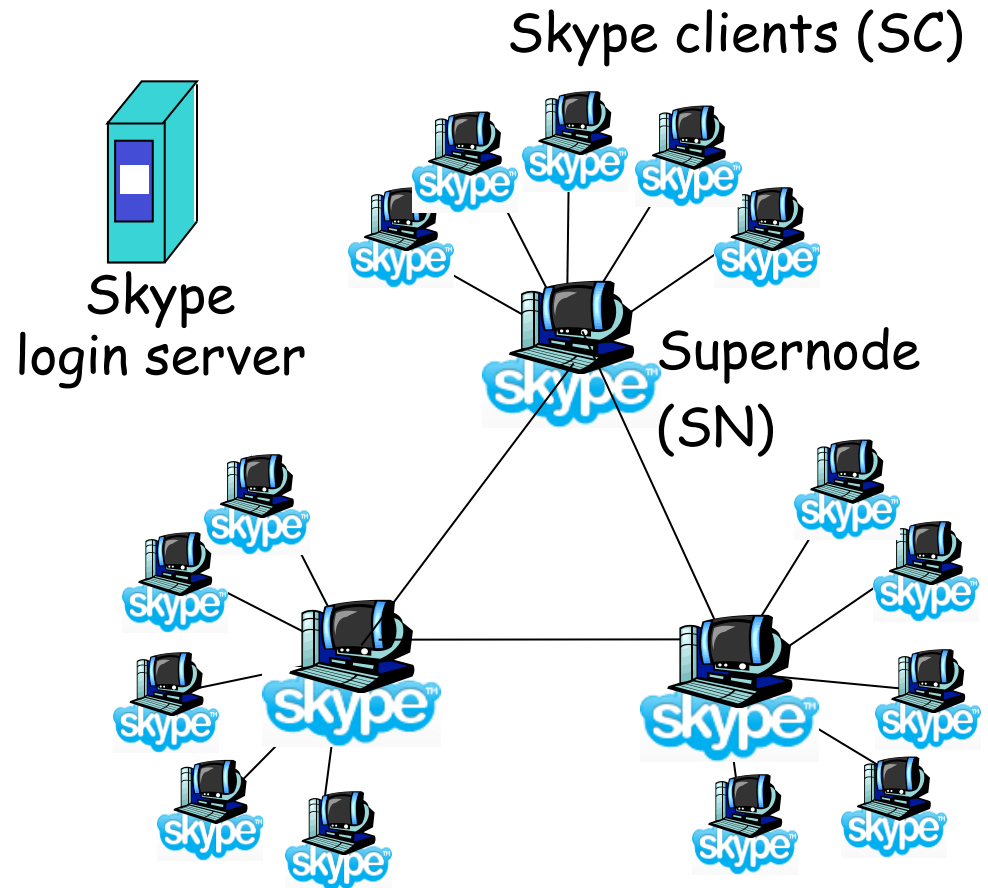
# Server-client vs. P2P: example

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



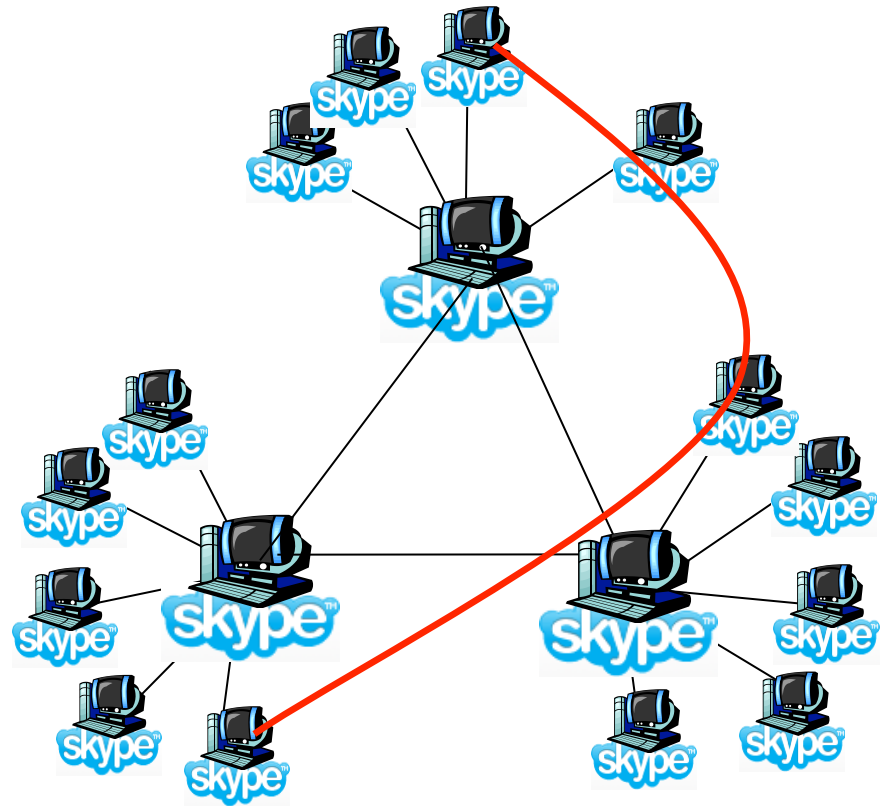
# P2P Case study: Skype

- ❖ inherently P2P: pairs of users communicate.
- ❖ proprietary application-layer protocol (inferred via reverse engineering)
- ❖ hierarchical overlay with SNs
- ❖ Index maps usernames to IP addresses; distributed over SNs



# Peers as relays

- ❖ problem when both Alice and Bob are behind “NATs”.
  - NAT prevents an outside peer from initiating a call to insider peer
- ❖ solution:
  - using Alice's and Bob's SNs, *relay* is chosen
  - each peer initiates session with relay.
  - peers can now communicate through NATs via relay





# A Brief Introduction To



Formerly known as ETHEREAL

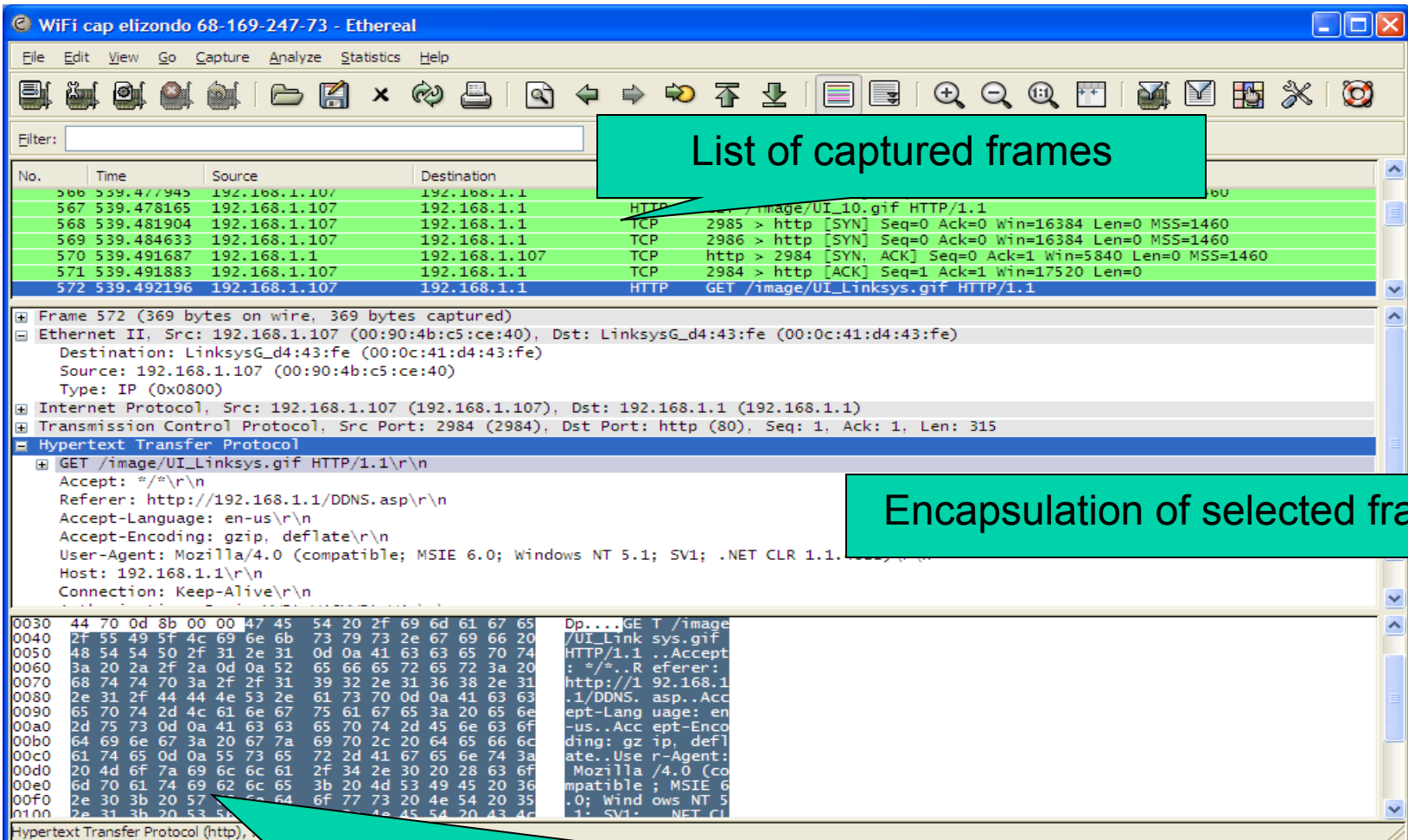
# So Ethereal, Is it?

- ❖ Actually, it's Wireshark to you! 😊
- ❖ A powerful GUI based Network Protocol Analyzer
- ❖ Runs on common o/s platforms: Linux, Unix, Mac, MS Windows
- ❖ Provides the ability to directly analyze network communications on your PC
- ❖ Supports most protocols and media, more than 472 currently
- ❖ It is free!

# What it does:

- ❖ Allows interactive examination of data arriving at, and leaving from, the Network Adapter on your host machine
- ❖ Displays source and destination IP addresses, ports, message types, and message contents
- ❖ Also allows selective filtering of particular frames for specific analysis

## Screen shot of the GUI



## Raw data from Physical (PHY) layer in HEX plus ASCII Text equivalent

- ❖ Screenshot presents the intercepted data in Hexadecimal representation at bottom of screen (Binary would not be efficient for display purposes!)
- ❖ Shows the encapsulation of different layers of the communication: addressing, ports, message type, payload
- ❖ Can expand (decodes or translates) each part of protocol into form meaningful to humans
- ❖ Particular example shows GET image request

# Example of connection to email server

The screenshot displays the Wireshark interface with a capture of network traffic. The packet list shows the following sequence of events:

No.	Time	Source	Destination	Protocol	Info
262	25.234879	192.168.1.102	207.200.89.193	TCP	2816 > http [ACK] Seq=540 Ack=936 Win=64601 Len=0
263	25.249163	192.168.1.102	207.200.89.193	TCP	2816 > http [FIN, ACK] Seq=540 Ack=936 Win=0 Len=0
35	7.786938	192.168.1.102	209.86.93.204	TCP	2806 > pop3 [SYN] Seq=0 Ack=119 Len=0
53	8.903792	192.168.1.102	209.86.93.204	TCP	2806 > pop3 [ACK] Seq=1 Ack=119 Len=0
56	9.005546	192.168.1.102	209.86.93.204	POP	Request: AUTH
64	9.086873	192.168.1.102	209.86.93.204	POP	Request: USER bartlett
67	9.304570	192.168.1.102	209.86.93.204	TCP	2806 > pop3 [ACK] Seq=22 Ack=119 Len=0

The packet details pane for frame 64 shows the following information:

- Frame 64 (69 bytes on wire, 69 bytes captured)
- Ethernet II, Src: ArimaCom\_12:94:26 (00:03:25:12:94:26), Dst: 192.168.1.1 (00:06:25:85:88:b3)
- Internet Protocol, Src: 192.168.1.102 (192.168.1.102), Dst: 209.86.93.204 (209.86.93.204)
- Transmission Control Protocol, Src Port: 2806 (2806), Dst Port: pop3 (110), Seq: 7, Ack: 119, Len: 15
- Source port: 2806 (2806)
- Destination port: pop3 (110)
- Sequence number: 7 (relative sequence number)
- [Next sequence number: 22 (relative sequence number)]
- Acknowledgement number: 119 (relative ack number)
- Header length: 20 bytes
- Flags: 0x0018 (PSH, ACK)
- Window size: 65417
- Checksum: 0x3a10 [correct]
- [SEQ/ACK analysis]
- Post Office Protocol
- USER bartlett\r\n
- Request: USER
- Request Arg: bartlett

The packet bytes pane shows the raw data of the USER request:

```
0000 00 06 25 85 88 b3 00 03 25 12 94 26 08 00 45 00  ..%....%...&..E.
0010 00 37 02 0a 40 00 80 06 07 86 c0 a8 01 66 d1 56  ]7..@... ..f.V
0020 5d cc 0a f6 00 6e a8 3e 45 77 a9 10 6e 5f 50 18  ]....n.> Ew..n_P.
0030 ff 89 3a 10 00 00 55 53 45 52 20 62 61 72 74 6c  ....US ER bartl
0040 65 74 74 0d 0a  ..ett..
```

IP address of Earthlinks' mail server 209.86.93.204

Assigned ports on mail server (110) and on my PC (2806)

# Email server connection comments

- ❖ The frame (#64 in this capture) shows that it is 69 bytes in length
- ❖ This particular frame is part of a connection to an email server using Post-Office-Protocol
- ❖ This is a request to the mail server identifying my user name
- ❖ You can see the destination port on the mail server is 110 (a good example of a well known port number), and the port opened on my machine is 2806
- ❖ Notice the Checksum field and Window size

# Example of Address Resolution Protocol event (housekeeping)

The image shows a Wireshark packet capture window titled "WiFi cap elizondo 68-169-247-73 - Ethereal". The packet list shows several packets, with packet 7 selected. The packet details pane shows the following information:

- Frame 7 (42 bytes on wire, 42 bytes captured)
- Ethernet II, Src: 192.168.1.107 (00:90:4b:c5:ce:40), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  - Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  - Source: 192.168.1.107 (00:90:4b:c5:ce:40)
  - Type: ARP (0x0806)
- Address Resolution Protocol (request)
  - Hardware type: Ethernet (0x0001)
  - Protocol type: IP (0x0800)
  - Hardware size: 6
  - Protocol size: 4
  - Opcode: request (0x0001)
  - Sender MAC address: 192.168.1.107 (00:90:4b:c5:ce:40)
  - Sender IP address: 192.168.1.107 (192.168.1.107)
  - Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
  - Target IP address: 192.168.1.1 (192.168.1.1)

Two callouts highlight specific information:

- A green callout bubble points to the "Destination: Broadcast (ff:ff:ff:ff:ff:ff)" field, containing the text: "Destination: ff.ff.ff.ff.ff.ff".
- A green callout bubble points to the "Sender MAC address" and "Sender IP address" fields, containing the text: "My machines' MAC and IP addresses".

The packet bytes pane at the bottom shows the raw data in hexadecimal and ASCII:

```
0000 ff ff ff ff ff ff 00 90 4b c5 ce 40 08 06 00 01 ..... K..@...
0010 08 00 06 04 00 01 00 90 4b c5 ce 40 c0 a8 01 6b ..... K..@...k
0020 00 00 00 00 00 00 c0 a8 01 01 ..... ..
```



# Details of ARP event

An example of how the network finds out MAC addresses

In the first slide, my machine is trying to find the MAC address of 192.168.1.1 (default router)

So it sends a “WHO HAS” broadcast message to MAC address ff.ff.ff.ff.ff.ff

Part of the message frame contains my machine's MAC address, hopefully for the default router to reply to

# ARP continuation

The image shows a Wireshark packet capture window titled "WiFi cap elizondo 68-169-247-73 - Ethereal". The packet list shows several ARP requests (frames 5-7) and one ARP reply (frame 8). The packet details pane for frame 8 shows the Ethernet II header, the ARP protocol type, and the ARP reply details. A speech bubble points to the ARP reply details, stating: "Reply from default router to my machine identifying its MAC address".

No.	Time	Source	Destination	Protocol	Info
5	68.912066	192.168.1.107	Broadcast	ARP	who has 192.168.1.1? Tell 192.168.1.107
6	71.488894	192.168.1.107	Broadcast	ARP	who has 192.168.1.1? Tell 192.168.1.107
7	73.824964	192.168.1.107	Broadcast	ARP	who has 192.168.1.1? Tell 192.168.1.107
8	73.828949	LinksysG_d4:43:fe	192.168.1.107	ARP	192.168.1.1 is at 00:0c:41:d4:43:fe
9	73.828963	192.168.1.107	24.48.217.226	DNS	Standard query PTR 107.1.168.192.in-addr.arpa
10	73.858094	24.48.217.226	192.168.1.107	DNS	Standard query response, No such name

Frame 8 (42 bytes on wire, 42 bytes captured)

- Ethernet II, Src: LinksysG\_d4:43:fe (00:0c:41:d4:43:fe), Dst: 192.168.1.107 (00:90:4b:c5:ce:40)
  - Destination: 192.168.1.107 (00:90:4b:c5:ce:40)
  - Source: LinksysG\_d4:43:fe (00:0c:41:d4:43:fe)
  - Type: ARP (0x0806)
- Address Resolution Protocol (reply)
  - Hardware type: Ethernet (0x0001)
  - Protocol type: IP (0x0800)
  - Hardware size: 6
  - Protocol size: 4
  - Opcode: reply (0x0002)
  - Sender MAC address: LinksysG\_d4:43:fe (00:0c:41:d4:43:fe)
  - Sender IP address: 192.168.1.1 (192.168.1.1)
  - Target MAC address: 192.168.1.107 (00:90:4b:c5:ce:40)
  - Target IP address: 192.168.1.107 (192.168.1.107)

0000 00 90 4b c5 ce 40 00 0c 41 d4 43 fe 08 06 00 01 ..K..@.. A  
0010 08 00 06 04 00 02 00 0c 41 d4 43 fe c0 a8 01 01 ..... A  
0020 00 90 4b c5 ce 40 c0 a8 01 6b ..... I

# ARP continued..

- ❖ The very next frame is the reply from the router telling my machine of the router's MAC address
- ❖ From this point onwards, both MAC addresses are known and therefore frames can be transferred by the Data Link Layer
- ❖ This shows how new nodes joining a network gain membership through Address Resolution Protocol

# How Ethereal works

- ❖ Puts the Network Adapter into Promiscuous mode
- ❖ Forces interface not to drop any frames
- ❖ Thus it allows all frames to be captured...
- ❖ And viewed
- ❖ If this particular machine was on a simple switch then all traffic to / from other machines would also be visible...

# Some more useful features

- ❖ Can capture traffic to a file for later analysis and filtering
- ❖ Useful when trying to debug or trace problems with networks or find malware
- ❖ Can filter traffic by IP and port number etc
- ❖ Can perform statistical analysis of captured frames

# Example of WiFi traffic

The screenshot displays the Wireshark network protocol analyzer interface. The title bar reads "WiFi cap elizondo 68-169-247-73 - Ethereal". The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, and Help. The toolbar contains various icons for file operations, capture control, and analysis. The filter bar is empty. The packet list pane shows a table of captured packets:

No.	Time	Source	Destination	Protocol	Info
357	533.467387	192.168.1.107	192.168.1.1	TCP	[TCP Dup ACK 355#1] 2968 > http [ACK] Seq=459 Ack=246 Win=17275 Len=0
358	533.540476	192.168.1.1	192.168.1.107	HTTP	[TCP Previous segment lost] Continuation or non-HTTP traffic
359	533.540564	192.168.1.107	192.168.1.1	TCP	[TCP Dup ACK 355#2] 2968 > http [ACK] Seq=459 Ack=246 Win=17275 Len=0
360	533.594258	192.168.1.1	192.168.1.107	HTTP	[TCP Retransmission] Continuation or non-HTTP traffic
361	533.594476	192.168.1.107	192.168.1.1	TCP	2968 > http [ACK] Seq=459 Ack=3166 Win=17520 Len=0 SLE=4626 SRE=6086
362	534.205849	192.168.1.107	192.168.1.1	TCP	2969 > http [SYN] Seq=0 Ack=0 Win=16384 Len=0 MSS=1460
363	534.207735	192.168.1.107	192.168.1.1	TCP	2970 > http [SYN] Seq=0 Ack=0 Win=16384 Len=0 MSS=1460

The packet details pane for packet 360 shows the following information:

- Transmission Control Protocol, Src Port: http (80), Dst Port: 2968 (2968), Seq: 246, Ack: 459, Len: 1460
- Source port: http (80)
- Destination port: 2968 (2968)
- Sequence number: 246 (relative sequence number)
- [Next sequence number: 1706 (relative sequence number)]
- Acknowledgement number: 459 (relative ack number)
- Header length: 20 bytes
- Flags: 0x0010 (ACK)
- Window size: 6432
- Checksum: 0xf983 [correct]
- [SEQ/ACK analysis]
- [TCP Analysis Flags]
- [This frame is a (suspected) retransmission]
- [The RTO for this segment was: 0.053782000 seconds]
- [RTO based on delta from frame: 358]
- Hypertext Transfer Protocol

The packet bytes pane shows the raw data in hexadecimal and ASCII format:

```
0000 00 90 4b c5 ce 40 00 0c 41 d4 43 fe 08 00 45 00 ..K..@.. A.C...E.
0010 05 dc e5 e7 40 00 06 cb 77 c0 a8 01 01 c0 a8 ....@.@. .w.....
0020 01 6b 00 50 0b 98 fa f0 27 f4 b0 d0 41 61 50 10 .k.P.... '...AaP.
0030 19 20 f9 83 00 00 3c 6d 65 74 61 20 68 74 74 70 . .<meta http
0040 2d 65 71 75 69 76 3d 22 63 61 63 68 65 2d 63 6f -equiv=" cache-co
0050 6e 74 72 6f 6c 22 20 63 6f 6e 74 65 6e 74 3d 22 ntrol" c ontent="
0060 6e 6f 2d 63 61 63 68 65 22 3e 0a 3c 6d 65 74 61 no-cache ">.<meta
0070 20 68 74 74 70 2d 65 71 75 69 76 3d 22 70 72 61 http-eq uiv="pra
0080 67 6d 61 22 20 63 6f 6e 74 65 6e 74 3d 22 6e 6f gma" con tent="no
0090 2d 63 61 63 68 65 22 3e 0d 0a 3c 4d 45 54 41 20 -cache"> ..<META
00a0 68 74 74 70 2d 65 71 75 69 76 3d 43 6f 6e 74 65 http-equ iv=Conte
00b0 6e 74 2d 54 79 70 65 20 63 6f 6e 74 65 6e 74 3d nt-Type content=
00c0 22 74 65 78 74 2f 68 74 6d 6c 3b 20 63 68 61 72 "text/ht ml; char
00d0 73 65 74 3d 69 73 6f 2d 38 38 35 39 2d 31 22 3e set=iso- 8859-1"
```

The status bar at the bottom shows: P: 1938 D: 1938 M: 0

- ❖ The previous slide shows a suspected retransmission of a frame.
- ❖ This particular traffic was captured from a wireless LAN belonging to a neighbor that I am within range of.
- ❖ As the signal strength is rather low, the connection integrity is rather poor leading to data corruption errors - lots of retries.

# PING

- ❖ An example of Internet Control Message Protocol (ICMP)
- ❖ Used to find out if a host is 'reachable'
- ❖ Run from CMD prompt on PC
- ❖ `C:\> ping 209.86.93.204`



# Ping 209.86.93.204

The screenshot displays the Wireshark (Ethereal) interface with a capture of a ping operation. The packet list shows eight packets: four requests and four replies. The selected packet (No. 1) is an ICMP Echo (ping) request from 192.168.1.101 to 209.86.93.204. The packet details pane shows the structure of the ICMP request, including the type (8), code (0), checksum (0x095b), identifier (0x0200), sequence number (0x4201), and 32 bytes of data. The packet bytes pane shows the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.101	209.86.93.204	ICMP	Echo (ping) request
2	0.071560	209.86.93.204	192.168.1.101	ICMP	Echo (ping) reply
3	0.998961	192.168.1.101	209.86.93.204	ICMP	Echo (ping) request
4	1.078235	209.86.93.204	192.168.1.101	ICMP	Echo (ping) reply
5	1.998891	192.168.1.101	209.86.93.204	ICMP	Echo (ping) request
6	2.079614	209.86.93.204	192.168.1.101	ICMP	Echo (ping) reply
7	2.998844	192.168.1.101	209.86.93.204	ICMP	Echo (ping) request
8	3.066248	209.86.93.204	192.168.1.101	ICMP	Echo (ping) reply

**Frame 1 (74 bytes on wire, 74 bytes captured)**  
Ethernet II, Src: 00:07:e9:f7:1c:62, Dst: 00:06:25:85:b3  
Internet Protocol, Src Addr: 192.168.1.101 (192.168.1.101), Dst Addr: 209.86.93.204 (209.86.93.204)  
Internet Control Message Protocol  
Type: 8 (Echo (ping) request)  
Code: 0  
Checksum: 0x095b (correct)  
Identifier: 0x0200  
Sequence number: 0x4201  
Data (32 bytes)

0000 00 06 25 85 b3 00 07 e9 f7 1c 62 08 00 45 00 ..%. .... ..b..E.  
0010 00 3c 39 82 00 00 80 01 10 0f c0 a8 01 65 d1 56 .<9. .... ..e.V  
0020 5d cc 08 00 09 5b 02 00 42 01 61 62 63 64 65 66 ]....[... B.abcdef  
0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 ghijklmn opqrstuv  
0040 77 61 62 63 64 65 66 67 68 69 wabcdefg hi

# Ping reply

The screenshot shows the Wireshark (Ethereal) interface with a packet capture of ICMP Echo (ping) traffic. The packet list at the top shows eight packets, with packet 2 (an ICMP Echo (ping) reply) selected. The packet details pane below shows the structure of this packet, including Ethernet II, Internet Protocol, and Internet Control Message Protocol (ICMP) fields. The packet bytes pane at the bottom displays the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.101	209.86.93.204	ICMP	Echo (ping) request
2	0.071560	209.86.93.204	192.168.1.101	ICMP	Echo (ping) reply
3	0.998961	192.168.1.101	209.86.93.204	ICMP	Echo (ping) request
4	1.078235	209.86.93.204	192.168.1.101	ICMP	Echo (ping) reply
5	1.998891	192.168.1.101	209.86.93.204	ICMP	Echo (ping) request
6	2.079614	209.86.93.204	192.168.1.101	ICMP	Echo (ping) reply
7	2.998844	192.168.1.101	209.86.93.204	ICMP	Echo (ping) request
8	3.066248	209.86.93.204	192.168.1.101	ICMP	Echo (ping) reply

Frame 2 (74 bytes on wire, 74 bytes captured)  
Ethernet II, Src: 00:06:25:85:88:b3, Dst: 00:07:e9:f7:1c:62  
Internet Protocol, Src Addr: 209.86.93.204 (209.86.93.204), Dst Addr: 192.168.1.101 (192.168.1.101)  
Internet Control Message Protocol  
Type: 0 (Echo (ping) reply)  
Code: 0  
Checksum: 0x115b (correct)  
Identifier: 0x0200  
Sequence number: 0x4201  
Data (32 bytes)

```
0000 00 07 e9 f7 1c 62 00 06 25 85 88 b3 08 00 45 00  ....b.. %.....E.  
0010 00 3c 37 91 40 00 f1 01 60 ff d1 56 5d cc c0 a8  :<7.@... ..V]...  
0020 01 65 00 00 11 5b 02 00 42 01 61 62 63 64 65 66  .e...[... B.abcdef  
0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76  ghijklmn opqrstuv  
0040 77 61 62 63 64 65 66 67 68 69                    wabcdefg hi
```

- ❖ First slide shows PING request
- ❖ Sends 32 bytes of data `abcdefghijklmnopghijkl...`
- ❖ Recipient responds back with data
- ❖ Provides measurement of loopback time
- ❖ Can also be used to illustrate DNS operation - see next slide

# PING www.csun.edu

ping-csun - Ethereal

File Edit View Go Capture Analyze Statistics Help

No.	Time	Source	Destination	Protocol	Info
37	37.392129	192.168.1.101	24.48.217.227	DNS	Standard query A www.csun.edu
38	37.411786	24.48.217.227	192.168.1.101	DNS	Standard query response A 130.166.246.41
39	37.418594	192.168.1.101	130.166.246.41	ICMP	Echo (ping) request
40	42.778179	192.168.1.101	130.166.246.41	ICMP	Echo (ping) request
41	48.277128	192.168.1.101	130.166.246.41	ICMP	Echo (ping) request
42	53.777518	192.168.1.101	130.166.246.41	ICMP	Echo (ping) request

Frame 37 (72 bytes on wire, 72 bytes captured)  
Ethernet II, Src: 00:07:e9:f7:1c:62, Dst: 00:06:25:85:88:b3  
Internet Protocol, Src Addr: 192.168.1.101 (192.168.1.101), Dst Addr: 24.48.217.227 (24.48.217.227)  
User Datagram Protocol, Src Port: 1046 (1046), Dst Port: domain (53)  
Domain Name System (query)  
Transaction ID: 0x0289  
Flags: 0x0100 (Standard query)  
Questions: 1  
Answer RRs: 0  
Authority RRs: 0  
Additional RRs: 0  
Queries  
= www.csun.edu: type A, class inet  
Name: www.csun.edu

0000 00 06 25 85 88 b3 00 07 e9 f7 1c 62 08 00 45 00 ..%. .... .b..E.  
0010 00 3a 37 4b 00 00 80 11 4f 47 c0 a8 01 65 18 30 ..7K... og...e.0  
0020 d9 e3 04 16 00 35 00 26 08 7b 02 89 01 00 00 01 .....5.& .{.....  
0030 00 00 00 00 00 00 03 77 77 77 04 63 73 75 6e 03 .....w ww.csun..  
0040 65 64 75 00 00 01 00 01 edu.....

Name of desired host

- ❖ Slide shows that DNS is used to find the IP address of [www.csun.edu](http://www.csun.edu)
- ❖ Next frame is response from DNS with IP: 130.166.246.41
- ❖ Ping can now proceed as before

## Use case..

- ❖ I found Ethereal after researching an unusually high traffic volume across my internet connection causing very slow page loads.
- ❖ Turns out that my machine was infected with a virus - a hidden server was running on my machine exchanging data with some remote host
- ❖ Examination of packets led to discovery of hidden server residing on my machine without my knowledge or consent!
- ❖ Ethereal is a very valuable tool to have

# Where to get it:

- ❖ Visit <http://www.wireshark.org/>
- ❖ Navigate to the 'download' section
- ❖ Select your particular machine type (Linux, Apple, Windows etc) and choose download.
- ❖ This will cause an installer executable file to download to your host.
- ❖ After completion, run the executable, follow the onscreen instructions and the installation of Ethereal tools will commence.