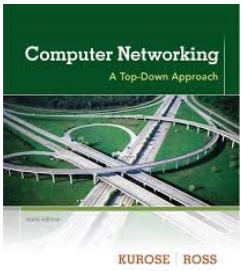
Chapter 2
Application Layer



Computer Networking: A Top Down Approach, 6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

A note on the use of these ppt slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:
- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Chapter 2: Application layer

Chapter 2: Application Layer

Our goals:

- conceptual, implementation aspects of network application protocols
  - transport-layer service models
  - client-server paradigm
  - peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
  - HTTP
  - FTP
  - SMTP / POP3 / IMAP
  - DNS
- programming network applications
  - socket API

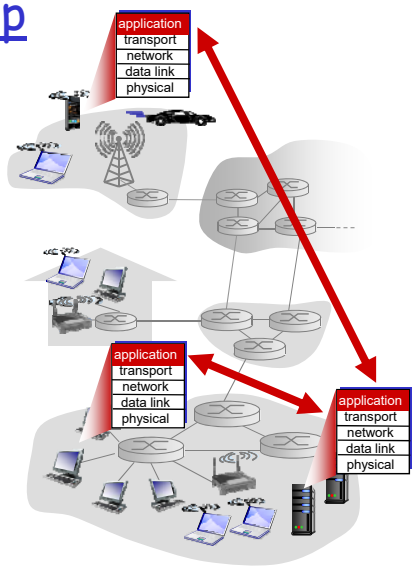
Some network apps

- e-mail
- web
- Text messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video (YouTube, Netflix)
- voice over IP
- real-time video conferencing
- Social netwroking
- cloud computing
- search
-

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



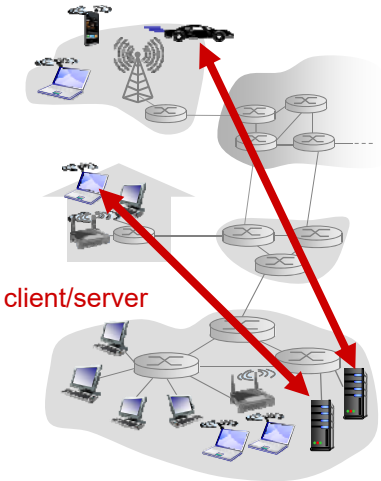
Chapter 2: Application layer

- 2.1 Principles of network applications
  - App architectures
  - Proc communication
  - Internet services
  - Security issues
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Convergent services
- 2.8 Network traffic characterization
- 2.9 Socket programming with TCP and UDP

Application architectures

- Client-server
- Peer-to-peer (P2P)
- Hybrid of client-server and P2P

Client-server architecture



server:

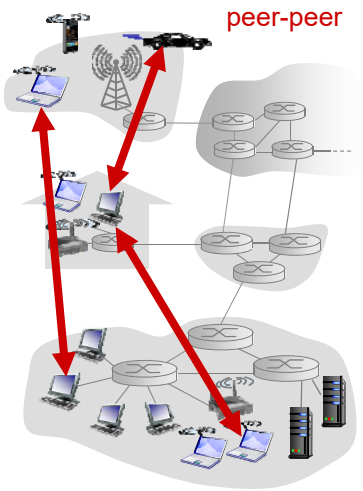
- always-on host
- permanent IP address
- datacenters for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Pure P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses
- self scalability – new peers bring new service capacity, as well as new service demands



Highly scalable but difficult to manage

Hybrid of client-server and P2P

Skype
- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging
- chatting between two users is P2P
- centralized service: client presence detection/location
  - user registers its IP address with central server when it comes online
  - user contacts central server to find IP addresses of buddies

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

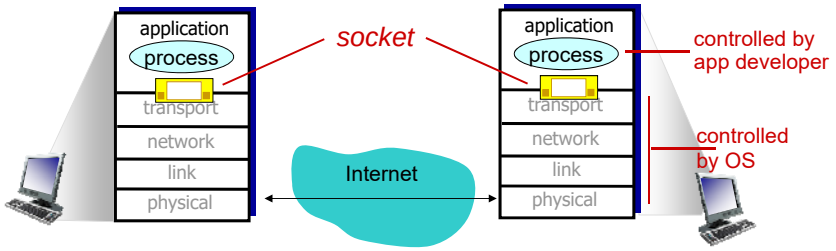
Client process: process that initiates communication

Server process: process that waits to be contacted

- Note: applications with P2P architectures have client processes & server processes

Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process



- API: (1) choice of transport protocol; (2) ability to fix a few parameters

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host suffice for identifying the process?

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- *Q:* does IP address of host on which process runs suffice for identifying the process?
  - *A:* No, *many* processes can be running on same host
- *identifier* includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - HTTP server: 80
  - Mail server: 25
- to send HTTP message to gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - Port number: 80
- more shortly…

App-layer protocol defines

- Types of messages exchanged,
  - e.g., request, response
- Message syntax:
  - what fields in messages & how fields are delineated
- Message semantics
  - meaning of information in fields
- Rules for when and how processes send & respond to messages

Public-domain protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

Proprietary protocols:

- e.g., Skype

What transport service does an app need?

Data integrity/loss

- some apps (e.g., audio) can tolerate some loss
- other apps (e.g., file transfer, telnet) require 100% reliable data transfer

Timing

- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"

Throughput

- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
- other apps ("elastic apps") make use of whatever throughput they get

Security

- Encryption, data integrity, …

Transport service requirements of common apps

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
text messaging	no loss	elastic	yes and no

Internet transport protocols services

TCP service:

- *connection-oriented:* setup required between client and server processes
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when network overloaded
- *does not provide:* timing, minimum throughput guarantees, security

UDP service:

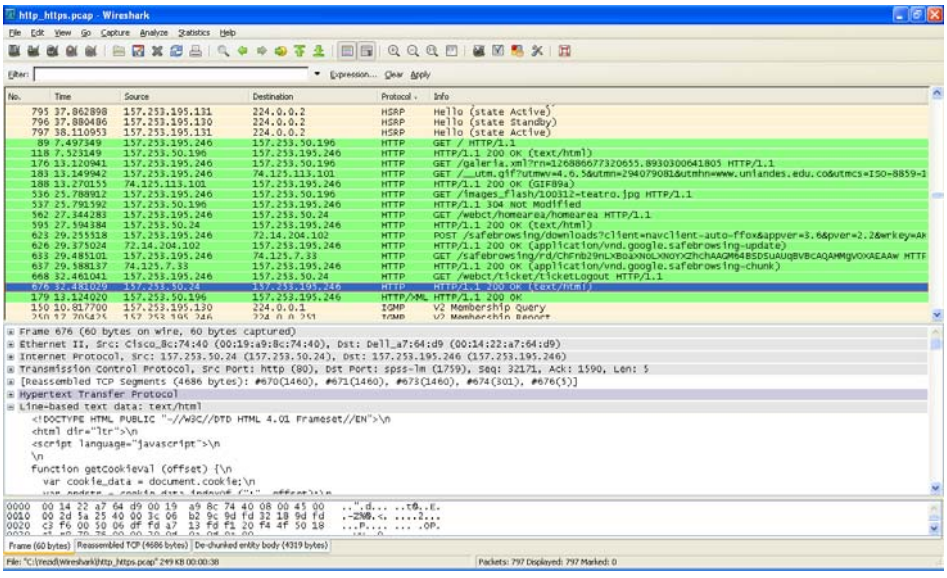
- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

Q: why bother? Why is there a UDP?

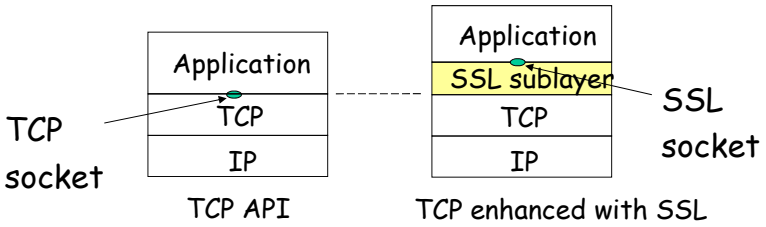
Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Security Problems in TCP/UDP

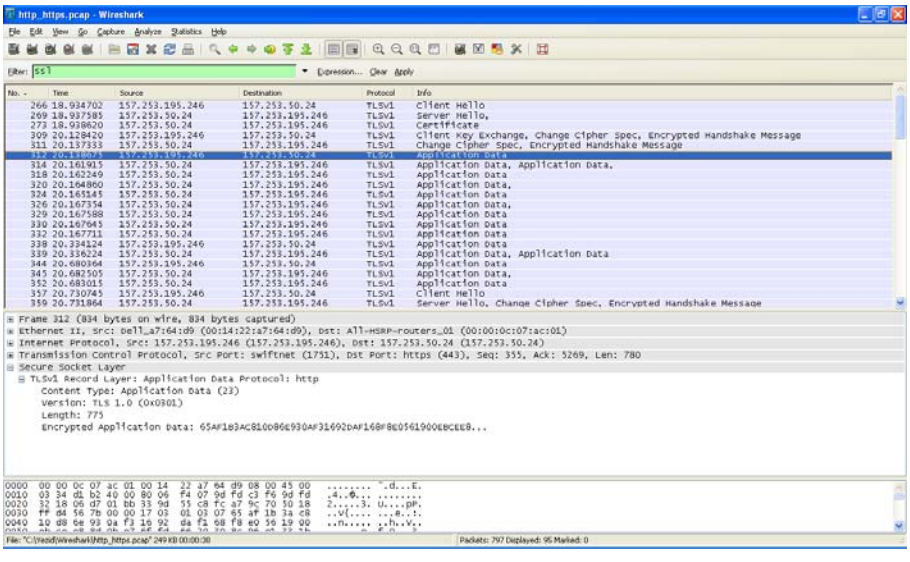


SSL sublayer



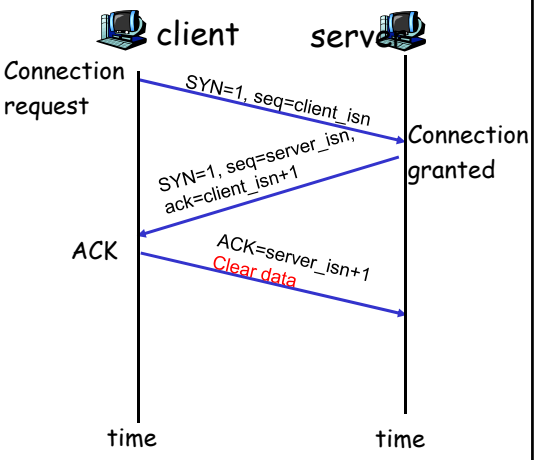
- provides transport layer security to any TCP-based application using SSL services.
  - e.g., between Web browsers, servers for e-commerce (https)
- security services:
  - server authentication, data encryption, data integrity, client authentication (optional)

SSL in action



Normal handshake in TCP

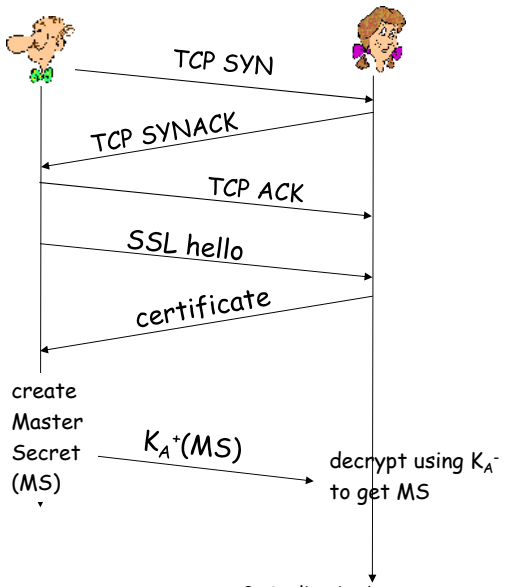
Handshake:



SSL protocol

1. Handshake:

- Bob establishes TCP connection to Alice
- authenticates Alice via CA signed certificate
- creates, encrypts (using Alice's public key), sends master secret key to Alice
  - nonce exchange not shown



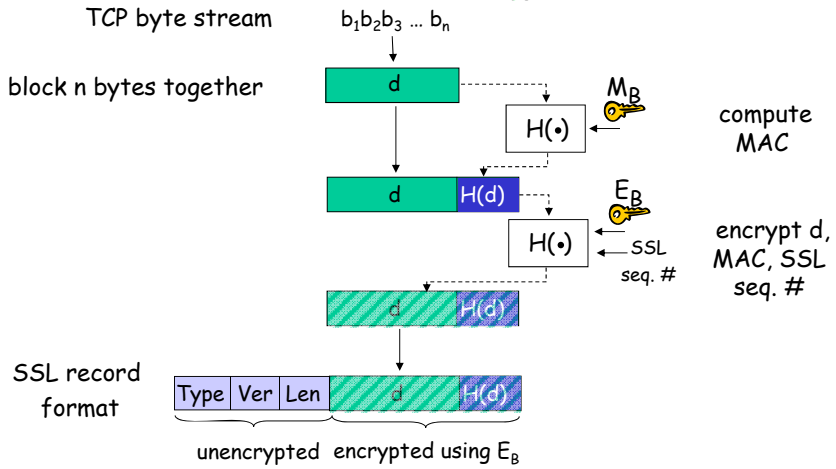
SSL protocol

2. Key Derivation:

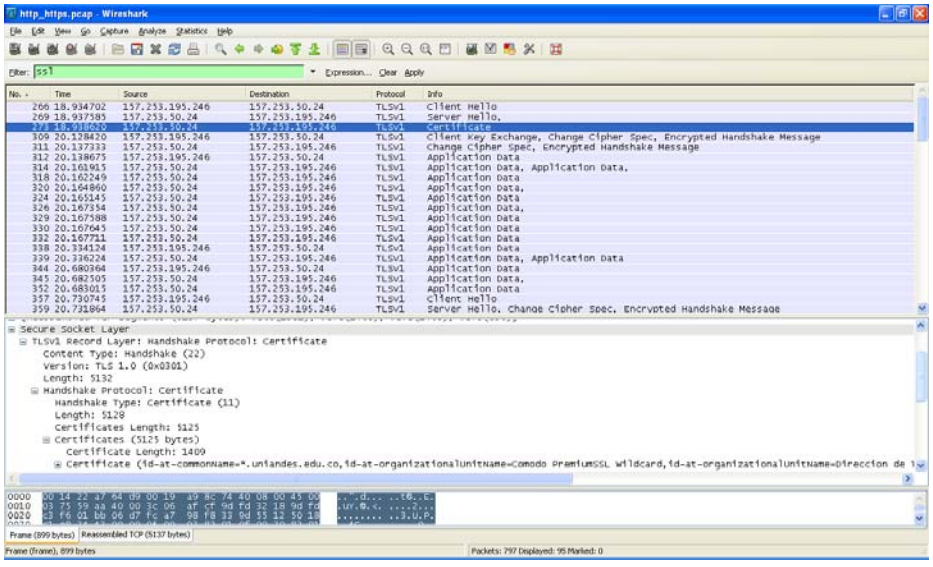
- Alice, Bob use shared secret (MS) to generate 4 keys:
  - E_B: Bob->Alice data encryption key
  - E_A: Alice->Bob data encryption key
  - M_B: Bob->Alice MAC key
  - M_A: Alice->Bob MAC key
- encryption and MAC algorithms negotiable between Bob, Alice

SSL: three phases

3. Data transfer



SSL in action



Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
  - HTTP connections
  - Cookies
  - Cache
  - Capacity planning
  - Security issues
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Convergent services
- 2.8 Network traffic characterization
- 2.9 Socket programming with TCP and UDP

Web and HTTP

First some jargon

- 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:

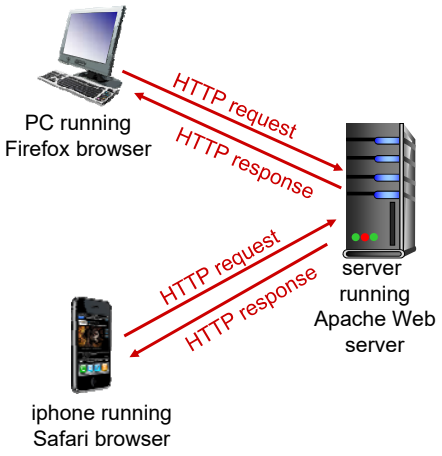
`www.someschool.edu/someDept/pic.gif`

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

Nonpersistent HTTP

- At most one object is sent over a TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

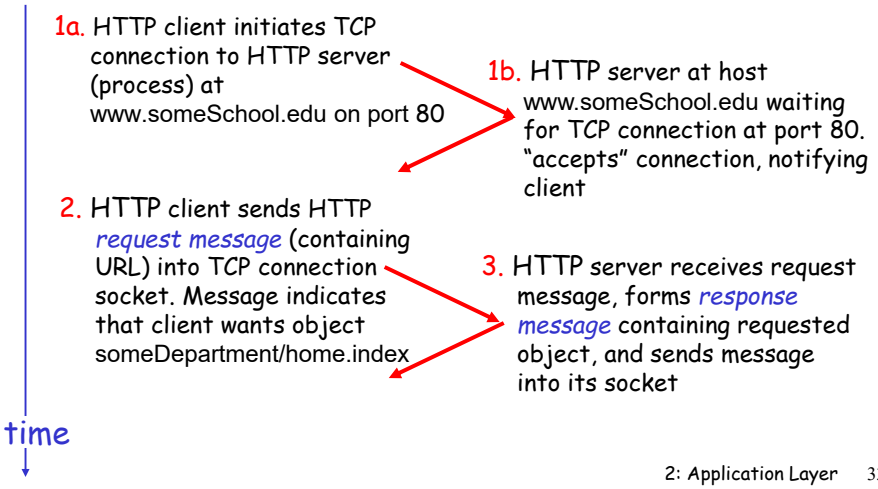
Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server

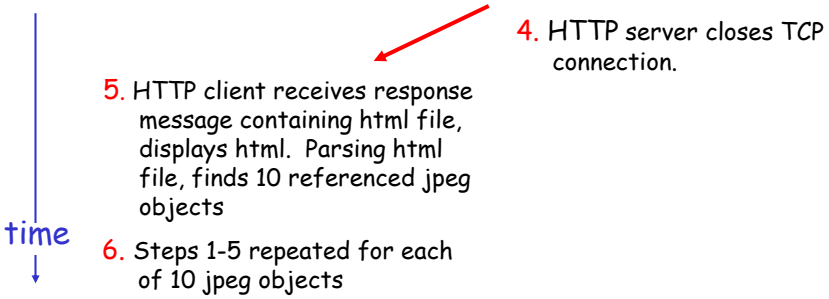
Nonpersistent HTTP

Suppose user enters URL
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)



Nonpersistent HTTP (cont.)



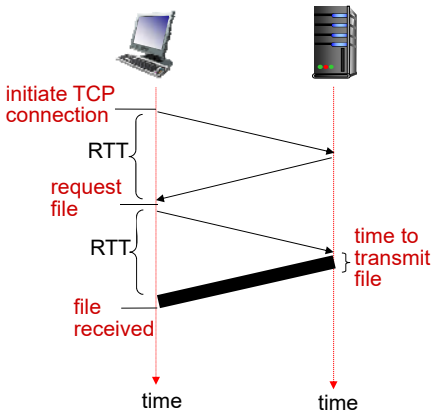
Non-Persistent HTTP: Response time

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+transmit time



Persistent HTTP

Nonpersistent 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

Persistent ***without*** pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent ***with*** pipelining:

- default in HTTP/1.1
- 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)

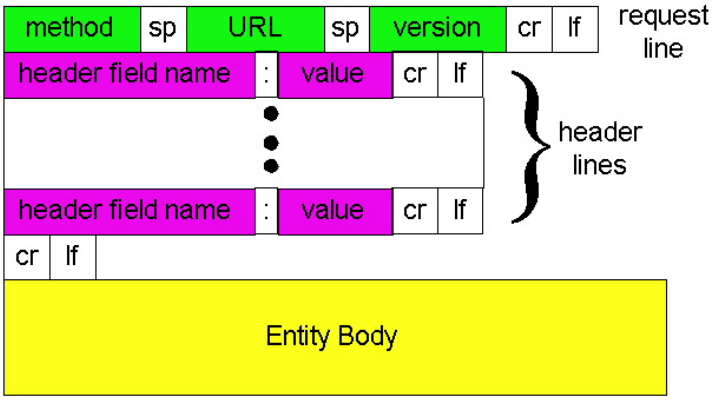
header lines

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Firefox/4.0
Connection: close
Accept-language:fr
```

Carriage return, line feed indicates end of message

(extra carriage return, line feed)

HTTP request message: general format



Uploading form input

Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

URL method:

- Uses GET method
- Input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```


Method types

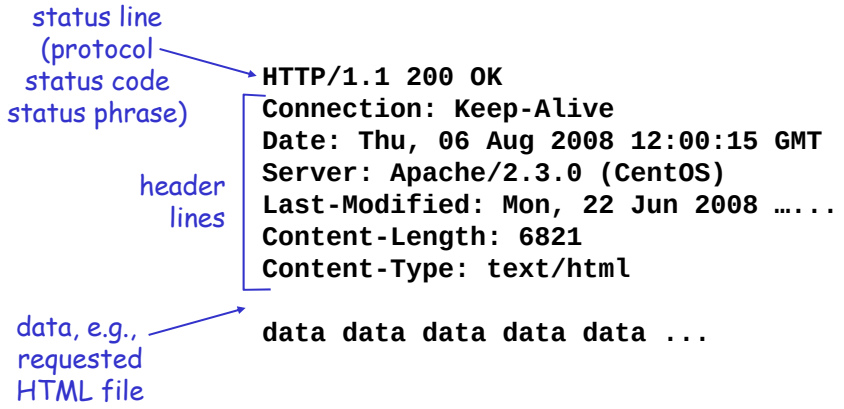
HTTP/1.0

- GET
- POST
- HEAD
  - asks server to leave requested object out of response

HTTP/1.1

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message



HTTP response status codes

In first line in server->client response message.
A few sample codes:

200 OK
- request succeeded, requested object later in this message

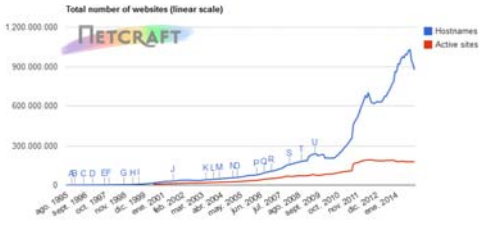
301 Moved Permanently
- requested object moved, new location specified later in this message (Location:)

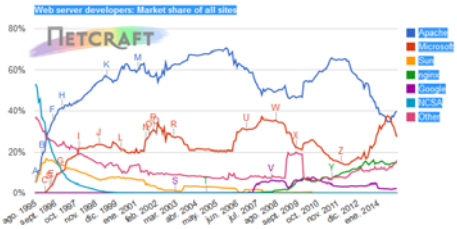
400 Bad Request
- request message not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

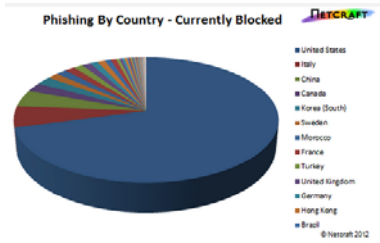
www.netcraft.com





Most Reliable Hosting Company Sites in December 2014

Rank	Performance Graph	OS	Outage hh:mm:ss	Failed Req%	DNS	Connect	First byte	Total
1	Qube Managed Services	Linux	0:00:00	0.007	0.097	0.037	0.075	0.075
2	Datapipe	Linux	0:00:00	0.011	0.104	0.017	0.034	0.047
3	Anexia	Linux	0:00:00	0.011	0.430	0.088	0.195	0.216
4	CWCS	Linux	0:00:00	0.011	0.226	0.099	0.181	0.183
5	iWeb	Linux	0:00:00	0.019	0.132	0.079	0.156	0.156
6	www.choopa.com	Linux	0:00:00	0.022	0.150	0.073	0.157	0.203
7	ServerStack	Linux	0:00:00	0.022	0.087	0.079	0.156	0.156
8	LeaseWeb	Linux	0:00:00	0.026	0.170	0.026	0.058	0.058
9	Bigstep	Linux	0:00:00	0.026	0.126	0.060	0.122	0.122
10	krystal.co.uk	Linux	0:00:00	0.026	0.149	0.062	0.141	0.141



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 to look at captured HTTP request/response)

User-server state: cookies

Many major Web sites use cookies

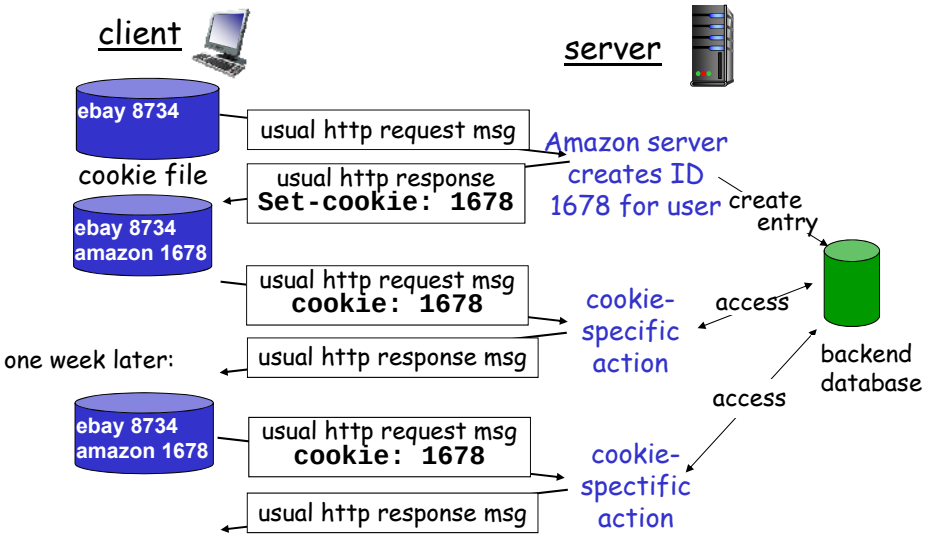
Four components:

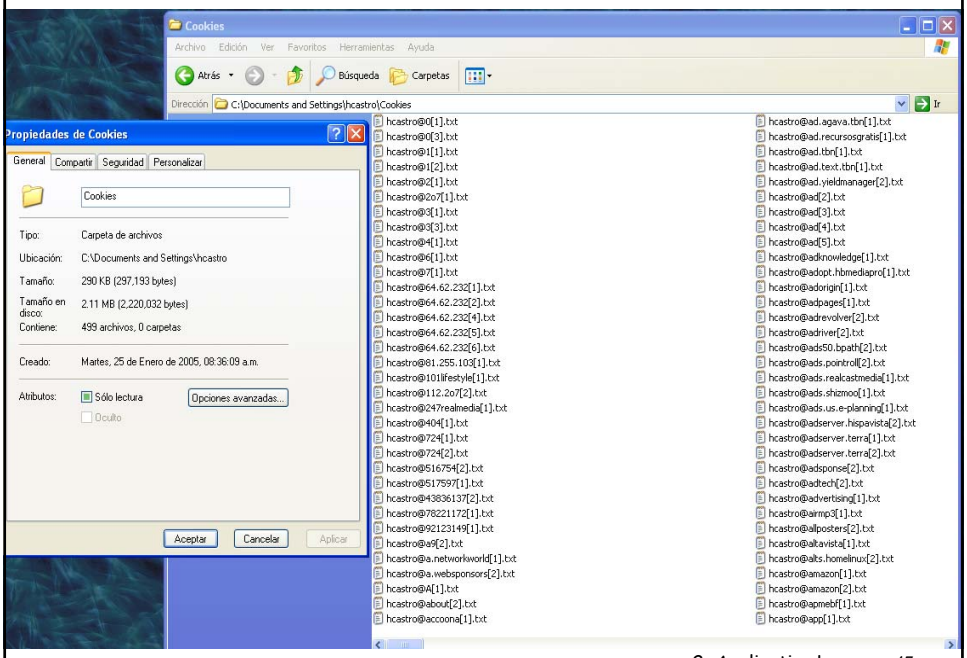
1) cookie header line of HTTP *response* message
2) cookie header line in HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

Example:

- Susan always access Internet always from PC
- visits specific e-commerce site for first time
- when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

Cookies: keeping "state" (cont.)





Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites

How to keep "state":

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

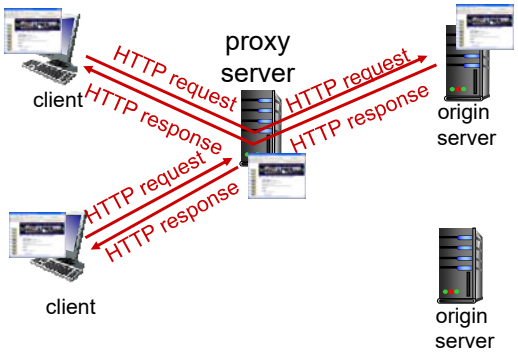
www.doubleclick.net (now a Google company)

- Transactions (between their server and your machine) are transparent to you
  - Clear gif of 1x1 pixels
- Cross-site profiling
  - Very rich anonymous profiles
- Can see what you search at search engines
- Can link anonymous profiles with personal information
- Can do whatever they want
  - Do you know the privacy policy of every site you visit?
- And technology is moving: omniture.com, hitbox.com

Web caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- cache acts as both client and server
- typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link.
- Internet dense with caches: enables "poor" content providers to effectively deliver content (but so does P2P file sharing)

Caching example

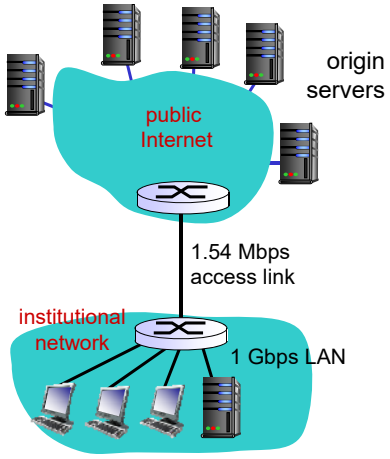
Assumptions

- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences

- utilization on LAN = 15% *problem!*
- utilization on access link = 99%
- total delay = Internet delay + access delay + LAN delay

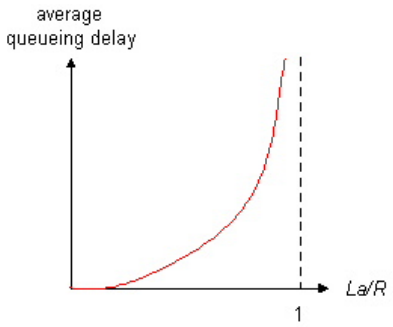
= 2 sec + minutes + microseconds



Queueing delay (revisited)

- R=link bandwidth (bps)
- L=packet length (bits)
- a=average packet arrival rate

traffic intensity = La/R



- La/R ~ 0: average queueing delay small
- La/R -> 1: delays become large
- La/R > 1: more "work" arriving than can be serviced, average delay infinite!

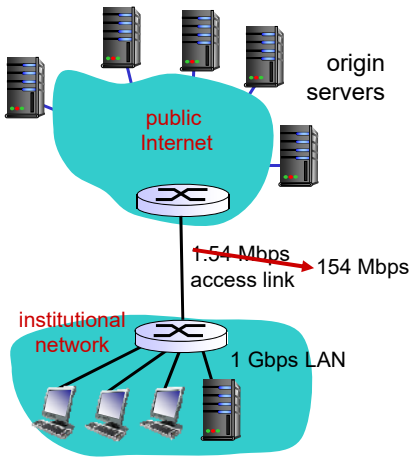
Caching example: fatter access link

assumptions:

- avg object size: 100K bits
- avg request rate from browsers to origin servers:15/sec
- avg data rate to browsers: 1.50 Mbps
- RTT from institutional router to any origin server: 2 sec
- access link rate: ~~1.54 Mbps~~ 154 Mbps

consequences:

- LAN utilization: 15%
- access link utilization = ~~99%~~ 9.9%
- total delay = Internet delay + access delay + LAN delay
  = 2 sec + ~~minutes~~ + usecs msecs

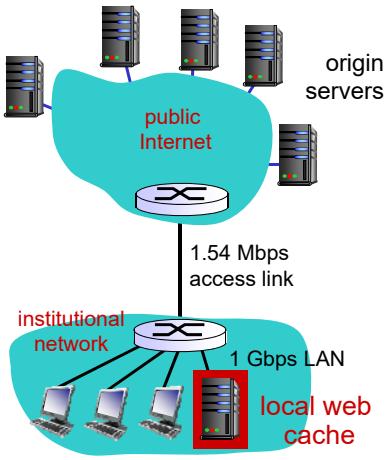


Cost: increased access link speed (not cheap!)

Caching example: install local cache

Calculating access link utilization, delay with cache:

- suppose cache hit rate is 0.4
  - 40% requests satisfied at cache, 60% requests satisfied at origin
  - access link utilization:
    - 60% of requests use access link
  - data rate to browsers over access link = 0.6*1.50 Mbps = .9 Mbps
    - utilization = 0.9/1.54 = .58
- total delay
  - = 0.6 * (delay from origin servers) +0.4 * (delay when satisfied at cache)
  - = 0.6 (2.01) + 0.4 (~msecs)
  - = ~ 1.2 secs
  - less than with 154 Mbps link (and cheaper too!)

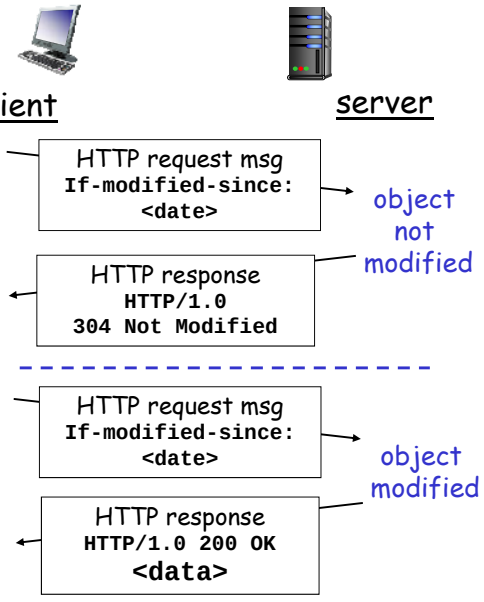


Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request

  `If-modified-since: <date>`
- server: response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`



Capacity Planning in Web Traffic

Traffic Generated by a Request for a 5-KB Page

Traffic Type	Bytes Sent
TCP Connection	180 (approx.)
GET Request	256 (approx.)
5-KB file	5,120
Protocols overhead	1,364 (approx.)
Total	**6,920**

Capacity Planning in Web Traffic

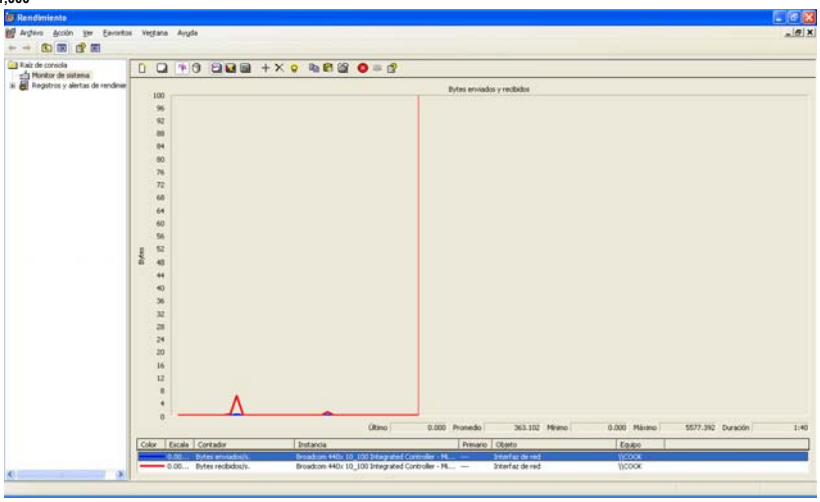
Relative Network Interface Speed

Connection Type	Connection Speed	5-KB Pages Sent per Second
Dedicated PPP/SLIP via modem	28.8 Kbps	Roughly half of 1 page
Frame Relay or fast modem	56 Kbps	Almost 1 page
ISDN	128 Kbps	Just over 2 pages
Typical DSL	640 Kbps	Almost 11 pages
DS1/T1	1.536 Mbps	26 pages
10-Mb Ethernet	8 Mbps (best case)	(Up to) 136 pages
DS3/T3	44.736 Mbps	760 pages
OC1	51.844 Mbps	880 pages
100-Mb Ethernet	80 Mbps (best case)	(Up to) 1,360 pages
OC3	155.532 Mbps	2,650 pages
OC12	622.128 Mbps	10,580 pages
1-Gbps Ethernet	800 Mbps (best case)	(Up to) 13,600 pages

Capacity Planning in Web Traffic

Web Traffic – Text Page http://datatracker.ietf.org/doc/rfc5905/

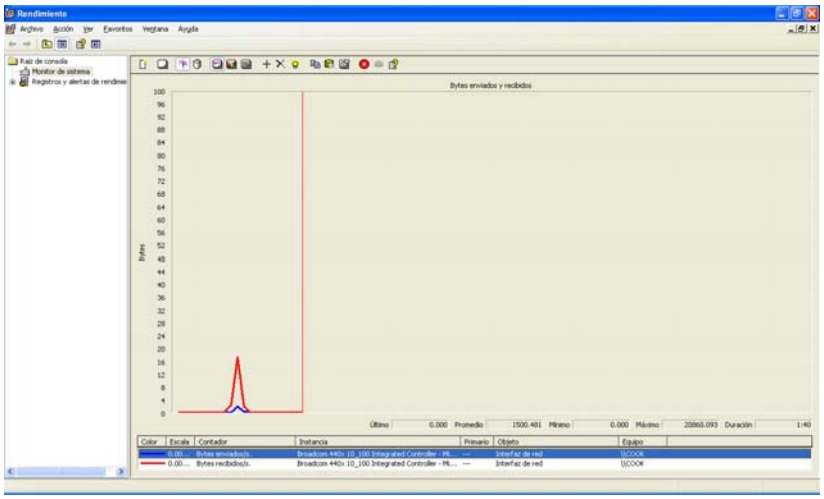
X 1,000



Capacity Planning in Web Traffic

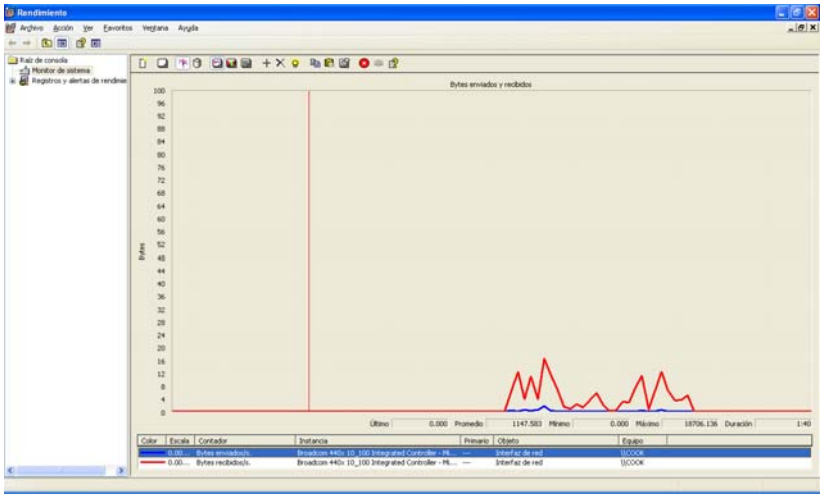
Web Traffic – image Page http://www.uniandes.edu.co

X 1,000



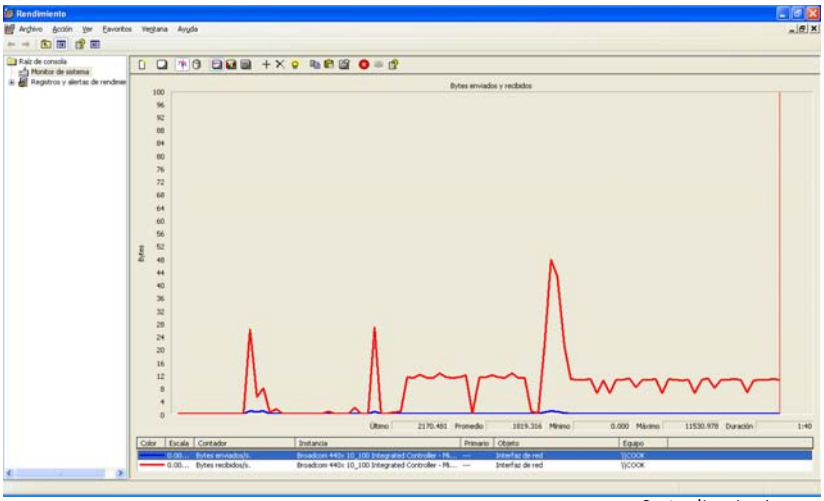
Capacity Planning in Web Traffic

Web Traffic – image Page http://www.cisco.com/
X 1,000



Capacity Planning in Web Traffic

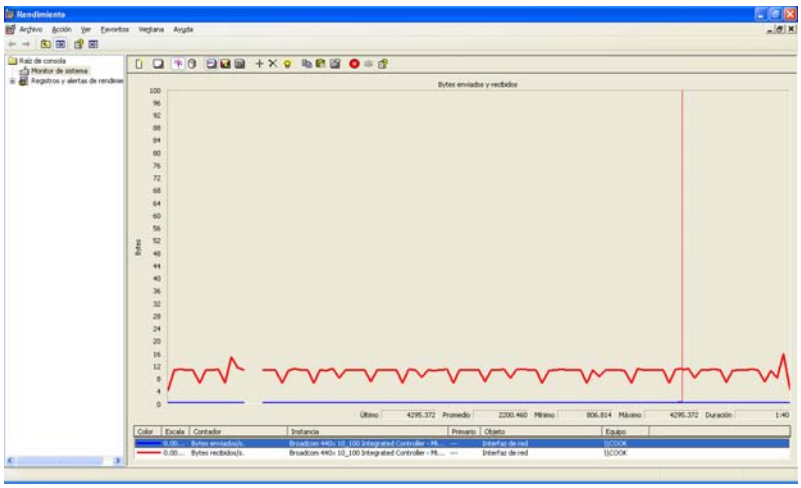
Web Traffic – Streaming http://video.uniandes.edu.co/
X 10,000



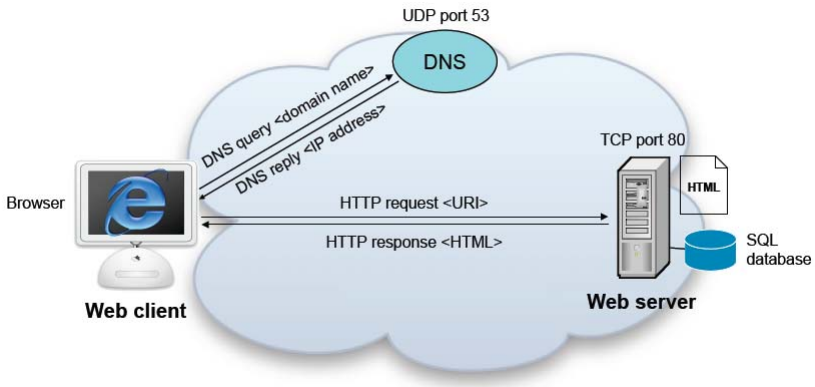
Capacity Planning in Web Traffic

Web Traffic – Streaming
X 10,000

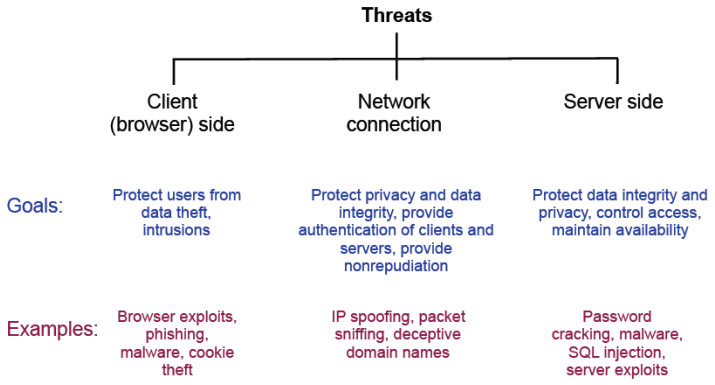
http://video.uniandes.edu.co/



Security Problems in HTTP



Security Problems in HTTP



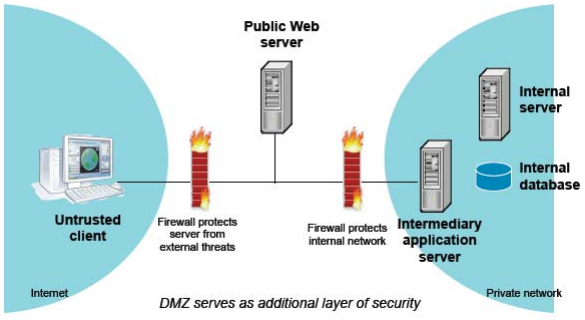
Security Problems in HTTP

- Why on browsers
  - Easy targets: many naïve users
  - Easy to find information: social engineering
  - Browsers are complex programs: may have vulnerabilities
    - Capable to manage hypermedia, files, applets: many possible attacks
    - Keep private data (cookies, history) making them attractive targets

Security Problems in HTTP

- Why on Web servers
  - Web servers accept input from clients making them vulnerable to attacks
  - Complex software
    - Not vulnerable proof
    - Attacks are easy to hide
  - Keep valuable data easy to reach
  - Trusted users: phishing
  - High impact

Security Problems in HTTP



Security Problems in HTTP

Hypertext Transfer Protocol Secure (HTTPS)

Familia:	Familia de protocolos de Internet
Función:	Transferencia segura de hipertexto
Puertos:	443/TCP

Ubicación en la pila de protocolos

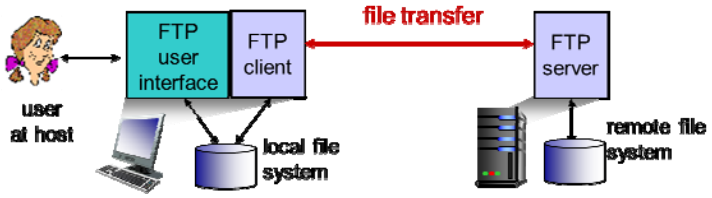
Aplicación	**HTTPS**
Transporte	SSL/TLS TCP
Red	IP

Estándares:	RFC 2818 – HTTP sobre TLS

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
  - Protocol
  - Capacity planning
  - Security issues
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Convergent services
- 2.8 Network traffic characterization
- 2.9 Socket programming with TCP and UDP

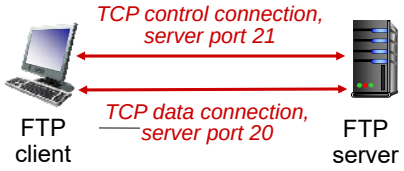
FTP: the file transfer protocol



- 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 2nd 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

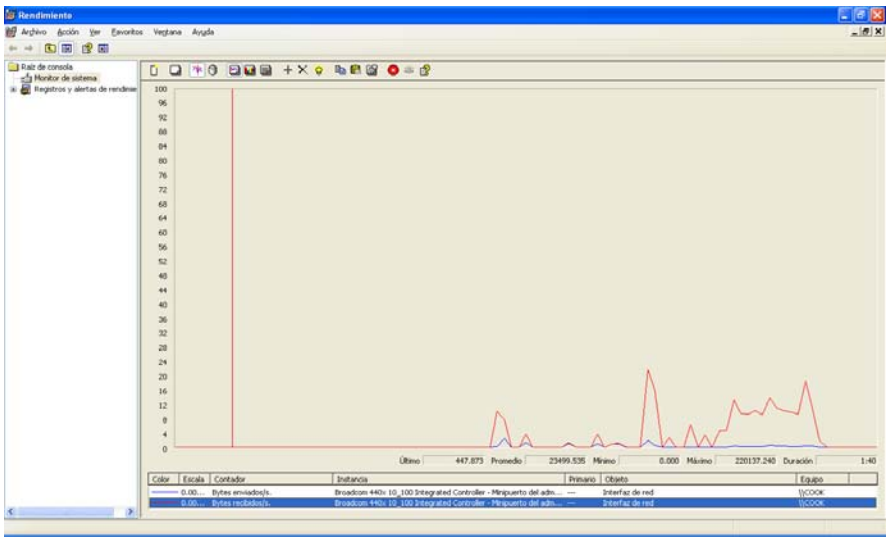
Sample commands:

- 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

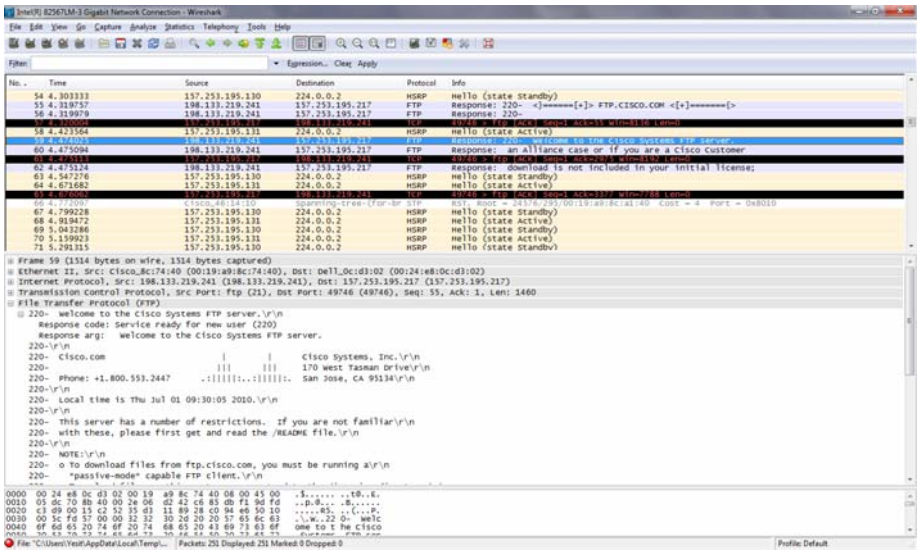
Sample return codes

- 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**

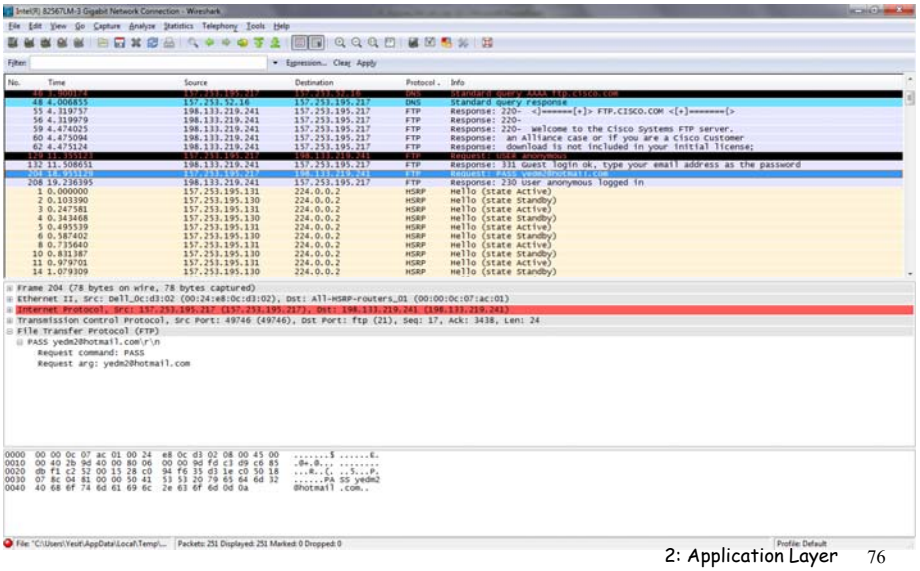
Capacity Planning in FTP Traffic



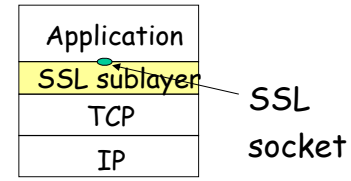
Security Problems in FTP Traffic



Security Problems in FTP Traffic



Security Problems in FTP Traffic



TCP enhanced with SSL

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - Protocols
  - Capacity planning
  - Security issues
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Convergent services
- 2.8 Network traffic characterization
- 2.9 Socket programming with TCP and UDP

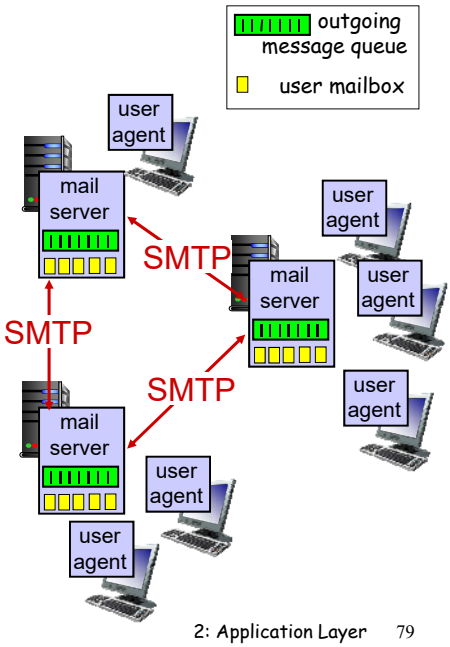
Electronic Mail

Three major components:

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

User Agent

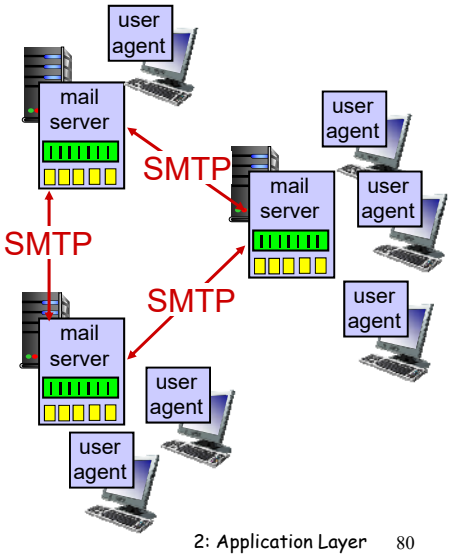
- "mail reader"
- composing, editing, reading mail messages
- e.g., Eudora, Outlook, Thunderbird, phone client
- outgoing, incoming messages stored on server



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

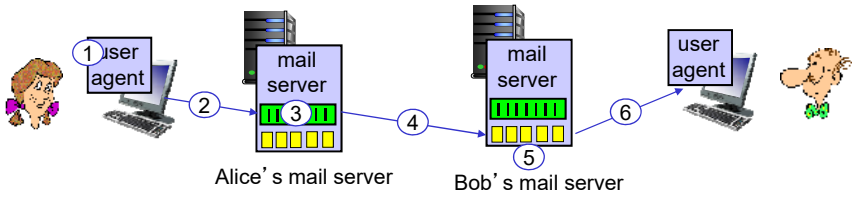


Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25 (SMTPS 465)
- 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



Sample SMTP interaction

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```


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)

SMTP: final words

- SMTP uses persistent connections
- SMTP requires message (header & body) to be in 7-bit ASCII
- SMTP server uses CRLF.CRLF to determine end of message

Comparison with HTTP:

- HTTP: pull
- SMTP: push
- both have ASCII command/response interaction, status codes
- HTTP: each object encapsulated in its own response msg
- SMTP: multiple objects sent in multipart msg

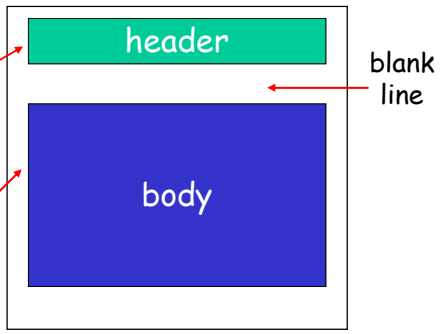
Mail message format

SMTP: protocol for exchanging email msgs

RFC 822: standard for text message format:

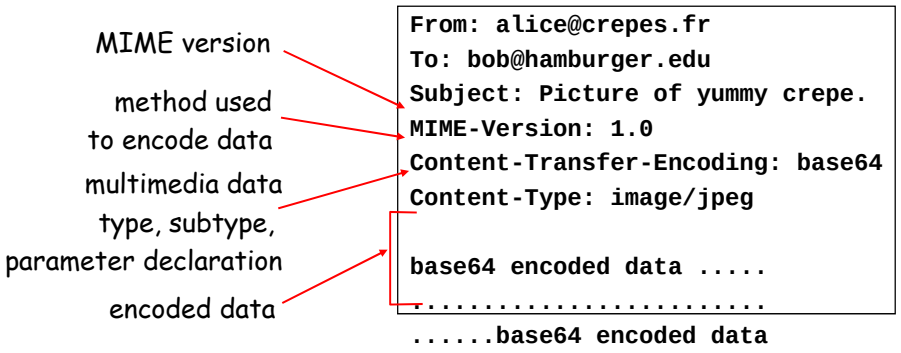
- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP commands!
- body
  - the "message", ASCII characters only



Message format: multimedia extensions

- MIME: multimedia mail extension, RFC 2045, 2056
- additional lines in msg header declare MIME content type



MIME types

Content-Type: type/subtype; parameters

Text

- example subtypes: **plain, html**

Image

- example subtypes: **jpeg, gif**

Audio

- example subtypes: **basic** (8-bit mu-law encoded), **32kadpcm** (32 kbps coding)

Video

- example subtypes: **mpeg, quicktime**

Application

- other data that must be processed by reader before "viewable"
- example subtypes: **msword, octet-stream**

text/plain; charset=us-ascii
text/plain; charset="ISO-8859-1"
text/html
application/msword

Multipart Type

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=StartOfNextPart

--StartOfNextPart
Dear Bob, Please find a picture of a crepe.
--StartOfNextPart
Content-Transfer-Encoding: base64
Content-Type: image/jpeg
base64 encoded data .....
.........................
......base64 encoded data
--StartOfNextPart
Do you want the recipe?
```

Delimiter

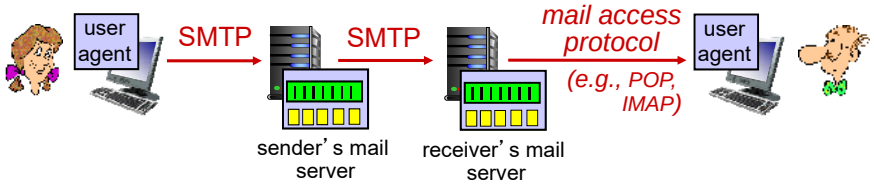
Two dashes

8-7 bits: quoted-printable encoding


```
From: Harold Castro <hcastro@uniandes.edu.co>
Subject: nena
To: Harold Castro <hcastro@uniandes.edu.co>
Message-id: <432705E8.5010802@uniandes.edu.co>
MIME-version: 1.0
Content-type: multipart/related; boundary="------------020601020701020006040809
X-Accept-Language: en-us, en
User-Agent: Mozilla Thunderbird 1.0 (Windows/20041206)
Original-recipient: rfc822;hcastro@store01.uniandes.edu.co

This is a multi-part message in MIME format.
--------------020601020701020006040809
Content-Type: text/html; charset=ISO-8859-1
Content-Transfer-Encoding: 7bit

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta content="text/html;charset=ISO-8859-1" http-equiv="Content-Type">
</head>
<body bgcolor="#ffffff" text="#000000">
<br>
y ahora aqu&iacute; mismo<br>

<img src="cid:part1.04060300.07090000@uniandes.edu.co" alt=""><br>
</body>
</html>

--------------020601020701020006040809
Content-Type: image/jpeg;
 name="moz-screenshot.jpg"
Content-Transfer-Encoding: base64
Content-ID: <part1.04060300.07090000@uniandes.edu.co>
Content-Disposition: inline;
 filename="moz-screenshot.jpg"

/9j/4AAQSkZJRgABAQIAJQAlAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UHRof
Hh0aHBwgJC4nICIsIxwcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/2wBDAQkJCQwLDBgNDRgyIRwh
MjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjL/wAAR
CACGAFkDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAA
```

uXmtYVuJY7pboeWuUWN8bRnk+/fivQoLn7WJIpY/Iu4cCaHdnbnoynjchwcNgdCCAQwFG/02
K5XEiA4OQcdK3hiHHSRlPDxkrxOFDW5Y27SFD1h3ybVj2+p9SDnHqaq/ZJLq0knkuo4gSVjW
QHnkhvoOK19Y0i5AKrbiSMjBMQAOPQ/zqhfxzGCCVrKZZVQiZMlVI6DafU8E11qcXszjcJR3
RTikvbm4D3EoYwRbYUON0iDjHvwMnviof7buv+eNnVKdERAmCxZQFkOflPfv17e9V8n+/H/3
yKq5J9VRrxSMMcUGU7eOtQmTJ5NefdHoWYOKiYc05pBiozIKkoYyioXUVIWFRs/HFFgOd8V2
DzaPLeWcKNqVkpntX8re+VIYxrghsOF2EA8hu/SrI8ueFJoXWSKRQyOhBDKeQQe4rUJyTmsP
TFFjd3OjsMRx/v7P0MLHlB/uNkYAwqNEOppNXQ07MintwwII/SqjwFCcAEf3TW5JESTxVWSD
1qLNGlzEm02yuxiWFN3oyj+dUv8AhEtM/wCfVfzrflt+2Kj+z/WmpyWzE4xe6O4kmG3AqsZs
Gq73I7moXuB61TZFi48/FV3uOetUpLnPQ1We5PPNJyHY0WuSO9MNz3zWSbnHU037VkdaVx8p
ri4z34rO1VJXNveWa7ru0k3KoIBkjPEkeeOq8gEgb1QngVCtyemacLkHvTUgcS9a30N9aQ3U
D74Zo1kjbBG5SMg4PPQ09lDd6520m+waw9gv/HvcpJdRDqUfePNBPoTIrDryX6AKK2hOM02x
IkaAHpSeQacJgcU/zh7UrILsxbfXbe9iMlvMrgNtbBwUburDqpGeQeRT31DI61yN1HY3kgln
tonlUYWUqN6+m1uowTkEHg1WcyWwzaahPGOvlzsZ0J9TuO/8AwHt1yrDu0dib8n+IVBJfn14
riJfEd1Cx862jI9IJ9x/8eCjH41XPisk82s//fUf/wAVT9nIXtIo7hrzd3pq3Jzwa46LxEXP
FncH6PF/8XV9NakYfLpt0f8AgcP/AMco5GhqaZ0onJPWpll965pdXnP/ADCrz/vuH/45Uw1e
cf8AMIvv++4f/jlKw+ZGnqcn2d7XU1PNu+yX08lyA+T2CkK5PpHjIBNawl965S5v5r2yntJN
J1ARzxtGxV4M4YYOP3nXmtbTpbg6da/bT/pXlJ53T7+Bu6cdc9Kb2EtzbWY+tP8AN96z1k56
0/zRUhY8Tl8XXDjCJisubWr2YkmVgD2BrPor0FFLY85zk92StczOctIx/GmeY/8Aeb86bRTJ
HCR1OQzA+oNWodUvYMbLh/xOap0UrIabRvW3iq+hPz4cflWxbeNI2wJUdf1riaWpdOL6FqrN
dT0218U2TkYnAPvWxDrUMgG2VT9DXjVPSaSM/I7L9DWboLoarEPqj21NTQ/xj86l/tJPX9a8
Zh1m/g+7OxHo3NWf+Em1D+8Kn2D7lrEx7FIxL6VE0WfwoorpucbQwoMD3pm3iiimITFLiiig
AAzS7eKKKAuOCZ704Q+9FFSND/s/+1SfZ/eiigD/2Q==
----------020601020701020006040809--

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
    - Port: 110/TCP Secure: 995/TCP
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - more features (more complex)
    - Port: 143/TCP 220/TCP (IMAP3) 993/TCP (IMAPS)
    - manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 protocol

authorization phase
- client commands:
  - **user:** declare username
  - **pass:** password
- server responses
  - **+OK**
  - **-ERR**

transaction phase, client:
- **list:** list message numbers
- **retr:** retrieve message by number
- **dele:** delete
- **quit**

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

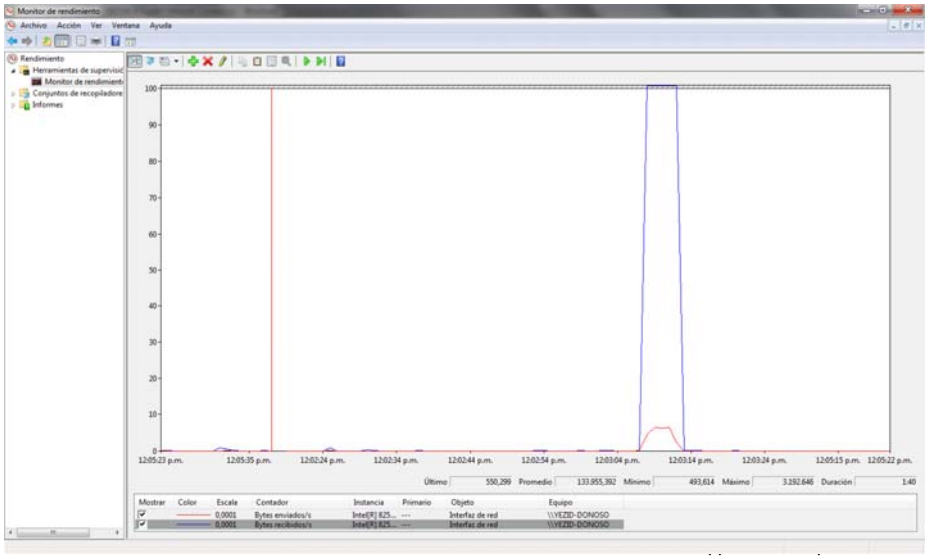
```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```


POP3 (more) and IMAP

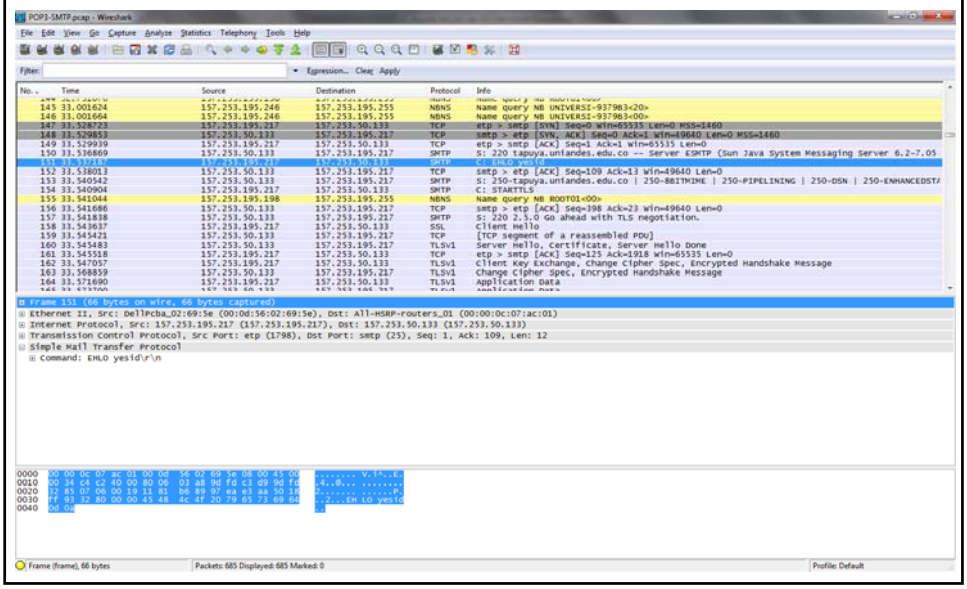
More about POP3
- Previous example uses "download and delete" mode.
  - Bob cannot re-read e-mail if he changes client
- "Download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP
- Keep all messages in one place: the server
- Allows user to organize messages in folders
- IMAP keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

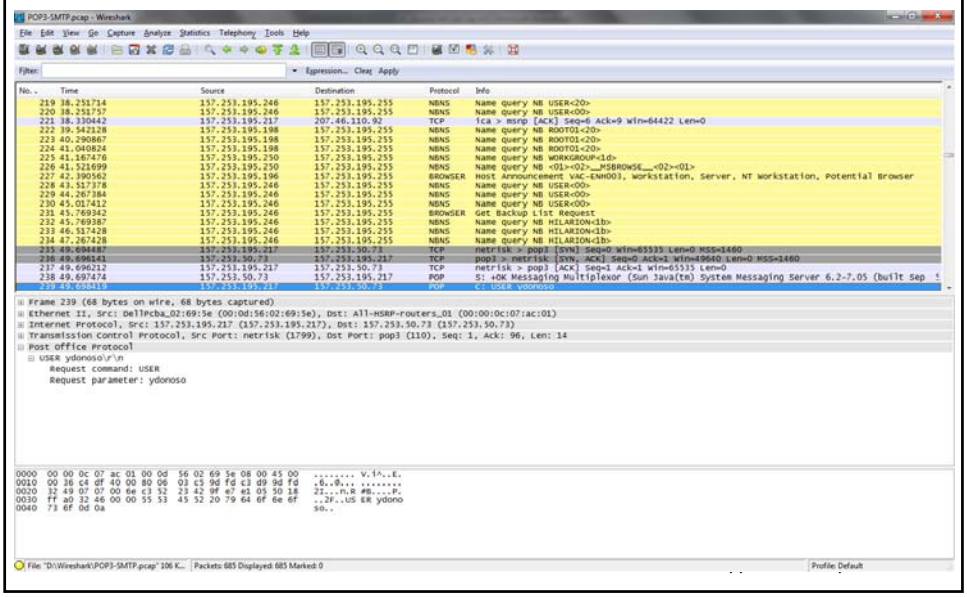
Capacity Planning in email



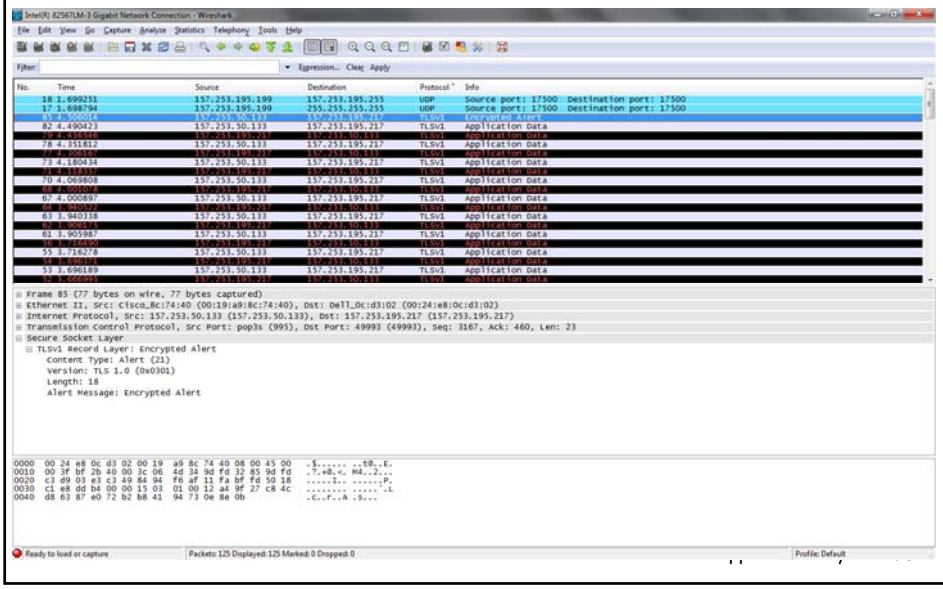
Security in email



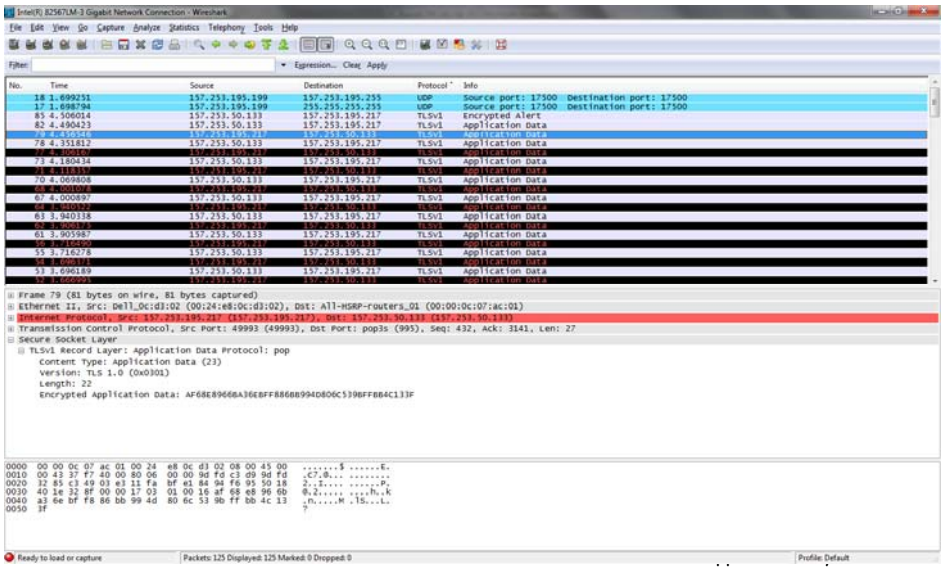
Security in email



Security in email

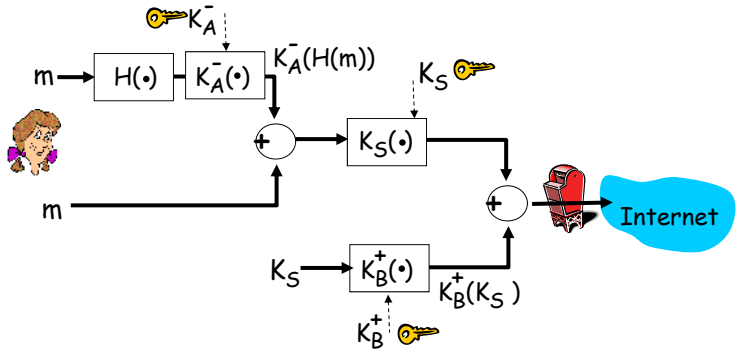


Security in email



Security in email

- Alice wants to provide secrecy, sender authentication, message integrity.



Alice uses three keys: her private key, Bob's public key, newly created symmetric key

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
  - Architecture
  - Security issues
- 2.6 P2P applications
- 2.7 Convergent services
- 2.8 Network traffic characterization
- 2.9 Socket programming with TCP and UDP

DNS: Domain Name System

People: many identifiers:
- SSN, name, passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., ww.yahoo.com - used by humans

Q: map between IP addresses and name ?

Domain Name System:
- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network's "edge"

DNS

DNS services

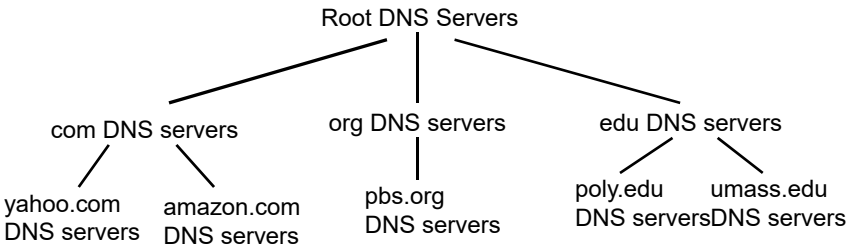
- hostname to IP address translation
- host aliasing
  - Canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: set of IP addresses for one canonical name

Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

doesn't *scale!*

Distributed, Hierarchical Database

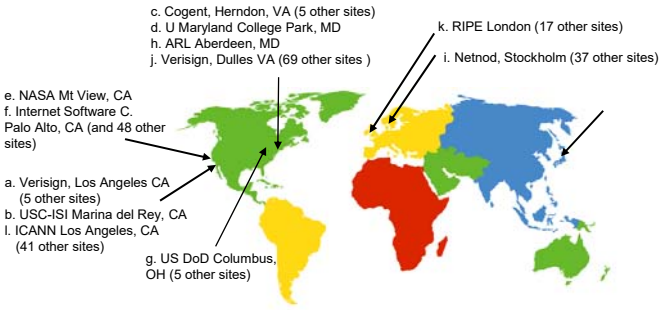


Client wants IP for www.amazon.com; 1st approx:

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

DNS: Root name servers

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



13 root name servers worldwide

TLD and Authoritative Servers

- Top-level domain (TLD) servers:
  - Two groups: generic (gTLDs) and countries (ccTLDs)
    - gTLDs: com, org, net, edu, museum, biz, name, etc.
    - ccTLDs: uk, fr, co,
  - Network Solutions maintains servers for com TLD, educause for edu TLD
- Authoritative DNS servers:
  - organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web, mail).
  - can be maintained by organization or service provider

Local Name Server

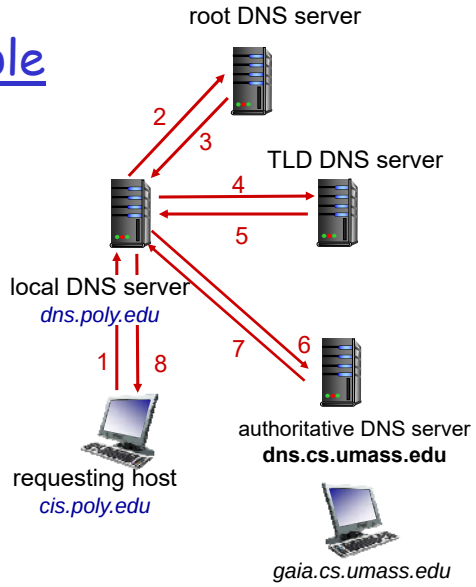
- does not strictly belong to hierarchy
- each ISP (residential ISP, company, university) has one.
  - also called "default name server"
- when host makes DNS query, query is sent to its local DNS server
  - acts as proxy, forwards query into hierarchy
  - has local cache of recent name-to-address translation pairs (but may be out of date!)

DNS name resolution example

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

iterated query:

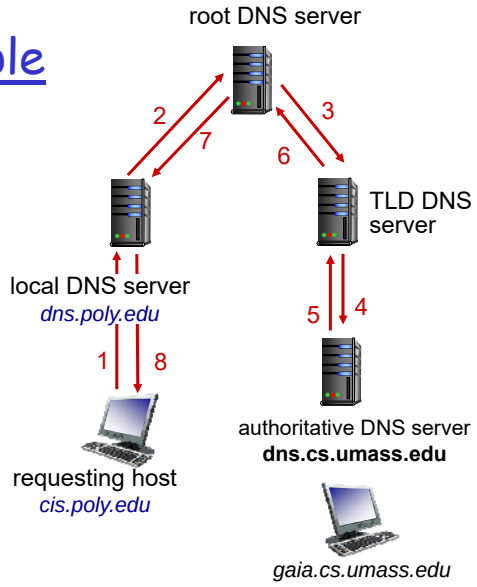
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



DNS name resolution example

recursive query:

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



DNS: caching and updating records

- 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 under design by IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

DNS records

DNS: distributed db storing resource records (RR)

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

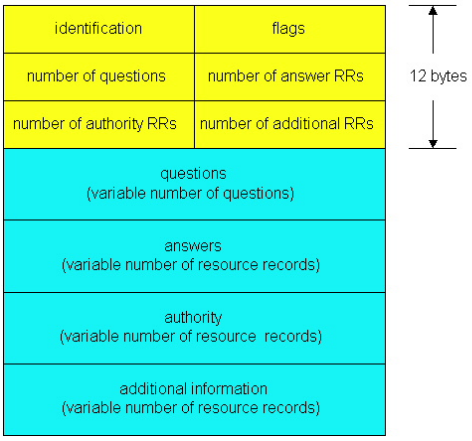
- Type=A
  - **name** is hostname
  - **value** is IP address
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is hostname of authoritative name server for this domain
- Type=CNAME
  - **name** is alias name for some "canonical" (the real) name `www.ibm.com` is really `servereast.backup2.ibm.com`
  - **value** is canonical name
- Type=MX
  - **value** is name of mailserver associated with **name**

DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

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



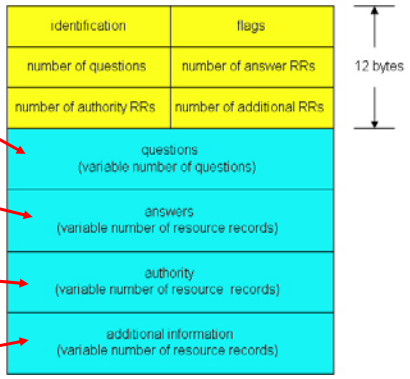
DNS protocol, messages

Name, type fields for a query

RRs in response to query

records for authoritative servers

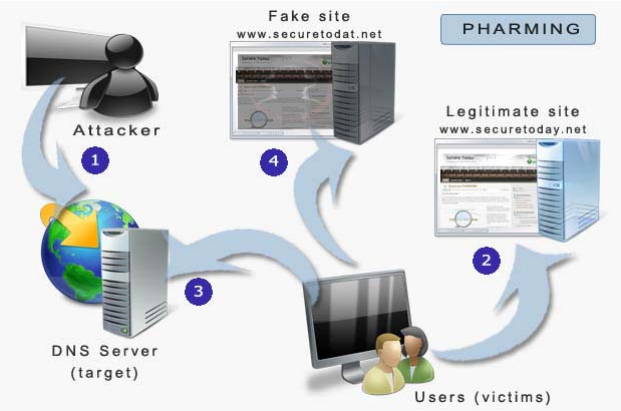
additional "helpful" info that may be used



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 two RRs into com TLD server:
    `(networkutopia.com, dns1.networkutopia.com, NS)`
    `(dns1.networkutopia.com, 212.212.212.1, A)`
- create authoritative server Type A record for www.networkuptopia.com; Type MX record for networkutopia.com
- How do people get IP address of your Web site?

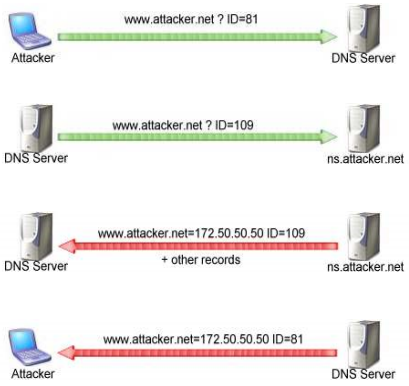
Security in DNS (DNS Spoofing)



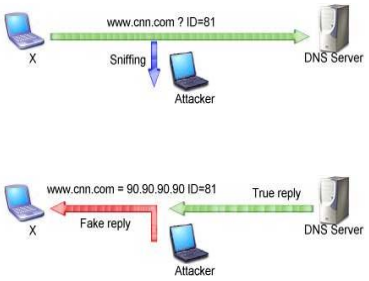
How difficult is the DNS spoofing attack?

1. /etc/hosts o c:\Windows\System32\drivers\etc\hosts

2. Cache poisoning



3. ID spoofing



Security in DNS (DNS Spoofing)

- To disable unused ports
- To implement MAC validations on switches
- To implement 802.1x
- To use DNSSec
  - Use cetificates to authenticate authoritative servers
- To use IPSec
  - Use encription to ensure confidenciality

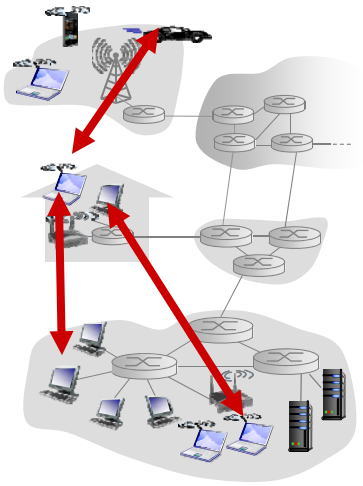
Chapter 2: Application layer

- 2.1 Principles of network applications
  - app architectures
  - app requirements
- 2.2 Web and HTTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
  - Content distribution
  - Examples: BitTorrent, skype
- 2.7 Convergent services
- 2.8 Network traffic characterization
- 2.9 Socket programming with TCP and UDP

Pure P2P architecture

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

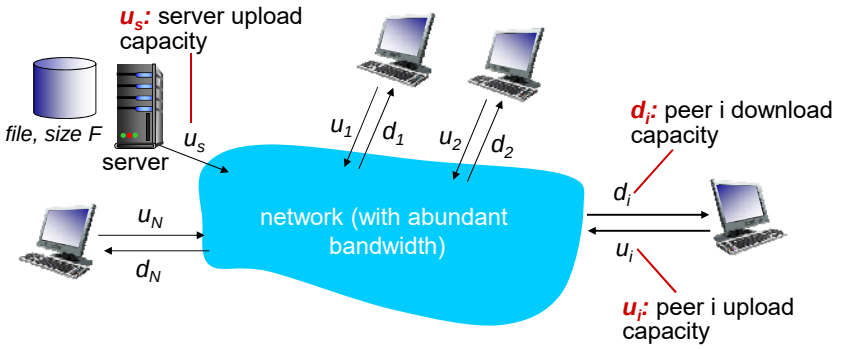
- Examples:
  - File distribution
  - Streaming
  - Skype



File Distribution: Server-Client vs P2P

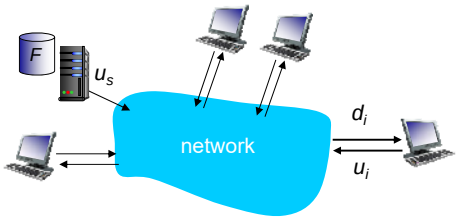
Question : How much time to distribute file from one server to *N peers*?

- peer upload/download capacity is limited resource



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}

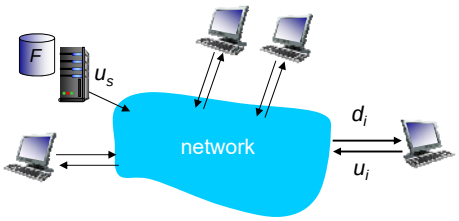


time to distribute F to N clients using client-server approach $D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$

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 (limting max download rate) is $u_s + \Sigma u_i$

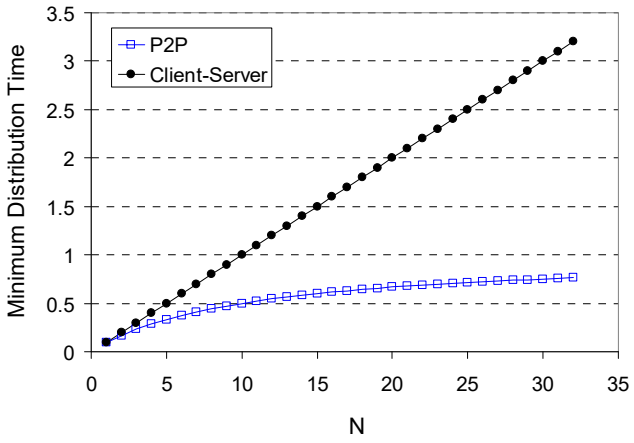


time to distribute F to N clients using P2P approach $D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$

increases linearly in N …
… but so does this, as each peer brings service capacity

Server-client vs. P2P: example

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

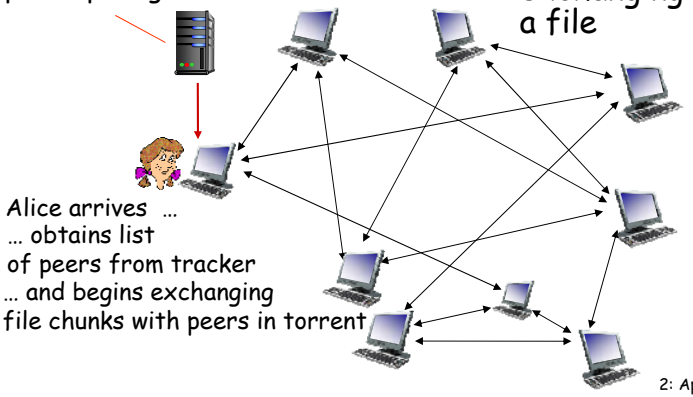


P2P file distribution: BitTorrent

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

tracker: tracks peers participating in torrent

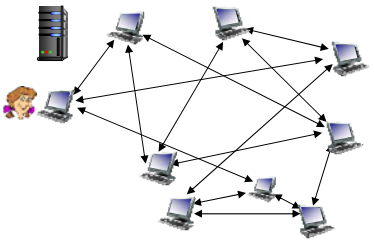
torrent: group of peers exchanging chunks of a file



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 questions

- Who should I ask for new chunks?
- Which chunks should I ask first?
- To whom should I answer?
- How to become attractive to others?
  - What if I'm not?

BitTorrent (2)

Requesting Chunks

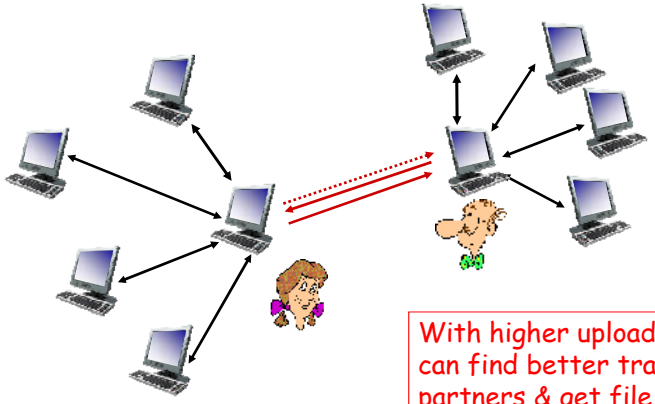
- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks *at the highest rate*
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

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



With higher upload rate, can find better trading partners & get file faster!

Distributed Hash Table (DHT)

- Hash table
- DHT paradigm
- Circular DHT and overlay networks
- Peer churn

Simple Database

Simple database with(key, value) pairs:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
…….	……
Lisa Kobayashi	177-23-0199

- key: movie title; value: IP address

Hash Table

- More convenient to store and search on numerical representation of key
- key = hash(original key)

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
…….		……
Lisa Kobayashi	9290124	177-23-0199

Distributed Hash Table (DHT)

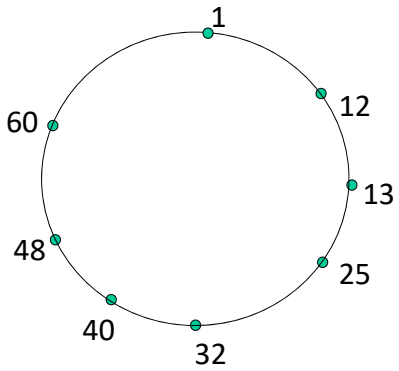
- Distribute (key, value) pairs over millions of peers
  - pairs are evenly distributed over peers
- Any peer can query database with a key
  - database returns value for the key
  - To resolve query, small number of messages exchanged among peers
- Each peer only knows about a small number of other peers
- Robust to peers coming and going (churn)

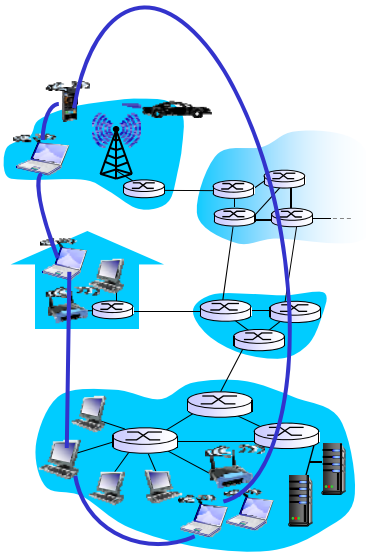
Assign key-value pairs to peers

- rule: assign key-value pair to the peer that has the *closest* ID.
- convention: closest is the *immediate successor* of the key.
- e.g., ID space {0,1,2,3,…,63}
- suppose 8 peers: 1,12,13,25,32,40,48,60
  - If key = 51, then assigned to peer 60
  - If key = 60, then assigned to peer 60
  - If key = 61, then assigned to peer 1

Circular DHT

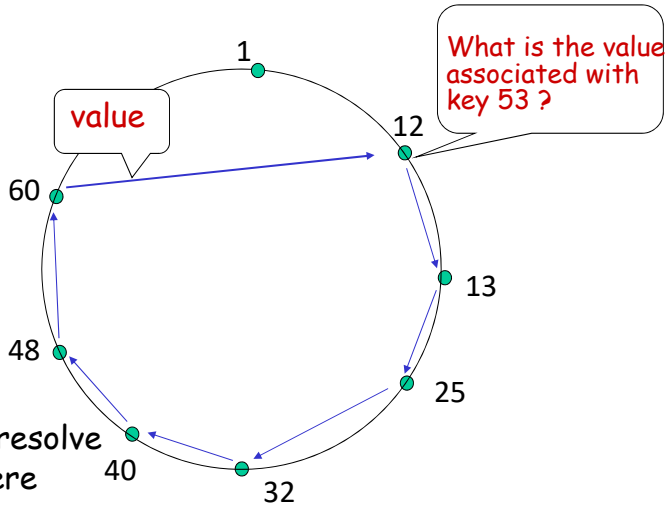
- each peer *only* aware of immediate successor and predecessor.





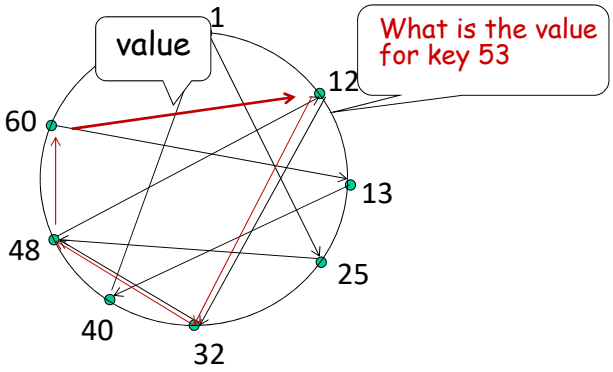
"overlay network"

Resolving a query



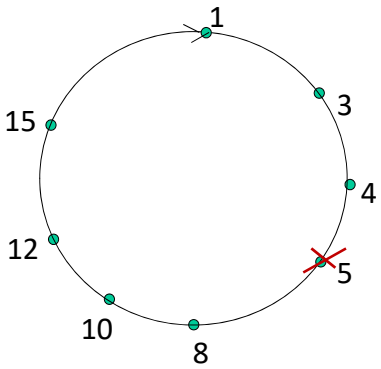
O(N) messages on avgerage to resolve query, when there are *N* peers

Circular DHT with shortcuts



- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 3 messages.
- possible to design shortcuts with *O(log N)* neighbors, *O(log N)* messages in query

Peer churn

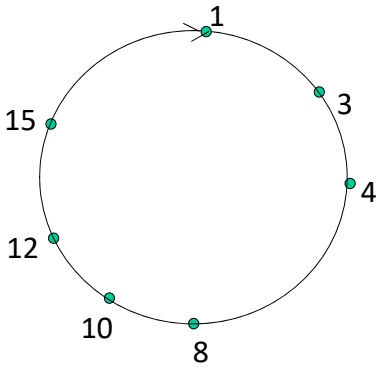


example: peer 5 abruptly leaves

handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

Peer churn



handling peer churn:

- peers may come and go (churn)
- each peer knows address of its two successors
- each peer periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- peer 4 detects peer 5's departure; makes 8 its immediate successor
- 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

Chapter 2: Application layer

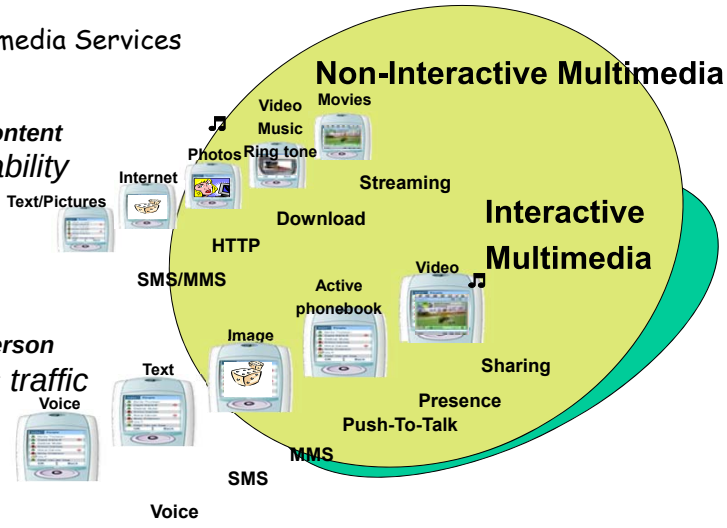
- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Convergent services
- 2.8 Network traffic characterization
- 2.9 Socket programming with TCP and UDP

Convergent Services

Towards Multimedia Services (IMS)

Person-to-Content
known usability patterns

Person-to-Person
dominates traffic growth

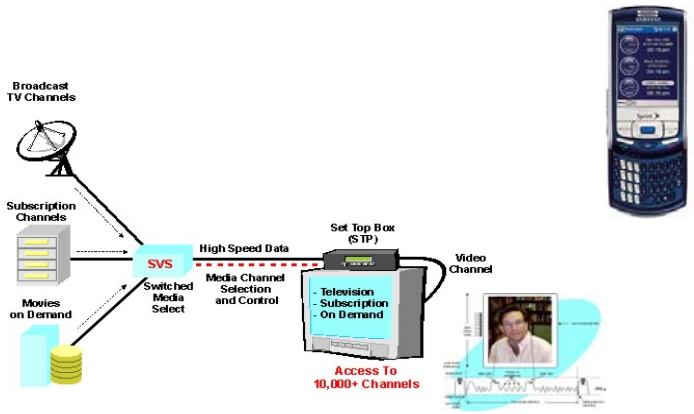


Convergent Services

Convergent Services

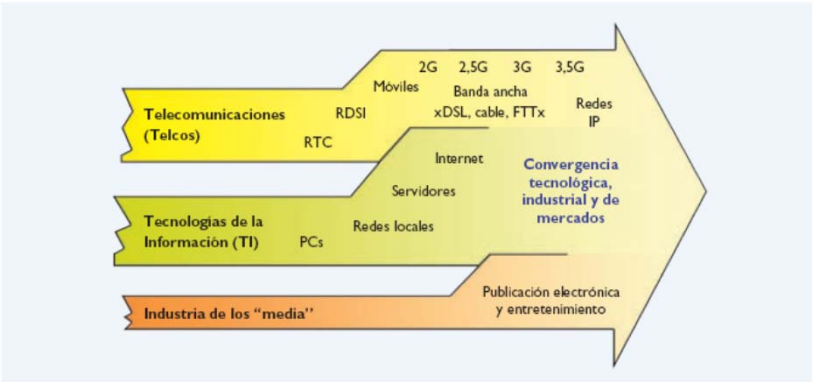
- Corporate voice and video services with high requirements on network resources. Those requirements are supported by the operator's network infrastructure.

Convergent Services

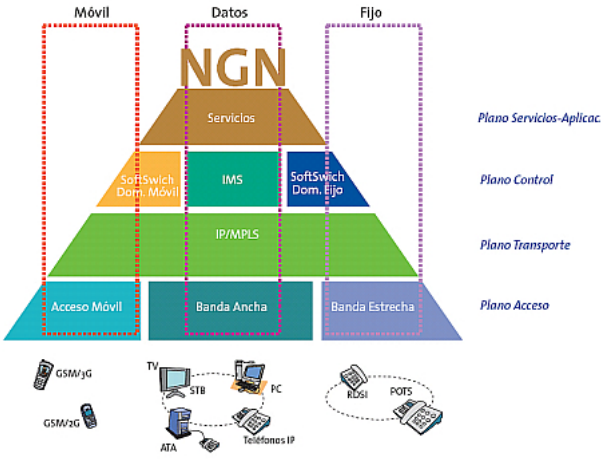


- TV convergence onto the IP network. Operators must offer a high available Backhaul
- Cell phones connected to the IP network. IP communication from/to the mobile end device

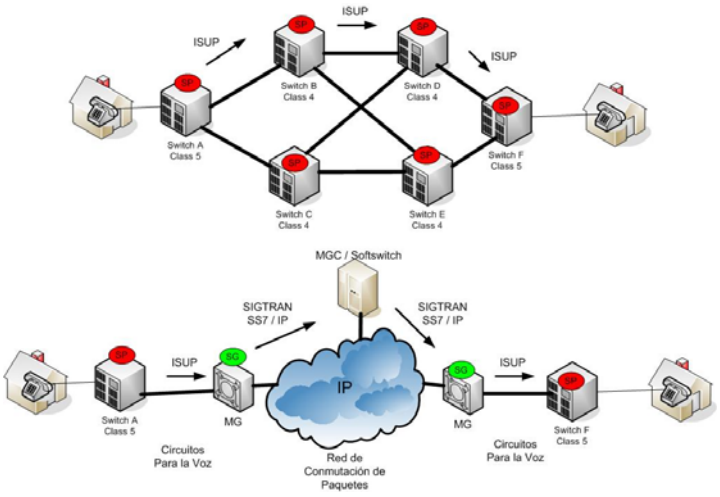
Convergent industries



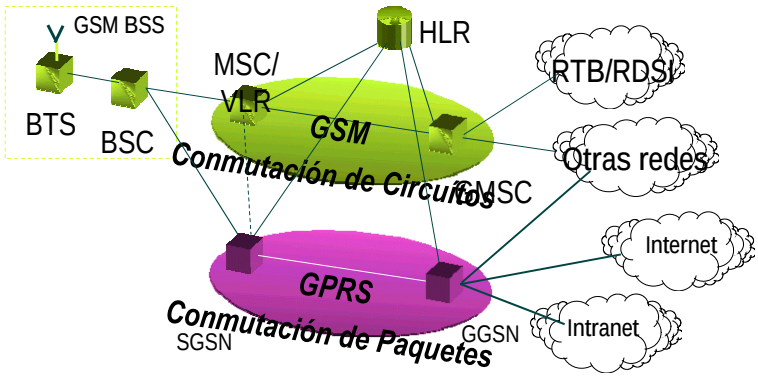
Architecture



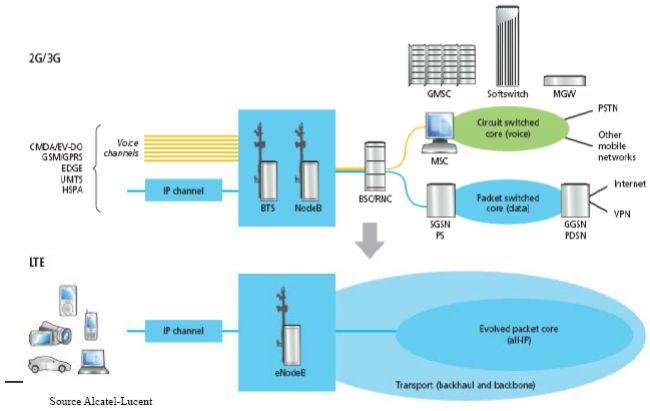
Convergent infrastructures



Convergent infrastructures

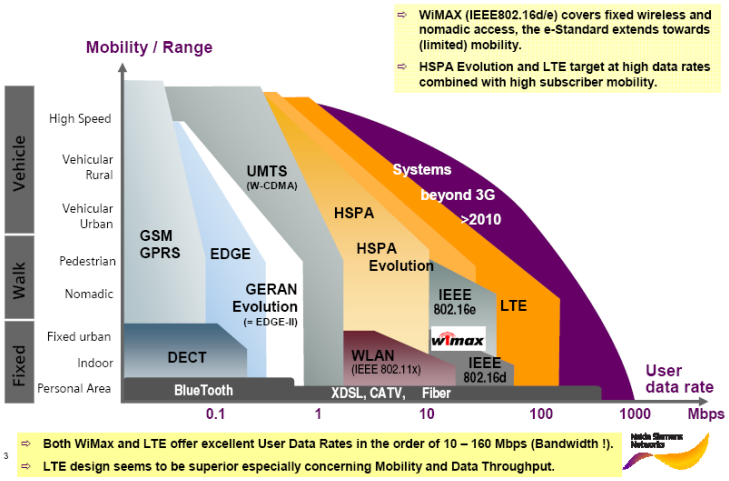


Convergent infrastructures

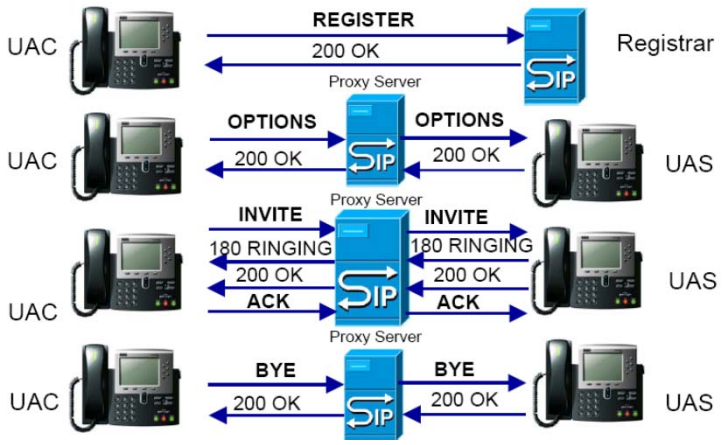


Source Alcatel-Lucent

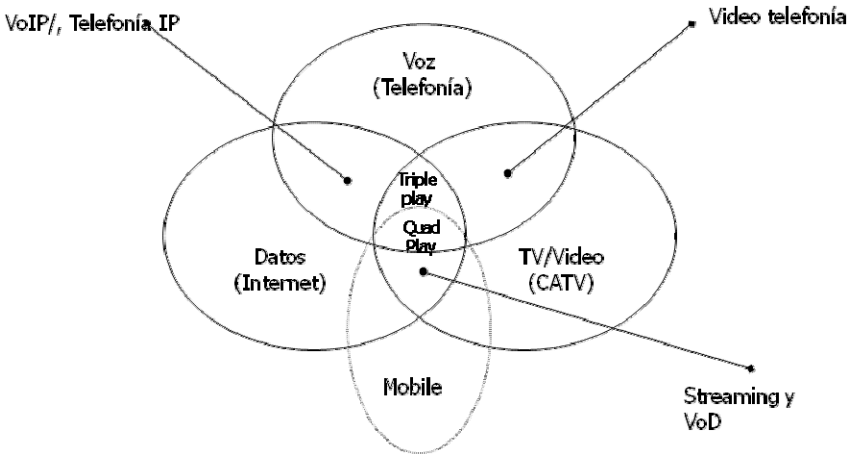
NGN: Evolution



New IP protocols

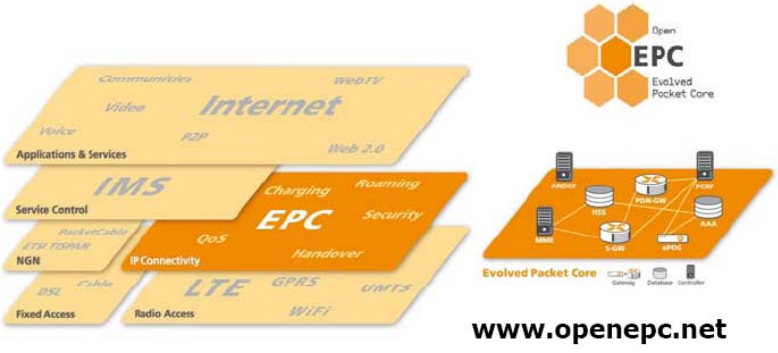


Convergence bills



Total integration

EPC



Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Convergent services
- 2.8 Network traffic characterization
- 2.9 Socket programming with TCP and UDP

Caracterización del tráfico de una Red

Identificar el origen de los Tráficos más altos en la red

Nombre Comunidad de Usuarios	Tamaño Comunidad (# usuarios)	Localización	Aplicaciones Usadas

Caracterización del tráfico de una Red

Identificar el origen de los más altos Almacenamientos

Almacenamiento de Datos	Localización	Aplicaciones	Datos Usados

Caracterización del tráfico de una Red

Determinación del Flujo de Tráfico en la Red

	Destino 1		Destino 2	
	Mbps	Ruta	Mbps	Ruta
Origen 1				
Origen 2				

Flujo de Tráfico entre Cliente-Servidor
Flujo de Tráfico entre peer-to-peer
Flujo de Tráfico entre Servidor-Servidor
Flujo de Tráfico Computación Distribuida

Caracterización del tráfico de una Red

Características de Tráfico de las Aplicaciones

Nombre de la Aplicación	Tipo del Flujo de Datos	Protocolos usados por la Aplicación	Comunidad de Usuarios que utilizan la Aplicación	Datos almacenados (Servidores, Hosts)	Ancho de Banda aproximado requerido por la aplicación	Requerimientos de QoS

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
  - SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Convergent services
- 2.8 Network traffic characterization
- 2.9 Socket programming with TCP and UDP

Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

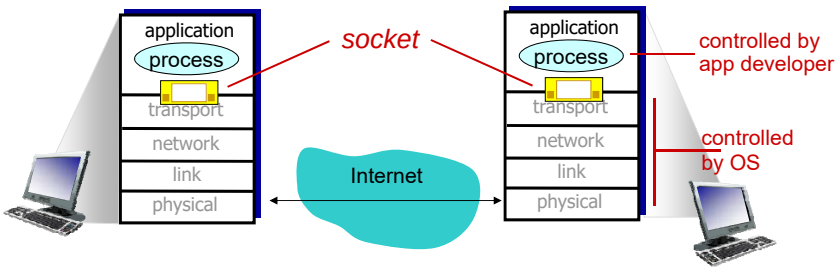
socket

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of **bytes** from one process to another



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:

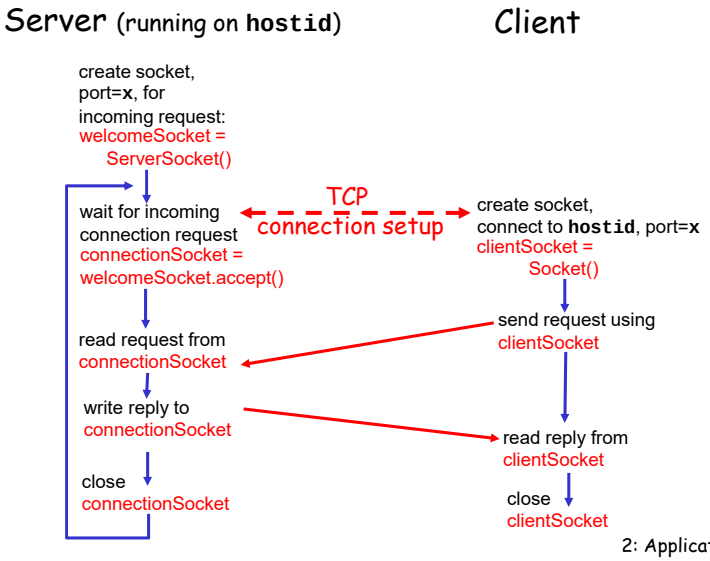
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint

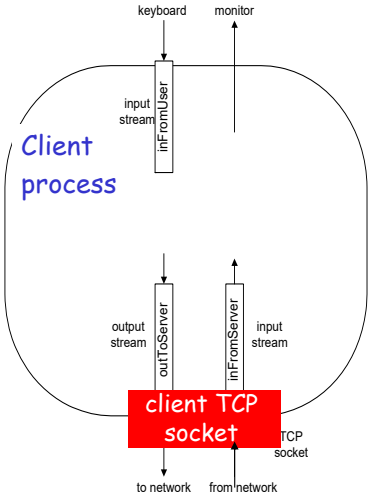
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Client/server socket interaction: TCP



Stream jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.



Socket programming with TCP

Example client-server app:

1) client reads line from standard input (**`inFromUser`** stream) , sends to server via socket (**`outToServer`** stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (**`inFromServer`** stream)

Example: Java client (TCP)

```
import java.io.*;
import java.net.*;
class TCPClient {

   public static void main(String argv[]) throws Exception
   {
      String sentence;
      String modifiedSentence;

      BufferedReader inFromUser =
       new BufferedReader(new InputStreamReader(System.in));

      Socket clientSocket = new Socket("hostname", 6789);

      DataOutputStream outToServer =
       new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java client (TCP), cont.

```
      BufferedReader inFromServer =
        new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));

      sentence = inFromUser.readLine();

      outToServer.writeBytes(sentence + '\n');

      modifiedSentence = inFromServer.readLine();

      System.out.println("FROM SERVER: " + modifiedSentence);

      clientSocket.close();

   }
}
```

Create input stream attached to socket

Send line to server

Read line from server

Example: Java server (TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java server (TCP), cont

```
      DataOutputStream  outToClient =
        new DataOutputStream(connectionSocket.getOutputStream());

      clientSentence = inFromClient.readLine();

      capitalizedSentence = clientSentence.toUpperCase() + '\n';

      outToClient.writeBytes(capitalizedSentence);
    }
  }
}
```

Create output stream, attached to socket

Read in line from socket

Write out line to socket

End of while loop, loop back and wait for another client connection

Socket programming with UDP

UDP: no "connection" between client and server

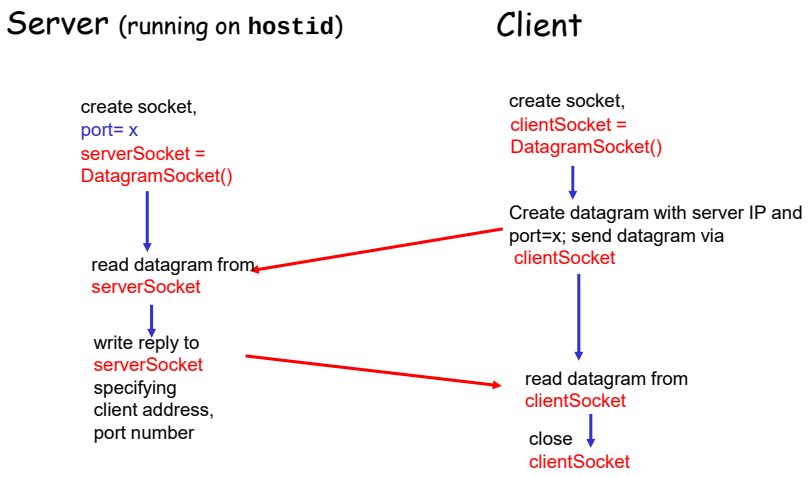
- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet

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

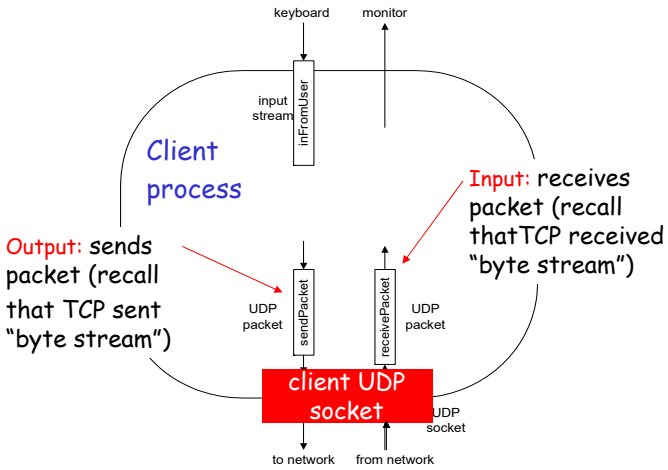
application viewpoint

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

Client/server socket interaction: UDP



Example: Java client (UDP)



Example: Java client (UDP)

```
import java.io.*;
import java.net.*;

class UDPClient {
   public static void main(String args[]) throws Exception
   {
      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

Example: Java client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
                DatagramPacket sendPacket =
                  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

                clientSocket.send(sendPacket);

                DatagramPacket receivePacket =
                   new DatagramPacket(receiveData, receiveData.length);

                clientSocket.receive(receivePacket);

                String modifiedSentence =
                    new String(receivePacket.getData());

                System.out.println("FROM SERVER:" + modifiedSentence);
                clientSocket.close();
                }
            }
```


Example: Java server (UDP)

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

```
            import java.io.*;
            import java.net.*;

            class UDPServer {
              public static void main(String args[]) throws Exception
                {

                DatagramSocket serverSocket = new DatagramSocket(9876);

                byte[] receiveData = new byte[1024];
                byte[] sendData  = new byte[1024];

                while(true)
                  {

                  DatagramPacket receivePacket =
                     new DatagramPacket(receiveData, receiveData.length);

                  serverSocket.receive(receivePacket);
```

Example: Java server (UDP), cont

```
                    String sentence = new String(receivePacket.getData());
Get IP addr
port #, of
sender              InetAddress IPAddress = receivePacket.getAddress();
                    int port = receivePacket.getPort();

                            String capitalizedSentence = sentence.toUpperCase();

                    sendData = capitalizedSentence.getBytes();
Create datagram
to send to client   DatagramPacket sendPacket =
                      new DatagramPacket(sendData, sendData.length, IPAddress,
                                         port);
Write out
datagram
to socket           serverSocket.send(sendPacket);
                  }
                }
              }
```

End of while loop, loop back and wait for another datagram

Chapter 2: Summary

our study of network apps now complete!

- application architectures
  - client-server
  - P2P
  - hybrid
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - FTP
  - SMTP, POP, IMAP
  - DNS
  - P2P: BitTorrent, Skype
- socket programming