Redes de Computadores

LEIC-Alameda

2 – Application Layer

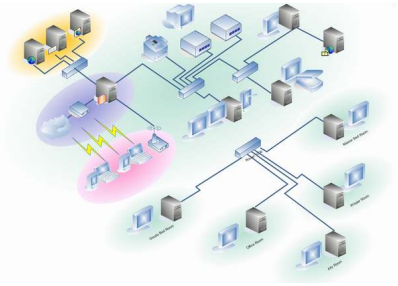
Prof. Paulo Lobato Correia

IST, DEEC – Área Científica de Telecomunicações

1

Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- FTP
- Electronic Mail
- DNS
- P2P Applications

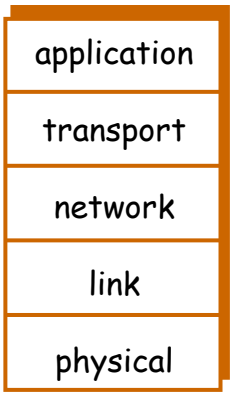


2

TÉCNICO LISBOA

Internet Protocol Stack

- Application: supporting network applications
  - FTP, SMTP, HTTP
- Transport: process-process data transfer
  - TCP, UDP
- Network: routing of datagrams from source to destination
  - IP, routing protocols
- Link: data transfer between neighboring network elements
  - PPP, Ethernet
- Physical: bits "on the wire"
  - RS-232c, V.92



3

TÉCNICO LISBOA

Applications are End-to-End

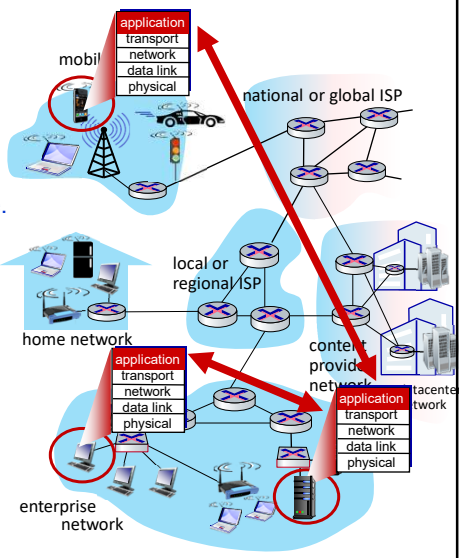
Networks applications:

- Run on (different) *end systems;*
- Communicate over network;

Example: web browser software communicates with web server software.

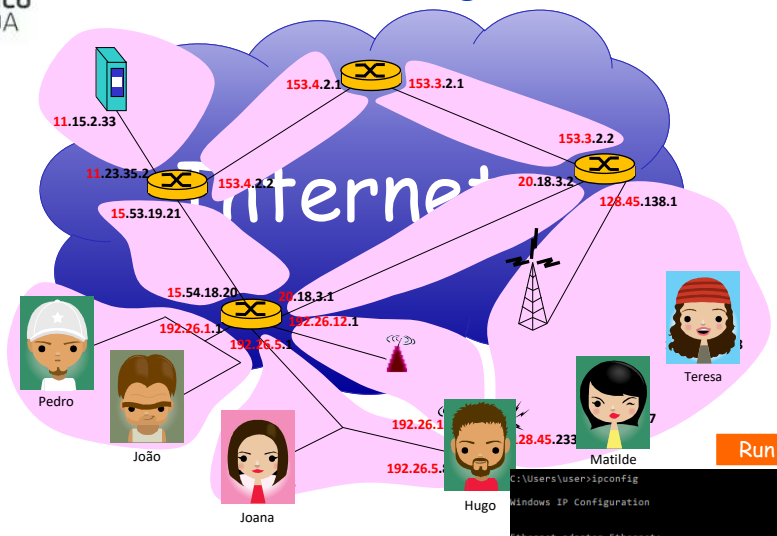
No need to write software for network-core devices !

- Network-core devices do not run user applications;
- Applications on end systems allow for rapid application development and dissemination.



4

Addressing



IPv4 address	193	32	216	9
	11000001	00100000	11011000	00001001

2^{32} = 4.294.967.296 addresses!

Addressing Processes

To receive messages, a process must have an *identifier.*

- Host has a unique 32-bit *IPv4* and/or 128-bit *IPv6* address;

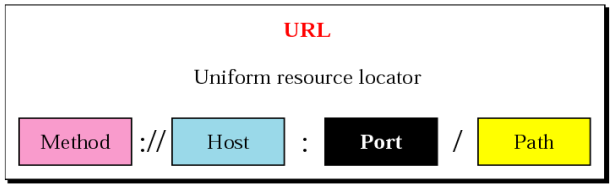
Q: Is the host IP address enough to identify the application process?

- No, *many* processes can be running on same host.

- *Identifier* includes both IP address and port numbers associated with process on host;

- Example of port numbers:
  - HTTP server: **80;**
  - Mail server: **25;**
- To send HTTP message to ***www.tecnico.ulisboa.pt*** web server:
  - IP address: 193.136.128.169
  - Port number: 80

URL: Universal Resource Locator



Methods:

Name	Used for	Example
http	Hypertext (HTML)	http://www.cs.vu.nl/~ast/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	file:///usr/suzanne/prog.c
news	Newsgroup	news:comp.os.minix
news	News article	news:AA0134223112@cs.utah.edu
gopher	Gopher	gopher://gopher.tc.umn.edu/11/Libraries
mailto	Sending e-mail	mailto:JohnUser@acm.org
telnet	Remote login	telnet://www.w3.org:80

Application Layer Protocol

Application layer **protocol defines**:

- Types of messages exchanged
  - e.g., request, response;
- Message **syntax**
  - What fields exist in messages and how the are delimited;
- Message **semantics**
  - Meaning of information in fields;
- **Rules** for when and how application processes send and respond to messages

Public-domain protocols:

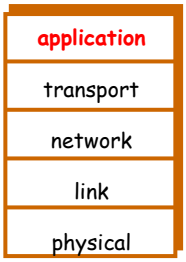
- Defined in RFCs, which allows for **interoperability**;
- e.g., HTTP, SMTP

Proprietary protocols:

- e.g., Skype

TÉCNICO LISBOA

Application Layer uses Transport Layer Services



What is required from the transport layer:

Data integrity

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

Throughput

- Some apps (e.g. multimedia) require a minimum amount of throughput to be "effective";
- Other apps ("elastic apps") make use of whatever throughput they get.

Security

- Encryption, data integrity, …

Timing

- Some apps (e.g. VoIP, interactive games) require low delay to be "effective".

TÉCNICO LISBOA

Transport Service Requirements

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
instant messaging	no loss	elastic	yes and no

Transport Layer Services

TCP service:

- *Connection-oriented:* setup required;
- *Reliable transport* between sending and receiving process;
- *Flow control:* sender won't overwhelm receiver;
- *Congestion control:* control transmission speed when network overloaded;
- Does not provide: timing, minimum throughput guarantees, security.

UDP service:

- *Unreliable data transfer* between sending and receiving processes;
- Does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security.

Q: Why bother? Why is there a UDP?

Application and Transport Protocols

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP 1.1 [RFC 7320]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary HTTP	TCP or UDP
streaming audio/video	[RFC 7320], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

TÉCNICO LISBOA

Securing TCP

Vanilla TCP & UDP sockets:

- no encryption
- cleartext passwords sent into socket traverse Internet in cleartext (!)

Transport Layer Security (TLS)

- provides encrypted TCP connections
- data integrity
- end-point authentication

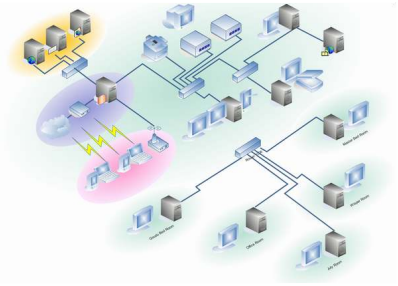
TLS implemented in application layer

- apps use TLS libraries, that use TCP in turn
- cleartext sent into "socket" traverse Internet *encrypted*
- more: Chapter 8

TÉCNICO LISBOA

Objectives

- Principles of Network Applications
- **Socket Programming with TCP and UDP**
- Web and HTTP
- FTP
- Electronic Mail
- DNS
- P2P Applications



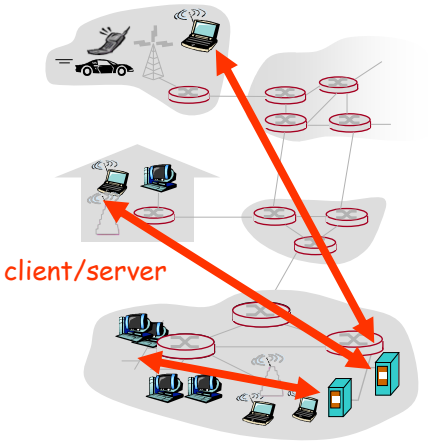
Client-Server Architecture

Server:

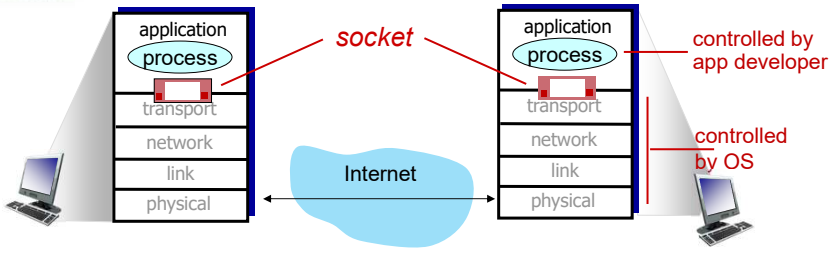
- Always-on host;

Clients:

- Initiate communication with server, specifying server's **IP address** and **port number**;
- May be intermittently connected;
- Do not communicate directly with each other.



Sockets API



- Process sends/receives messages to/from its **socket**
- Socket analogy to a door:
  - Sending process sends message out of the door;
  - Sending process **relies on transport infrastructure** on other side of door which brings message to socket at receiving process;

Sockets API:

- (1) Choice of transport protocol;
- (2) Ability to set a few parameters.

Socket Programming using TCP

Client must contact server:

- Server process must first be running;
- Server must have created socket (door) to welcome client contacts.

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 communication between server and client:
  - Allows server to talk with multiple clients;
  - Source port numbers are used to distinguish clients.

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

Socket Programming with UDP

UDP – no "connection" between client and server:

- No handshaking;
- Sender explicitly includes IP address and port of destination to each packet;
- Server must extract IP address and port of client from the received packet.

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

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

Socket Programming: TCP vs UDP

TCP:

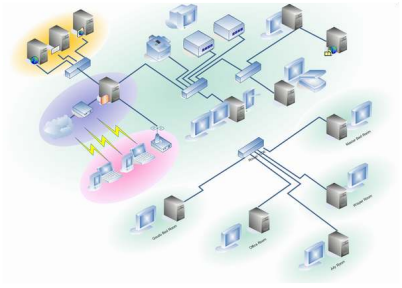
- `read()` and `write();`
- Byte stream (and no byte is lost);
- Bytes read with `read()` may correspond to several `write();`
- Bytes written with `write()` may need to be read with several `read();`

UDP:

- `sendto()` and `recvfrom();`
- Preserves boundary between messages;
- Each message read with `recvfrom()` corresponds to a single `sendto();`
- A message may be lost.

Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- **Web and HTTP**
- FTP
- Electronic Mail
- DNS
- P2P Applications



TÉCNICO LISBOA

WWW: World Wide Web

- The *World Wide Web* (WWW):
  - WEB pages and other resources accessible through the Internet.
- The most popular Internet service; client-server architecture.
- Some dates:
  - **1989** – The **concept** appeared in CERN (*Centre Européan pour Rechèrche Nucleaire*), when **Tim Berners-Lee** concluded that he could not create a research database using a single computer.
    The solution was to have the information spread over a number of computers, but interconnected using **hypertext** and using **URLs** (*Universal Resource Locators*);
  - 1993 – First graphical *browser*: ***Mosaic***;
  - 1994 – ***Netscape***;
  - ...

TÉCNICO LISBOA

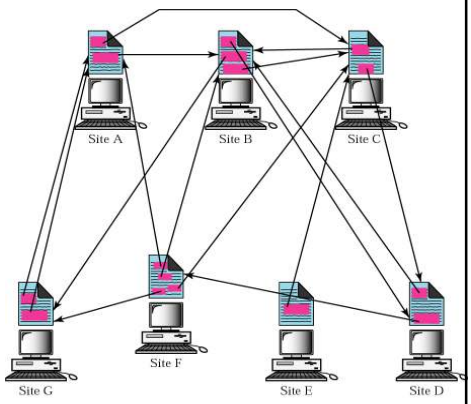
WWW: World Wide Web (HTML + HTTP)

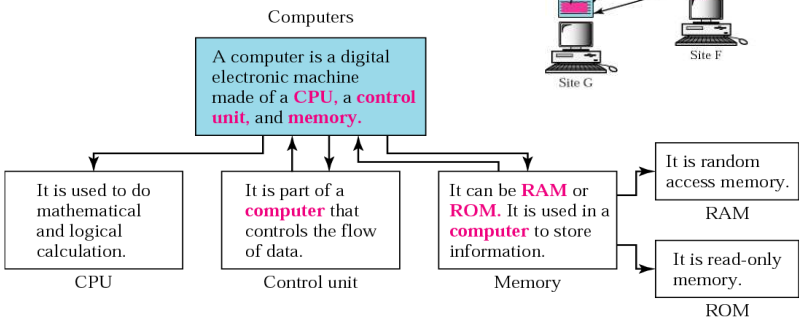
- *HyperText Markup Language* (**HTML**)
  Language used to create (simple) WEB pages.
  Other options:
  - Dynamic HTML – allows mouse-over techniques, layers, …
  - **XML** (*Extensible Markup Language*) – describes how to create a document, including its definition and contents; The syntax is similar to HTML, but allows defining new tags with their own properties.

- *HyperText Transport Protocol* (**HTTP**)
  Protocol used to transfer WEB pages.

TÉCNICO LISBOA

HTML: Hypertext





TÉCNICO LISBOA

HTML

Some HTML tags:

Tag	Description
<html> ... </html>	Declares the Web page to be written in HTML
<head> ... </head>	Delimits the page's head
<title> ... </title>	Defines the title (not displayed on the page)
<body> ... </body>	Delimits the page's body
<h *n*> ... </h*n*>	Delimits a level *n* heading
...	Set ... in boldface
...	Set ... in italics
...	Center ... on the page horizontally
...	Brackets an unordered (bulleted) list
...	Brackets a numbered list
	Starts a list item (there is no)
 	Forces a line break here
<p>	Starts a paragraph
<hr>	Inserts a Horizontal rule
<img src="...">	Displays an image here
 ... 	Defines a hyperlink

Example:

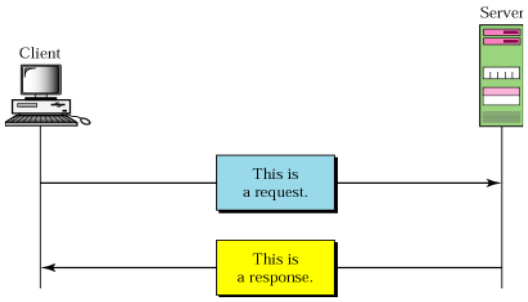
```
<html>

<head>
<title>RC</title>
</head>

<body>
<H2> RC </H2>

<P> Informações: </P>

...

</body>
</html>
```

The WWW Protocol: HTTP

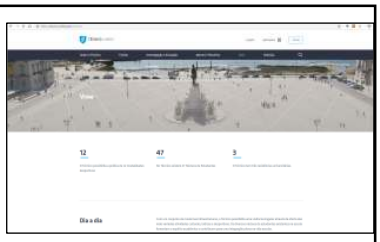
HTTP is the application layer protocol used in the WWW.
It uses **TCP** at the transport layer (default: port 80).

- Client: *browser* makes requests, receives and presents replies;
- Server: answers requests;

stateless - keeps no information about previous requests;



Web and HTTP



Some definitions:

- Web page consists of objects:
  - HTML file;
  - JPEG images;
  - Audio files, …

(objects can be stored on different web servers)

- The base object (HTML file) may reference **other objects**;
- Each object is addressable by a URL (Uniform Resource Locator)

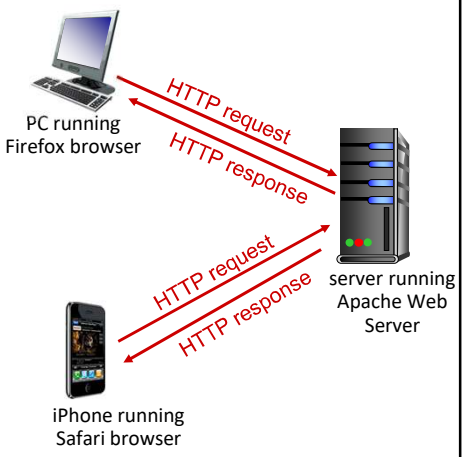
Example: `http://tecnico.ulisboa.pt/pt/viver/`

host name path name

HTTP Overview

HTTP: hypertext transfer protocol:

- Web's application layer protocol;
- Client/Server model:
  - ***Client:*** browser that requests, receives and "displays" Web objects;
  - ***Server:*** Web server sends objects in response to requests.



HTTP Overview

HTTP is "stateless":

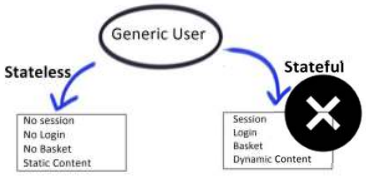
- Server maintains no information about past client requests.

Uses TCP:

- Client initiates TCP connection (*creates socket*) to server, on 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.

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

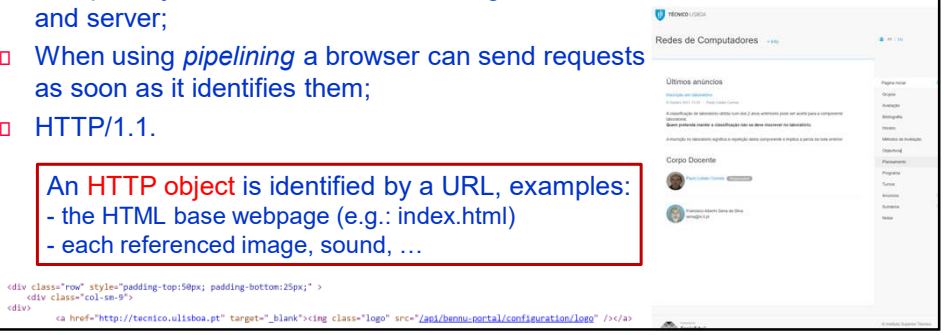
Non-persistent HTTP

- At most one object is sent over a TCP connection;
- Browsers can open parallel connections (typically 5-10);
- HTTP/1.0.

Persistent HTTP

- Multiple objects can be sent over a single TCP connection between client and server;
- When using *pipelining* a browser can send requests as soon as it identifies them;
- HTTP/1.1.

An HTTP object is identified by a URL, examples:
- the HTML base webpage (e.g.: index.html)
- each referenced image, sound, …



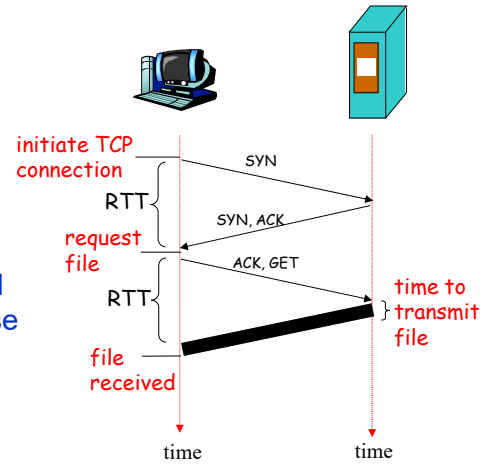
Non-persistent HTTP: Response time

RTT (round trip time):
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 = 2.RTT+transmit time



HTTP Connections

Non-persistent HTTP

HTML + 1 image

Persistent HTTP

Persistent HTTP

TPC: Prof. 3

Non-persistent HTTP issues:

- Requires **2 RTTs per object**;
- OS overhead for *each* TCP connection (e.g., allocate buffers and variables in client and server);
- 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 are sent over the open connection (requiring **1 RTT per each object after the first**);
- With **pipelining**: client sends requests as soon as it encounters a referenced object - **as little as one RTT for all the referenced objects**.

TÉCNICO LISBOA

HTTP Request Message

Two types of HTTP messages: ***(1) request, (2) response***

(1) HTTP request message:

- ASCII (human-readable format)

request line (GET, POST, HEAD commands)

header lines

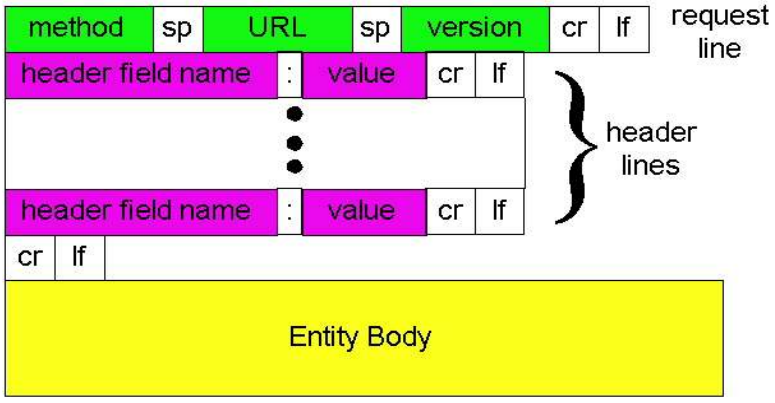
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

(message body)

Check out the online interactive exercises for examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

TÉCNICO LISBOA

HTTP Request Message



HTTP: Method Types

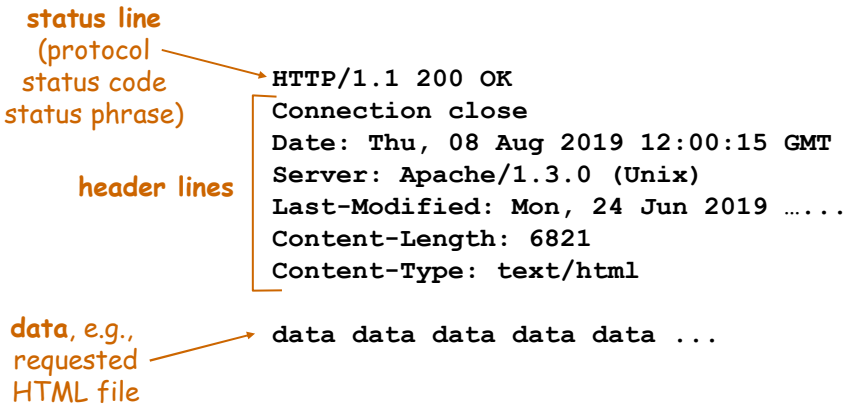
HTTP/1.0 (RFC 1945)

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

HTTP/1.1 (RFC 2616)

- 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 Line Codes

The HTTP response is in the first line of server → client response message.
A few examples of status line 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

Trying out HTTP (client side)

1. Netcat to your favorite Web server:

```
nc -v gaia.cs.umass.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at gaia.cs.umass.edu

2. Type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

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

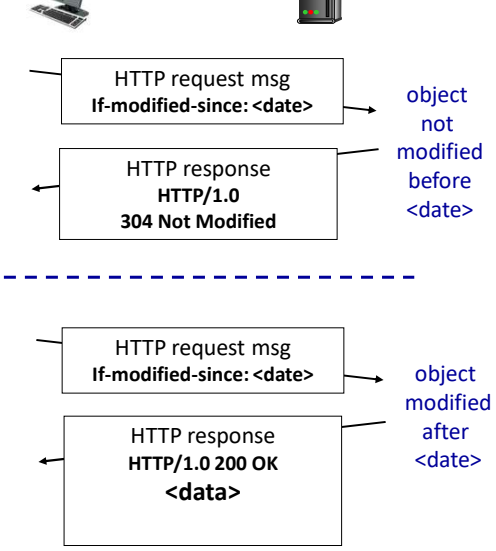
3. Look at response message sent by HTTP server!
   (or use ***Wireshark*** to look at captured HTTP request/response)

HTTP: Browser Cache + Conditional GET

Goal: don't send object if cache already has up-to-date version.

```
HTTP/1.1 200 OK
Date: Mon, 08 Oct 2018 12:00:15 GMT
Last-Modified: Thu, 04 Oct 2018 12:45:26 GMT
…
```

- Client: specify date of cached copy in HTTP request header:
  `If-modified-since: <date>`
- Server: response contains no object if cached copy is up-to-date:
  `HTTP/1.0 304 Not Modified`



HTTP: Cookies
Maintaining State

Many major websites use cookies to maintain some state between transactions.

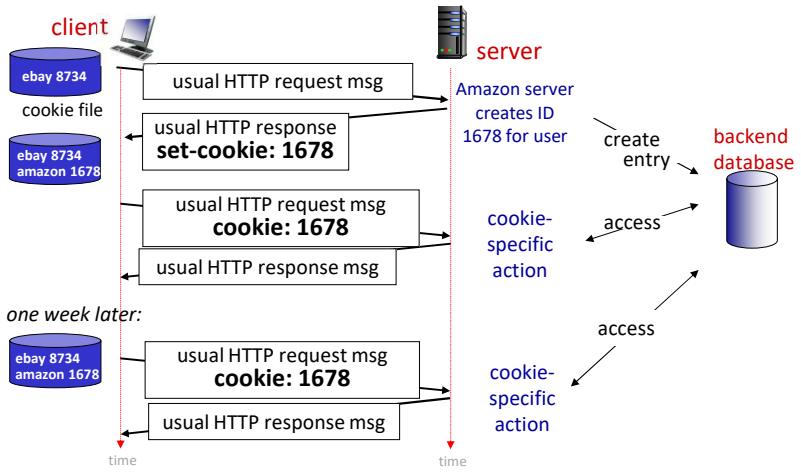
Four components:

1) Cookie header line of HTTP *request* message;
2) Cookie header line in HTTP *response* message;
3) Cookie file kept on user's host, managed by user's browser;
4) Back-end database at Web server.

Example:

- Susan always accesses Internet from her 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.
- Subsequent HTTP requests from Susan to this site will contain cookie ID value, allowing site to "identify" Susan

Maintaining user/server state: cookies



HTTP Cookies: Comments

What cookies can bring:

- Authorization;
- Shopping carts;
- Recommendations;
- User session state (Web e-mail).

How to keep "state":

- Protocol endpoints:
  - **maintain state at sender/receiver** over multiple transactions;
- **Cookies**: HTTP messages carry state.

aside

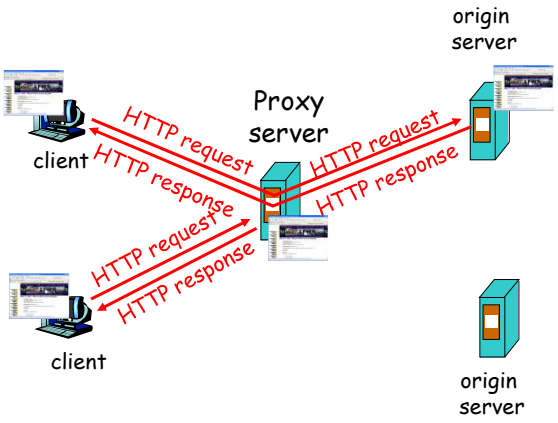
Cookies and privacy:

- Cookies permit sites to learn a lot about you;
- You may supply name and e-mail to sites.

Web Caching: Proxy Server

Goal: satisfy client request without involving the origin server.

- User sets browser to access the Web via a cache;
- Browser sends all HTTP requests to the cache:
  - If object is in cache: cache returns object;
  - Otherwise, cache requests object from origin server, then returns object to client.



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.

TÉCNICO LISBOA

HTTP/2

HTTP 1.0 – RFC 1945, May 1996, T. Berners-Lee, et. al

HTTP 1.1 – RFC 2068, Jan 1997
RFC 2616, Jun 1999
…
RFC 7230, Jun 2014

"HTTP/1.0 does not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for **persistent connections**, or virtual hosts."

HTTP/2 – RFC 7540, May 2015

"This specification describes an optimized expression of the semantics of the Hypertext Transfer Protocol (HTTP), referred to as HTTP version 2 (HTTP/2). HTTP/2 enables a **more efficient use of network resources** and a reduced perception of latency by introducing **header field compression** and **allowing multiple concurrent exchanges on the same connection.** It also introduces unsolicited push of representations from servers to clients.

This specification is an alternative to, but **does not obsolete, the HTTP/1.1 message syntax.** HTTP's existing semantics remain unchanged."

"… it allows interleaving of request and response messages on the same connection and uses an efficient coding for HTTP header fields. It also allows **prioritization** of requests, letting more important requests complete more quickly, further improving performance."
"… **fewer TCP connections can be used.** …"
"HTTP/2 also enables more efficient processing of messages through use of **binary message framing.**"

TÉCNICO LISBOA

HTTP/2

HTTP/2

- **reduce latency** by enabling full request and response multiplexing;
- **minimize protocol overhead** via efficient compression of HTTP header fields;
- add support for **request prioritization**;
- add **server push**.

- **does not modify semantics of HTTP**. All the core concepts, such as HTTP methods, status codes, URIs, and header fields, remain in place.

- **modifies how the data is formatted** (framed) **and transported** between client and server.
  - introduces a new binary framing layer that is not backward compatible with previous HTTP/1.x

HTTP/2

HTTP/2 Request

```
GET / HTTP/1.1
Host: server.example.com
Connection: Upgrade, HTTP2-Settings
Upgrade: h2c
HTTP2-Settings: <base64url encoding of HTTP/2 SETTINGS payload>
```

HTTP 1.1 Response

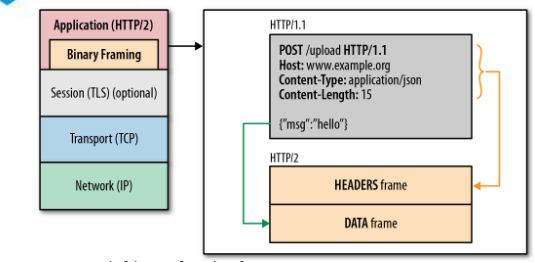
```
HTTP/1.1 200 OK
Content-Length: 243
Content-Type: text/html
...
```

HTTP/2 Response

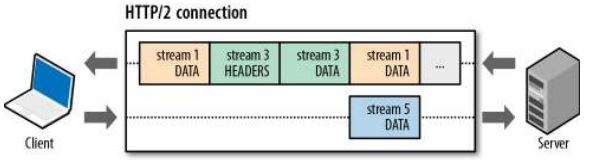
```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: h2c
...
```


66

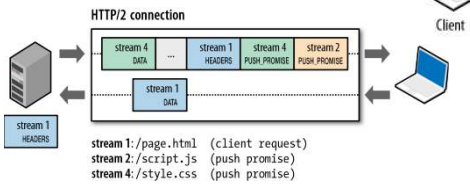
HTTP/2



HTTP/2 binary framing layer



HTTP/2 request and response multiplexing within a shared connection



stream 1:/page.html (client request)
stream 2:/script.js (push promise)
stream 4:/style.css (push promise)

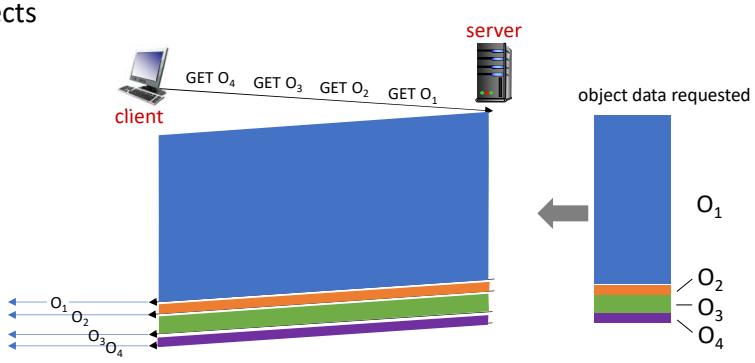
Server initiates new streams (promises) for push resources

[High Performance Browser Networking: What every web developer should know about networking and web performance, *Ilya Grigorik*, O'Reilly Media, Sep 2013]

67

HTTP/2: mitigating HOL blocking

HTTP 1.1: client requests 1 large object (e.g., video file) and 3 smaller objects



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2: mitigating HOL blocking

HTTP/2: objects divided into frames, frame transmission interleaved

O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

TÉCNICO LISBOA

HTTP/3 (2019)

HTTP/2 introduced binary framing and multiplexing to **improve latency without modifying the transport layer**. However, a *lost or reordered packet causes all active transactions to experience a stall* regardless of whether that transaction was impacted by the lost packet.

HTTP/3 adds **security**, **per object error- and congestion-control** (more pipelining) **over UDP.**
Evolution of HTTP over Google's QUIC protocol - *UDP-based, stream-multiplexing, encrypted transport protocol*.

HTTP/3 **will not work on top of TCP**. HTTP/3 will speed the initial connection time taking advantage of *SSL session reuse*, which reduces overhead when multiple substreams are sent over a single connection. Adoption is expected to be gradual, as HTTP/2.

TÉCNICO LISBOA

HTTP/3

HTTP/3 is designed for QUIC, which is a transport protocol that handles streams by itself.

HTTP/2 is designed for TCP, and therefore handles streams in the HTTP layer.

- **HTTP/3 has much faster handshakes** thanks to QUIC vs TCP + TLS.

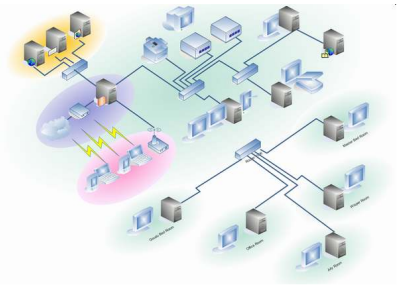
- **HTTP/3 does not exist in an insecure or unencrypted version.**
  HTTP/2 can be implemented and used without HTTPS - even if this is rare on the Internet.

Browser support for HTTP/3

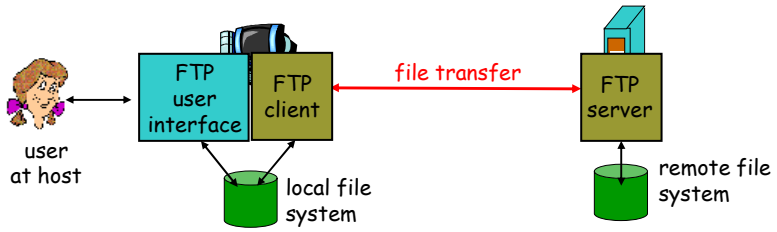
Browser	Version implemented (disabled by default)		Version shipped (enabled by default)		Comment
Chrome	Stable build (79)	December 2019	87[6]	April 2020[25]	Earlier versions implemented other drafts of QUIC
Edge	Stable build (79)	December 2019	87	April 2020	Edge 79 was the first version based on Chromium
Firefox	Stable build (72.0.1)	January 2020	88[9]	April 2021[26]	
Safari	Safari Technology Preview 104	April 2020	–	–	

Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- **FTP**
- Electronic Mail
- DNS
- P2P Applications

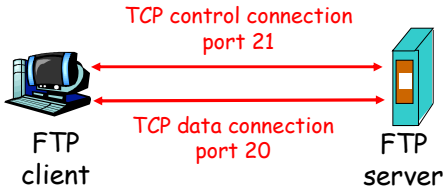


FTP: File Transfer Protocol



- 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: TCP port 21.

FTP: Control and 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 (e.g., *put*, *get*), server opens ***2nd* TCP connection (per file) to client (port 20)**;
- 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: Some User Commands

ascii	*set the mode of file transfer to ASCII (default - transmits seven bits per character)*
binary	*set transfer mode to binary (transmits eight bits per byte - used to transmit files other than ASCII)*
bye	*to exit the FTP environment (same as* quit*)*
cd	*to change directory on the remote machine*
close	*to terminate a connection with another computer*
delete	*to delete a file in the current remote directory (same as* rm *in UNIX)*
get	*to copy one file from the remote machine to the local machine*
help	*to request a list of all available FTP commands*
lcd	*to change directory on your local machine (same as UNIX* cd*)*
ls	*to list the names of the files in the current remote directory*
mkdir	*to make a new directory within the current remote directory*
mget	*to copy multiple files from the remote machine to the local machine*
mput	*to copy multiple files from the local machine to the remote machine*
open	*to open a connection with another computer*
put	*to copy one file from the local machine to the remote machine*
pwd	*to find out the pathname of the current directory on the remote machine*
quit	*to exit the FTP environment (same as* bye*)*
rmdir	*to remove (delete) a directory in the current remote directory*

FTP Commands and Responses

Some FTP commands:

Sent as ASCII text over control channel:

- `USER username`
- `PASS password`
- `LIST` - return list of files in current directory
- `RETR filename` - retrieves (gets) file
- `STOR filename` - stores (puts) file onto remote host.

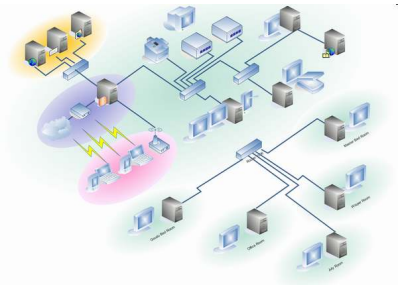
Some return codes:

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`

Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- FTP
- **Electronic Mail**
- DNS
- P2P Applications



TÉCNICO LISBOA

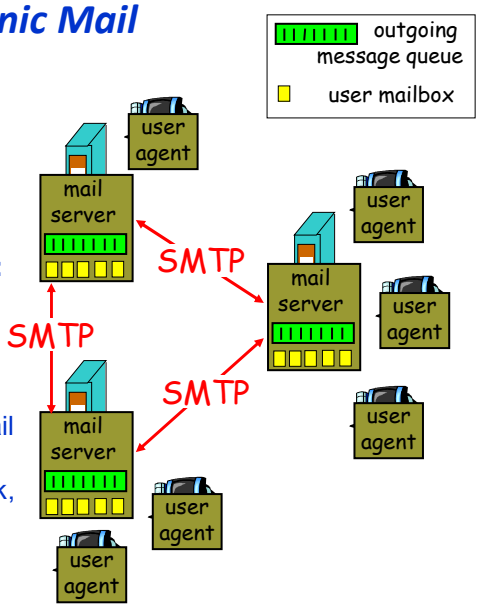
Electronic Mail

Three major components:

- **User agents**
- **Mail servers**
- **Simple mail transfer protocol: SMTP**

User Agent

- The "mail reader"
- Composing, editing, reading mail messages
- Examples: Thunderbird, Outlook, eM, mailbird, Hiri, Spike, …

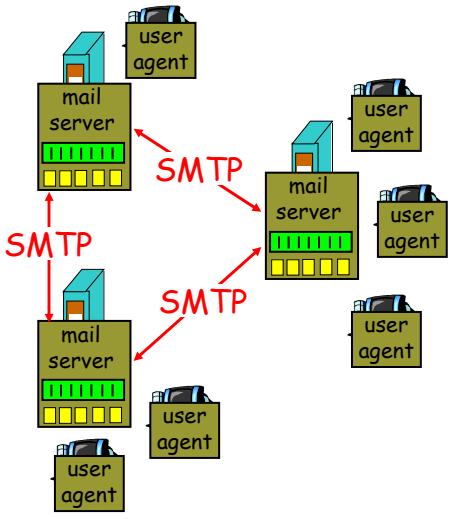


TÉCNICO LISBOA

Electronic Mail: Mail Servers

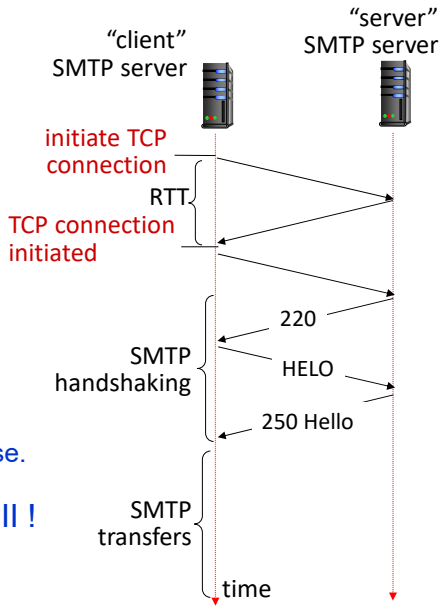
Mail Servers

- 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.

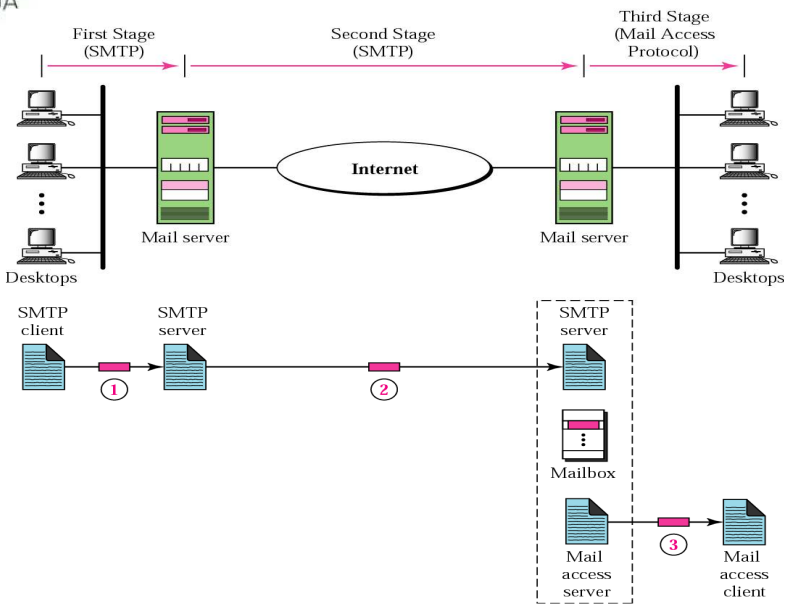


Electronic Mail: SMTP Protocol [RFC 5321]

- Uses **TCP** to reliably transfer email message from client to server;
- Well-known **port**: **25**;
- Three phases of transfer:
  - Handshaking ("greeting");
  - Transfer of messages;
  - Closure.
- Command/Response interaction:
  - Commands: ASCII text;
  - Responses: status code and phrase.
- Messages must be in 7-bit ASCII !



E-mail: Sending Message



SMTP: Example

- **nc servername 25**
- See 220 reply from server;
- Enter commands:
  - HELO or EHLO
  - MAIL FROM
  - RCPT TO
  - DATA
  - QUIT

to send e-mail without using an e-mail client (reader).

83

SMTP: Example

```
S: 220 lx.it.pt
C: HELO meu.computador.pt
S: 250  Hello meu.computador.pt, pleased to meet you
C: MAIL FROM: <aluno@meu.computador.pt>
S: 250 aluno@meu.computador.pt... Sender ok
C: RCPT TO: <plc@lx.it.pt>
S: 250 plc@lx.it.pt ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Teste do SMTP
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 lx.it.pt closing connection
```

Exemplo: Telnet servidor.ist.utl.pt 25

84

TÉCNICO LISBOA

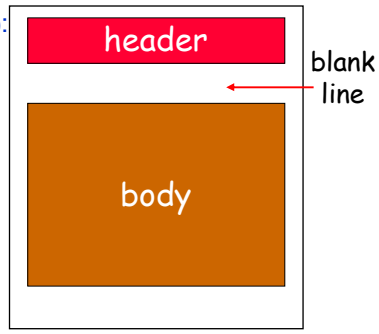
Mail Message Format

SMTP: protocol for ***exchanging email messages*** (RFC 5321, 7504). SMTP

Standard for ***text message format*** (RFC 5322):

- Header lines, e.g.:
  - From:
  - To:
  - Date:
  - Subject:
  - Received: (added by receiving SMTP server)

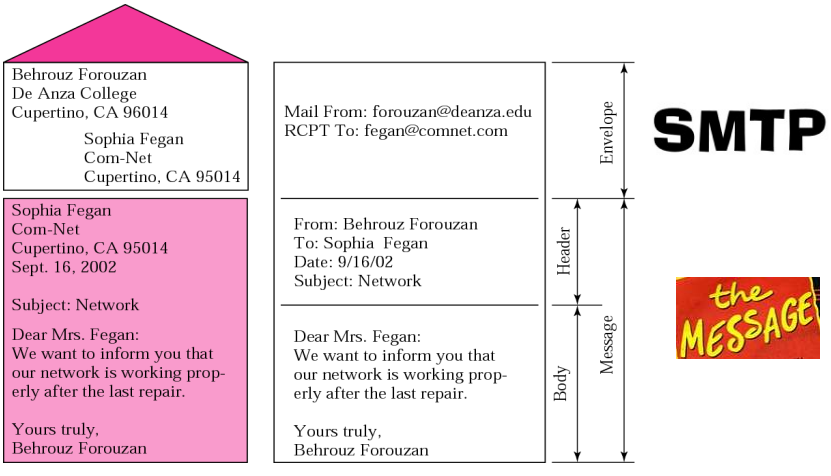
  Different from SMTP commands !
- Body:
  - The "message", ASCII characters only.



TÉCNICO LISBOA

E-mail: Formato da Mensagem

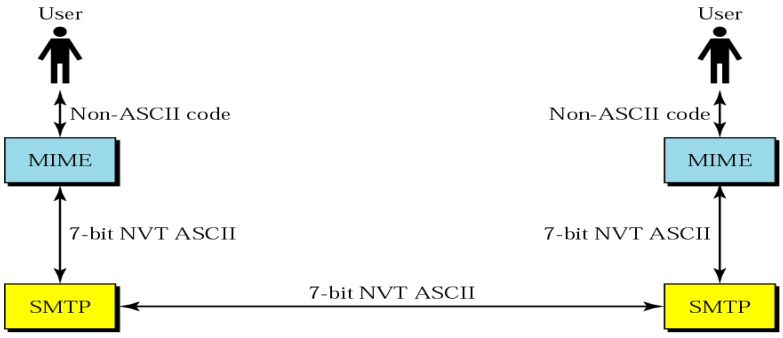
WIRESHARK



SMTP

Message Format: Multimedia Extensions

- MIME: multimedia mail extension, RFC 2045, 2056.



87

Message Format: Multimedia Extensions

- Additional lines in message header declare MIME content type.

RFC 822 headers added by MIME:

Header	Meaning
MIME-Version:	Identifies the MIME version
Content-Description:	Human-readable string telling what is in the message
Content-Id:	Unique identifier
Content-Transfer-Encoding:	How the body is wrapped for transmission
Content-Type:	Type and format of the content

MIME version

method used to encode data

multimedia data type, subtype, parameter declaration

encoded data

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Picture of yummy crepe.
MIME-Version: 1.0
Content-Transfer-Encoding: base64
Content-Type: image/jpeg

base64 encoded data .....
.........................
.....base64 encoded data
```


88

Message Format: Multimedia Extensions

MIME types and subtypes defined in RFC 2045 :

Type	Subtype	Description
Text	Plain	Unformatted text
	Enriched	Text including simple formatting commands
Image	Gif	Still picture in GIF format
	Jpeg	Still picture in JPEG format
Audio	Basic	Audible sound
Video	Mpeg	Movie in MPEG format
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in PostScript
Message	Rfc822	A MIME RFC 822 message
	Partial	Message has been split for transmission
	External-body	Message itself must be fetched over the net
Multipart	Mixed	Independent parts in the specified order
	Alternative	Same message in different formats
	Parallel	Parts must be viewed simultaneously
	Digest	Each part is a complete RFC 822 message

89

Message Format: Multimedia Extensions

MIME multipart message containing formatting and audio.

```
From: elinor@abcd.com
To: carolyn@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abcd.com>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Earth orbits sun integral number of times

This is the preamble. The user agent ignores it. Have a nice day.

--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/enriched

Happy birthday to you
Happy birthday to you
Happy birthday dear <bold> Carolyn </bold>
Happy birthday to you

--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
        access-type="anon-ftp";
        site="bicycle.abcd.com";
        directory="pub";
        name="birthday.snd"

content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```


90

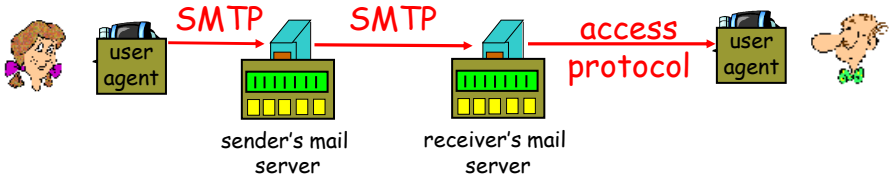
SMTP: Simple Mail Transfer Protocol

- SMTP uses persistent connections;
- SMTP requires message (header & body) to be in 7-bit ASCII.

Comparison with HTTP:

- HTTP: pull application;
- SMTP: push application;
- Both use ASCII command/response interaction, status codes;
- HTTP: objects are encapsulated in response messages;
- SMTP: multiple objects are sent in multipart messages.

Mail Access Protocols

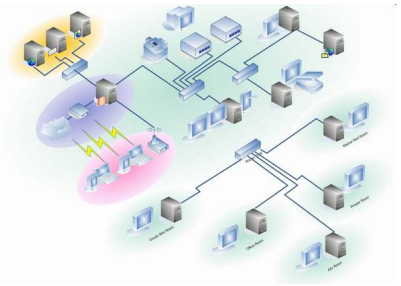


- SMTP: delivery/storage to receiver's server;
- Mail access protocols are used to retrieve e-mail from server:
  - **POP**: Post Office Protocol [RFC 1939]
    - Authorization (agent <-->server) and download;
  - **IMAP**: Internet Mail Access Protocol [RFC 1730]
    - More features (more complex);
    - Manipulation of stored messages on server;
  - **HTTP**: Gmail, Hotmail, Yahoo! Mail, etc.

TÉCNICO LISBOA

Objectives

- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- FTP
- Electronic Mail
- **DNS**
- P2P Applications



TÉCNICO LISBOA

DNS: Domain Name System

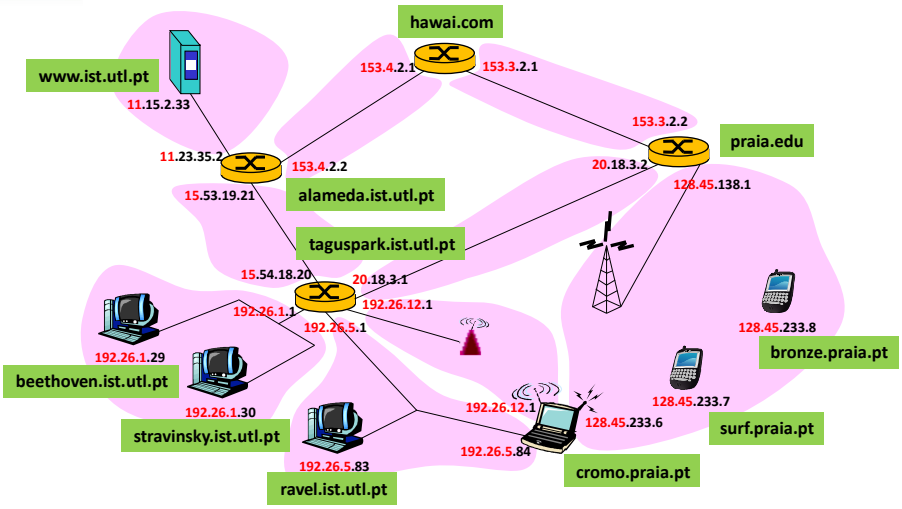
People use many identifiers:
- name, #IST, #cartão cidadão, #passport, …

Internet hosts, routers:
- **IP address** (32 bit) - used for addressing datagrams; e.g., 193.136.128.1;
- **Hostname**, e.g., www.yahoo.com - used by humans;

Q: How to map between IP addresses and hostnames?

```
struct addrinfo hints,*res;
n = getaddrinfo("tejo.tecnico.ulisboa.pt",PORT,&hints,&res);

struct addrinfo{                          // (item in a linked list)
            ...
            socklen_t        ai_addrlen;   // address length (bytes)
            struct sockaddr *ai_addr;      // socket address
            ...   };
struct sockaddr_in {
            sa_family_t    sin_family;    // address family: AF_INET
            u_int16_t      sin_port;      // port (16 bits)
            struct in_addr sin_addr;      // internet address    };
struct in_addr{
            uint32_t       s_addr;        // IPv4 (32 bits)
            };
```

Internet Names: DNS

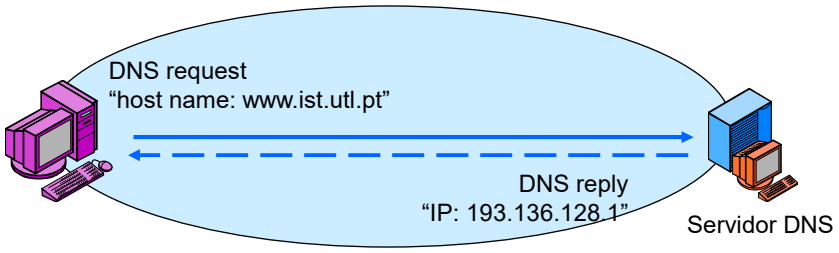


DNS: Domain Name System

How to map between IP addresses and hostnames?

Domain Name System (DNS):

- *Distributed database:* implemented in hierarchy of *name servers;*
- *Application-layer protocol:*
  hosts, routers, name servers communicate to *resolve* names
  - (*Name → Address*) The translation uses the services of **UDP (port 53).**



TÉCNICO LISBOA

DNS

DNS services

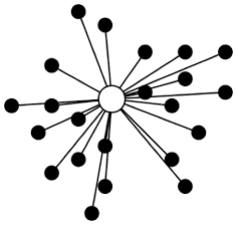
- Hostname to IP address translation;
- Host aliasing:
  - Canonical hostname and alias names;

- 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 issues.

Doesn't *scale!*



TÉCNICO LISBOA

Thinking about the DNS

Humongous distributed database:

- ~ billion records, each simple

Handles many *trillions* of queries/day:

- *many* more reads than writes
- *performance matters:* **almost every Internet transaction interacts with DNS** - msecs count!

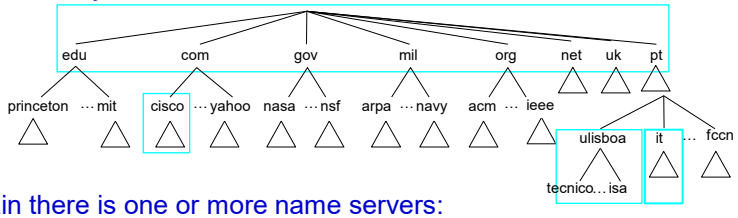
Organizationally, physically decentralized:

- millions of different organizations responsible for their records

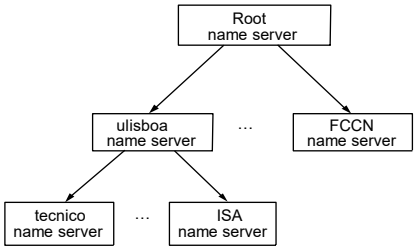
"Bulletproof": reliability, security

DNS: Distributed, Hierarchical Database

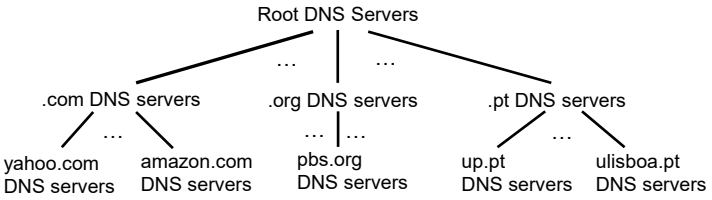
The naming hierarchy is divided into domains:



In each domain there is one or more name servers:



DNS: a distributed, hierarchical database



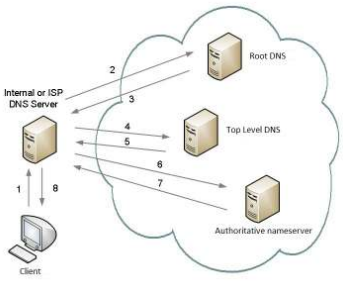
Root

Top Level Domain (TLD)

Authoritative

Client wants IP address for www.amazon.com:

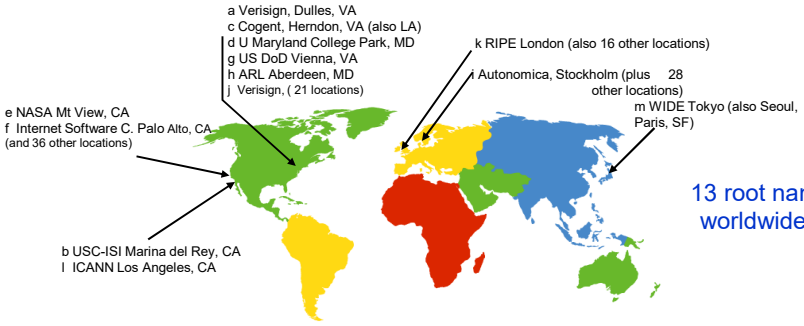
1. client queries **root DNS server** to find .com DNS server
2. client queries .com **TLD DNS server** to get amazon.com authoritative DNS server
3. client queries amazon.com **authoritative DNS server** to get IP address for www.amazon.com



TÉCNICO LISBOA

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

TÉCNICO LISBOA

Root Servers

The authoritative name servers that serve the DNS root zone, commonly known as the "root servers", are a network of hundreds of servers in many countries around the world. They are configured in the DNS root zone as 13 named authorities, as follows.

List of Root Servers

Hostname	IP Addresses	Manager
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	192.228.79.201, 2001:500:84::b	University of Southern California (ISI)
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4	US Department of Defence (NIC)
h.root-servers.net	128.63.2.53, 2001:500:1::803f:235	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:3::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

[https://www.iana.org/domains/root/servers]

DNS: Root Name Servers

13 root name servers, replicated worldwide

107

DNS: Root Name Servers

Porto, Portugal	
Operator	ICANN
IPv4	199.7.83.42
IPv6	2001:500:9f::42
ASN	20144

ICANN

L Root

ISC Internet Systems Consortium

F-root

Lisbon, PT	
Operator	Internet Systems Consortium, Inc.
IPv4	192.5.5.241
IPv6	2001:500:2f::f
ASN	3557

Lisbon, Portugal	
Operator	University of Maryland
IPv4	199.7.91.13
IPv6	2001:500:2D::D
ASN	10886

D Root

Lisbon, PT	
Operator	Verisign, Inc.
IPv4	192.58.128.30
IPv6	2001:503:c27::2:30
ASN	26415

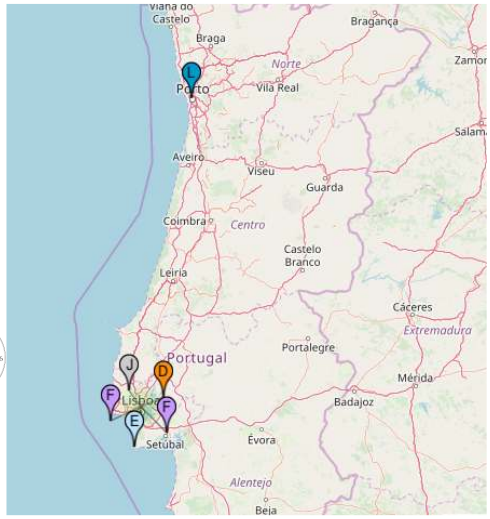
VERISIGN

J Root

Lisbon, PT	
Operator	NASA Ames Research Center
IPv4	192.203.230.10
IPv6	2001:500:a8::e
ASN	21556

NASA Ames Research Center

E Root



[http://www.root-servers.org/]

108

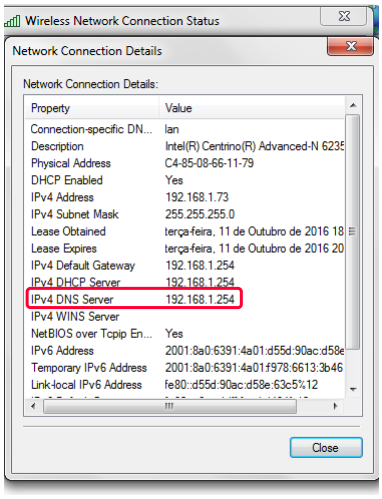
DNS: TLD and Authoritative Servers

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

DNS: Local Name Server

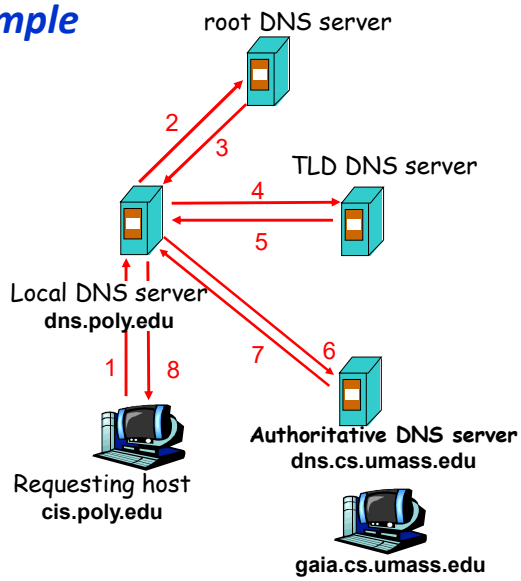
- Does not strictly belong to hierarchy;
- Each ISP (residential ISP, company, university) has one:
  - Also called "**default name server**";
- When host makes DNS query, query is sent to its local DNS server:
  - Acts as proxy;
  - Forwards query into hierarchy.

IPv4 DNS servers:	212.113.177.241 (Unencrypted) 62.169.70.160 (Unencrypted) 192.168.68.1 (Unencrypted)



DNS: 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"

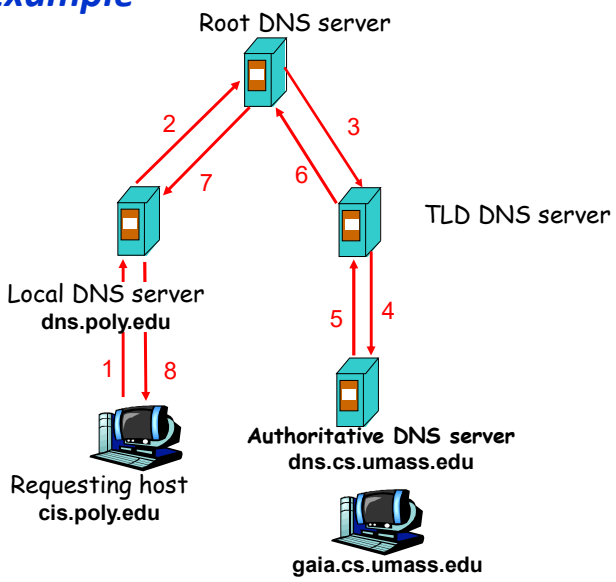
Test applet at:
https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/recursive-iterative-queries-in-dns/index.html

111

DNS: Example

Recursive query:

- Puts burden of name resolution on contacted name server
- Heavy load?

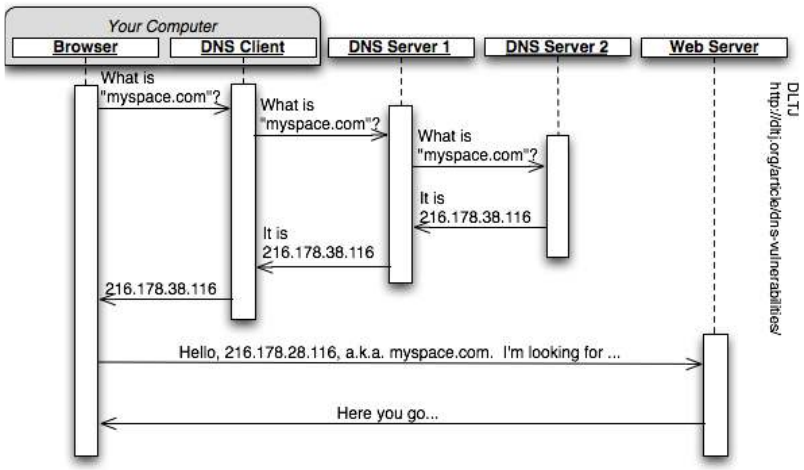


112

TÉCNICO LISBOA

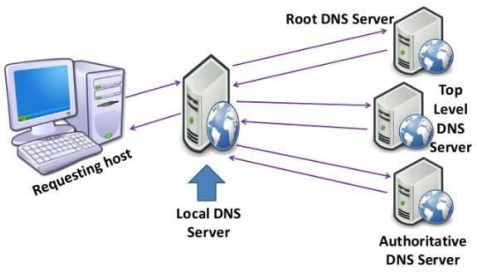
DNS: Example

TPC: Prob. 4



TÉCNICO LISBOA

DNS Caching



- Once (any) name server learns a mapping, it *caches* that mapping:
  - Cache entries timeout (disappear) after some time (typically 2 days);
  - TLD servers are typically cached in local name servers:
    - Thus, *root name servers are not so often visited*.

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 the canonical name
- Type=MX
  - **value** is name of a mailserver associated with **name**

nslookup Test

```
>nslookup www.google.pt
Server:         193.136.128.1
Address:        193.136.128.1#53

Non-authoritative answer:
Name:   www.google.pt
Address: 173.194.45.31
Name:   www.google.pt
Address: 173.194.45.23
Name:   www.google.pt
Address: 173.194.45.24
```

Using "dig" – DNS Lookup Utility

```
>dig www.google.pt

;; ANSWER SECTION:
www.google.pt.          269019  IN      CNAME   www.google.com.
www.google.com.         603749  IN      CNAME   www.l.google.com.
www.l.google.com.       3       IN      A       74.125.39.103
www.l.google.com.       3       IN      A       74.125.39.104
www.l.google.com.       3       IN      A       74.125.39.147
www.l.google.com.       3       IN      A       74.125.39.99

;; AUTHORITY SECTION:
l.google.com.           86391   IN      NS      f.l.google.com.
l.google.com.           86391   IN      NS      g.l.google.com.
l.google.com.           86391   IN      NS      a.l.google.com.
l.google.com.           86391   IN      NS      b.l.google.com.
l.google.com.           86391   IN      NS      c.l.google.com.
l.google.com.           86391   IN      NS      d.l.google.com.
l.google.com.           86391   IN      NS      e.l.google.com.

;; ADDITIONAL SECTION:
a.l.google.com.         171858  IN      A       209.85.139.9
b.l.google.com.         12148   IN      A       74.125.45.9
c.l.google.com.         9179    IN      A       64.233.161.9
d.l.google.com.         17235   IN      A       74.125.77.9
e.l.google.com.         80225   IN      A       209.85.137.9
f.l.google.com.         18235   IN      A       72.14.235.9
g.l.google.com.         86333   IN      A       74.125.95.9
```


Using "dig" – DNS Lookup Utility

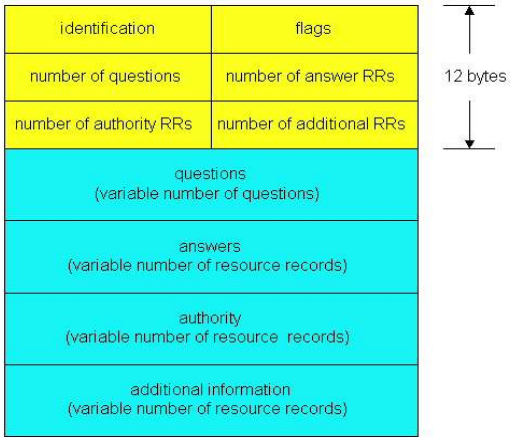
```
>dig www.tecnico.ulisboa.pt

;; QUESTION SECTION:
;www.tecnico.ulisboa.pt.                IN      A

;; ANSWER SECTION:
www.tecnico.ulisboa.pt. 3600    IN      A       193.136.128.169

;; AUTHORITY SECTION:
tecnico.ulisboa.pt.     3600    IN      NS      a.ul.pt.
tecnico.ulisboa.pt.     3600    IN      NS      ns1.tecnico.ulisboa.pt.
tecnico.ulisboa.pt.     3600    IN      NS      ns2.tecnico.ulisboa.pt.

;; ADDITIONAL SECTION:
a.ul.pt.                115     IN      A       194.117.0.150
ns1.tecnico.ulisboa.pt. 3600    IN      A       193.136.128.1
ns2.tecnico.ulisboa.pt. 3600    IN      A       193.136.128.2
a.ul.pt.                115     IN      AAAA    2001:690:21c0:a::150
ns1.tecnico.ulisboa.pt. 3600    IN      AAAA    2001:690:2100:1::53:1
ns2.tecnico.ulisboa.pt. 3600    IN      AAAA    2001:690:2100:1::2
```

DNS Protocol, Messages

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

Message header:

- **Identification**: 16 bit number identifying the query; the reply to this query uses the same number;
- **Flags**:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS uses UDP, port 53

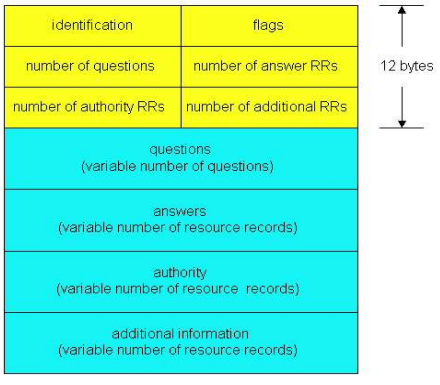
DNS Protocol, Messages

Name and type fields for a query

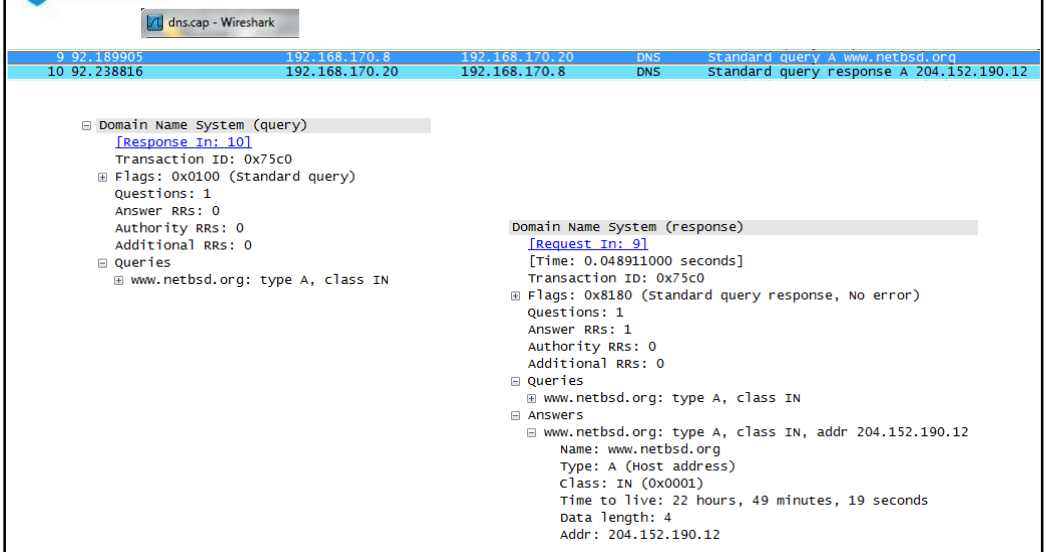
RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



DNS Protocol



Inserting Records into DNS

- Example: new startup “Network Utopia”
- Register name networkuptopia.com at *DNS registrar* :
  - Provide names and IP addresses of authoritative name servers (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

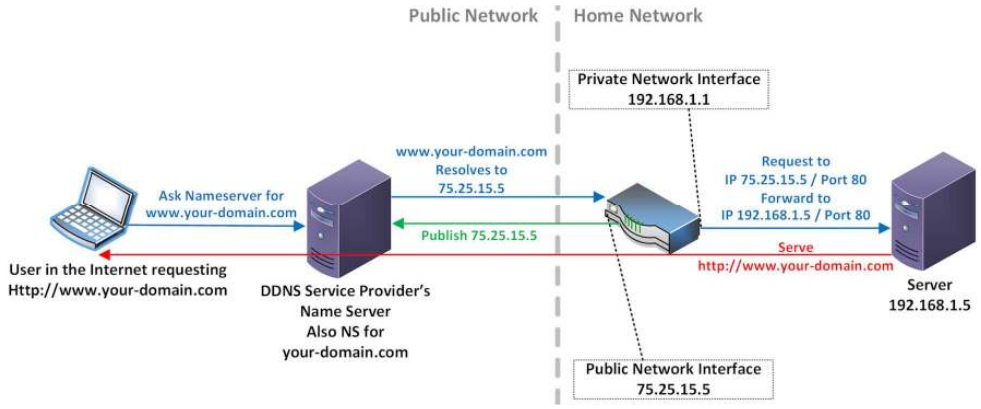
Dynamic DNS

- DDNS allows to update the IP address of a domain in real time (instead of in a few days);
- Domain name can be assigned to a computer with a varying IP address;

Dynamic DNS is a method that allows you to **notify a Domain Name Server (DNS) to change your active DNS configuration on a device** such as a router or computer of its configured hostname and address. It is most useful when your computer or network obtains a new IP address lease and you would like to dynamically associate a hostname with that address, without having to manually enter the change every time. Since there are situations where an IP address can change, it helps to have a way of ***automatically updating hostnames that point to the new address every time***.

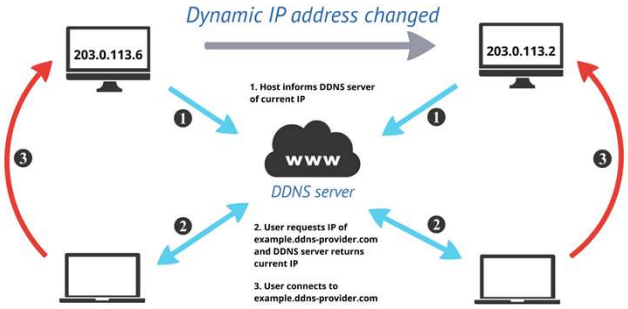
Dynamic DNS

- DDNS allows to update the IP address of a domain in real time (instead of in a few days);
- Domain name can be assigned to a computer with a varying IP address;



Dynamic DNS

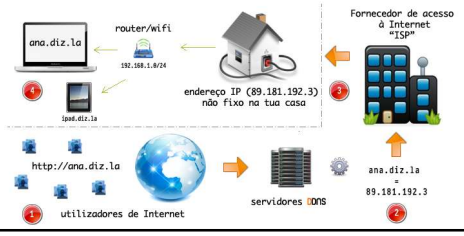
- Other sites on the Internet can establish connections to a machine, without needing to track the IP address themselves;
- It makes servers with dynamic IP addresses accessible on the Internet;
- Allows a domain name to point to a PC whose IP address changes;
- Allows to run servers at home – Internet, Email, RC project, …



DDNS: How to Use

Dynamic NS services can be found in many places online, for varying costs, but they all work more or less the same way:

- Sign up for a DDNS account and pick a hostname (check: dnslookup.me/dynamic-dns/);
  - Signing up will get you a simple hostname, and by configuring your router, your ISP-assigned and ever-changing IP will be updated automatically.
- Enter DDNS registration information in the router or use a DDNS client software, and setup the router and Web service to use the DDNS configuration
  - Enable DDNS and specify the hostname you created;



DNS security

DDoS attacks

(DNS Denial of Service)

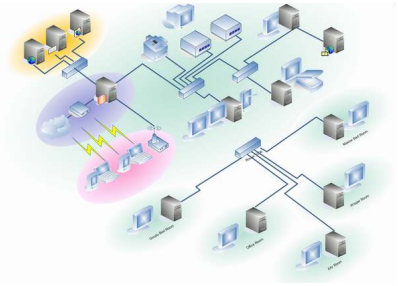
- Bombard root servers with traffic
  - not successful to date
  - traffic filtering
  - local DNS servers cache IPs of TLD servers, allowing root server bypass
- Bombard TLD servers
  - potentially more dangerous

Spoofing attacks

- Intercept DNS queries, returning bogus replies
  - DNS cache poisoning
  - RFC 4033: DNSSEC authentication services

Objectives

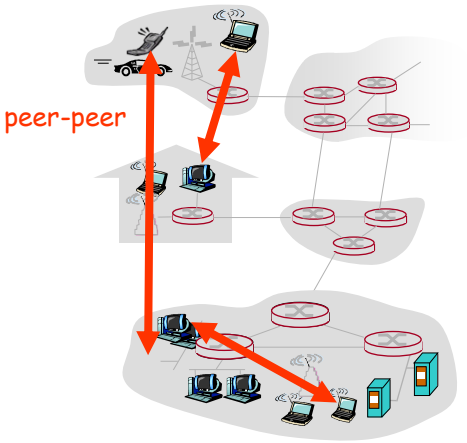
- Principles of Network Applications
- Socket Programming with TCP and UDP
- Web and HTTP
- FTP
- Electronic Mail
  - SMTP, POP3, IMAP
- DNS
- **P2P Applications**



Pure P2P Architecture

- There is *not* an always-on server;
- Arbitrary end systems communicate directly;
- Peers are intermittently connected and can change IP addresses.

- Three examples:
  - File distribution;
  - Searching for information;
  - Skype.



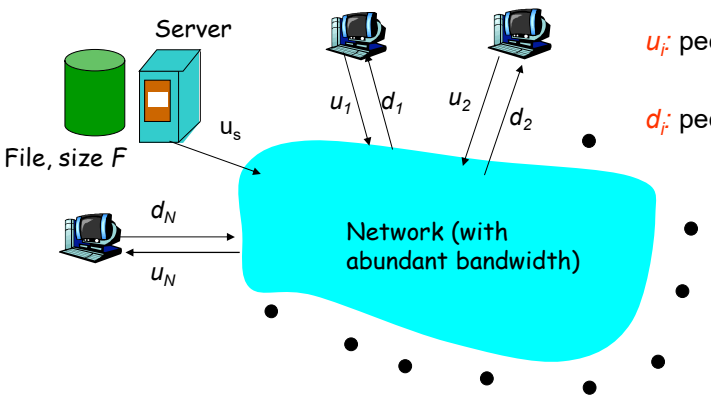
File Distribution: Server-Client vs P2P

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

u_s: server upload bandwidth

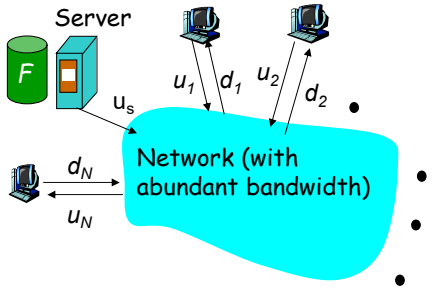
u_i: peer i upload bandwidth

d_i: peer i download bandwidth



File Distribution Time: Server-Client

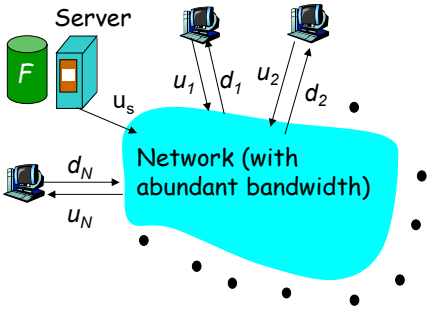
Client-Server



136

File Distribution Time: Server-Client

- Server 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 $= t_{cs} = \max\{ NF/u_s, F/\min(d_i) \}$

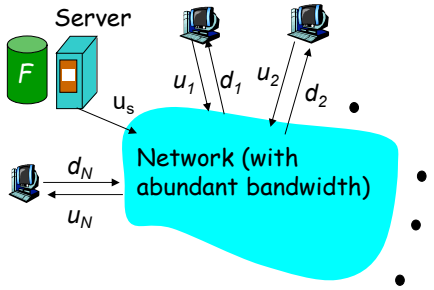
increases linearly in N (for large N)

137

TÉCNICO LISBOA

File Distribution Time: P2P

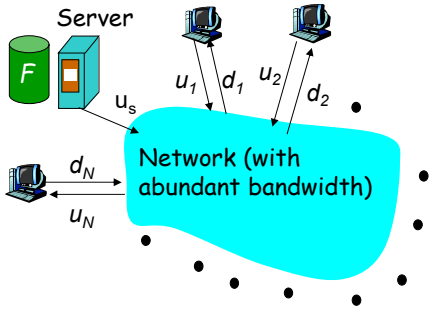
Peer to peer



TÉCNICO LISBOA

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: $u_s + \Sigma u_i$

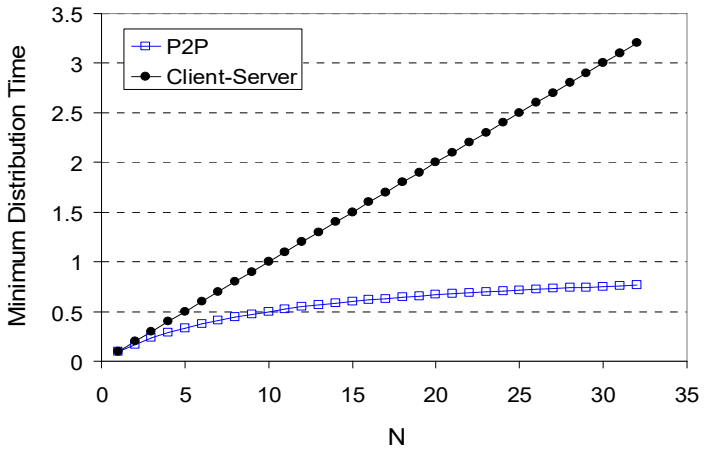


$$t_{P2P} = \max \{ F/u_s, F/\min(d_i) , NF/(u_s + \Sigma u_i) \}$$

Server-Client vs. P2P: Example

TPC: Prob. 5

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



140

Summary

- 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, DNS;
  - P2P.

143

TÉCNICO LISBOA

Summary

Learned about *protocols*

- Typical request/reply message exchange:
  - **Client** requests info or service;
  - **Server** responds with data, status code.
- Message formats:
  - **Headers**: fields giving information about data;
  - **Data**: info being communicated.

Important issues:

- Control vs. data messages: in-band, out-of-band;
- Centralized vs. decentralized;
- Stateless vs. stateful;
- Reliable vs. unreliable message transfer.