Chapitre 3
Application Layer
(Elastic Applications)

Laurent Schumacher (UNamur)

Dernière mise-à-jour : 11 mars 2021

UNIVERSITÉ DE NAMUR

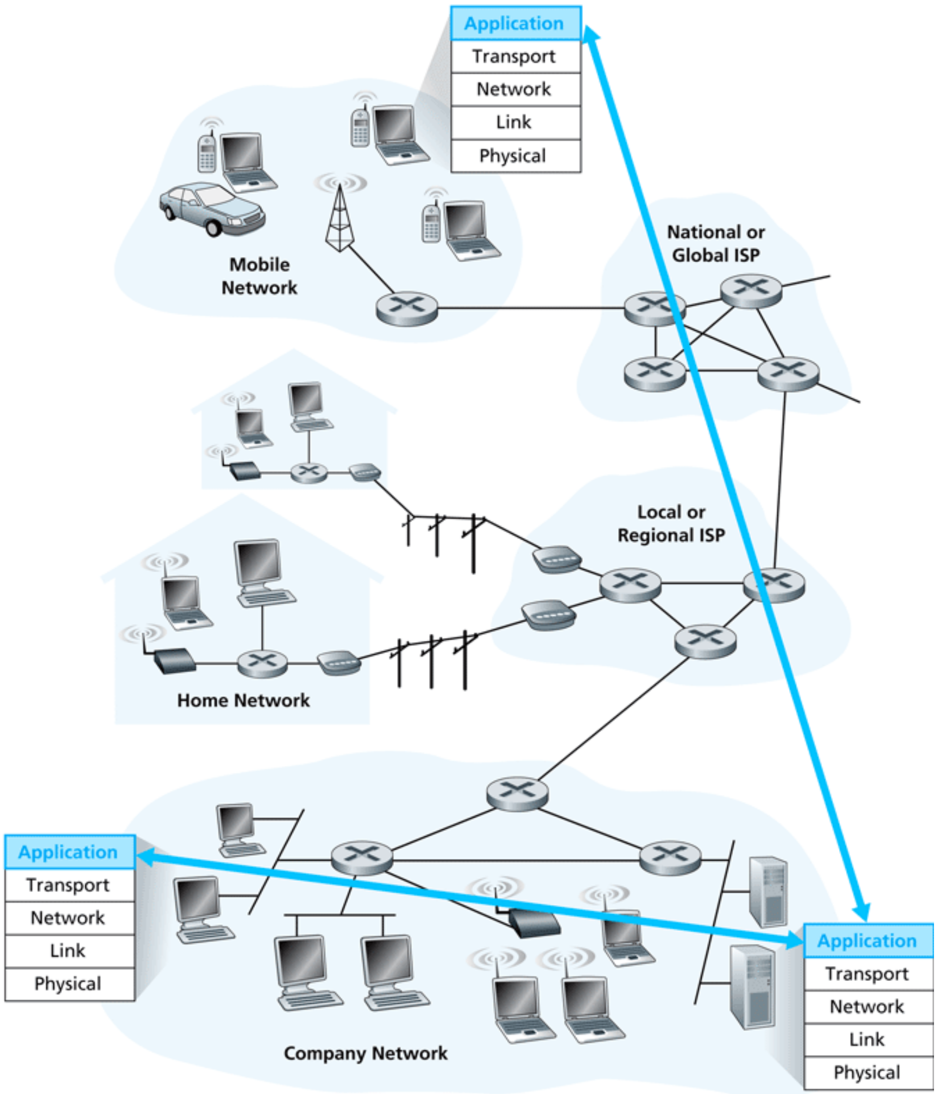
Outline

- Principles of application layer protocols
- Web browsing and HTTP
- Domain name resolution (DNS)
- File transfer (FTP)
- Electronic Mail: SMTP, POP3, IMAP
- Content distribution: Web caching, CDN, P2P
- Socket programming with TCP
- Socket programming with UDP
- Building a simple Web server

Principles of application layer protocols
Some jargon

- *Process*: program running within a host
  - Within same host, two processes communicate using interprocess communication (defined by OS).
    - Example: **`du –k –s * | grep somebody`**
  - Processes running in different hosts communicate with an application-layer protocol
- *User agent*: interfaces with user “above” and network “below”.
  - Implements user interface and application-level protocol
  - Examples
    - Web: browser
    - E-mail: mail reader
    - Streaming audio/video: media player

Principles of application layer protocols
Some jargon



- Network applications
  - Distributed communicating processes
  - Running on end systems
  - Exchange messages to implement application
  - Examples: web browsing, e-mail, P2P file sharing, instant messaging, etc.
- Application layer protocol
  - Define format and order of messages exchanged by applications
  - Define actions taken on transmission or receipt of a message
- Web application

  = browser (Chrome, Safari, IE)

  + server (Apache, Microsoft)

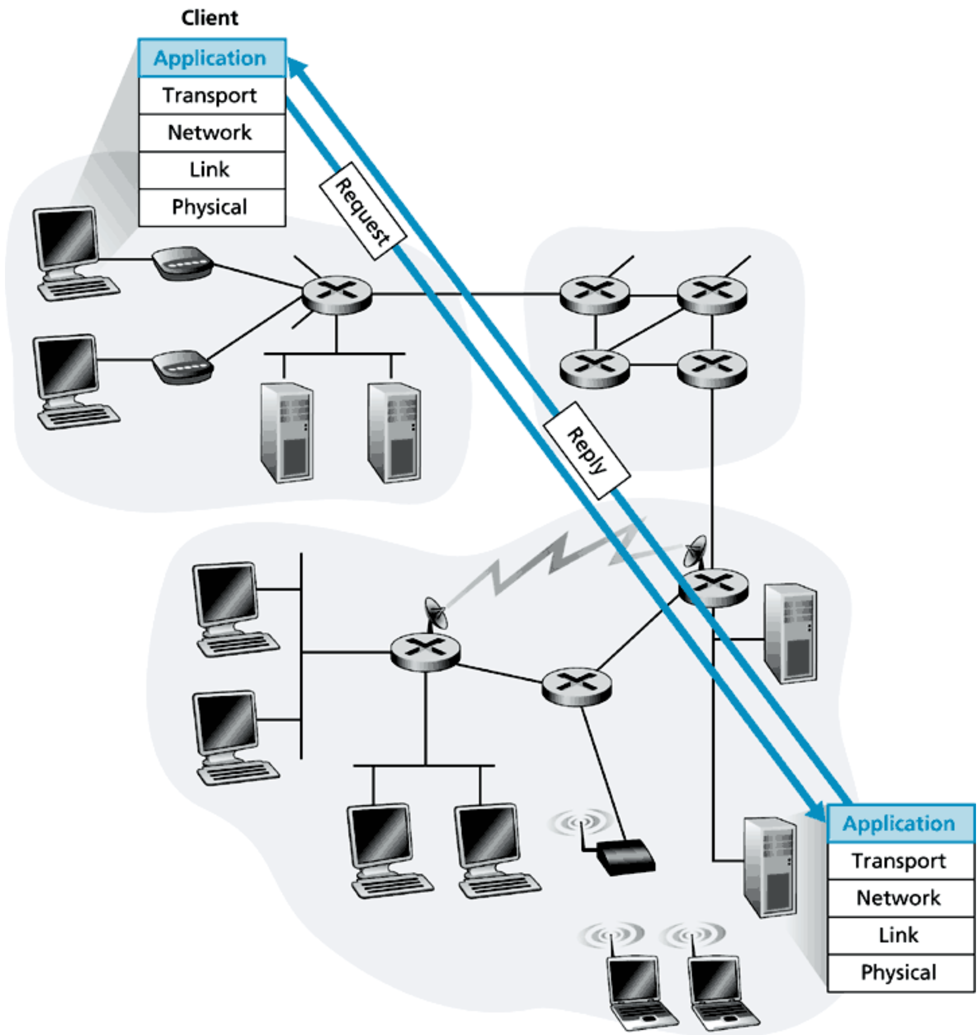
  + application-layer protocol (HTTP)

  + document format (HTML)

Principles of application layer protocols
Role of protocol

- Defines how application's processes running on different end systems pass messages to each other
- Defines
  - Types of messages exchanged, request and responses
  - **Syntax** of various message types, such as fields in the message and fields are delineated
  - **Semantics** of the fields, e.g. their meaning
  - **Rules** for determining when and how a process sends/responds to a message
- Two types of protocol
  - Public-domain
    - Described in RFCs
    - Enable interoperability
    - Example: HTTP, SMTP
  - Proprietary
    - Usually no interoperability
    - Example: multimedia content, Skype, KaZaA, etc

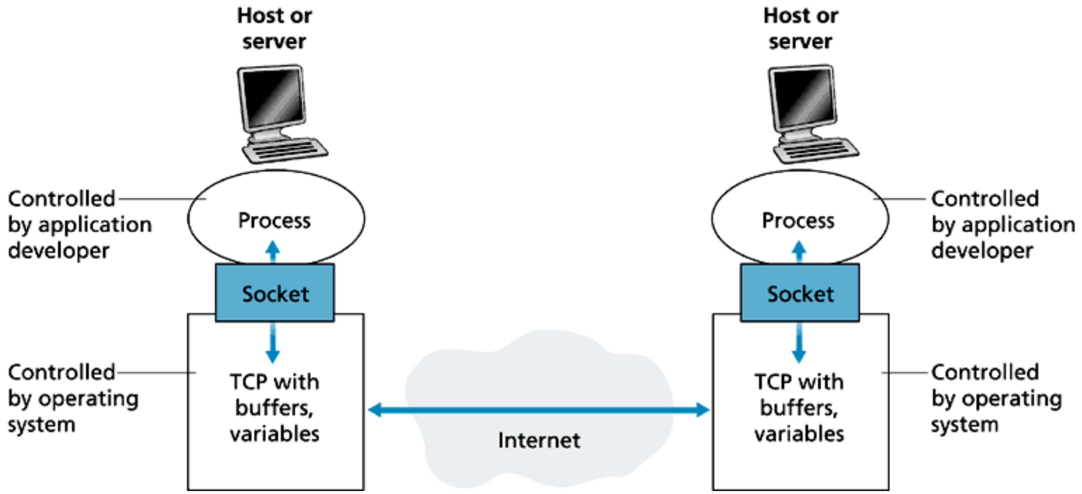
Principles of application layer protocols
Client-server paradigm



- The typical network application has two pieces, a client and a server
- Client
  - Typically requests service from server
  - Initiates contact with server ("speaks first")
  - Web browser, mail reader
- Server
  - Provides requested service to client
  - Web server sends requested Web page
  - Mail server delivers e-mail

Principles of application layer protocols
Processes communicating accross networks

- Socket
  - Interface between application and transport layer within a host
  - Also referred to as API (Application Programming Interface)
  - Process sends/receives messages to/from its socket
  - The socket is analogous to a door
    - Sending process shoves message out door
    - Sending process assumes transport infrastructure on other side of door which brings message to socket at receiving process



- Application developper controls
  - Choice of transport protocol
  - Transport-layer parameters (buffer size, segment size)

Principles of application layer protocols
Adressing processes

- For a process to receive messages, it must have an identifier
- Every host has a unique **IP address**
- Does the IP address of the host on which the process runs suffice for identifying the process?
- No, many processes can be running on same host
- Identifier includes both the IP address and **port numbers** associated with the process on the host.
- Example well-known port numbers
  - HTTP(S) server: 80/443
  - FTP server: 20, 21
  - Mail server: 25/587 (sending), 110 (POP3 reading)
- Ephemeral ports (random, > 1,024) at client

Principles of application layer protocols
What communication services for an App?

- **Reliable data transfer (no data loss)**
  - Some applications can tolerate losses (audio, video)
  - Other expect 100% reliability (file transfer, e-mail, finance)
- **Bandwidth**
  - Some apps have minimum bandwidth requirements to be effective (multimedia)
  - Example: telephony PCM 64 kbps, GSM 9-13 kbps, streaming HDTV 8 Mbps, SDTV 1 Mbps
  - Other apps use available bandwidth → **elastic** apps

- **Latency**
  - Some apps require low delays to be effective (VoIP telephony, interactive games)
  - End-to-end (E2E) delays in the order of a few hundreds ms

Principles of application-layer protocols
Transport service requirements of common apps

Application	Data loss	Bandwidth	Latency
Web	No loss	Elastic	Not time-sensitive
E-mail			
File transfer			
IPTV, webradio – Live audio/video	Loss tolerant	Audio: up to 1 Mbps Video: up to 8 Mbps	Yes (MAX 400ms mouth-to-ear delay)
VoD, podcast – Stored audio/video			Yes (MAX 10s buffering time)
Interactive games	Loss tolerant	Up to 10 kbps	Yes
Instant messaging	No loss	Elastic	Variable

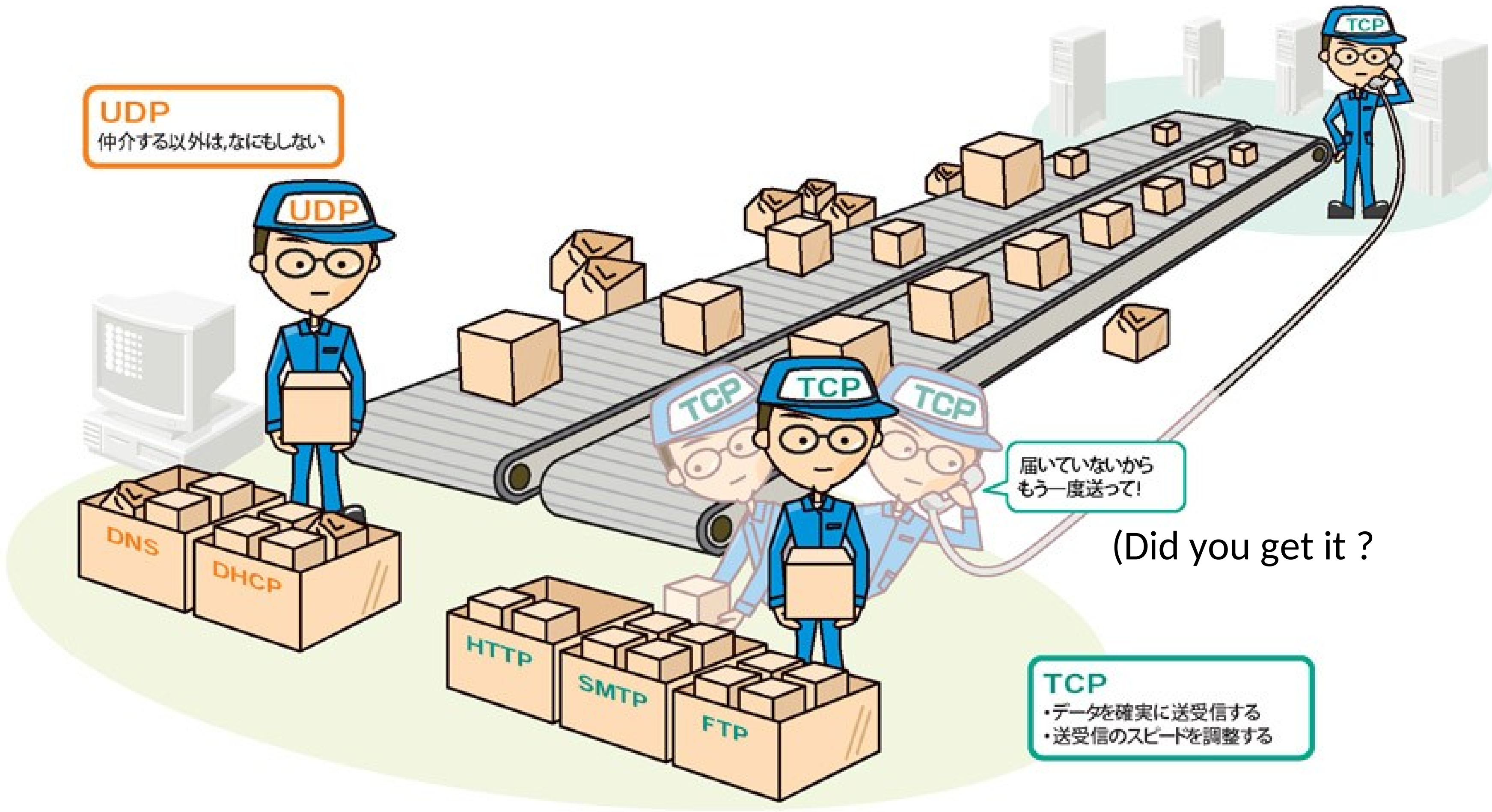
Principles of application-layer protocols
Services provided by Internet transport protocols

TCP	UDP
Connection-oriented: set-up required between client and server processes Reliable transport between sending and receiving process Flow control: sender will not overwhelm receiver Congestion control: sender reduces rate when network overloaded	Lightweight transport protocol with minimal service model
Does not provide - Timing guarantee - Bandwidth guarantee	Does not provide - Connection setup - Reliability - Flow control - Congestion control - Timing guarantee - Bandwidth guarantee

Principles of application-layer protocols
Protocoles for popular applications (1/2)

Applications	Application-layer protocol	Transport protocol
Web	HTTP, HTTPS	TCP
E-mail	SMTP	TCP
Remote login	Telnet, SSH	TCP
File transfer	FTP, SFTP	TCP
Streaming	RTP/RTCP/RTSP	UDP
	DASH	TCP
VoIP – Internet telephony	Proprietary	Typically UDP
Domain name resolution	DNS	UDP (Requests)
		TCP (DB updates)

Principles of application-layer protocols
Protocoles for popular applications (2/2)



Source : https://xtech.nikkei.com/it/article/lecture/20070305/263897/zu1.jpg

Outline

Principles of application layer protocols

Web browsing and HTTP

Domain name resolution (DNS)

File transfer (FTP)

Electronic Mail: SMTP, POP3, IMAP

Content distribution: Web caching, CDN, P2P

Socket programming with TCP

Socket programming with UDP

Building a simple Web server

Web browsing and HTTP
Some jargon for web browsing

- Web page consists of **objects**
- Object can be HTML file, JPEG image, Java applet, web app, social media widget, multimedia file, etc
- Web page consists of **base HTML-file** which includes several referenced objects
- HTTP Archive: typical web page in 2013
  - Embeds 88 objects from 30+ hosts (WebPageTest)
  - Contains 1,280 kB in total
- Each object is addressable by a **URL (Uniform Resource Locator)**

$$\underbrace{\texttt{www.someschool.edu}}_{\text{Host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{Path name}}$$

Web browsing and HTTP
And much more...

- Substrate for applications other than Web browsing
- Representational State Transfer (REST) APIs
  - **`http://sensor10.example.com/config/sleeptime`**
  - Default entry point: "**`/.well-known/core`**" URI identifier
- Motivations
  - Familiarity by implementers, specifiers, administrators, developers and users
  - Availability of client, server and proxy implementations
  - Ease of use
  - Feature reuse like multiplexing, caching, authentication, security, etc
  - Ability to traverse firewalls
- Often ad-hoc developments
- Best Common Practice: draft-ietf-httpbis-bcp56bis

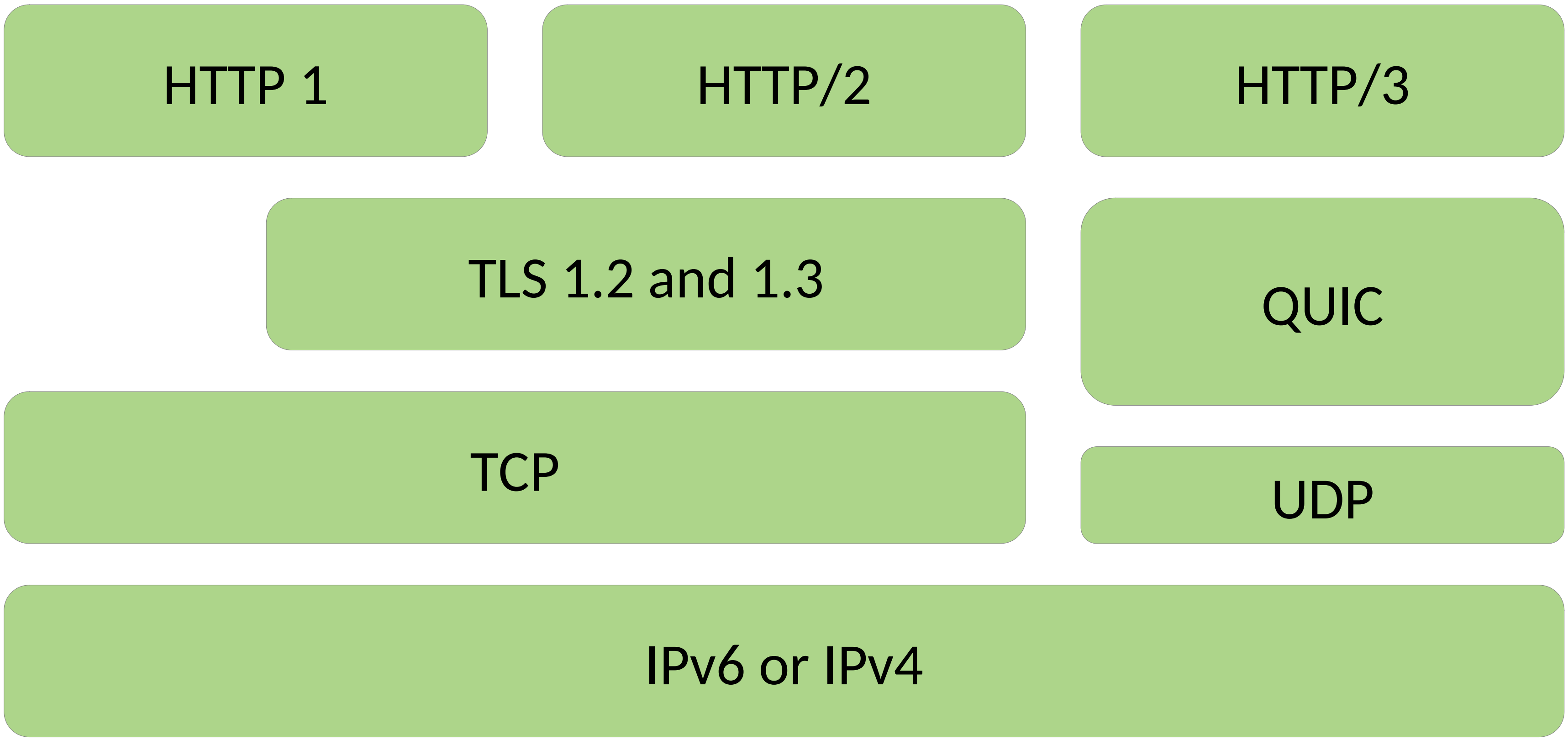
Web browsing and HTTP
Hypertext Transfer Protocol (HTTP) overview

- Web’s application layer protocol
- Client/server model
  - Client: browser that requests, receives and renders objects
  - Server: sends objects in response to requests (demographics)
- Originally based on TCP
  - HTTP client initiates TCP connection (creates **socket**) to HTTP server, well-known port 80
  - Server accepts TCP connection from client
  - (application-layer protocol) messages exchanged between browser (HTTP client) and Web server (HTTP server)
  - TCP connection closed
- Stateless
  - Server maintains no information about past client requests
  - Server sends twice the same object if the client requests it

Web browsing and HTTP
Standard versions

<table>
<tr><th>HTTP 1.0</th><th>HTTP 1.1</th><th>HTTP 2.0</th><th>HTTP 3.0</th></tr>
<tr><td>RFC 1945, standardised in May 1996</td><td>RFC 2616, standardised in June 1999</td><td>RFC 7540, standardised in May 2015</td><td>Work in progress</td></tr>
<tr><td colspan="3">TCP</td><td>QUIC</td></tr>
<tr><td></td><td>Backward compatible with HTTP 1.0</td><td>Backward compatible with HTTP 1.1</td><td>Unlikely</td></tr>
<tr><td rowspan="2">Non persistent connections
At most one object sent over a TCP connection</td><td colspan="2">Persistent connections
Multiple objects can be sent over a single TCP connection</td><td>Connectionless</td></tr>
<tr><td>FIFO pipelining at server</td><td>Asynchronous pipelining</td><td>Stream based</td></tr>
<tr><td colspan="2"></td><td>Header compression (gzip/DEFLATE)</td><td>TBD</td></tr>
</table>

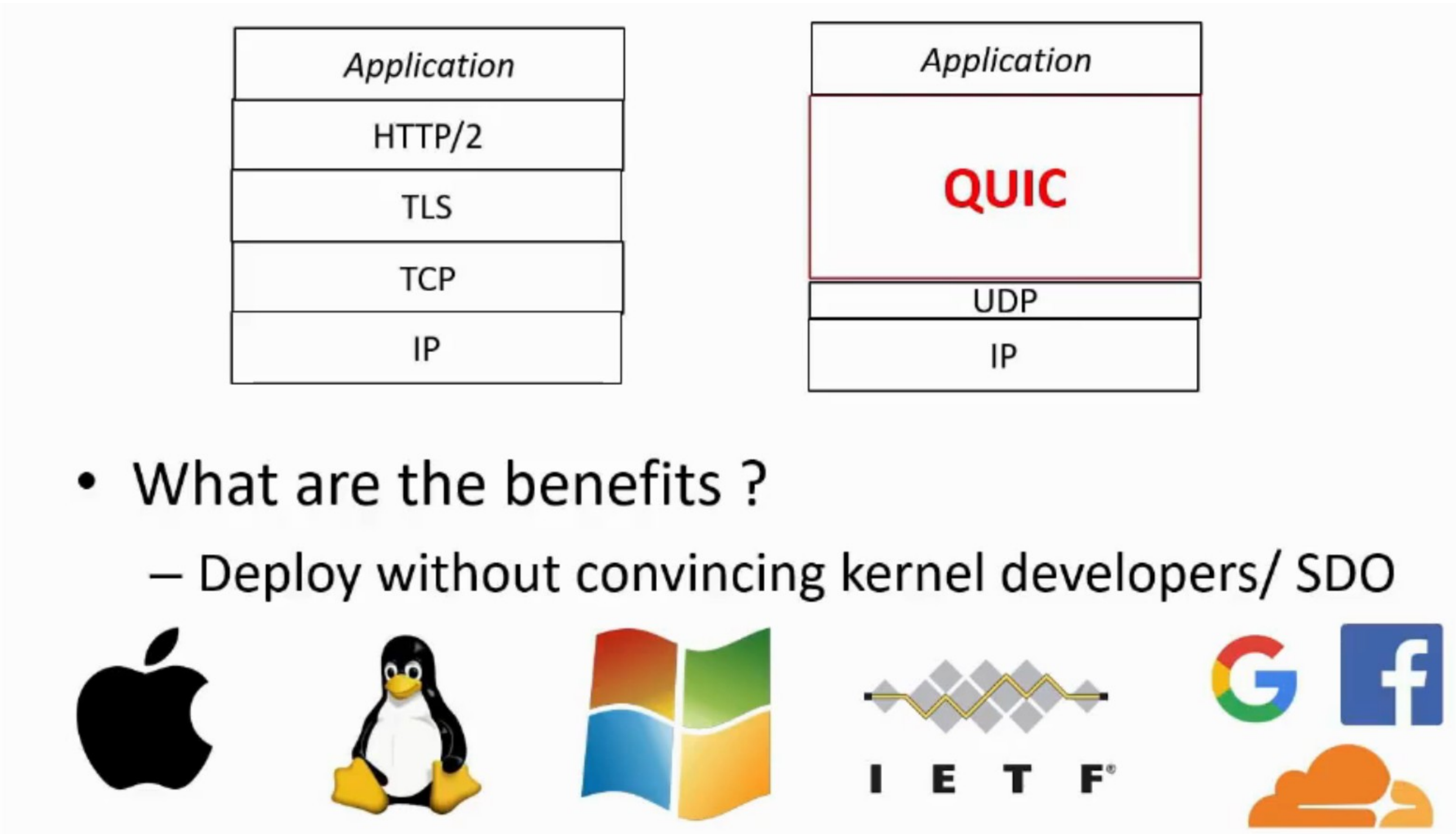
Web browsing and HTTP
Standard versions



Issue:
HTTP HoL blocking

Issue:
TCP HoL blocking

Web browsing and HTTP
The QUIC Revolution



Source : Olivier Bonaventure (UCLouvain), keynote at LCN 2019 - https://www.youtube.com/watch?v=W0lZXYJqYB4

Web browsing and HTTP
Non persistent connection

Suppose user enters URL

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

Page consists of a base HTML file and 10 JPEG images

Client initiates TCP connection to server at **`www.someSchool.edu`** on port 80

Server at **`www.someSchool.edu`** waiting for TCP connection at port 80 accepts connection, notifying client

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

Server receives request message, forms response message containing requested object, sends message into its socket and closes TCP connection

Client receives response message containing html file, displays html. Parsing html file, finds 10 JPEG objects.

Repeat for each of 10 JPEG

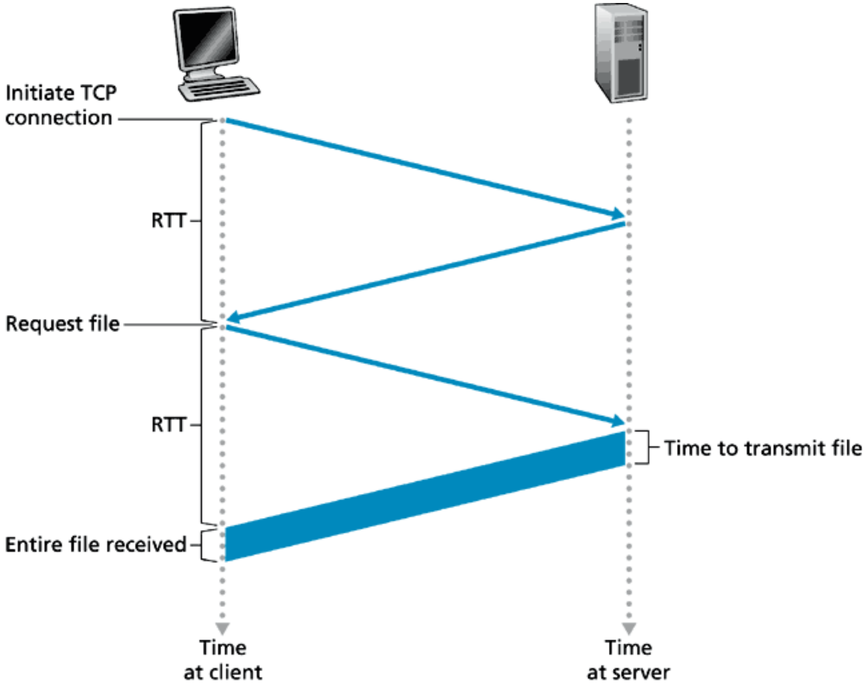
Web browsing and HTTP
Round-trip and response times

Round-trip time (RTT)

- Time it takes for a small packet to travel from client to server, and back to client
- Includes nodal processing, queuing and propagation delays

HTTP 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 + file transmission time



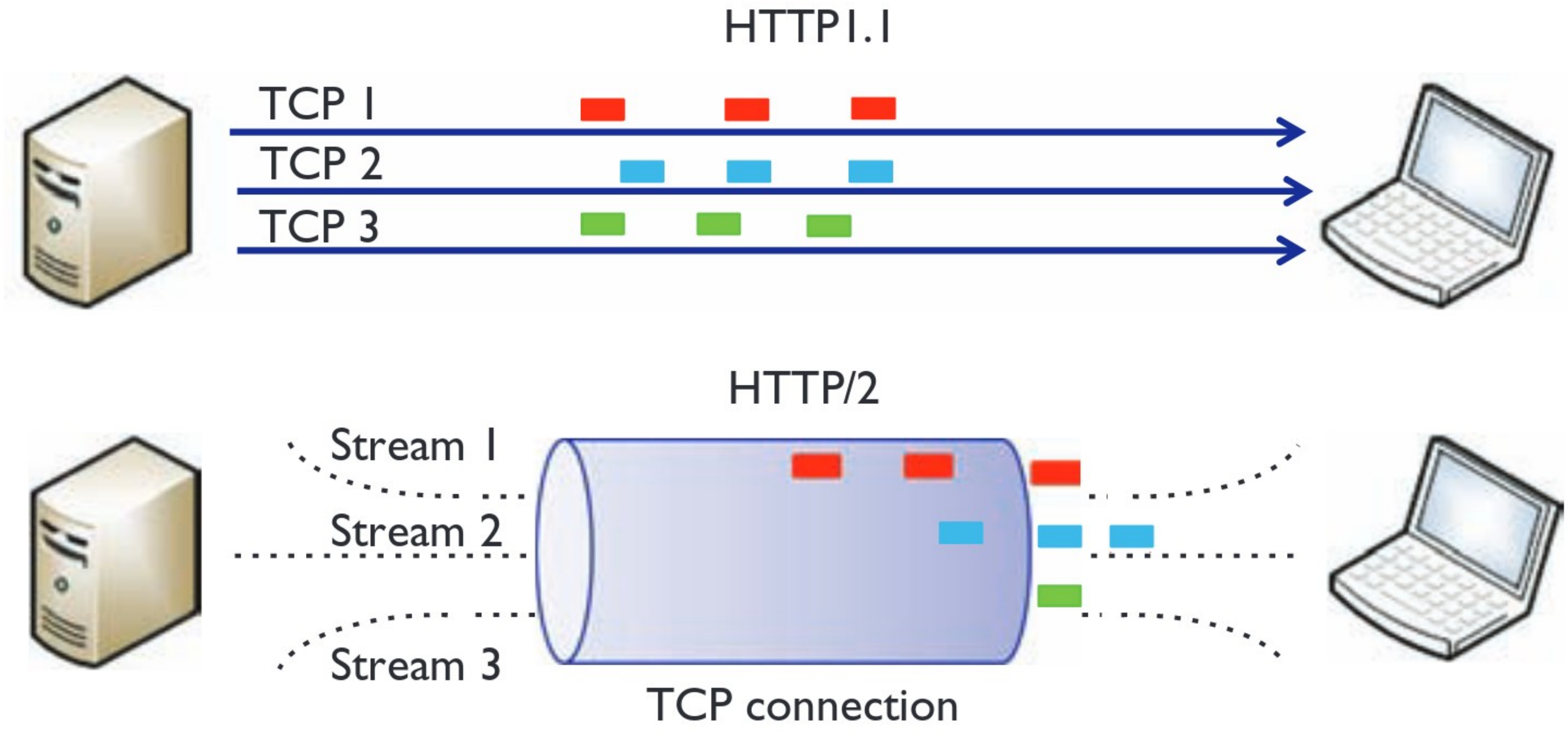
Web browsing and HTTP
Persistent connection and pipelining

Non persistent connections
- Requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection

Persistent connections
- Server leaves TCP connection open after answering
- Persistent connections without pipelining
  - Client issues new request only when previous response has been received
  - One RTT for each referenced object
- Persistent connections with pipelining
  - Client sends requests as soon as it encounters a referenced object
  - As little as one RTT for all the referenced objects
  - FIFO pipelining in HTTP 1.1, asynchronous in HTTP 2.0

HTTP Delay Estimation Applet

Web browsing and HTTP
Asynchronous pipelining



Source: Innovating Transport with QUIC: Design Approaches and Research Challenges, IEEE Internet Computing, Mar.-Apr. 2017

Web browsing and HTTP
Request message

- Two types of HTTP messages: request, response
- HTTP request message

ASCII (human-readable format)
No longer true with header compression (HTTP 2.0)

Request line (GET, POST, HEAD commands)

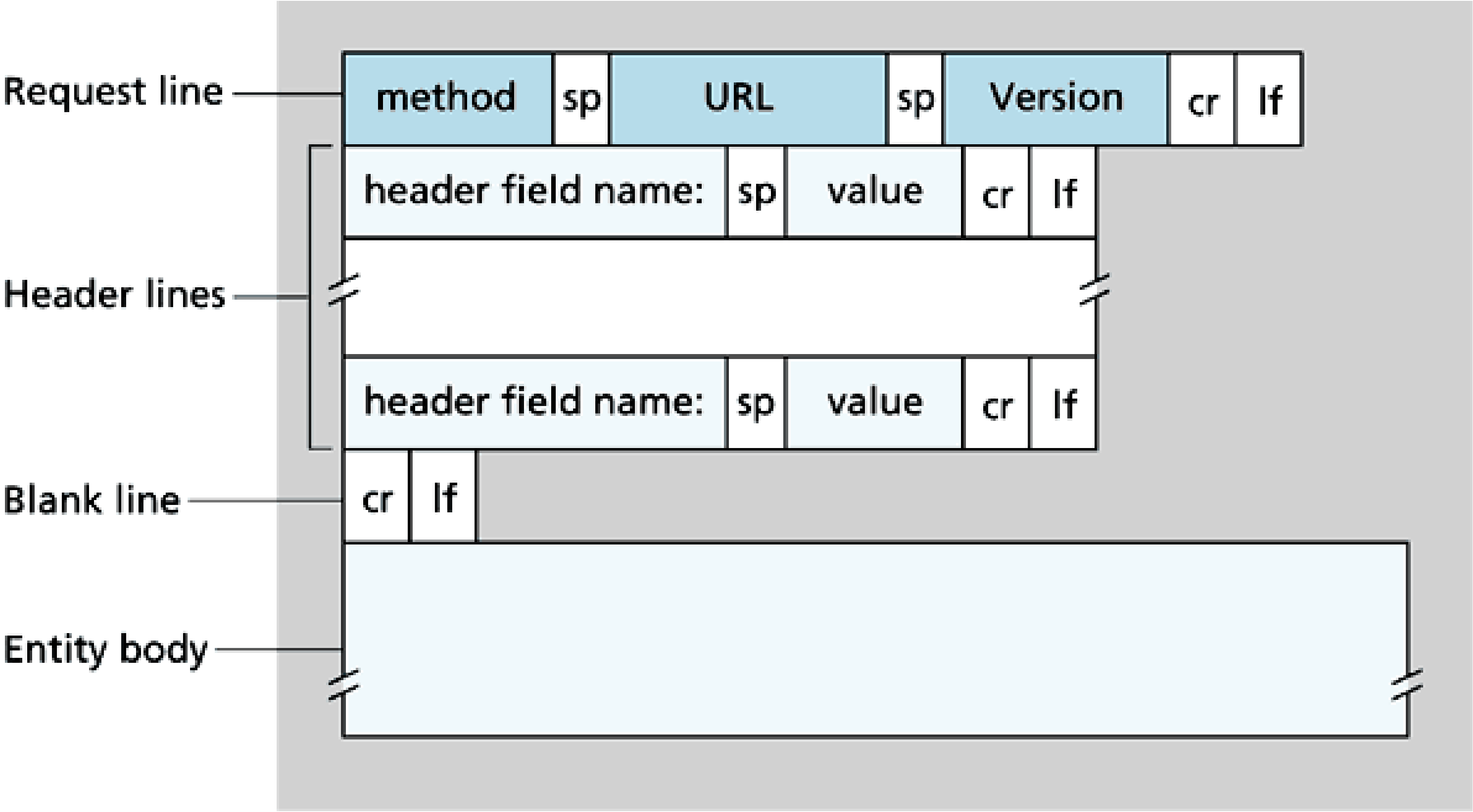
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/5.0
Connection: close
Accept-language: fr
```

Header lines

Carriage return, line feed indicates end of message

(extra carriage return, line feed)

Web browsing and HTTP
Request message – General format



Web browsing and HTTP
Methods

- Originally
  - GET – Request
  - POST – Upload content of “body” in form

    ```
    <FORM ACTION="http://www.somesite.com/Formulaire"
    METHOD="POST">
    ```
  - HEAD – Response limited to header
- Since HTTP 1.1
  - PUT – Upload content of « body » to path specified in URL field
  - DELETE – Delete file specified in URL field
- Some other, less common commands. Refer to RFC for details.

Web browsing and HTTP
Response message

- Two types of HTTP messages: request, response
- HTTP reponse message

ASCII (human-readable format)
No longer true with header compression

Status line (protocol, status code, status phrase)

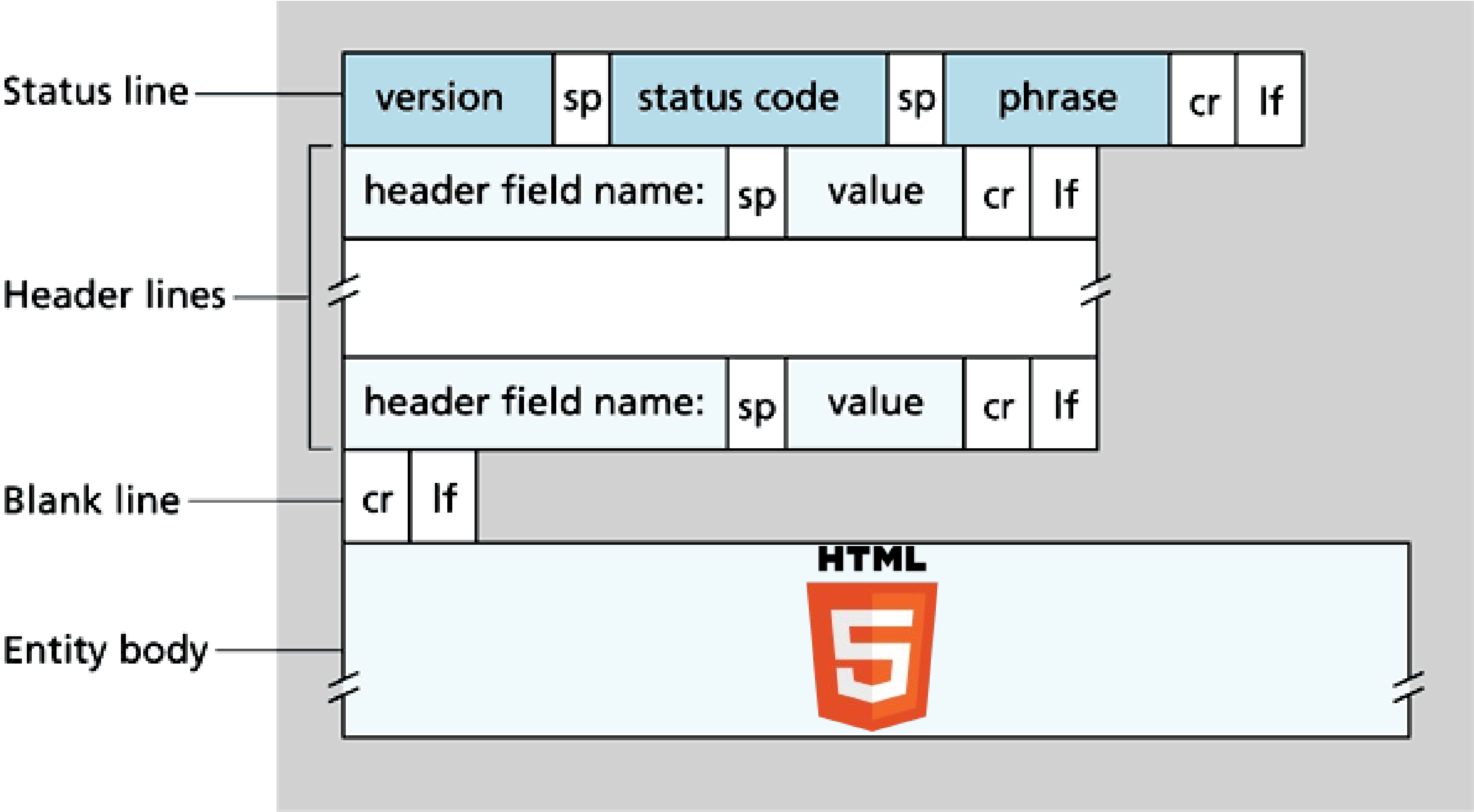
Header lines

Data

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 2012 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2012
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

Web browsing and HTTP

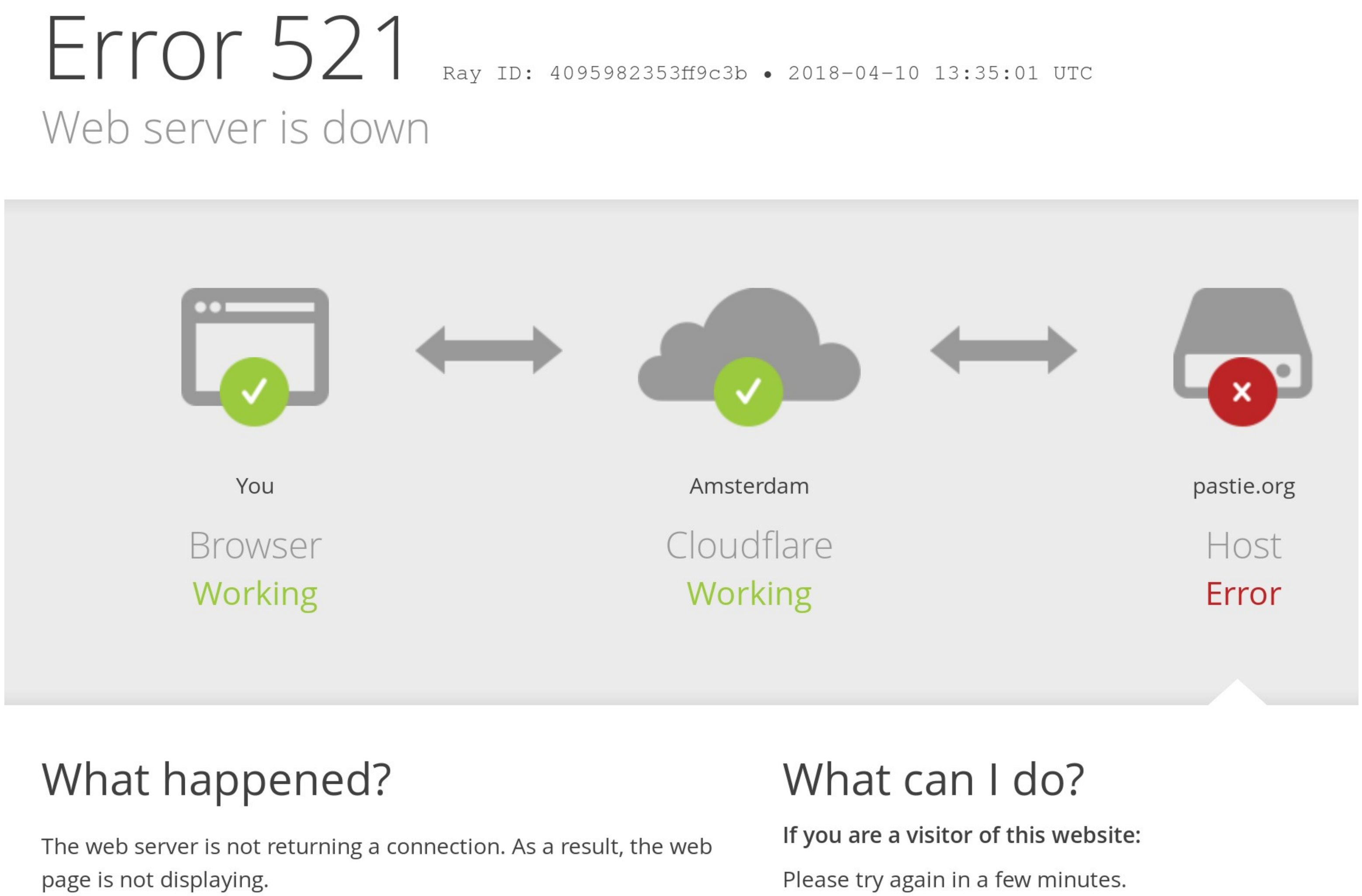
Request message – General format



Web browsing and HTTP
3-digit status codes (1/2)

Reply	Description
1xx	Positive preliminary reply – Action started, but expect another reply before sending another command
2xx	Positive completion reply – A new command can be sent **200 Command OK**
3xx	Positive intermediate reply – Command accepted, but another command must be sent **301 Moved permanently**
4xx	Transient negative completion reply – Requested action did not take place, but error condition temporary so the command can be reissued later **404 Not Found**
5xx	Permanent negative completion reply – Command was not accepted and should not be retried **505 HTTP Version Not Supported**

Web browsing and HTTP
3-digit status codes (2/2)



Web browsing and HTTP
Trying out HTTP client side for yourself

1. Telnet to a Web server

```
telnet gaia.cs.umass.edu 80
```

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

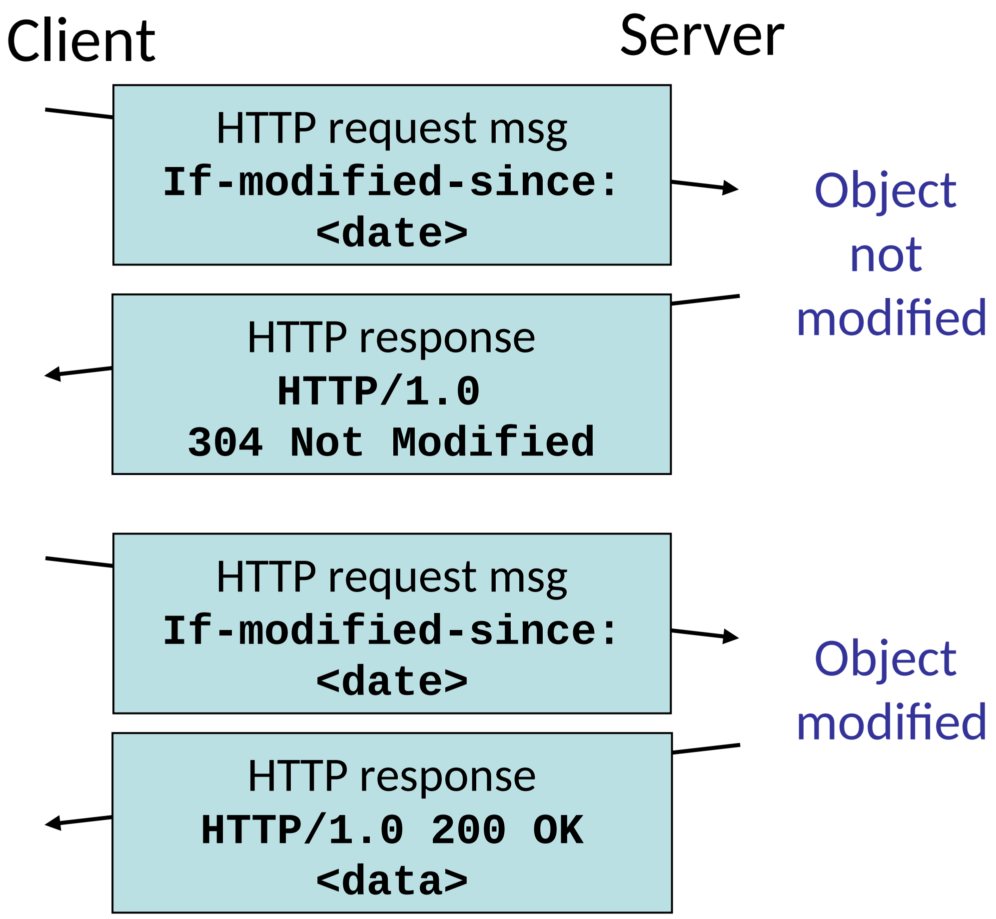
2. Type in a GET HTTP request

```
GET  /wireshark-labs/HTTP-wireshark-file1.html  HTTP/1.1
Host: gaia.cs.umass.edu:80
```

By typing this in (hit carriage return twice), you send a GET request to HTTP server

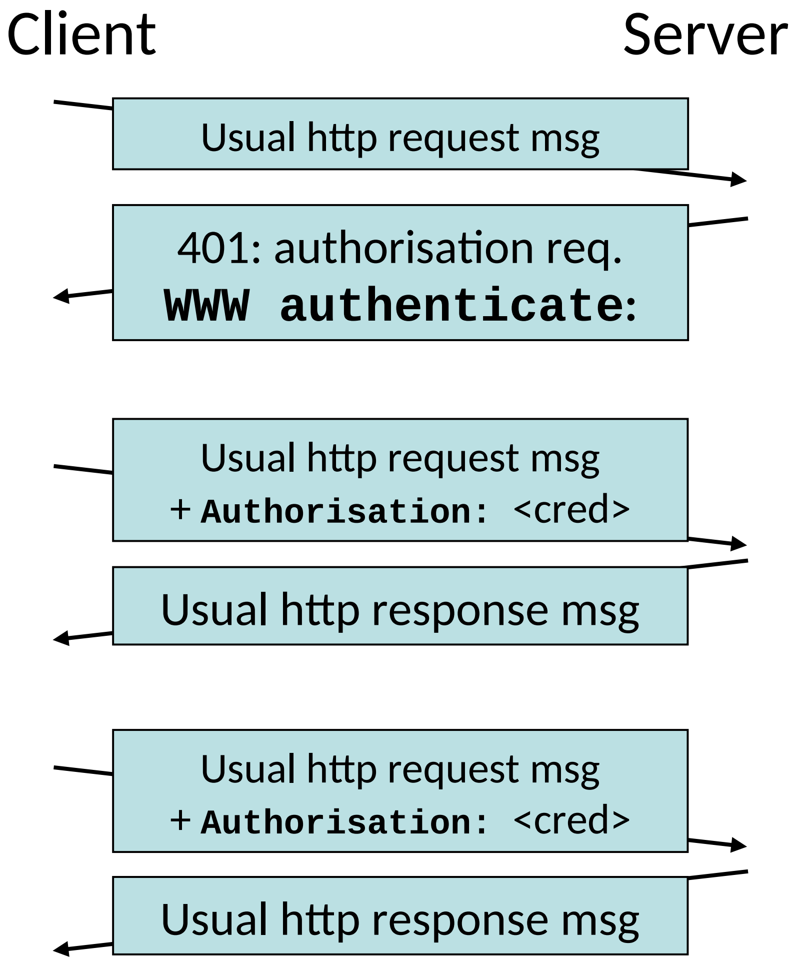
3. Look at response message sent by HTTP server!

Web browsing and HTTP
Conditional GET



- Avoid sending object if client has up-to-date cached version
- The client specifies date of cached copy in HTTP request
- The response contains no object if cached copy is up-to-date
- If not, modified object sent back

Web browsing and HTTP
Authorisation

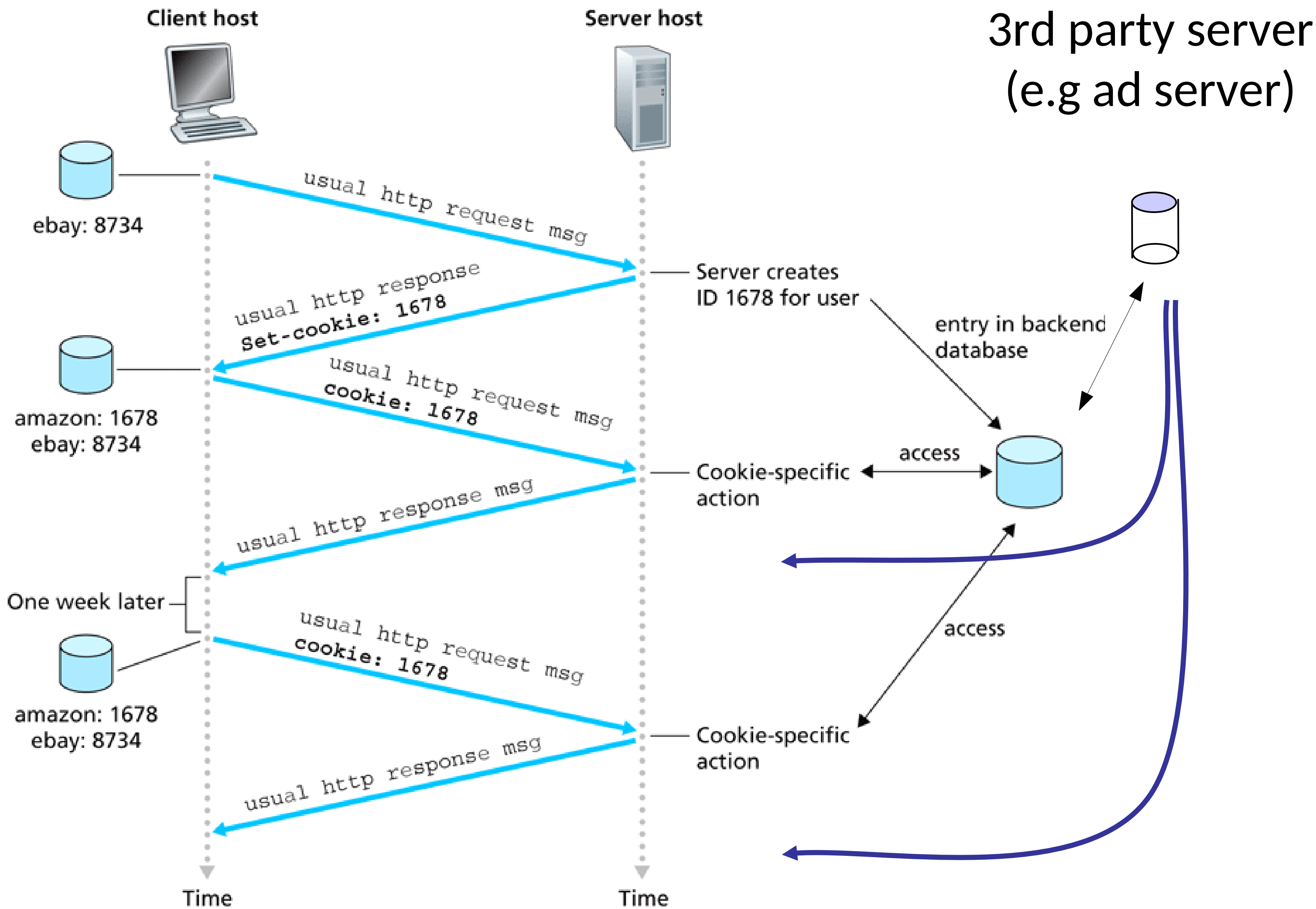


- Control access to server content
- Credentials: typically name, password
- Stateless: client must present authorisation in each request
  - Authorisation header line in each request
  - If no authorisation header line, server refuses access, sends **WWW authenticate**
- Ex: http://gaia.cs.umass.edu/wireshark-labs/protected_pages/HTTP-wireshark-file5.html

Web browsing and HTTP
Cookies

- HTTP stateless
- Web sites are wishing to serve content in function of user identity (Internet banking, shopping carts, recommendations)
- To manage state information, they use cookies
  - "A cookie is a well-known computer science term that is used when describing an opaque piece of data held by an intermediary" (Lou Montulli, Netscape 1.0 specification, 1997)
- Four components of cookie technology
  - Cookie header line in the HTTP response message
  - Cookie header line in the HTTP request message
  - Cookie file kept on user's host and managed by user's browser (MAX 300 cookies of 4 kB each, RFC 2109)
  - Back-end database
- Session cookie (expires at end of session) vs. permanent cookie (remains until expiry)

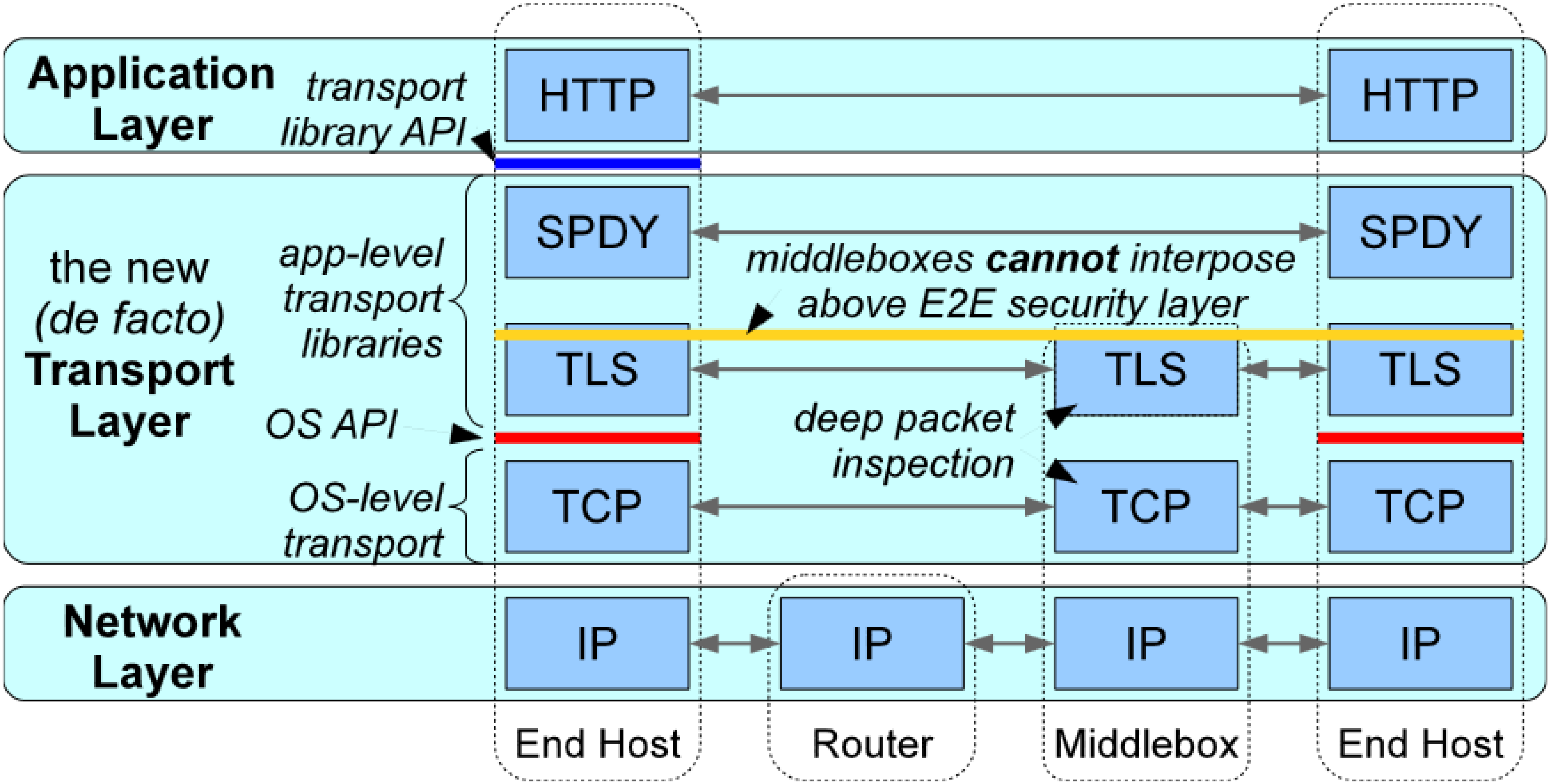
Web browsing and HTTP
Cookies



Web browsing and HTTP
Browser fingerprinting

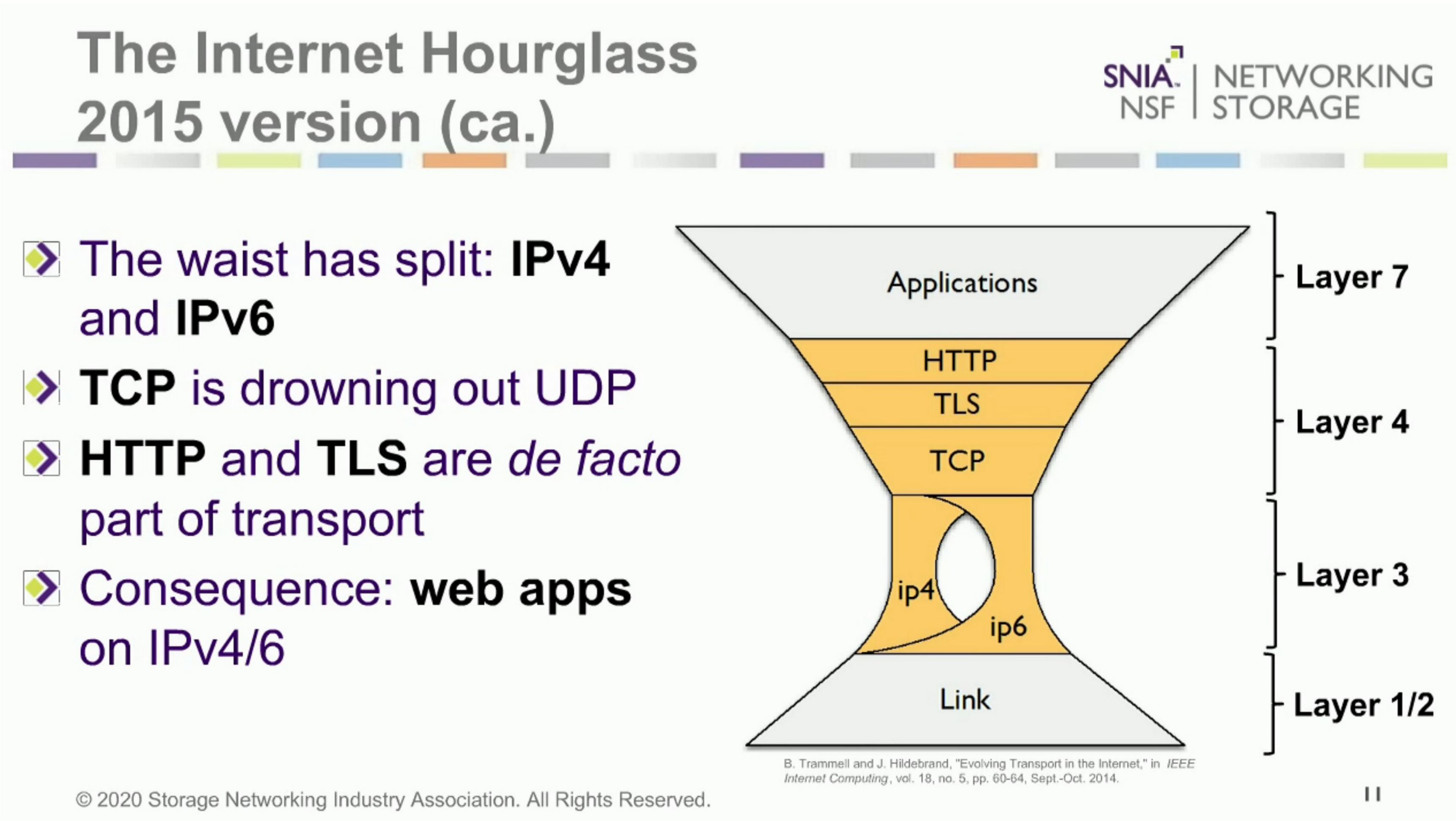
- Cookies are outdated and regulated (GDPR)
- Generation of a tracker based on
  - Hardware
  - Operating System
  - Display settings
  - Browser settings
- Check on amiunique.org

Today's « de facto » set-up
Source : ACM SIGCOMM eBook 2013

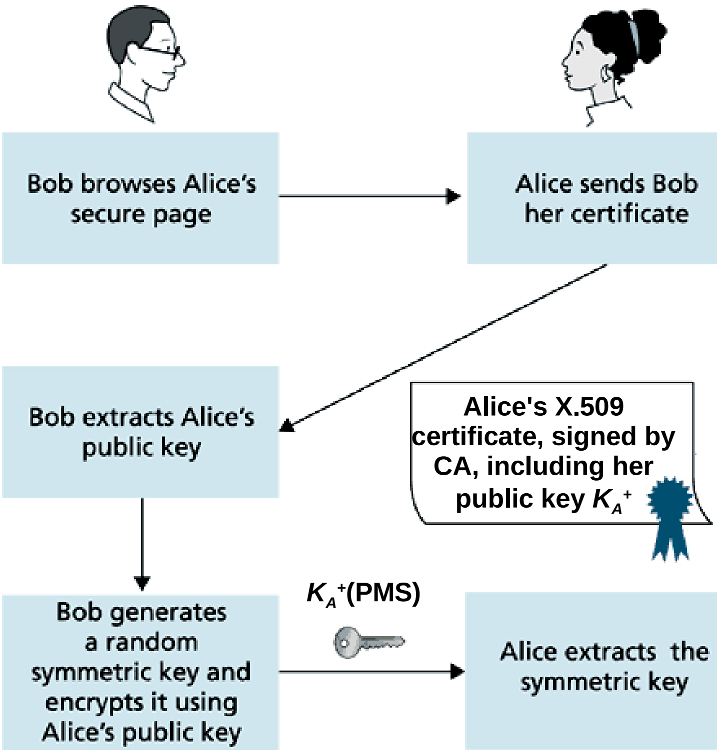


The Internet Hourglass

Source : SNIA presentation, April 2020



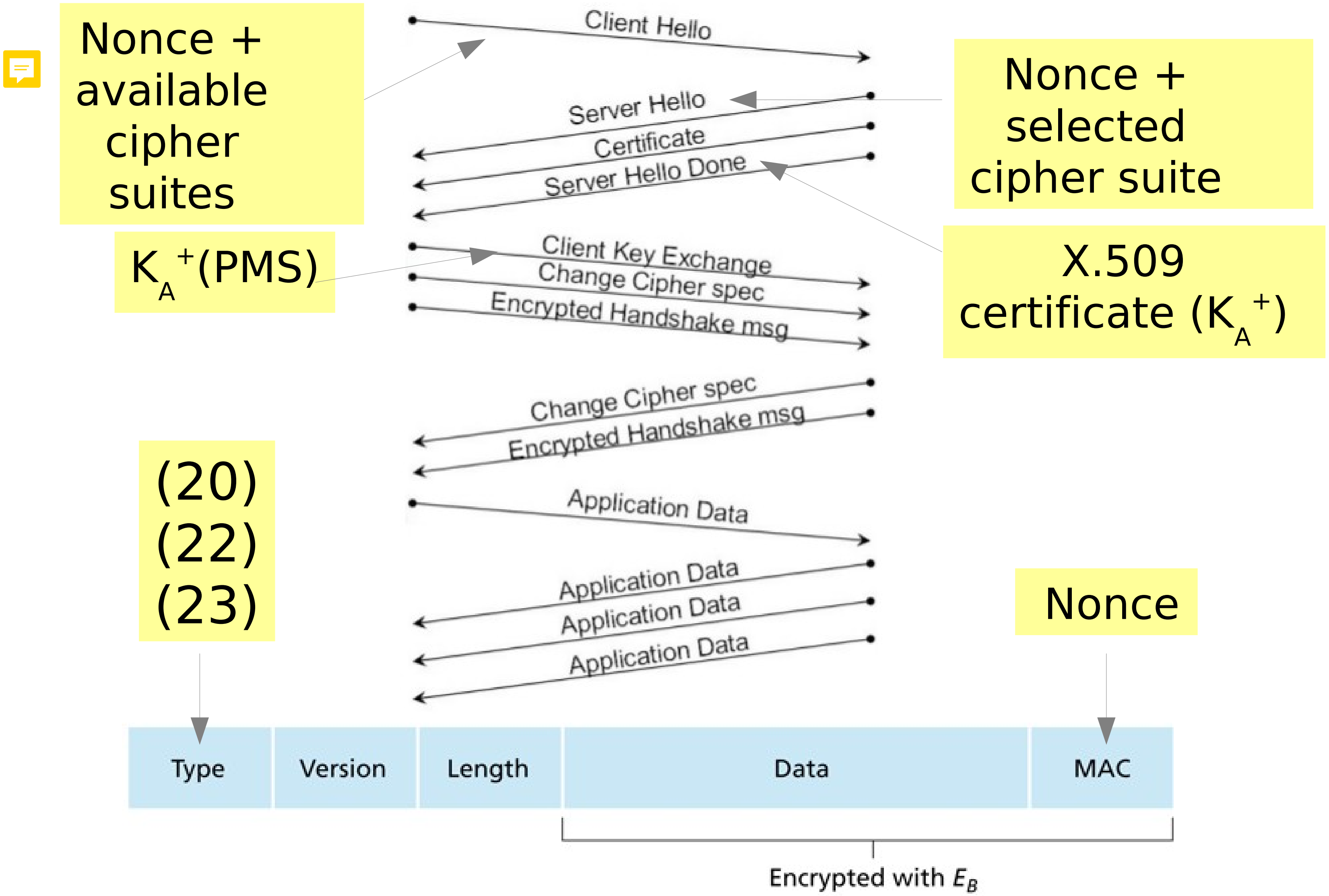
Web browsing and HTTP
HTTPS – Fighting phishing



- An SSL-compliant browser embeds public keys of CAs
- Peers initiate a cipher suite negotiation
- Bob requests Alice's certificate
- Bob uses K_{CA}^+ to authenticate Alice and extract her K_A^+ from the certificate
- Bob's browser generates a Pre-Master Secret (PMS) as seed for cyphering and Message Authentication (MAC)

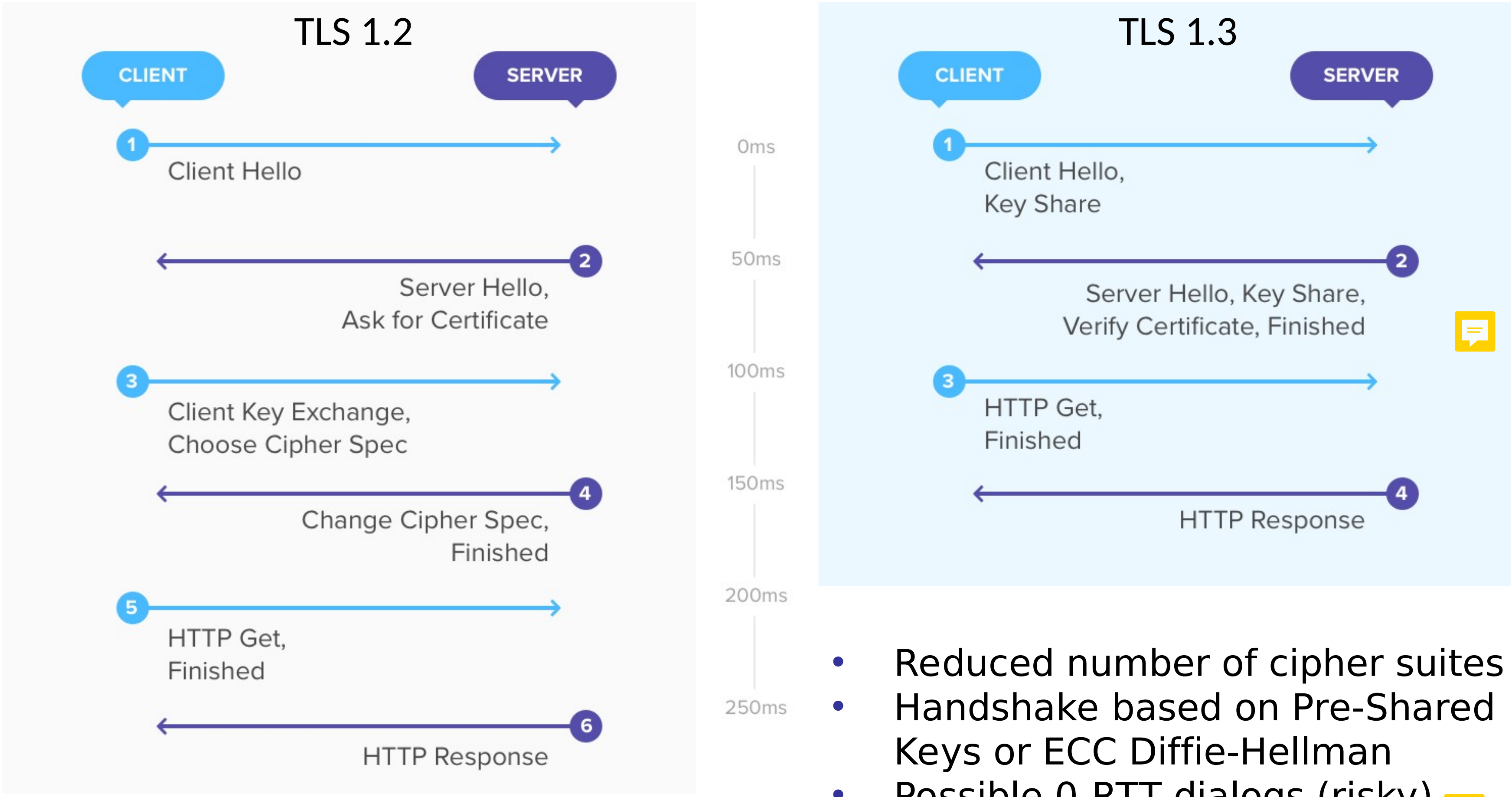
Web browsing and HTTP

Transport Layer Security v1.2 (RFC 5246, August 2008)



Web browsing and HTTP

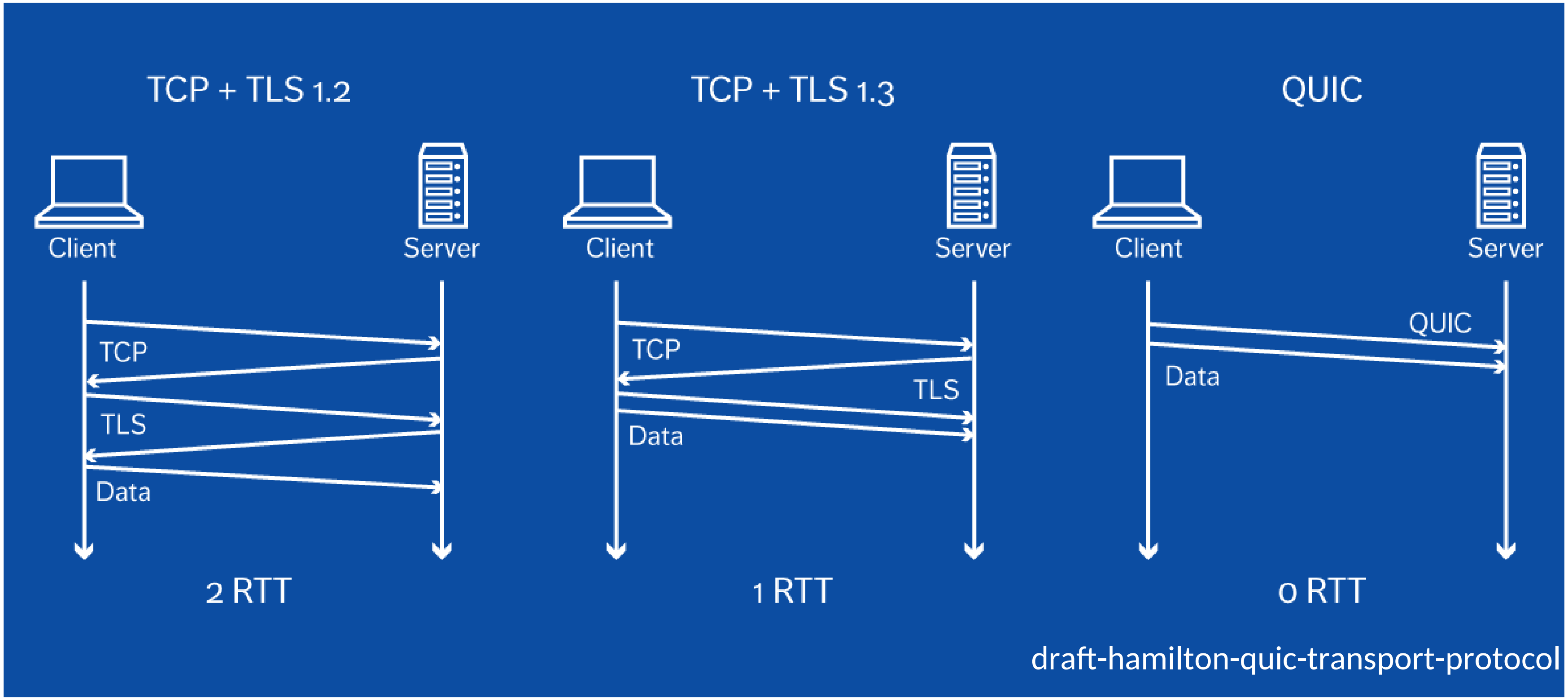
Transport Layer Security v1.3 (RFC 8446, August 2018)



- Reduced number of cipher suites
- Handshake based on Pre-Shared Keys or ECC Diffie-Hellman
- Possible 0-RTT dialogs (risky)

Source : https://www.cdn77.com/blog/cdn77-tls-1-3-supported/

Web browsing and HTTP
Evolutions



Source: Cryptography in an All Encrypted World, Ericsson Technology Review, December 2015

SSL/TLS and PKI Timeline, June 2016

Outline

Principles of application layer protocols

Web browsing and HTTP

Domain name resolution (DNS)

File transfer (FTP)

Electronic Mail: SMTP, POP3, IMAP

Content distribution: Web caching, CDN, P2P

Socket programming with TCP

Socket programming with UDP

Building a simple Web server

Domain Name System (DNS)
Overview

- Many identifiers for people: name, national register number, ID card number, passport number, etc.
- Two identifiers for hosts and routers
  - Hostname, e.g., **`gaia.cs.umass.edu`** – Used by humans
  - IP address (32 bits IPv4, 128 bits IPv6) – Used for addressing datagrams
- Need for mapping between hostname and IP address
- Domain Name System (DNS)
  - Distributed database, implemented in a hierarchy
  - Core Internet function, running as application-layer protocol
    - Host, routers, name servers communicate to resolve names (address/name translation)
    - Most commonly used implementation: Berkeley Internet Name Domain (BIND)
  - Runs over both UDP (queries) and TCP (DB updates) on port 53

Domain Name System (DNS)
Services

1. Mapping hostnames – IP addresses
2. Host aliasing: DNS translates mnemonic hostnames **`www.foo.com`** into actual canonical hostname **`relay1.west-coast.foo.com`**
3. Mail server aliasing: DNS translates mnemonic e-mail address **`bob@foo.com`** into actual address **`bob@relay1.west-coast.foo.com`**
4. Load balancing
   - Several IP addresses associated with a given canonical hostname
   - Whenever DNS query on hostname, server picks one of the IP addresses according to a given scheme (random, rotation)
   - Distribution of traffic among replicated servers

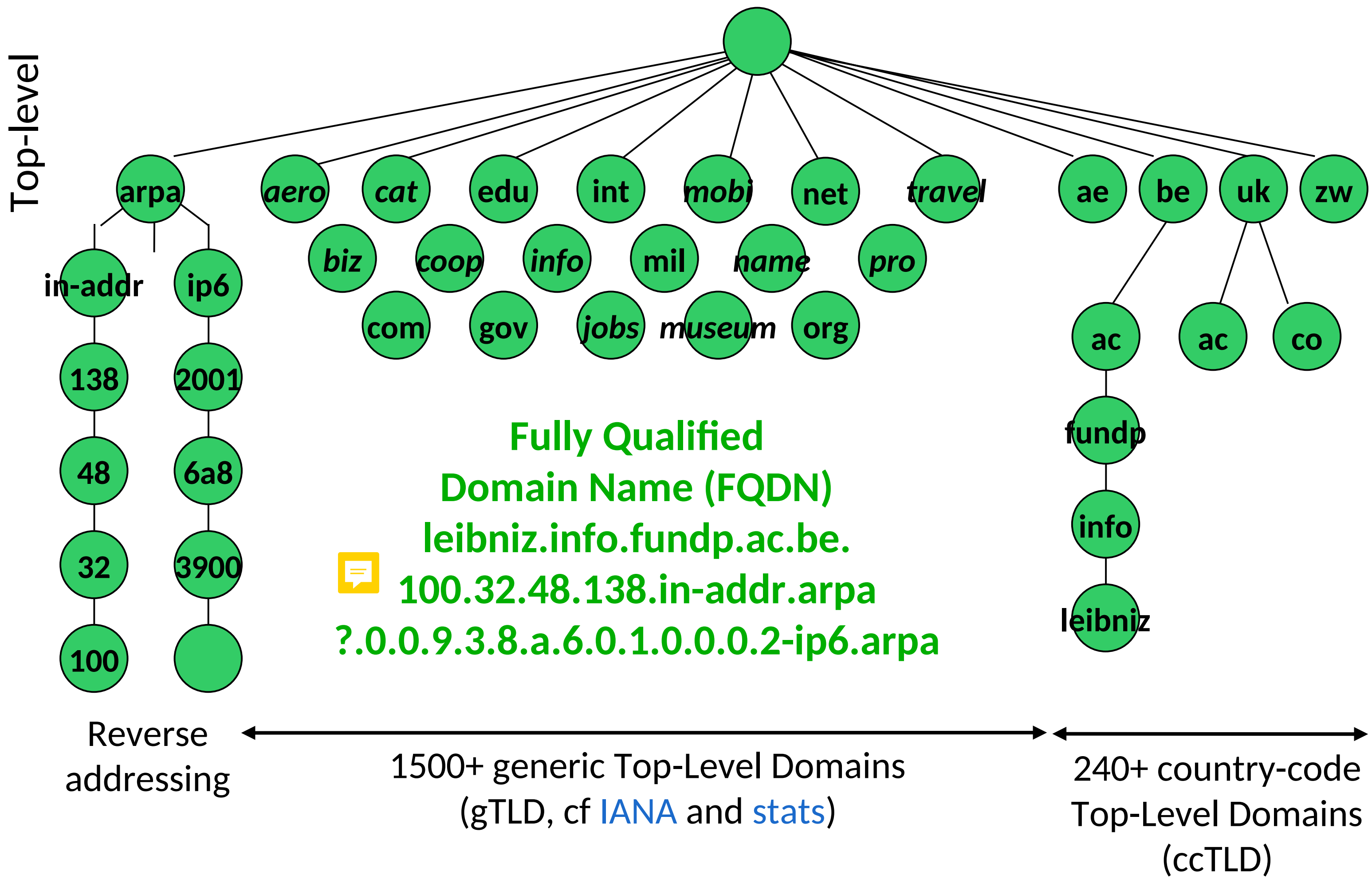
Domain Name System (DNS)
How it works

- Simple design would be one Internet name server containing all mappings
- Drawbacks of this centralised design
  - Single Point of Failure (SPOF)
  - Huge traffic volume
  - Significant delays for distant queries
  - Maintenance and authentication
  - Not scalable
- Solution
  - Large number of name servers
  - Hierarchically organised
  - Distributed around the world

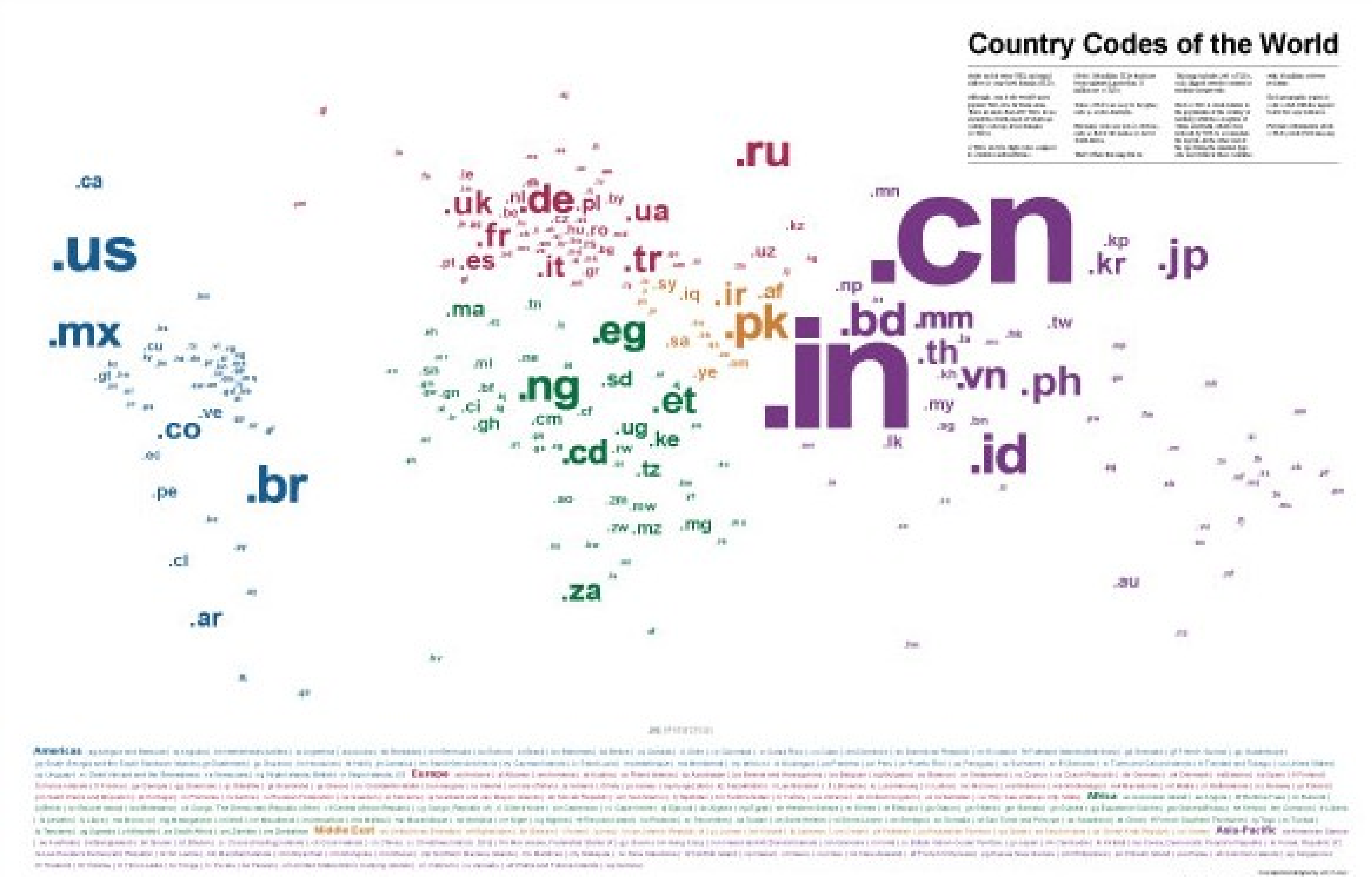
Domain Name System (DNS)
Four types of name servers

1. Local name servers
   - Each ISP, company has a local (default) name server
   - Host DNS query first goes to local name server
2. Root name servers
   - A “dozen” of root name servers
   - Queried by local name servers when local name servers can not reply to a query
   - Root name server sends record back
   - Root name server refers to authoritative name server
3. Top-Level Domain (TLD) servers
   - Responsible for Top-Level Domain
4. Authoritative name servers
   - Server is authoritative for a host if it can perform hostname/address translation for that host’s name
   - Authoritative name server for a host is name server in host’s local ISP

Domain Name System (DNS)
Hierarchical organisation – Top-Level Domains

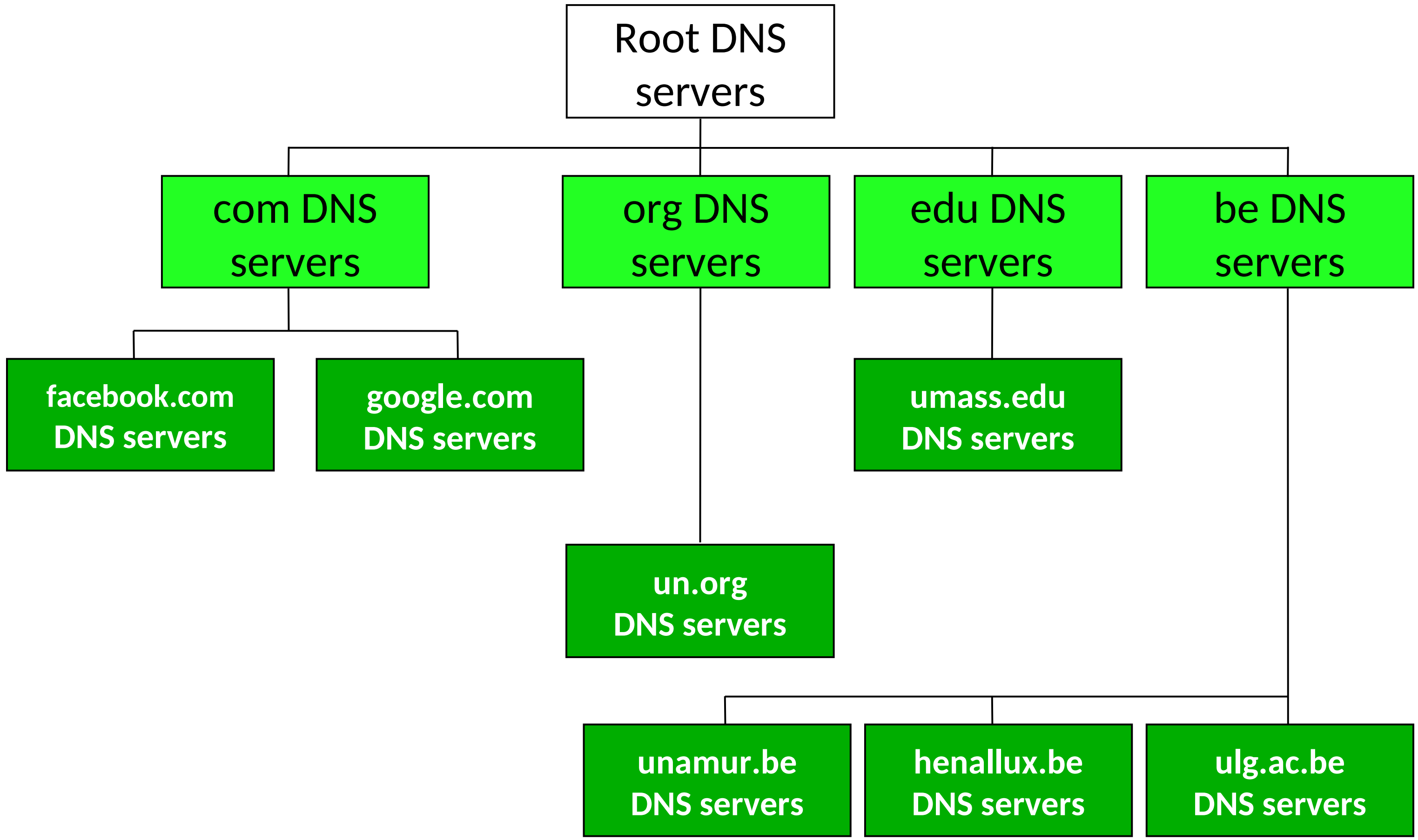


Domain Name System (DNS)
Hierarchical organisation – ccTLD



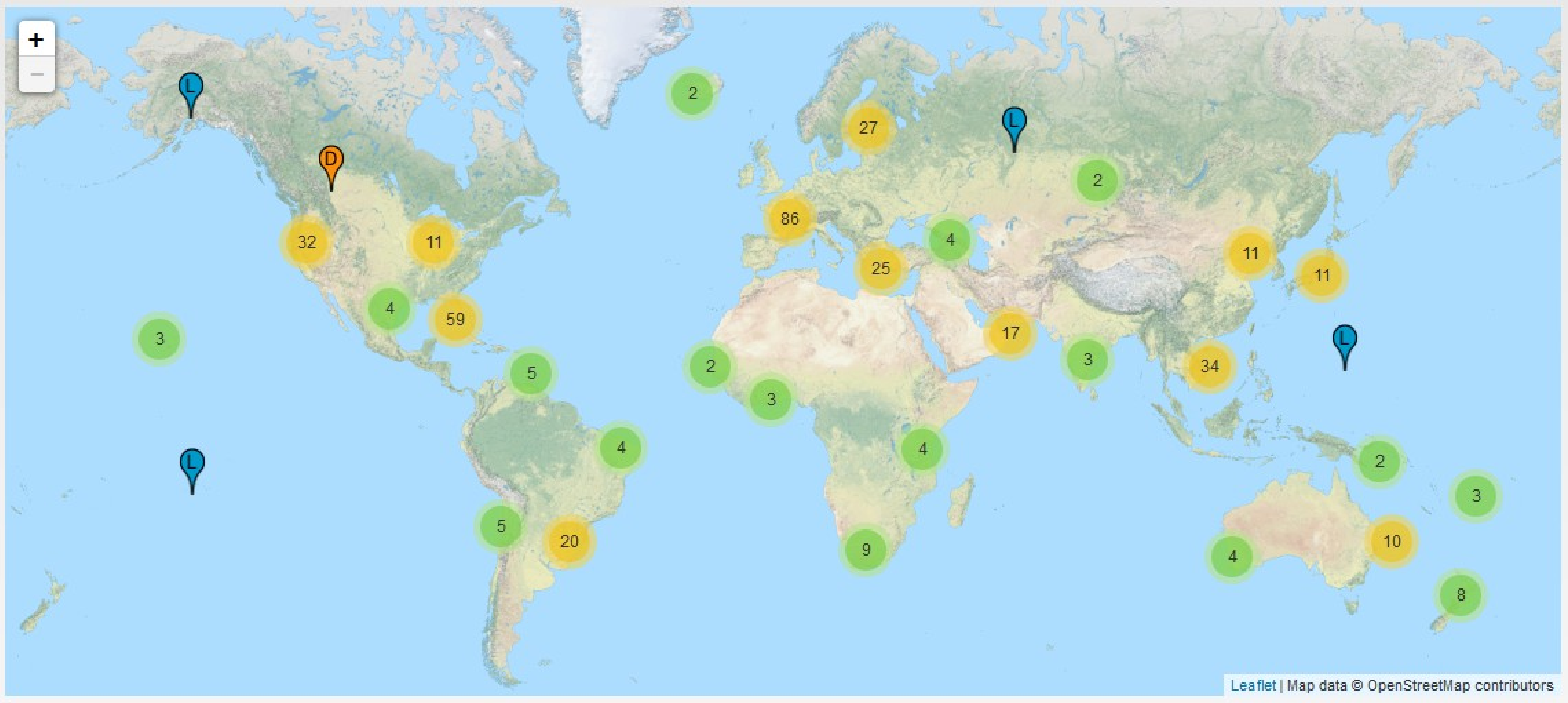
Source: http://www.bytelevel.com/map/ccTLD.html

Domain Name System (DNS)
Hierarchical organisation



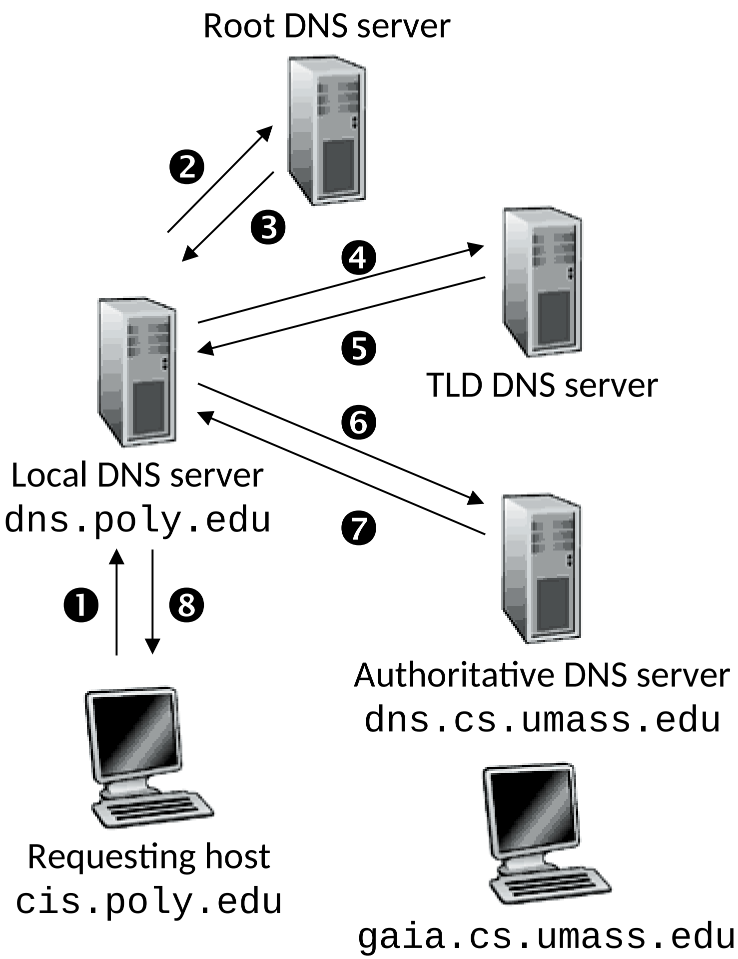
Domain Name System (DNS)
Root Name Servers

- 12 root server operators
- Operating from 800+ sites (October 2014)
- BELNET hosting a “**`i`**” root server since May 2004



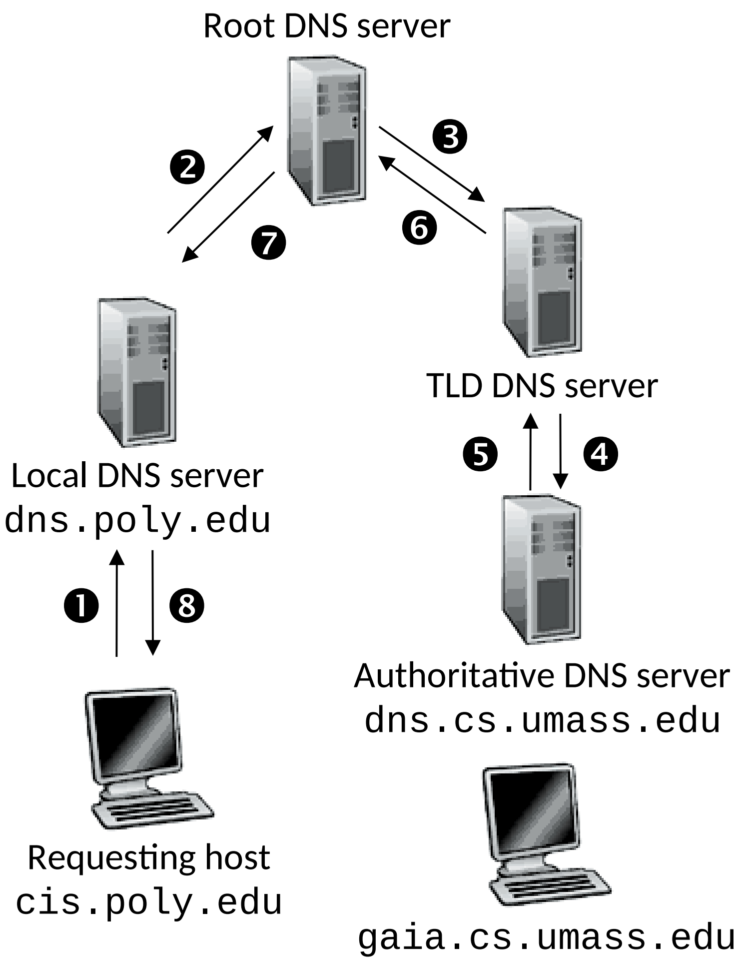
Domain Name System

Simple DNS Example



- Host **`cis.poly.edu`** wants IP address of **`gaia.cs.umass.edu`**
- Contacts its local DNS server, **`dns.poly.edu`**
- **`dns.poly.edu`** contacts root name server, if necessary
- Root name server refers to **`edu`** TLD server
- Local name server queries TLD server
- TLD server refers to **`umass.edu`** authoritative name server
- Local name server queries authoritative name server
- IP address sent back

Domain Name System
Iterative vs. recursive



- Iterative query
  - Contacted server replies with name of server to contact
  - “I don’t know this name, but ask this server”
- Recursive query
  - Puts burden of name resolution on contacted name server

Recursive/Iterative queries in DNS

Domain Name System
DNS and censorship

9TO5Google

SEARCH

ANDROID CHROME/OS GUIDES GOOGLE APPS ANDROID TV YOUTUBE

MARCH 21, 2014

Google DNS provides workaround as Turkish government blocks access to twitter

Ben Lovejoy - 3 years ago @benlovejoy

YOUTUBE GOOGLE TWITTER INTERNET SERVICE PROVIDER TURKEY

4 Comments Facebook Twitter Google Pinterest Reddit

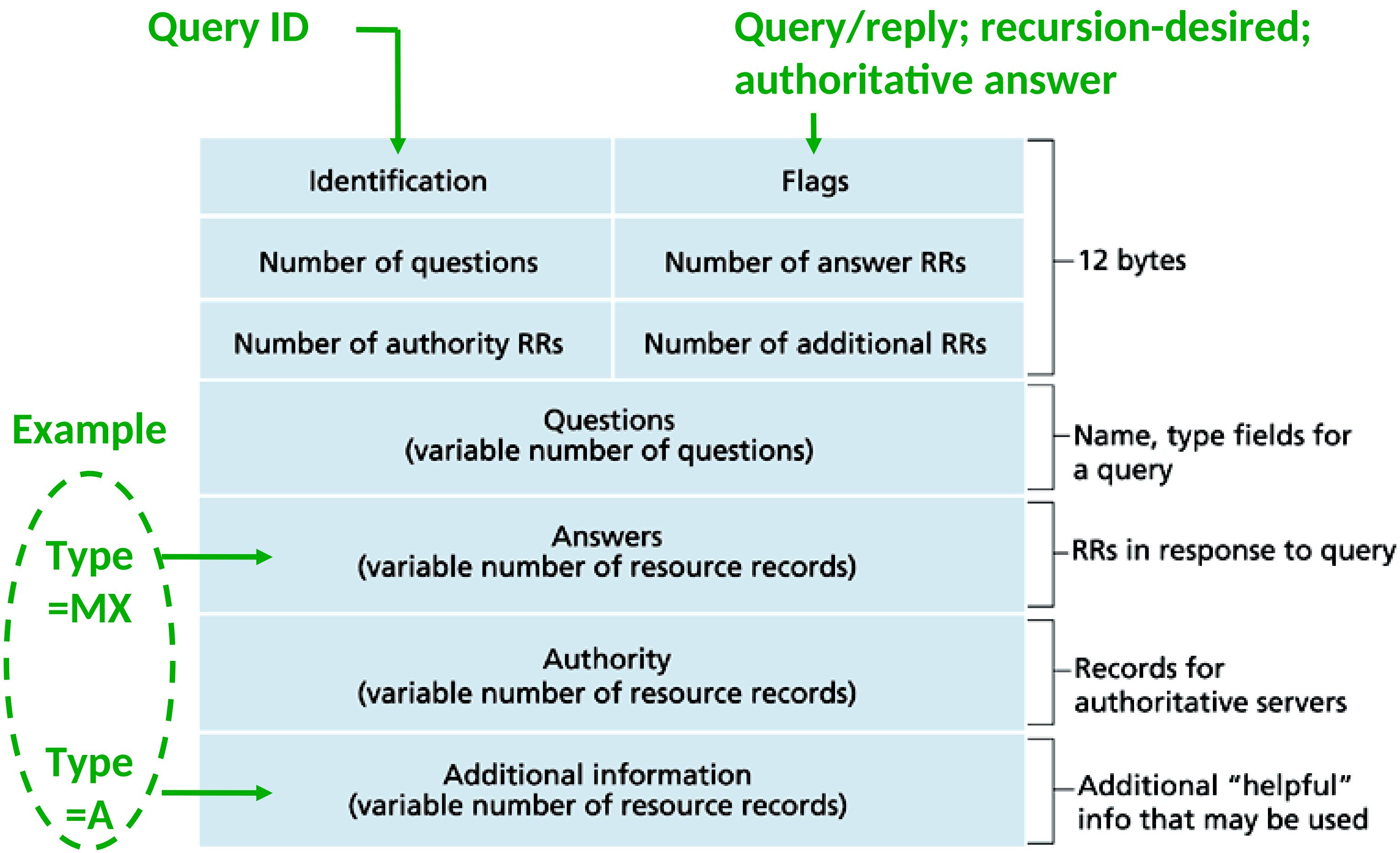
Turkish citizens, who found access to Twitter blocked yesterday in an apparent attempt by prime minister Recep Erdoğan to stem the spread of corruption allegations against him, have been able to

Follow

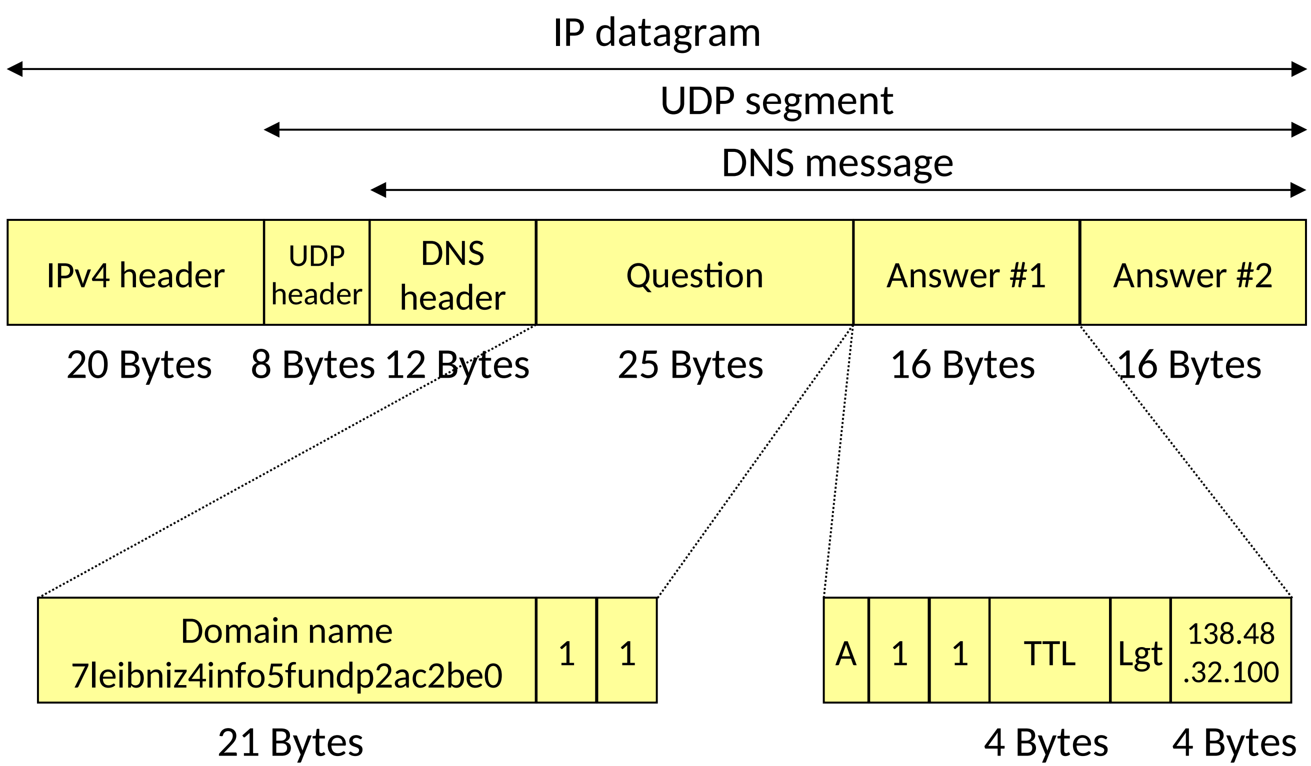
Domain Name System (DNS)
DNS Records

- DNS is a distributed database storing Resource Records (RR)
- RR is a 4-tuple (Name, Value, Type, TTL)
- Type=A(AAA)
  - name is hostname, value is IP address
  - (**`relay1.west-coast.foo.com`**, 145.37.93.126, A, TTL)
  - (**`ipv6.l.google.com`**, 2a00:1450:8003::6a, AAAA, TTL)
- Type=CNAME
  - name is alias name for the “canonical” (real) name
  - (**`www.foo.com`**, **`relay1.west-coast.foo.com`**, CNAME, TTL)
- Type=MX
  - name is domain (e.g. **`foo.com`**)
  - value is name of mailserver associated with name
  - (**`foo.com`**, **`mail.foo.com`**, MX, TTL)
- Type=NS
  - value is hostname of authoritative name server for this domain
  - (**`foo.com`**, **`dns.foo.com`**, NS, TTL)
- Full list

Domain Name System (DNS)
DNS messages



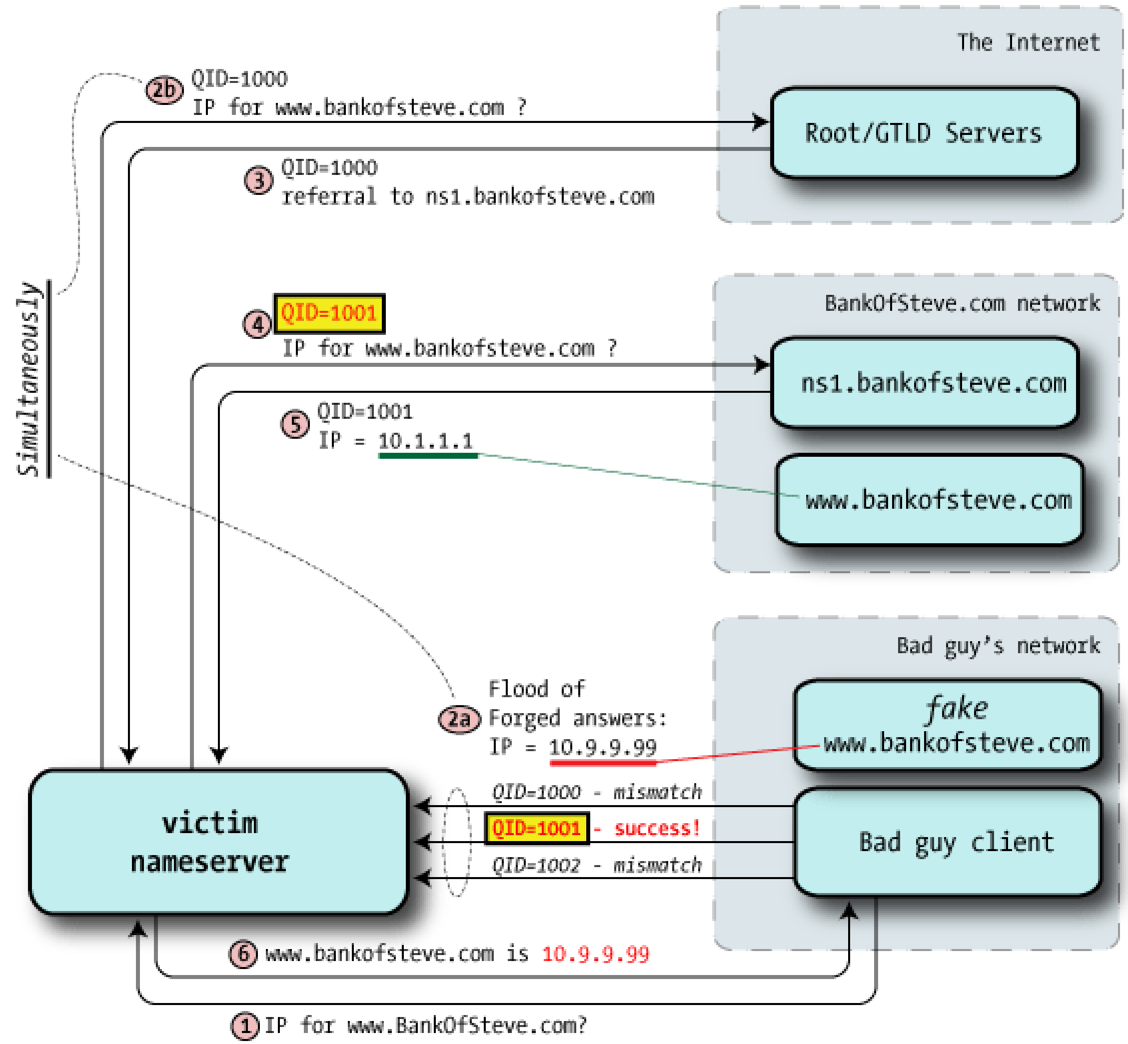
Domain Name System (DNS)
DNS reply



Domain Name System (DNS)
DNS Caching

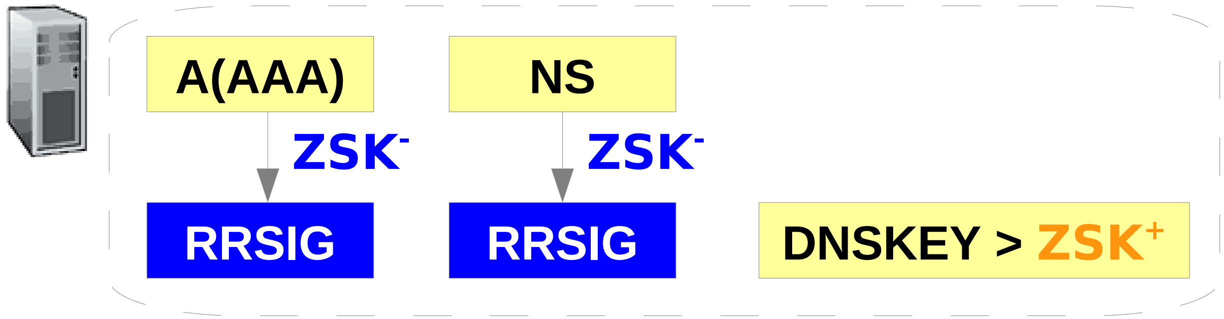
- Whenever a name server receives a mapping, it caches it while passing it along
- Server is able to answer queries without being authoritative answer for queried hostname
- Cached entries discarded after period of time (usually two days)
- Threat: cache poisonning/forgery (Dan Kaminsky, July 2008)
- Solution: DNSSEC (RFC 4033, March 2005)
    - Signed DNS replies
    - Key management (chain of trust)
    - Issue: key renewal

Domain Name System (DNS)
DNS cache poisonning



Domain Name System (DNS)
DNSSEC – Island of security

- Goal: validate Resource Records (RRs) provided by an authoritative server
- Solution: sign RRs with a *Zone Signing Key* (ZSK)



- New RRs
  - RRSIG : signature of an RR. Content: RR type, algorithm, expiry date, IDs and signature

    `dig www.belnet.be RRSIG`

  - DNSKEY: public key ZSK^+ of authoritative server

    `dig belnet.be DNSKEY`

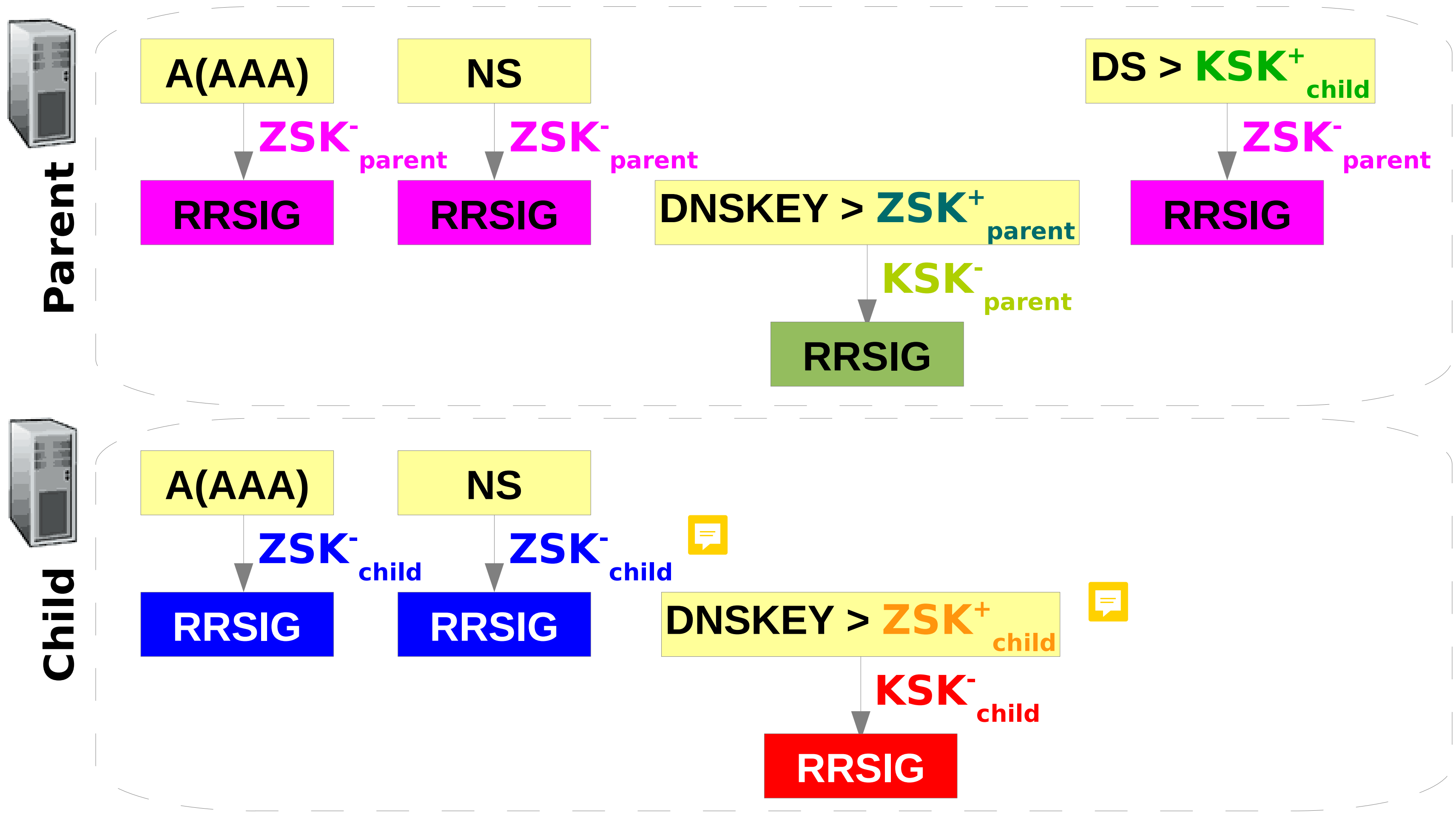
  - NSEC: counter measure to DNS survey

Domain Name System (DNS)
DNSSEC – Chain of trust (1/2)

- Goal: create a trust relationship from the root downto a given domain
- Another key, the *Key Signing Key* (KSK)
- Introduction of DS-type RRs
- Parent zone
  - ($\mathbf{KSK^{+}_{parent}}$, $\mathbf{KSK^{-}_{parent}}$)
  - ($\mathbf{ZSK^{+}_{parent}}$, $\mathbf{ZSK^{-}_{parent}}$)
  - $\mathbf{KSK^{-}_{parent}}$ (RR DNSKEY embedding $\mathbf{ZSK^{+}_{parent}}$)
  - $\mathbf{ZSK^{-}_{parent}}$(RRs) including $\mathbf{ZSK^{-}_{parent}}$(RR DS embedding $\mathbf{KSK^{+}_{child}}$)
- Child zone
  - ($\mathbf{KSK^{+}_{child}}$, $\mathbf{KSK^{-}_{child}}$)
  - ($\mathbf{ZSK^{+}_{child}}$, $\mathbf{ZSK^{-}_{child}}$)
  - $\mathbf{KSK^{-}_{child}}$ (RR DNSKEY embedding $\mathbf{ZSK^{+}_{child}}$)
  - $\mathbf{ZSK^{-}_{child}}$(RRs)

Domain Name System (DNS)

DNSSEC – Chain of trust (2/2)



Domain Name System (DNS)
DNSSEC – Example (1/2)

- Download of ZSK^{+}_{root}

```
dig . DNSKEY | grep -Ev '^($|;)' > root.keys
```

- DNSSEC sigchase on www.eurid.eu

```
dig @194.150.168.168 +sigchase
        fiorano.belnet.be AAAA
```

```
https://dnssectest.net/fiorano.belnet.be/AAAA
```

- Source: http://backreference.org/2010/11/17/dnssec-verification-with-dig/

KSK Key Signing Ceremony

Domain Name System (DNS)

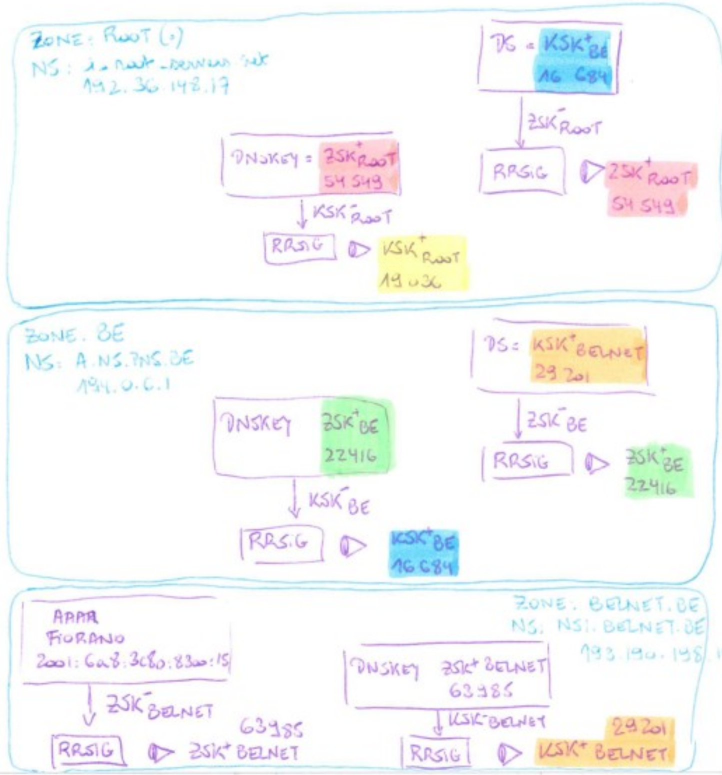
DNSSEC – Example (2/2)

Fichier : /run/media/lschumac/EXCHANGE/...DNSSEC_20160117_simplified.txt Page 1 sur 2

```
ns name: 192.36.148.17

Launch a query to find a RRset of type AAAA for zone: fiorano.belnet.be. with nameservers:
.               494692  IN      NS      i.root-servers.net.

no response but there is a delegation in authority section:be.

Launch a query to find a RRset of type DNSKEY for zone: .

;; DNSKEYset:
.               172800  IN      DNSKEY  256 3 8 [hash]
.               172800  IN      DNSKEY  257 3 8 [hash]

;; RRSIG of the DNSKEYset:
.               172800  IN      RRSIG   DNSKEY 8 0 172800 20160125235959 20160111000000 19036 . [hash]

;; Ok, find a Trusted Key in the DNSKEY RRset: 54549
;; Ok, find a Trusted Key in the DNSKEY RRset: 19036
;; VERIFYING DNSKEY RRset for . with DNSKEY:19036: success

;; DSset:
be.             86400   IN      DS      12664 8 1 [hash]
be.             86400   IN      DS      12664 8 2 [hash]
be.             86400   IN      DS      16684 8 1 [hash]
be.             86400   IN      DS      16684 8 2 [hash]

;; RRSIGset of DSset
be.             86400   IN      RRSIG   DS 8 1 86400 20160127050000 20160117040000 54549 . [hash]

;; VERIFYING DS RRset for be. with DNSKEY:54549: success
ns name: 194.0.6.1

Launch a query to find a RRset of type AAAA for zone: fiorano.belnet.be. with nameservers:
be.             172800  IN      NS      a.ns.dns.be.

no response but there is a delegation in authority section:belnet.be.

Launch a query to find a RRset of type DNSKEY for zone: be.

;; DNSKEYset:
be.             86400   IN      DNSKEY  256 3 8 [hash]
be.             86400   IN      DNSKEY  256 3 8 [hash]
be.             86400   IN      DNSKEY  256 3 8 [hash]
be.             86400   IN      DNSKEY  257 3 8 [hash]
be.             86400   IN      DNSKEY  257 3 8 [hash]

;; RRSIG of the DNSKEYset:
be.             86400   IN      RRSIG   DNSKEY 8 1 86400 20160126092452 20151217082452 16684 be. [hash]

;; OK a DS valids a DNSKEY in the RRset
;; Now verify that this DNSKEY validates the DNSKEY RRset
;; OK a DS valids a DNSKEY in the RRset
;; Now verify that this DNSKEY validates the DNSKEY RRset
;; OK a DS valids a DNSKEY in the RRset
;; Now verify that this DNSKEY validates the DNSKEY RRset
;; VERIFYING DNSKEY RRset for be. with DNSKEY:16684: success

;; DSset:
belnet.be.      86400   IN      DS      29201 8 1 [hash]
belnet.be.      86400   IN      DS      29201 8 2 [hash]


;; RRSIGset of DSset
belnet.be.      86400   IN      RRSIG   DS 8 2 86400 20160121013532 20160111010901 22416 be. [hash]

;; VERIFYING DS RRset for belnet.be. with DNSKEY:22416: success
ns name: 193.190.198.14

Launch a query to find a RRset of type AAAA for zone: fiorano.belnet.be. with nameservers:
belnet.be.      86400   IN      NS      ns1.belnet.be.

Launch a query to find a RRset of type DNSKEY for zone: belnet.be.
```

Fichier : /run/media/lschumac/EXCHANGE/...DNSSEC_20160117_simplified.txt Page 2 sur 2

```
;; DNSKEYset:
belnet.be.      3600    IN      DNSKEY  256 3 8 [hash]
belnet.be.      3600    IN      DNSKEY  256 3 8 [hash]
belnet.be.      3600    IN      DNSKEY  257 3 8 [hash]
belnet.be.      3600    IN      DNSKEY  256 3 8 [hash]

;; RRSIG of the DNSKEYset:
belnet.be.      3600    IN      RRSIG   DNSKEY 8 2 3600 20160128090019 20160107043348 29201 belnet.be.
[hash]

;; OK a DS valids a DNSKEY in the RRset
;; Now verify that this DNSKEY validates the DNSKEY RRset
;; VERIFYING DNSKEY RRset for belnet.be. with DNSKEY:29201: success
;; VERIFYING AAAA RRset for fiorano.belnet.be. with DNSKEY:63985: success

;; The Answer:
fiorano.belnet.be.      300     IN      AAAA    2001:6a8:3c80:8300::15

;; FINISH : we have validate the DNSSEC chain of trust: SUCCESS
```



Domain Name System (DNS)
DNS over HTTPS (DOH, RFC 8484, October 2018)

- Goals
  - Achieve confidentiality between DNS clients
  - Avoid UDP fragmentation in case of large DNS payloads
  - Avoid DDoS with spoofed UDP source addresses
  - Combine web and DNS services
    - through a single HTTP/2 session
    - going through a single TCP connection to port 443
- JSON formatting available. Example

```
curl -s -H 'accept: application/dns+json'
'https://dns.google.com/resolve?
name=www.potaroo.net&type=A' | jq
```

Domain Name System (DNS)

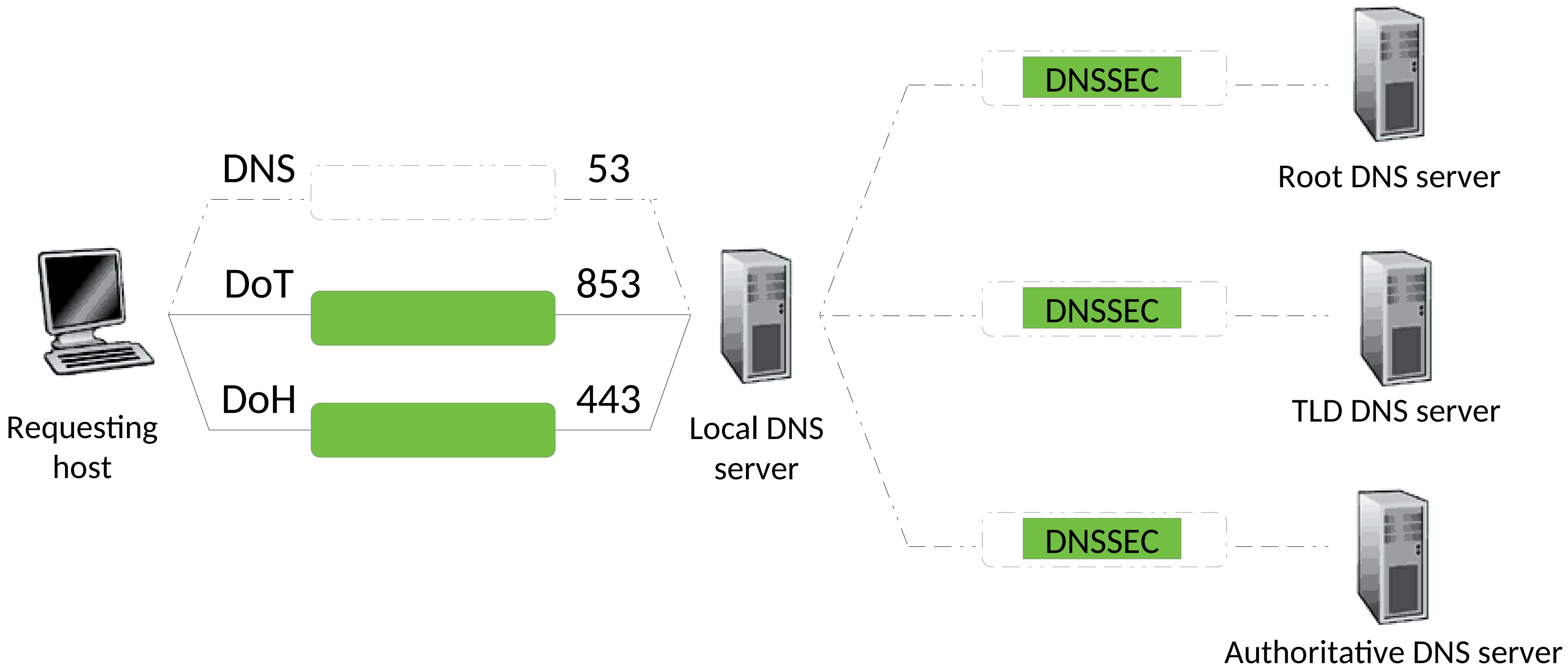
DNS over HTTPS (DOH, RFC 8484, October 2018)

```
curl -s -H 'accept: application/dns+json'
'https://dns.google.com/resolve?name=www.potaroo.net&type=A'
| jq
```


```
{
  "Status": 0,
  "TC": false,
  "RD": true,
  "RA": true,
  "AD": true,
  "CD": false,
  "Question": [
    {
      "name": "www.potaroo.net.",
      "type": 1
    }
  ],
  "Answer": [
    {
      "name": "www.potaroo.net.",
      "type": 1,
      "TTL": 6275,
      "data": "203.133.248.2"
    }
  ]
}
```

Domain Name System (DNS)

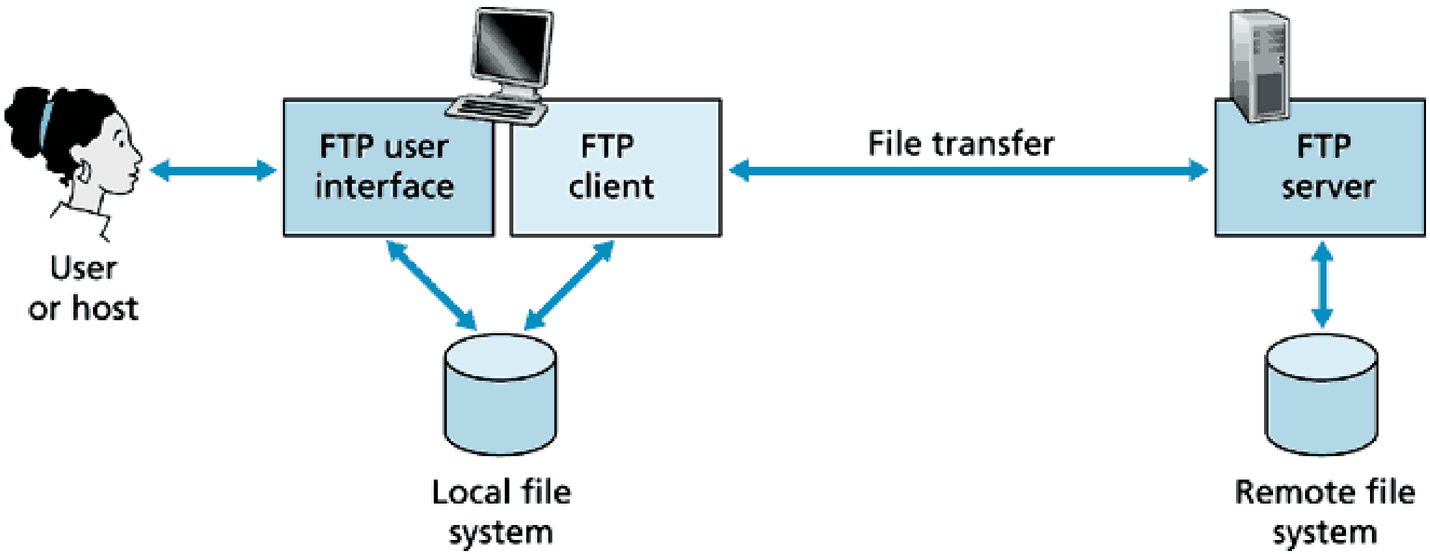
Overall sketch



Outline

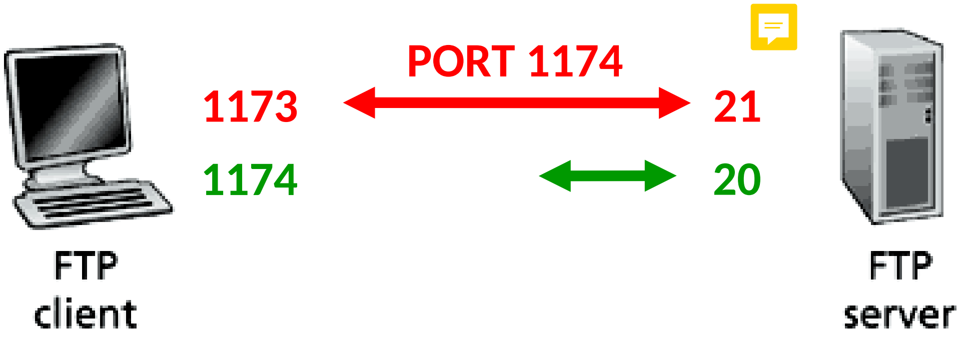
Principles of application layer protocols
Web browsing and HTTP
Domain name resolution (DNS)
File transfer (FTP)
Electronic Mail: SMTP, POP3, IMAP
Content distribution: Web caching, CDN, P2P
Socket programming with TCP
Socket programming with UDP
Building a simple Web server

File Transfer Protocol (FTP)



- Transfer file to/from remote host
- Client/server model
- Originally described in RFC 959
- Uses server ports 20 and 21

File Transfer Protocol (FTP)
Separate Data Plane and Control Plane



Control

- FTP client contacts FTP server at port 21, specifying TCP as transport protocol and port 1174 as ephemeral data port
- Client obtains authorisation over control connection
- Client browses remote directory by sending commands over control connection.

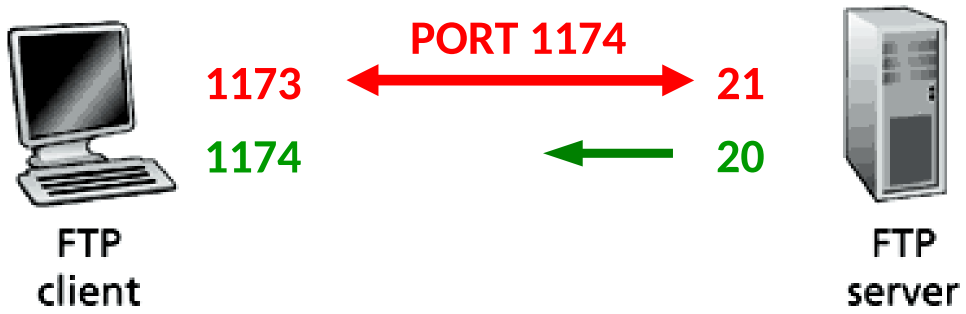
Data

- When server receives a command for a file transfer, the server opens a TCP data connection to client at its port 20
- After transferring one file, the server closes TCP data connection.
- Server opens a second TCP data connection to transfer another file.

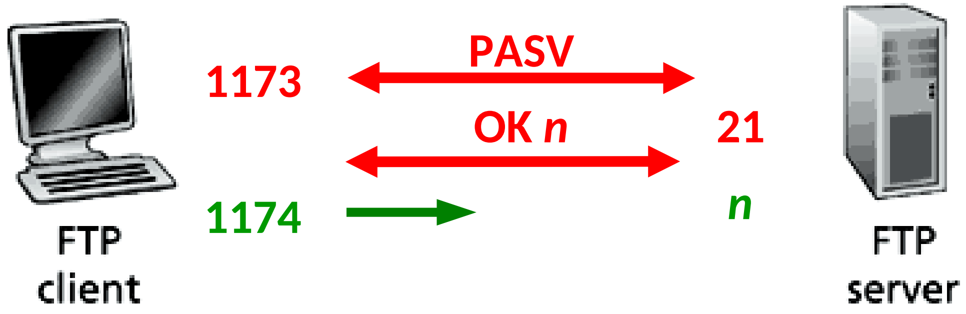
- Control connection: “**out of band**”
- FTP server maintains “**state**”: current directory, earlier authentication

File Transfer Protocol (FTP)
Active versus Passive Mode

- Active mode



- Passive mode



File Transfer Protocol (FTP)
Commands and responses

- Sample commands
  - Sent as ASCII text over control channel
  - **`USER`** ***`username`***
  - **`PASS`** ***`password`***
  - **`LIST`** returns 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`**

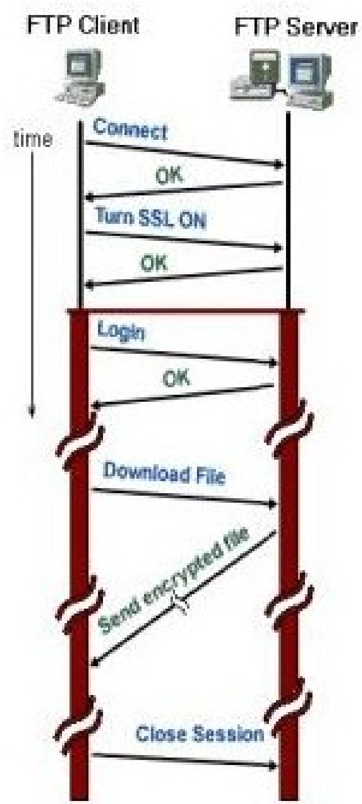
File Transfer Protocol (FTP)
Anonymous FTP

- Before Cloud Computing, Dropbox, etc.
- Log to ftp servers using e-mail address as password
  - Name: anonymous
  - Password: pierre.dupont@info.unamur.be
- Some servers check the received source IP address
- Perform an address-to-hostname mapping
- If this check fails (no mapping),

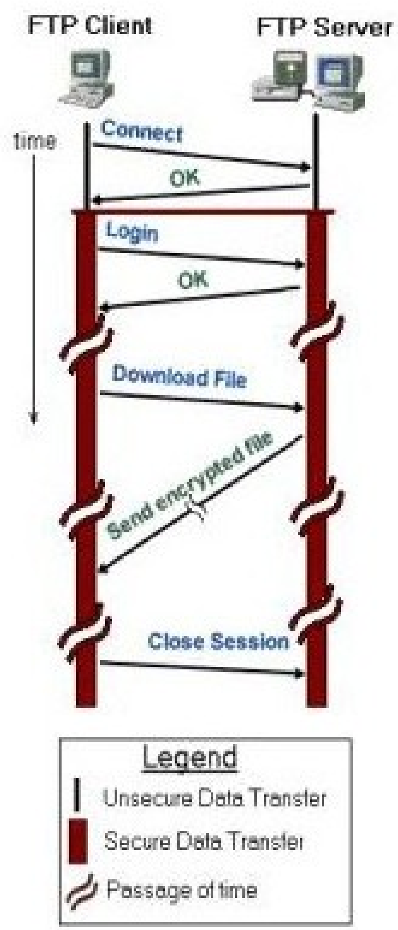
```
530 User anonymous access denied.
```

File Transfer Protocol (FTP)
FTP over SSL (FTPS)

Explicit – Port 21



Implicit – Port 990

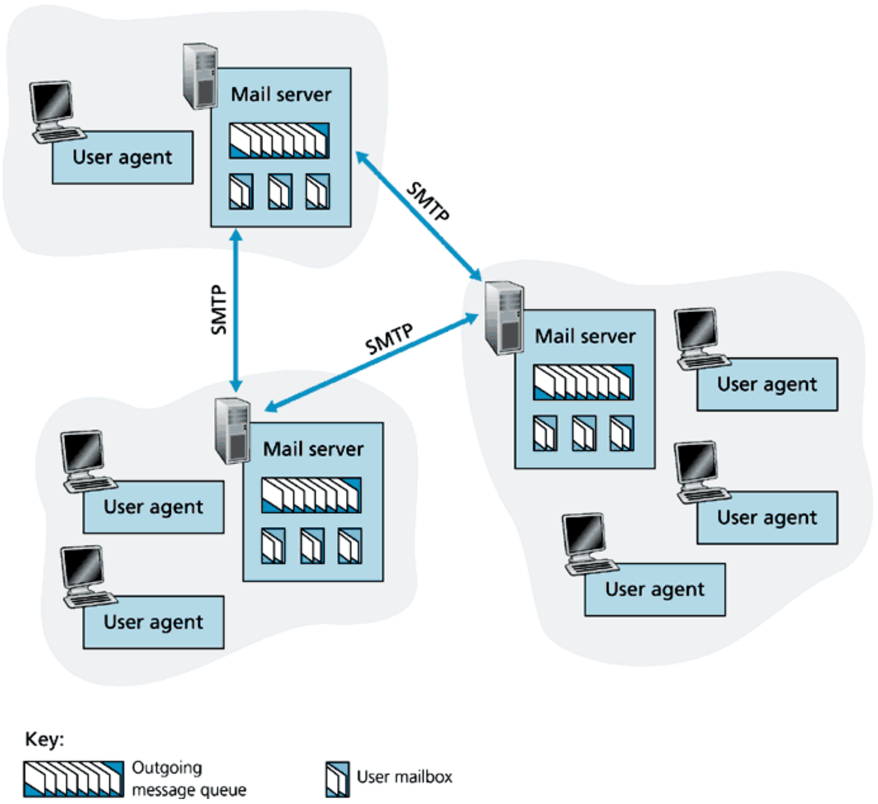


- Two methods for invoking security
- Explicit
  - Port 21
  - New FTP command AUTH to trigger SSL cipher negotiation
- Implicit
  - Port 990
  - Client immediately sends SSL Client Hello message
- Also SSH File Transfer Protocol (SFTP)

Outline

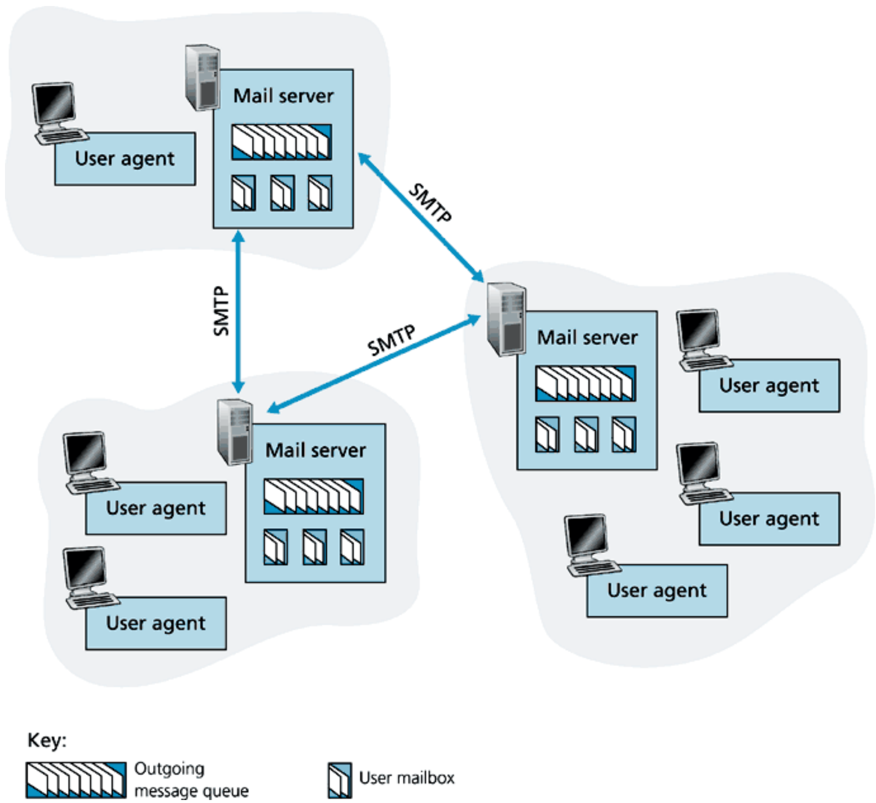
Principles of application layer protocols
Web browsing and HTTP
Domain name resolution (DNS)
File transfer (FTP)
Electronic Mail: SMTP, POP3, IMAP
Content distribution: Web caching, CDN, P2P
Socket programming with TCP
Socket programming with UDP
Building a simple Web server

Electronic mail
Components – User agent



- Three major components:
  - User agents
  - Mail servers
  - Simple Mail Transfer Protocol (SMTP)
- User agent
  - Composing, editing, reading mail messages
  - Examples: Thunderbird, Outlook, pine
  - Incoming and outgoing messages stored on server

Electronic mail
Mail server



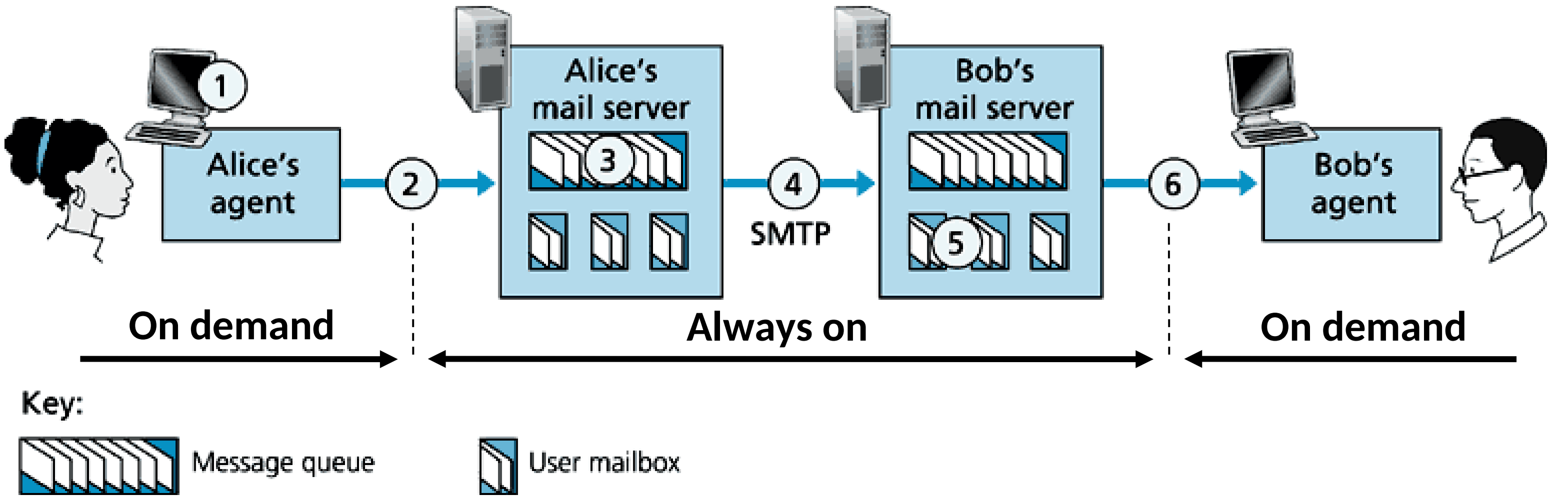
- Mailbox contains incoming messages for user
- Message queue of outgoing (to be sent) mail messages
- SMTP protocol between mail servers to exchange e-mail messages
  - Client: sending mail server
  - Server: receiving mail server
- Relaying possible if SMTP relay known from DNS

Electronic mail
Simple Mail Transfer Protocol (SMTP)

- Originally described in RFC 821 (August 1982)
- Uses TCP to reliably transfer email messages
- Cleartext Considered Obsolete: Use of TLS for Email Submission and Access (RFC 8314, January 2018)
- Default port : 587 (25 in unsecured SMTP)
- 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

Electronic mail

Scenario: Alice sends message to Bob



- Alice uses her agent to compose message "to" bob@someschool.edu
- Alice's user agent sends message to her mail server; message placed in outgoing message queue
- Client side of SMTP opens TCP connection with Bob's mail server
- SMTP client sends Alice's message over the TCP connection
- Bob's mail server places the message in Bob's mailbox
- Bob invokes his user agent to read message

Electronic mail
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
```

Electronic mail
Try SMTP interaction

- The following commands enable you to send e-mail without using an e-mail client (reader)
  - **`telnet servername 25`**
  - See **`220 reply`** from server
  - Enter **`HELO`**, **`MAIL FROM`**, **`RCPT TO`**, **`DATA`**, **`QUIT`** commands

Electronic mail

HTTP vs. SMTP

HTTP	SMTP
Pull	Push
7-bit ASCII command/response interaction, status codes	
No restriction on content	7-bit ASCII only
Encapsulates each object in its own response message	Multiple objects sent in a single multipart message

Electronic mail

Internationalisation

- Messages must be in 7-bit ASCII
- Issue with globalisation (Unicode)
- Encoding schemes
  - Quoted-printable: 1 Byte → 3 x 7-bit characters

Character	8-bit encoding	Quoted-printable
é	0xe9	=E9
[space]	0x20	=20

  - Base 64: 3 Bytes → 4 x 6-bit values according to table

U	N	A	M	U	R
0x55	0x4e	0x41	0x4d	0x55	0x52
01010101	01001110	01000001	01001101	01010101	01010010

0x15	0x14	0x39	0x01	0x13	0x15	0x15	0x12
P	O	n	B	N	P	P	M

Electronic mail
Multimedia Extensions (MIME)

- Multimedia Mail Extension (MIME, originally RFC 2045)
- Additional lines in message header declare MIME content type

MIME version

Method used to encode data

Multimedia data type, subtype, parameter declaration

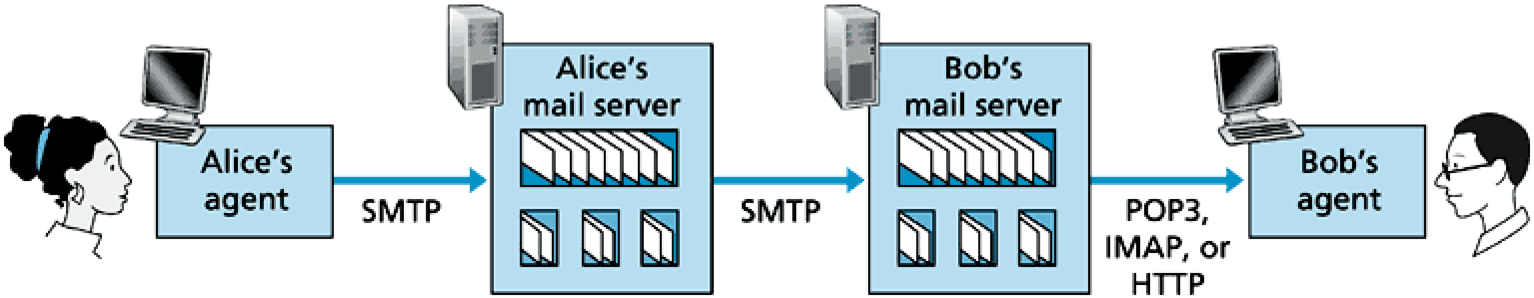
Encoded data

```
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
```

Electronic mail
MIME – 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?
```

Electronic mail
Mail access protocols (1/2)



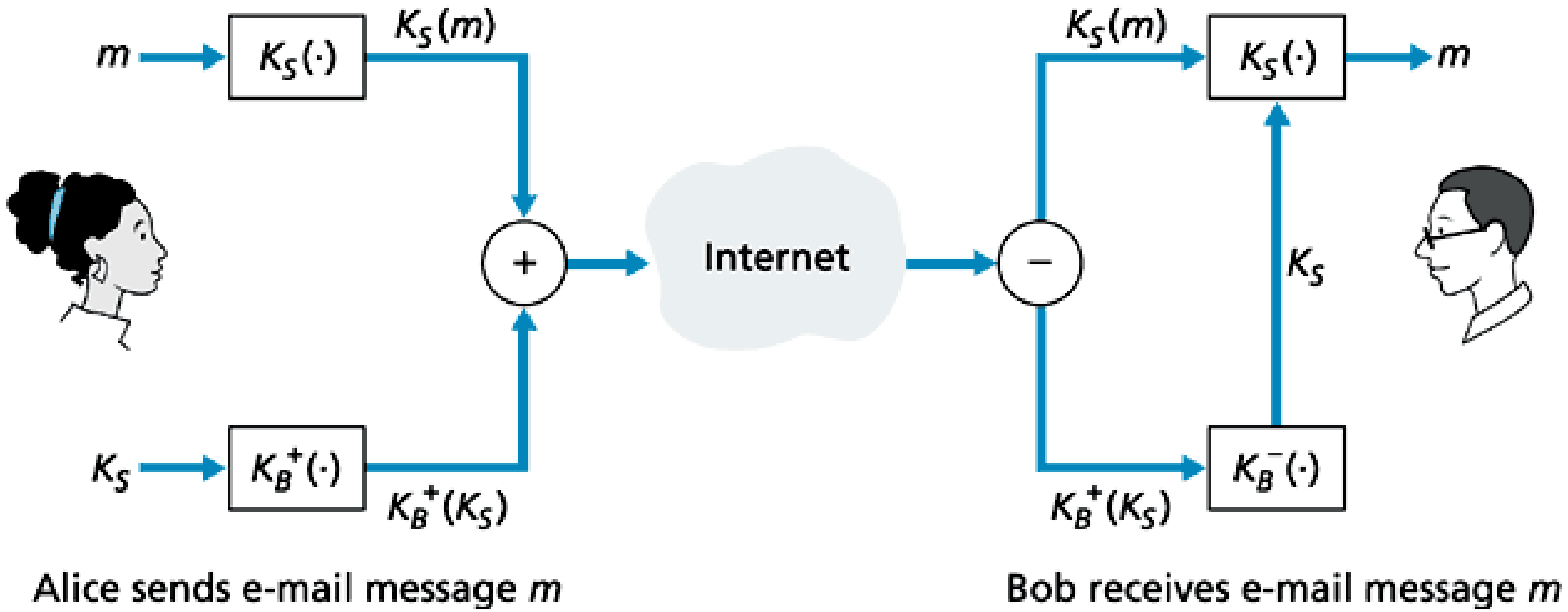
- SMTP – Delivery/storage to receiver's server
- Mail access protocol: retrieval from server
  - Post Office Protocol (POP version 3, RFC 1939): authorisation, download and delete or keep on server
  - Internet Mail Access Protocol (IMAP, RFC 1730)
    - Manipulation of stored messages on server
    - More convenient for nomadic users
  - Web-based e-mail (e.g. GMail): relies on HTTP

Electronic mail
Mail access protocols (2/2)

- POP3
  - Two modes
    - “Download and delete” - Bob cannot re-read e-mail if he changes client
    - “Download-and-keep” - Bob generates copies of messages on the different clients he uses
  - POP3 is stateless across sessions
- IMAP
  - Keep all messages in one place: the server
  - Allows user to organize messages in folders
  - IMAP is stateful, i.e. keeps user state across sessions
    - Names of folders
    - Mappings between message IDs and folder name

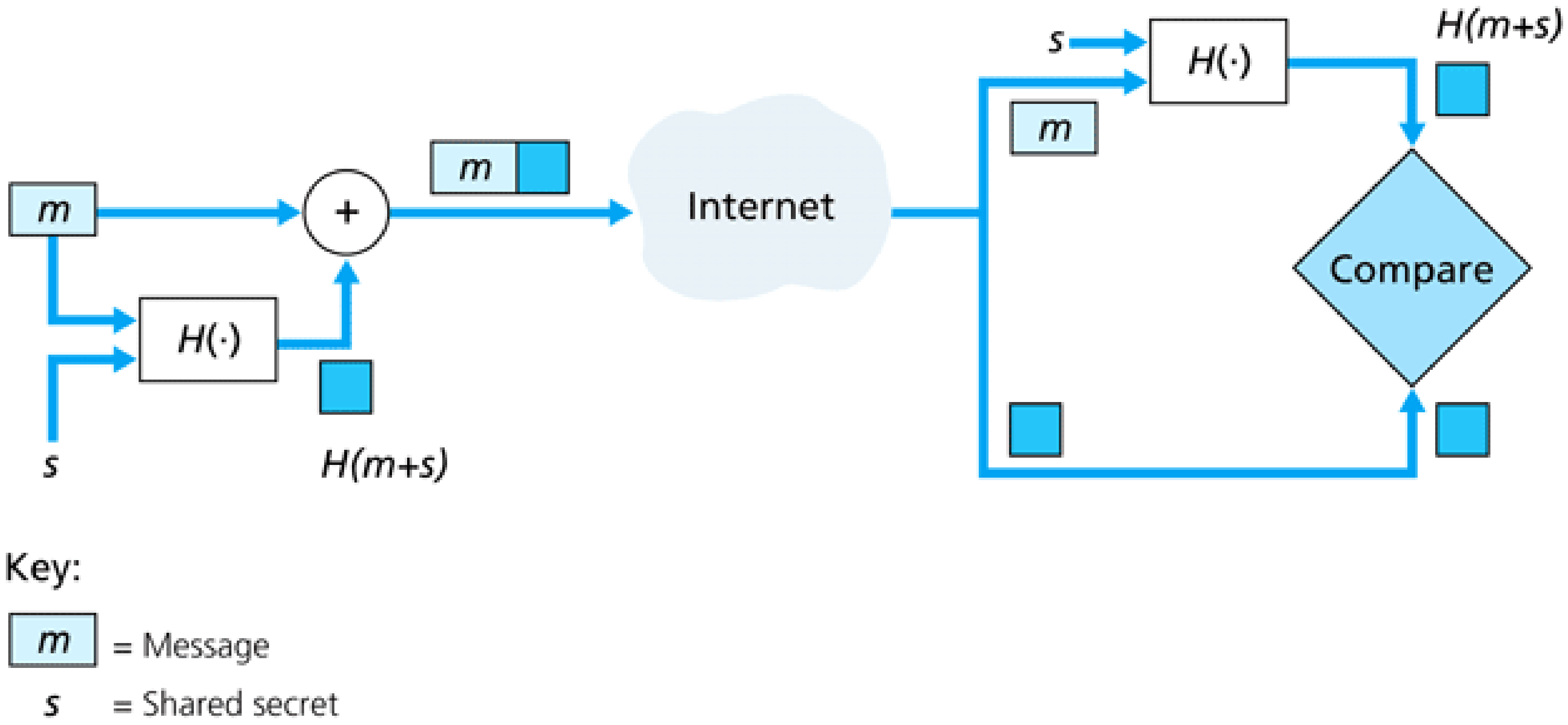
Electronic mail
Confidentiality

- The message *m* is ciphered with symmetric key K_S
- Symmetric key K_S is shared with public key K_B^+



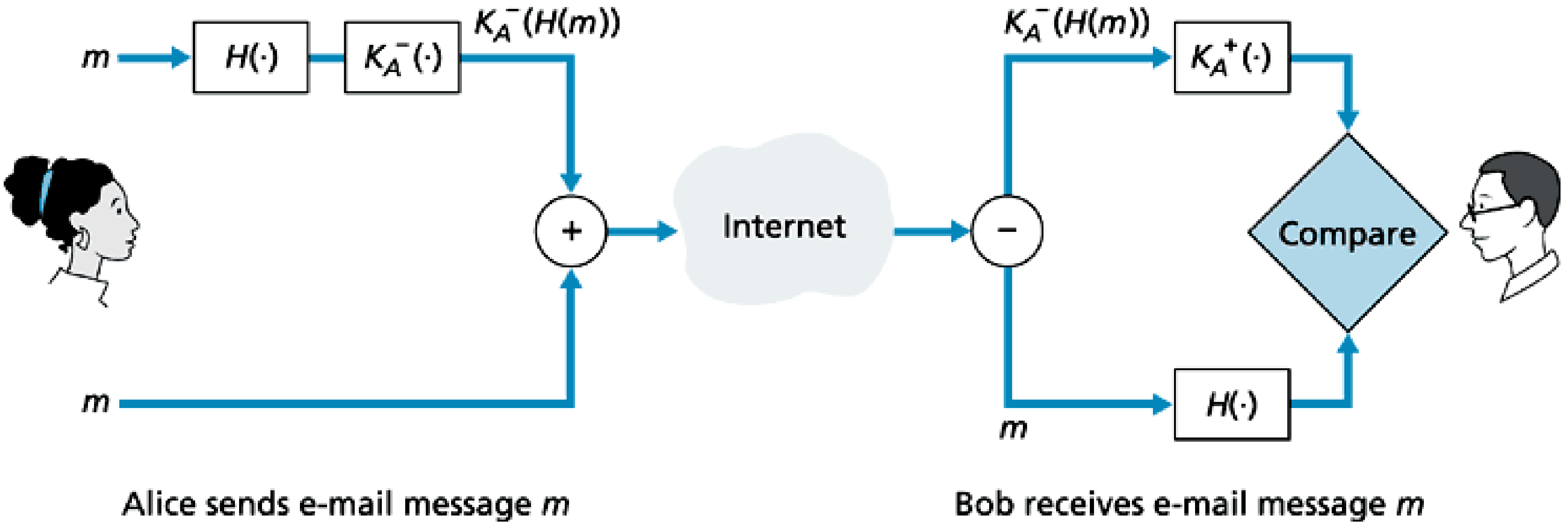
Electronic mail
Integrity – *Message Authentication Code* (MAC)

The message *m* is complemented with a digest $H(m+s)$



Electronic mail
Authentication

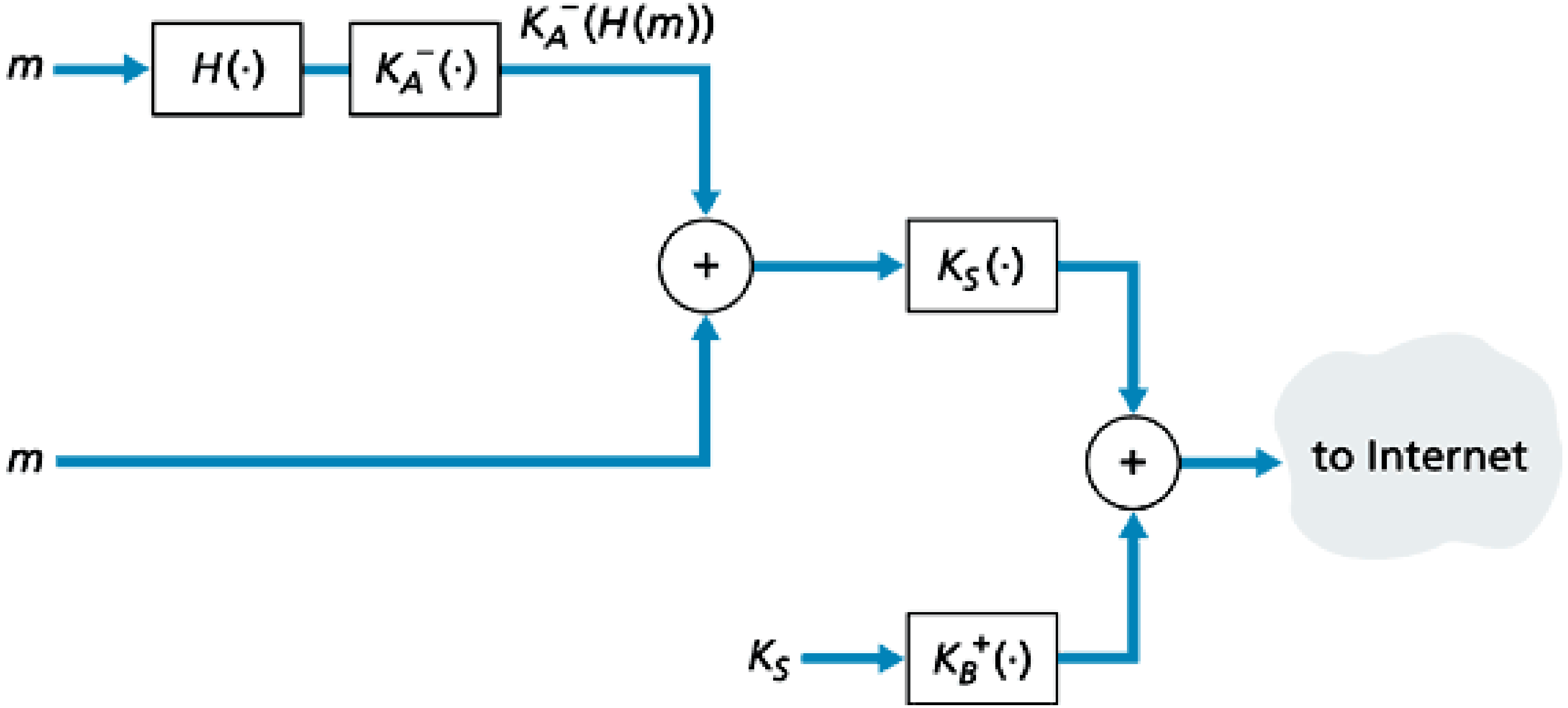
- Digest $H(m)$ is signed with private key K_A^-
- Bob compares digests, received and computed



Electronic mail

CIA = Confidentiality + Integrity + Authentication

- Digest $H(m)$ is signed with private key K_A^-
- Message and digest ciphered with symmetric key K_S is shared with public key K_B^+



Electronic mail

Pretty Good Privacy (PGP)

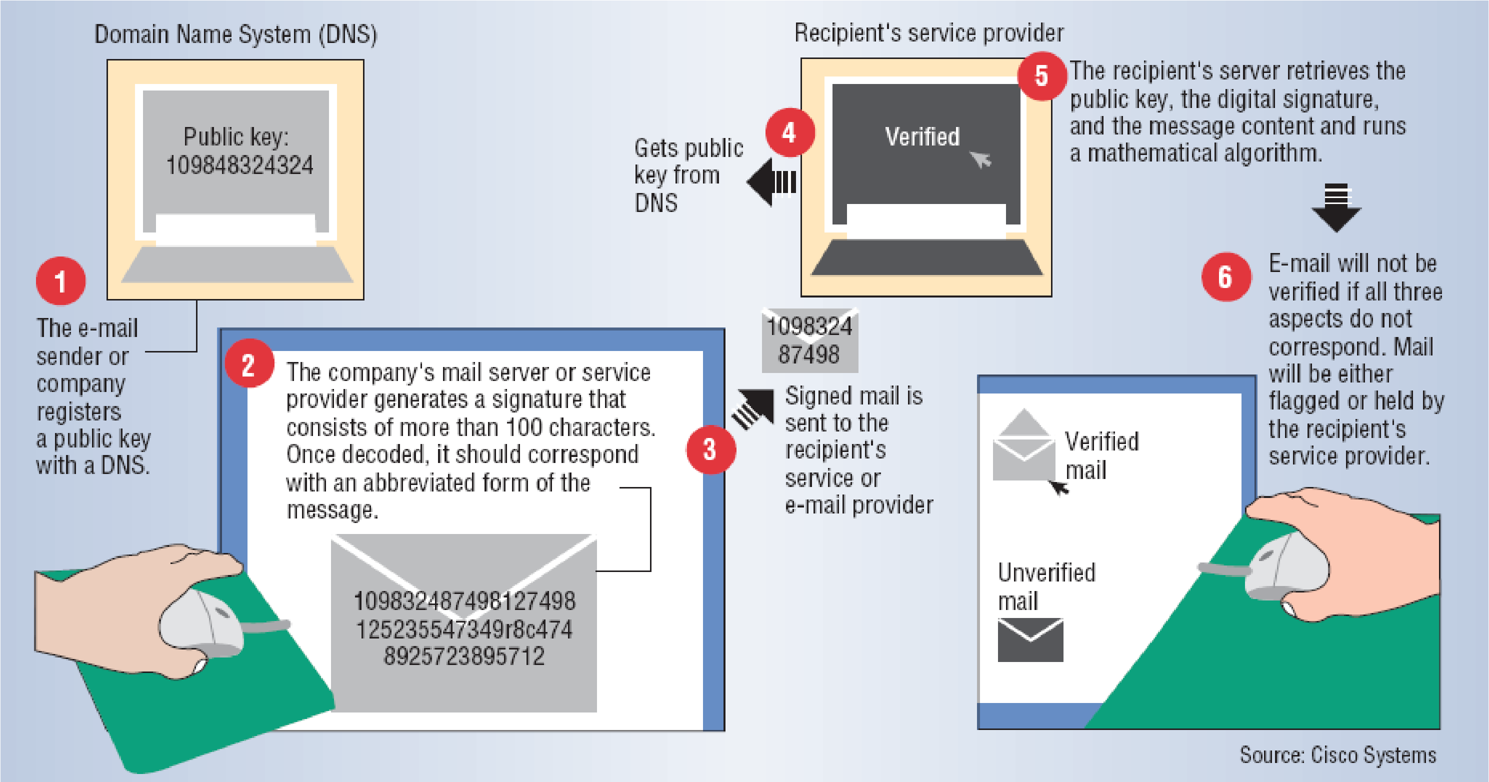
```
---BEGIN PGP SIGNED MESSAGE---
Hash: SHA1

Bob: My husband is out of town
   tonight. Passionately yours,
   Alice

---BEGIN PGP SIGNATURE---
Version: PGP 5.0
Charset: noconv
yhHJRHhGJGhgg/
   12EpJ+lo8gE4vB3mqJhFEvZP9t6n
   7G6m5Gw2
---END PGP SIGNATURE---
```

$K_A^-(H(m))$

- Standard *de facto*
- Ensures
  - Confidentiality (IDEA, CAST, 3DES, AES)
  - Authentification (signature RSA, ECC)
  - Integrity (hash MD5, SHA)
- Hack a PGP message of 1,024 bits requires 300 billion MIPS years
- Keys certified through « chains of trust »

Electronic mail
DomainKeys, SenderID

- Messages signed by outgoing SMTP server
- Public keys available through DNS request



Source: IEEE Computer Magazine, November 2005

Outline

Principles of application layer protocols

Web browsing and HTTP

Domain name resolution (DNS)

File transfer (FTP)

Electronic Mail: SMTP, POP3, IMAP

Content distribution: Web caching, CDN, P2P

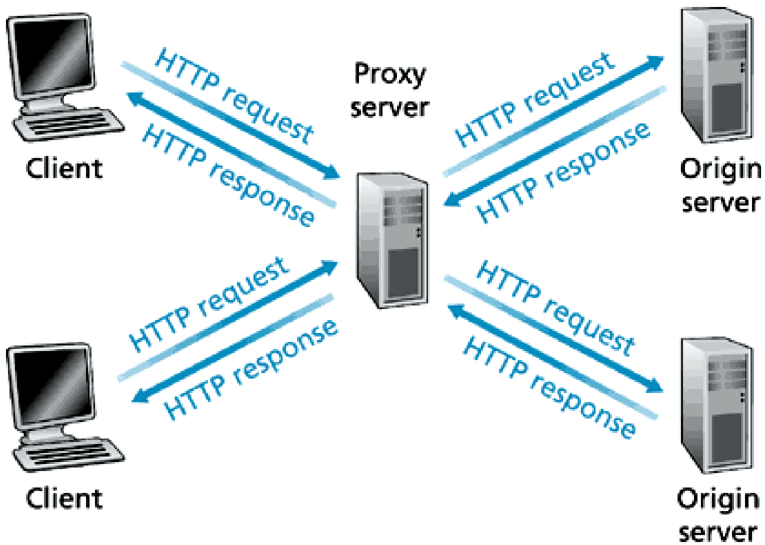
Socket programming with TCP

Socket programming with UDP

Building a simple Web server

Content distribution

Web caching, an example of proxying



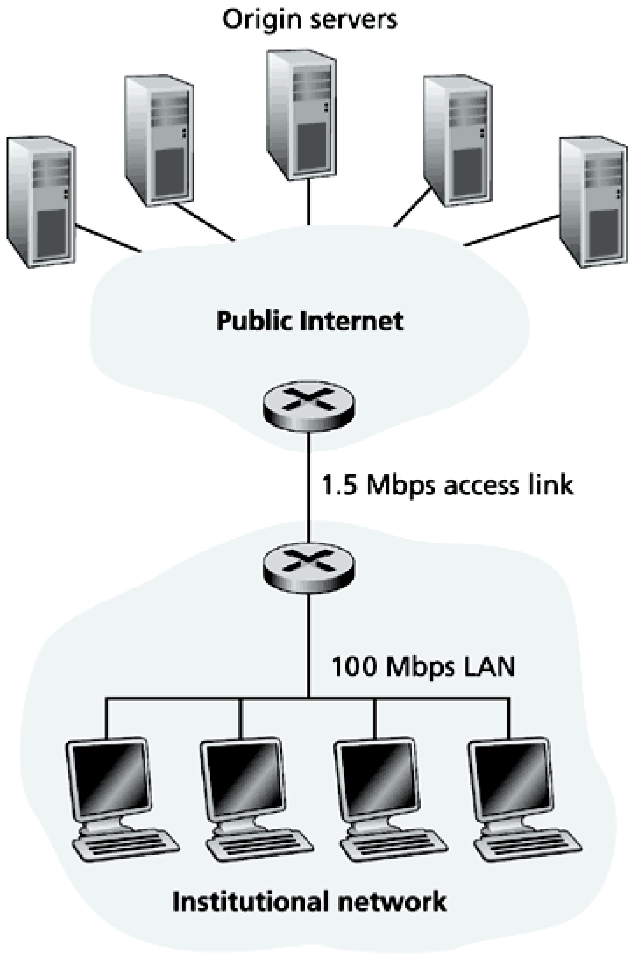
- Goal: satisfy client request without involving origin server
- User sets browser: Web accesses via cache
- Browser sends all HTTP requests to cache
    - Object in cache: cache returns object
    - Else, cache requests object from origin server, then returns object to client

Middlebox (RFC 3234, Feb. 2002): any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host

Content distribution
Web caching – Discussion

- Cache acts as both client and server
- Typically proxy server is installed by ISP
- Web caching = content distribution
    - Locally replicate content on demand
    - Benefit for content provider: Internet dense with caches enables “poor” content providers to effectively deliver content
    - Benefit for institution: reduce traffic on outbound link
    - Benefit for end user: reduce response time
- Cache can do up-to-date check using **`If-modified-since`** HTTP header
    - Issue: should cache take risk and deliver cached object without checking?
    - Heuristics are used

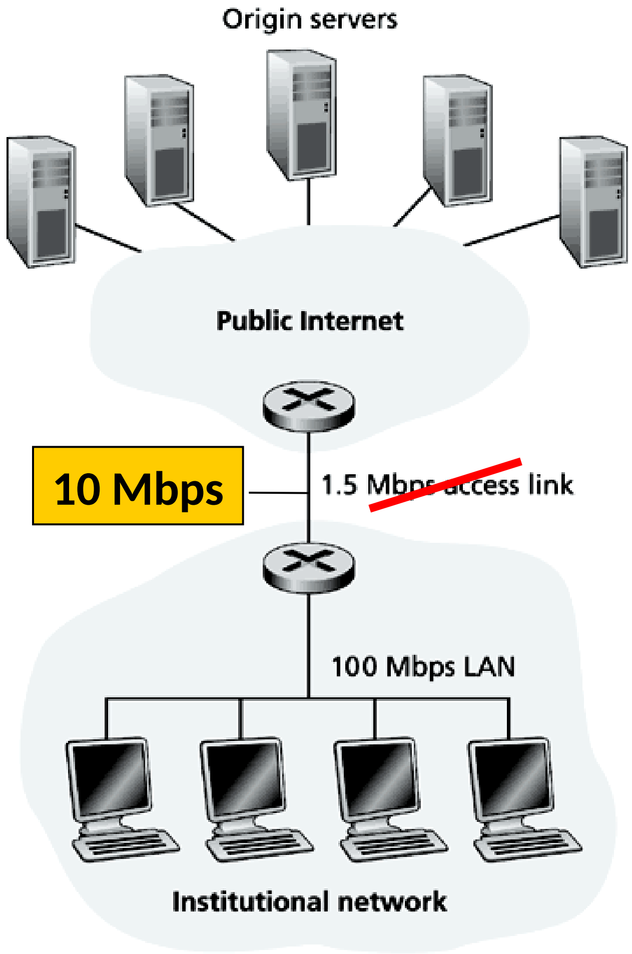
Content distribution
Web caching example (1/3)



- Assume
  - Average object size = 100 kbits
  - Average request rate from institution's browser to origin servers = 15/s
  - Delay from institutional router to any origin server and back to router = 2 s

- LAN utilisation = 1.5%
- Access link utilisation = 100%
- Total delay
  = LAN delay + access link delay
  + Internet delay
  = ms + minutes + 2 s

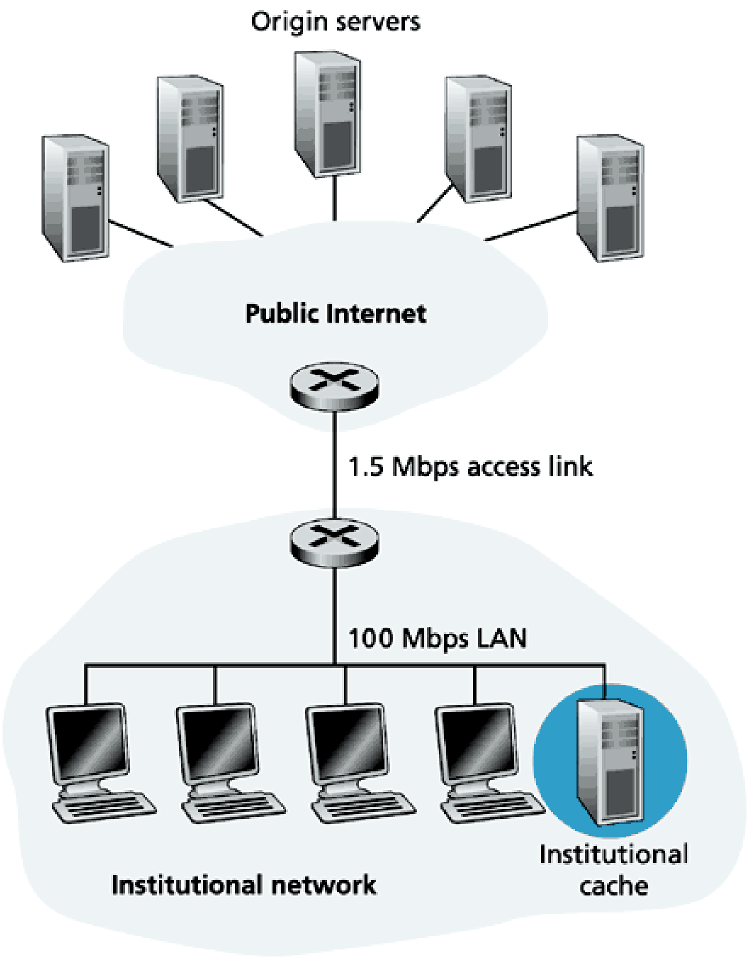
Content distribution
Web caching example (2/3)



- Assume
  - Average object size = 100 kbits
  - Average request rate from institution's browser to origin servers = 15/s
  - Delay from institutional router to any origin server and back to router = 2 s

- LAN utilisation = 1.5%
- Access link utilisation = 15%
- Total delay
  = LAN delay + access link delay
  + Internet delay
  = ms + ms + 2 s

Content distribution
Web caching example (3/3)

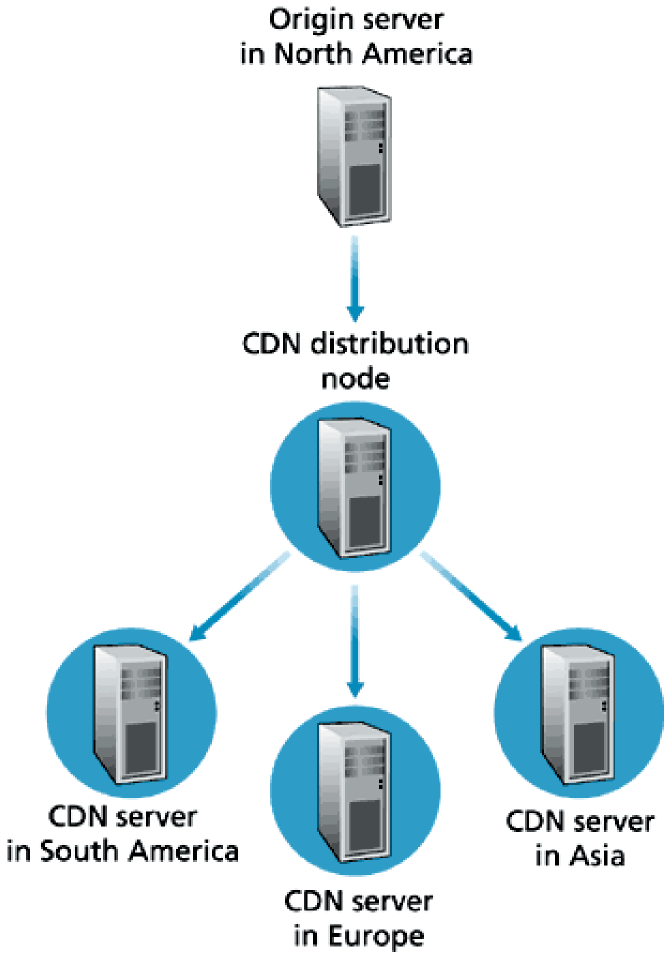


- Assume
  - Average object size = 100 kbits
  - Average request rate from institution's browser to origin servers = 15/s
  - Delay from institutional router to any origin server and back to router = 2 s
  - Proxy server hit rate = 0.4

- LAN utilisation = 1.5%
- Access link utilisation = 60%
- Total delay
  = LAN delay + access link delay + Internet delay
  = 0.4 * 1 ms + 0.6 (2 s + 10 ms)
  = 1.206 s

Content distribution

Content Distribution Network (CDN)

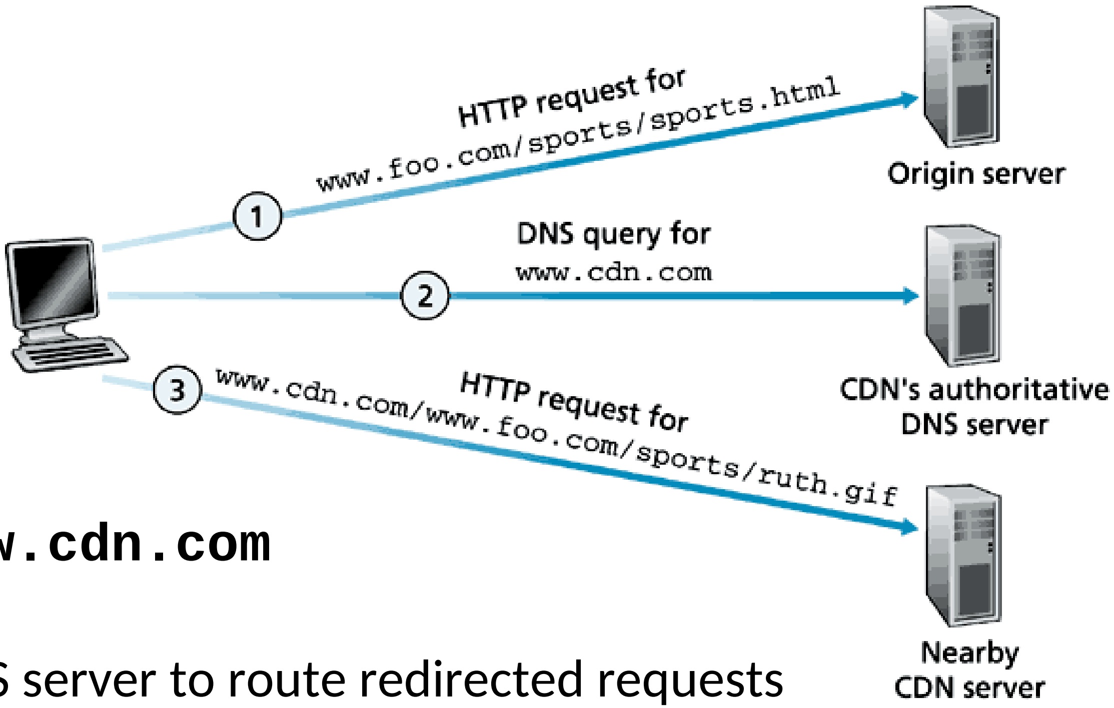


- The content providers are the CDN customers.
- CDN company installs hundreds of CDN servers throughout Internet
- In lower-tier ISPs, close to users
- CDN replicates its customers' content in CDN servers. When provider updates content, CDN updates servers

Content distribution
CDN Example

Origin server **`http://www.foo.com`**

- Distributes HTML
- Replaces **`http://www.foo.com/sports/ruth.gif`**
  with **`http://www.cdn.com/www.foo.com/sports/ruth.gif`**



CDN company **`http://www.cdn.com`**

- Distributes GIF files
- Uses its authoritative DNS server to route redirected requests

Content distribution
How to determine best CDN server?

- CDN creates a “map”, indicating distances from leaf ISPs and CDN nodes
- When a query arrives at authoritative DNS server
  - Server determines ISP from which query originates
  - Uses “map” to determine best CDN server
- CDN does not distribute only Web pages
  - Streaming stored audio/video
  - Streaming real-time audio/video

Content distribution
Operational CDNs (January 2018)

- Commercial
  - Akamai
    - Market leader – 15-30% web traffic (April 2015)
    - 240,000+ servers in 1,600 networks across 130+ countries
  - Limelight Networks: 950+ access networks around the world
  - Both use DNS-based request-routing for redirection
- Academic
  - CoDeeN

Content distribution

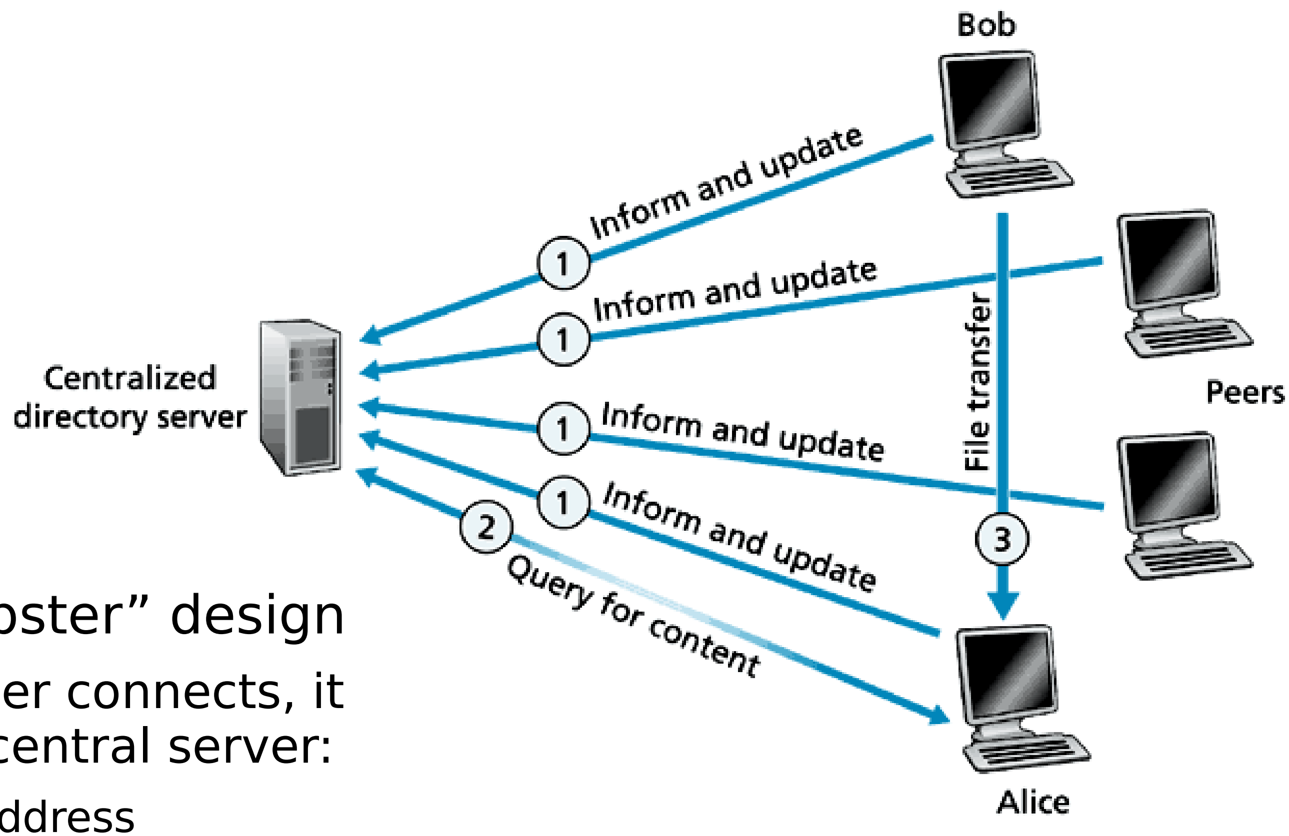
P2P Taxonomy (RFC 5694, November 2009)

- P2P is more than file sharing. From IAB :
  - “[T]he elements that form the [P2P] system **share their resources** in order to provide the service the system has been designed to provide.
  - The elements in the system **both** provide services to other elements and request services from other elements.”
- P2P applications
  - File sharing like KaZaa, Gnutella, BitTorrent, eDonkey/eMule, etc.
  - Distributed computing: parallelism, grid/cloud computing. Caution: SETI@home and Folding not regarded P2P by IAB because service requests centrally generated by master node.
  - Collaboration: instant messaging, Skype, p2PSIP
  - Platforms like JXTA (peer discovery, grouping and peer communication)

Content distribution
P2P file sharing

- Scenario
  - Alice runs P2P client application on her computer
  - Intermittently connects to Internet; gets new IP address for each connection
  - Asks for a given file
  - Alice chooses one of the peers, Bob.
  - File is copied from Bob's device to Alice's device
  - While Alice downloads, other users uploading from Alice
  - Alice's peer is both a Web client and a transient Web server
- Architectures
  - Centralised: Napster, BitTorrent
  - Query flooding: original Gnutella
  - Decentralised: Kazaa

Content distribution
P2P – Centralised directory

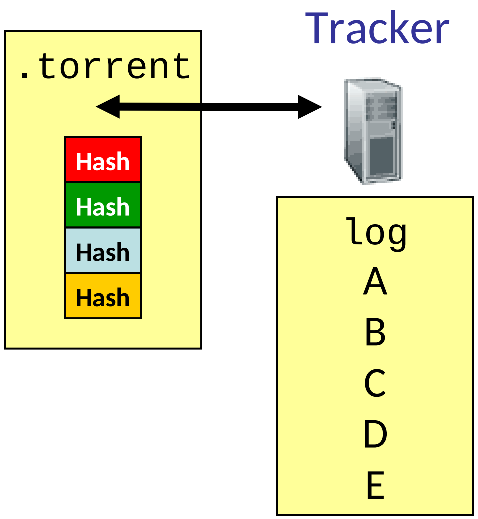


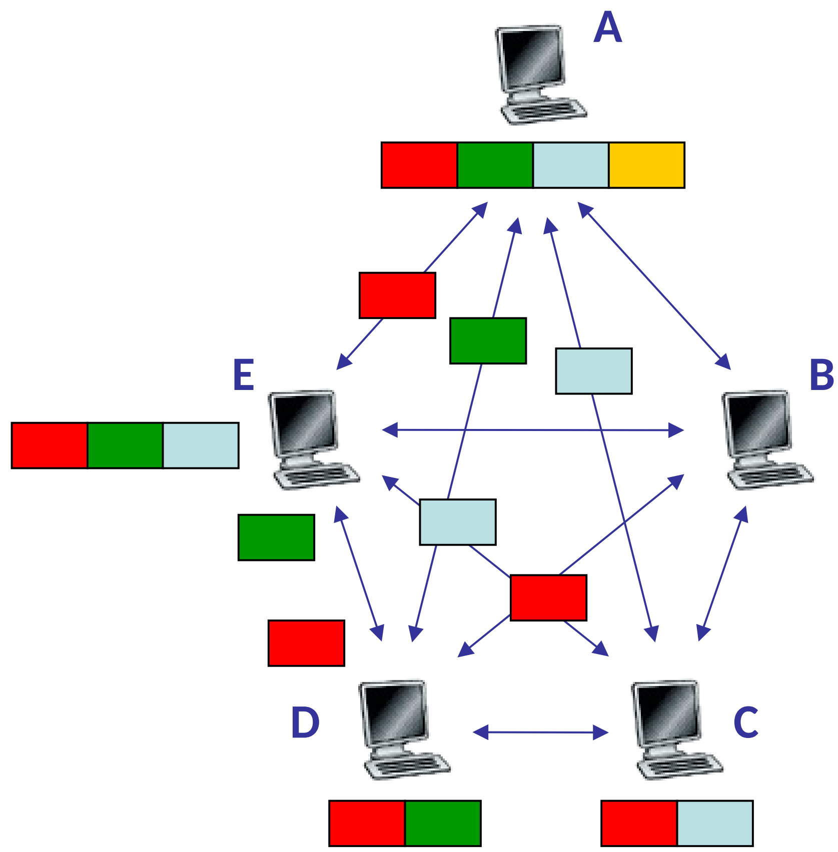
- Original “Napster” design
  - When peer connects, it informs central server:
    - IP address
    - Content
  - Alice queries for “Hey Jude”
  - Alice requests file from Bob

Content distribution
P2P – Centralised directory

- Swarming = collect chunks of files instead of whole file from multiple nodes
- Relies on tracker servers logging all users of users interested in the file
- Three main actions
  - Publish = run a tracker server
  - Join = get a list of peers from centralised tracker server
  - Fetch = down-/up-load chunks between peers

Content distribution
P2P – Swarming



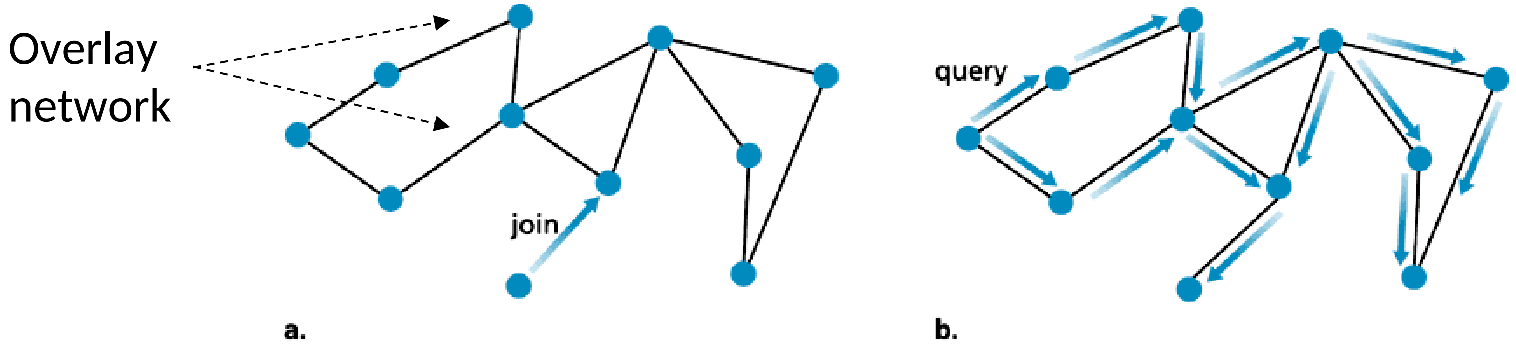


- Peers A, B, C, D and E
- A is seed (stores whole torrent file)

Content distribution
P2P – Drawbacks of centralised directory

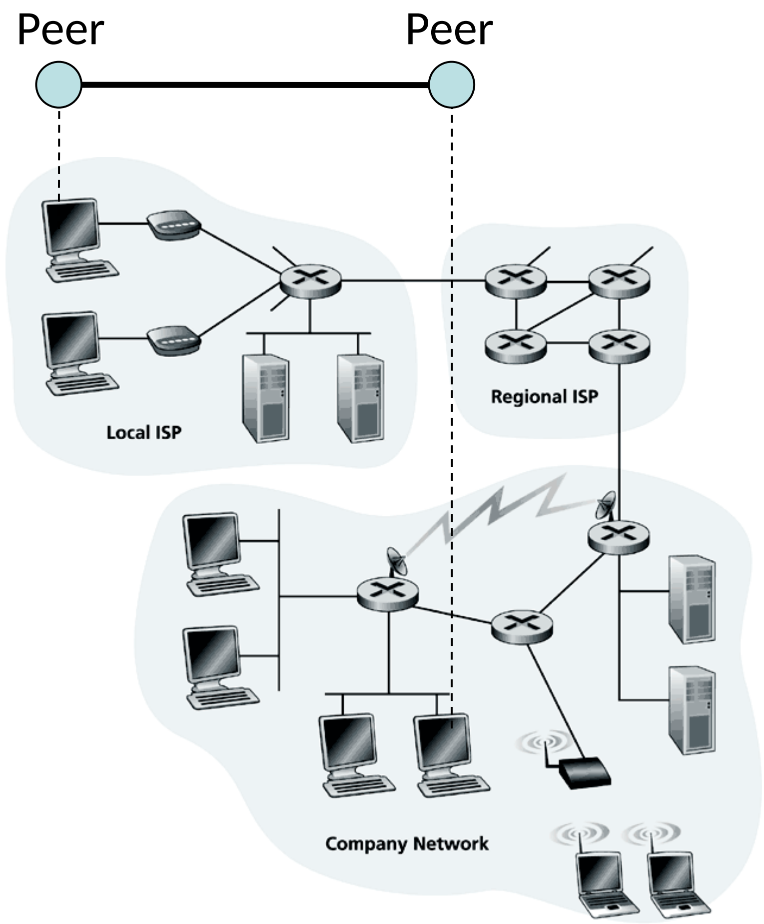
- File transfer is decentralised, but locating content is highly centralised
- SPOF (Single Point of Failure)
- Performance bottleneck – Traffic problems at centralised server
- Copyright infringement – Architecture vulnerable to legal proceedings

Content distribution
P2P – Query Flooding



- Original Gnutella design (up to version 0.4)
- Flat, no hierarchy
- Overlay network: use bootstrap node to join peers
- Query
    - Send query to neighbours
    - Neighbours forward query
    - If queried peer has object, it sends message back (query hit) to querying peer
    - Peers exchange file through direct connection

Content distribution
P2P – Centralised directory



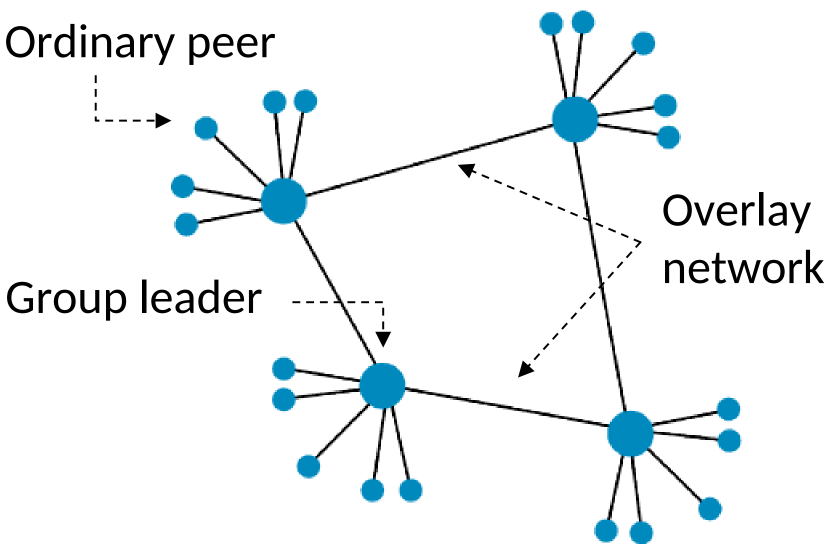
- Overlay network
  - Virtual neighbours
  - In graph-theoretic terms
    - Peers are nodes
    - Edges connect peers
- Bootstrap nodes
  - Necessary to construct the overlay network
  - Connecting peer is assigned to neighbouring peers

Content distribution
P2P – Query Flooding

Pros	Cons
Highly decentralised	Overlay network: bootstrap nodes needed, maintenance/churn
No peer maintains directory info	Excessive query traffic due to greedy users (95 % consumers only)
Peers have similar responsibilities	Query radius: may not have content when present

Content distribution
P2P – Decentralised directory

- Each peer is either a group leader or assigned to a group leader
- Group leader tracks the content in all its children.
- Peer queries group leader
- Group leader may query other group leaders



- KaZaA/FastTrack
  - Peers/elected super-peers or supernodes
- eDonkey/eMule
  - Leaves/hubs
  - Hubs are dedicated servers

Content distribution
P2P – Decentralised directory

Pros	Cons
No centralised directory server	Fairly complex protocol
Location service distributed over peers	Group leaders can get overloaded and become bottlenecks
More difficult to shut down	Group leaders not fully on par with ordinary peers
	Overlay network: bootstrap nodes needed, maintenance/churn

Content distribution

P2P – Summary

Centralised directory	Decentralised directory	Query flooding
Napster BitTorrent	Gnutella2 Kazaa/FastTrack eDonkey	Gnutella
n/a	Overlay network Relying on bootstrap nodes	
Not scalable	Scalable	Not scalable

- In 2008, P2P traffic share in Internet's total bandwidth: from 45% (Middle East) to 70% (Eastern Europa) - Source: ipoque
- OTT streaming (Over The Top), both legal (Netflix, Hulu) and illegal (Kodi) is taking over P2P users for video content.

Content distribution
P2P – OTT streaming taking over P2P in USA

Upstream		Downstream		Aggregate	
BitTorrent	18.37%	Netflix	35.15%	Netflix	32.72%
YouTube	13.13%	YouTube	17.53%	YouTube	17.31%
Netflix	10.33%	Amazon Video	4.26%	HTTP - OTHER	4.14%
SSL - OTHER	8.55%	HTTP - OTHER	4.19%	Amazon Video	3.96%
Google Cloud	6.98%	iTunes	2.91%	SSL - OTHER	3.12%
iCloud	5.98%	Hulu	2.68%	BitTorrent	2.85%
HTTP - OTHER	3.70%	SSL - OTHER	2.53%	iTunes	2.67%
Facebook	3.04%	Xbox One Games Download	2.18%	Hulu	2.47%
FaceTime	2.50%	Facebook	1.89%	Xbox One Games Download	2.15%
Skype	1.75%	BitTorrent	1.73%	Facebook	2.01%
	69.32%		74.33%		72.72%

sandvine

- BitTorrent down from 31% in 2008 to 3% in 2016
- Source : Sandvine Global Internet Phenomena Report, June 2016

Outline

Principles of application layer protocols

Web browsing and HTTP

Domain name resolution (DNS)

File transfer (FTP)

Electronic Mail: SMTP, POP3, IMAP

Content distribution: Web caching, CDN, P2P

Socket programming with TCP

Socket programming with UDP

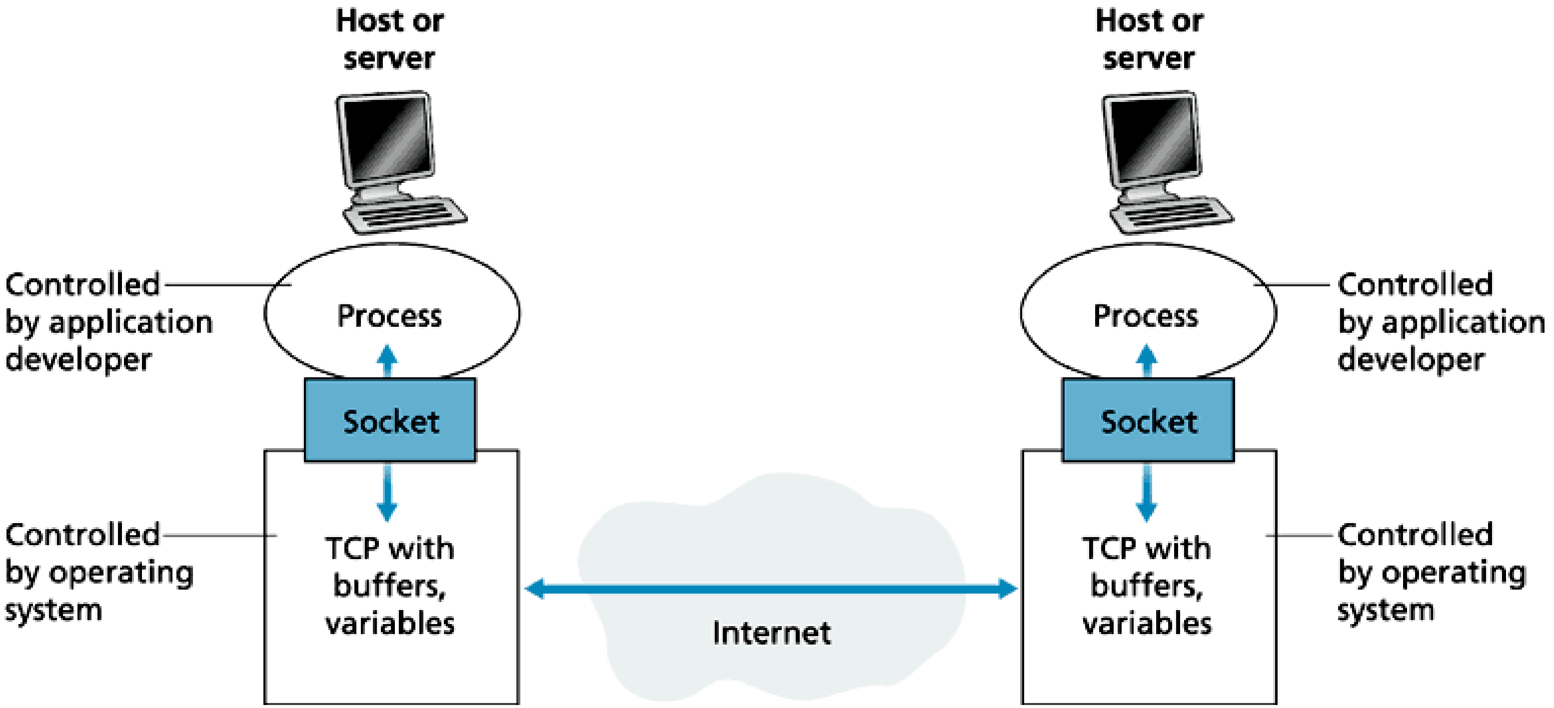
Building a simple Web server

Socket programming in Java
Introduction – System requirements

- Learn how to build client/server application that communicate using sockets
- Socket API
    - A door between application process and end-end-transport protocol (TCP or UDP)
    - Human analogy: room service in hostels
    - Explicitly created, used, released by network application
    - Client/server paradigm
    - Two types of transport service via socket API
        - Reliable, byte stream-oriented
        - Unreliable datagram
- System requirements
    - Java Runtime Environment (JRE)
    - Eclipse open source software development project

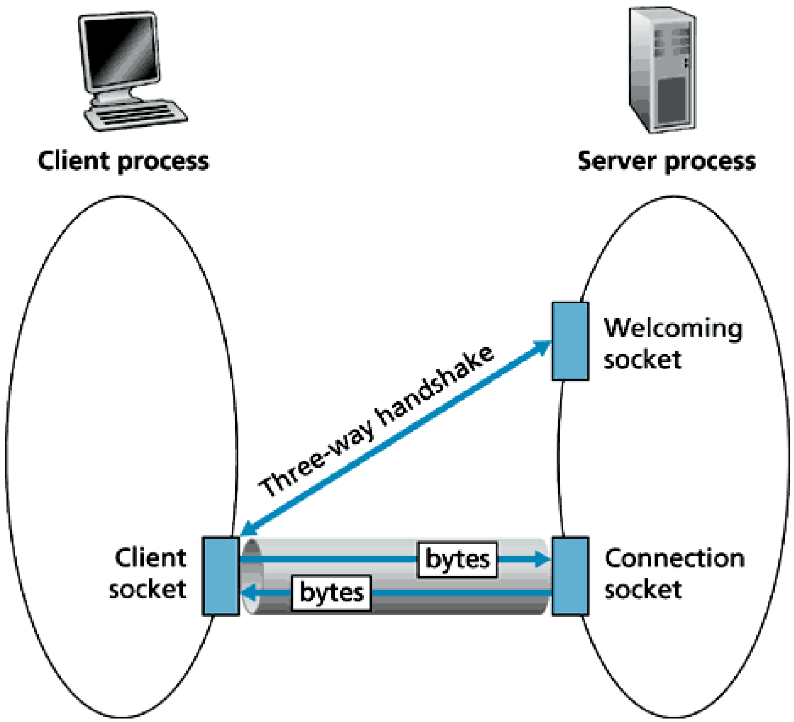
Socket programming in TCP

- TCP service: reliable, in-order transfer of bytes from one process to another
- TCP creates a pipe between client and server

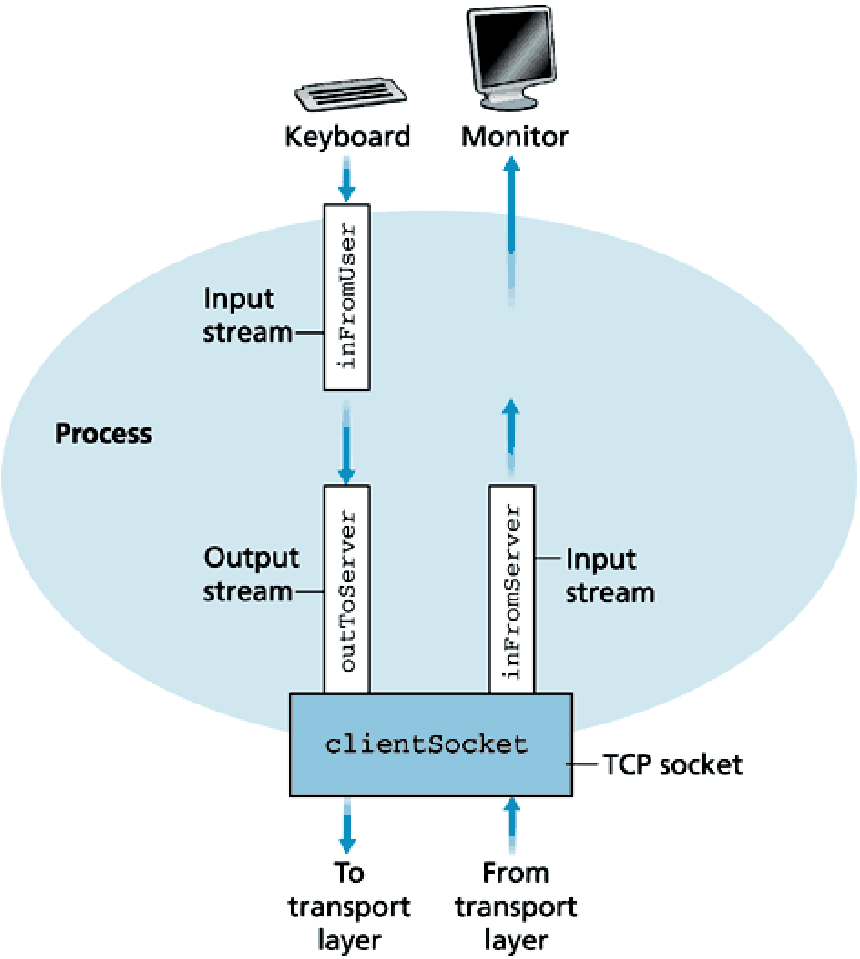


Socket programming in TCP
Sockets

- Server process must first be running
  - Server must have created <u>welcome socket</u>
- 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 <u>connection socket</u> for server process to communicate with client
    - Allows server to talk with multiple clients
    - Source port numbers used to distinguish clients

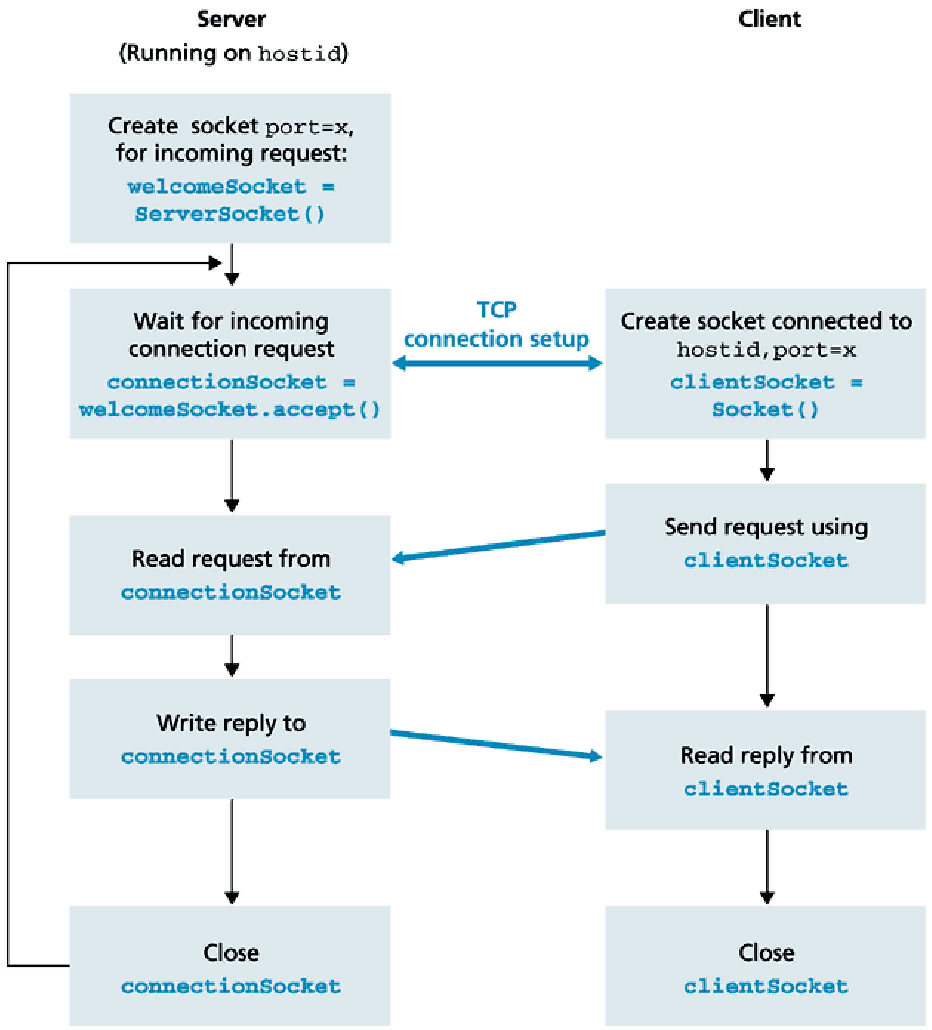


Socket programming in TCP
Streams



- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to an input source, eg, keyboard or socket.
- An output stream is attached to an output source, eg, monitor or socket
- Example client-server application: client reads line from standard input, sends to server, server converts line to uppercase, sends back to client, client prints modified line

Socket programming in TCP
Client-server socket interaction



Socket programming in TCP

TCPClient.java

Eclipse Workspace - Computer Networking/_tcp/TCPClient.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Main.java | main_01.js | main_08.js | TCPClient.java | TCPServer.java | UDPClient.java | UDPServer.java

```java
/*
 * Created on 22-ao�t-2003
 *
 */
package _tcp;

import java.io.*;

/**
 * @author Kurose and Ross page 138
 *
 * Client side of the application
 */
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("localhost",6789);
        DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
        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();
    }
}
```

Strings, streams and sockets declaration

Keyboard > Socket > (Server) > Socket > Screen

Socket programming in TCP

`TCPServer.java`

Eclipse Workspace - Computer Networking/_tcp/TCPServer.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Main.java | main_01.js | main_08.js | TCPClient.java | TCPServer.java | UDPClient.java | UDPServer.java

```java
import java.io.*;

/**
 * @author Kurose and Ross page 141
 *
 * Server side of the application
 */
class TCPServer {
    public static void main(String[] args) throws Exception
    {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            System.out.println("Handling remote client at " +
                connectionSocket.getInetAddress().getHostAddress() + " on port " +
                connectionSocket.getPort() + " from " +
                InetAddress.getLocalHost().getHostAddress() + " on port " +
                connectionSocket.getLocalPort());
            BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream())
            DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();
            System.out.println("Received: " + clientSentence);
            capitalizedSentence = clientSentence.toUpperCase() + "\n";
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

First (welcome) socket

Second (connection) socket

Outline

Principles of application layer protocols

Web browsing and HTTP

Domain name resolution (DNS)

File transfer (FTP)

Electronic Mail: SMTP, POP3, IMAP

Content distribution: Web caching, CDN, P2P

Socket programming with TCP

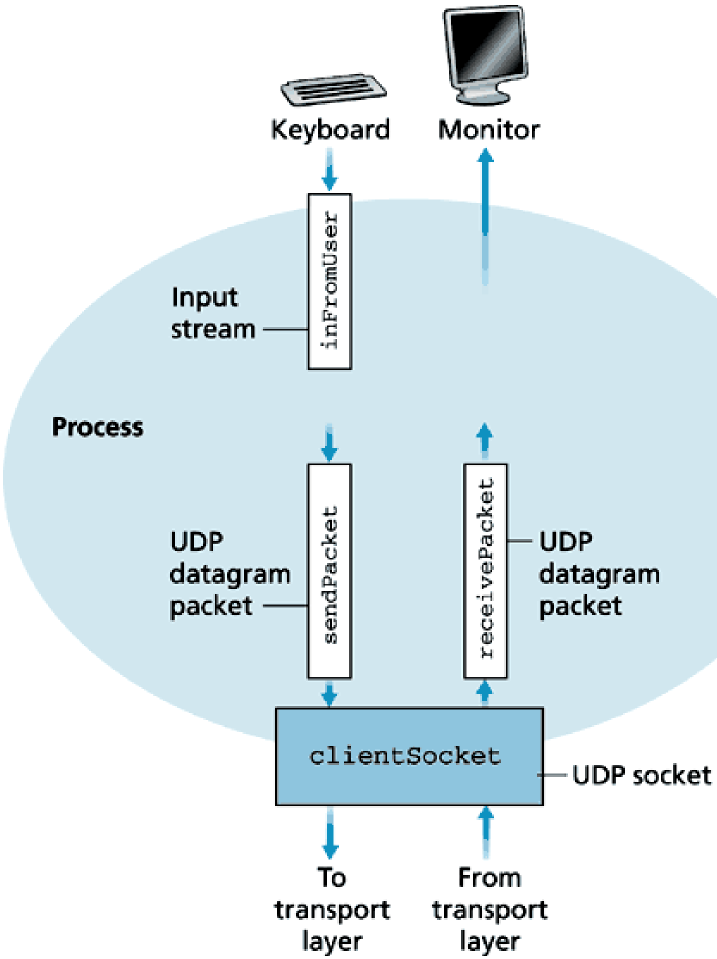
Socket programming with UDP

Building a simple Web server

Socket programming in UDP

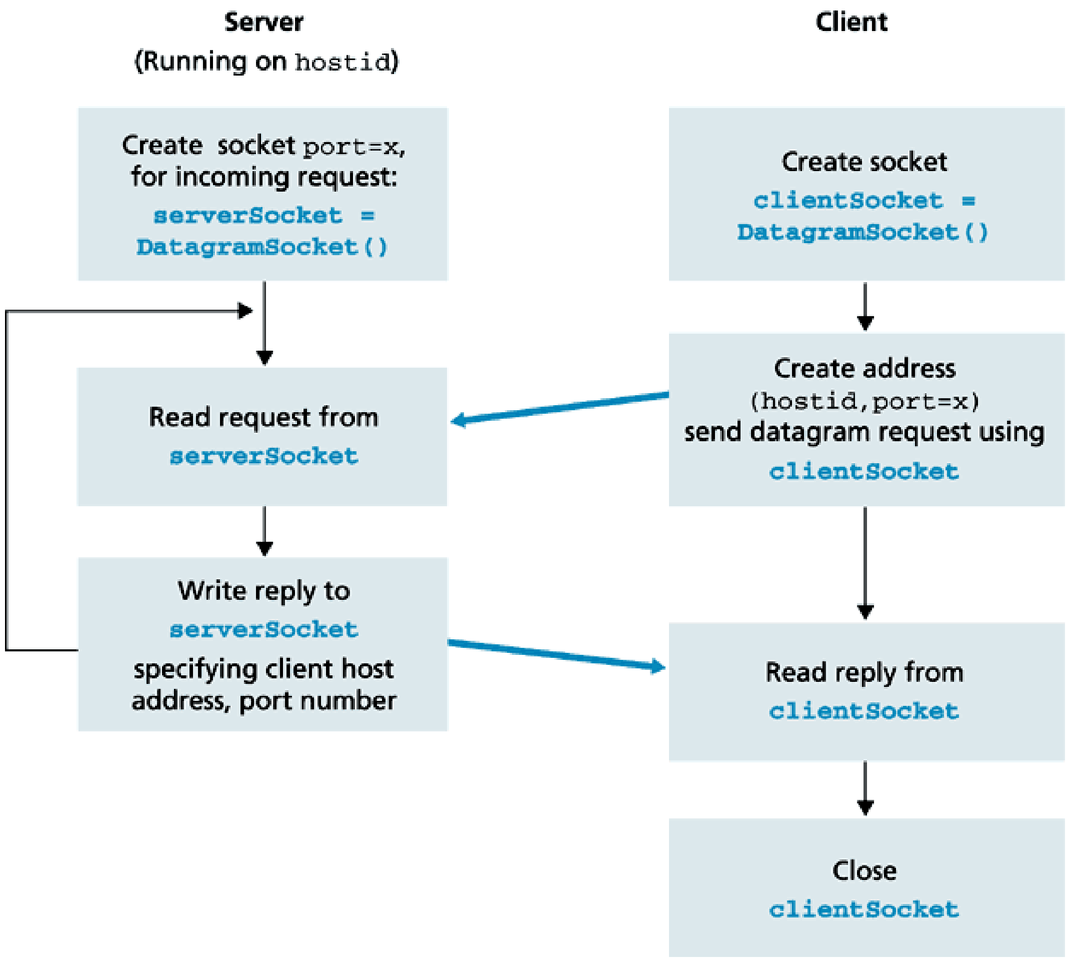
- UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server
- No "connection" between client and server, UDP doesn't have a pipe between client and server
- No handshaking
- Sender explicitly attaches IP address and port of destination to each packet
- Server must extract IP address, port of sender from received packet
- Transmitted data may be received out of order, or lost

Socket programming in UDP
Only one stream



- With UDP, only one stream, connected to the keyboard
- No streams (input or output) connected to the socket
- The socket accepts/delivers individual packets from/to the process
- UDP creates packets including the adress of the server whereas TCP inserts string into stream logically connected to server

Socket programming in UDP
Client-server socket interaction



Socket programming in UDP

UDPClient.java

Eclipse Workspace - Computer Networking/_udp/UDPClient.java - Eclipse Platform

```java
 * Created on 22-ao�t-2003
package _udp;

import java.io.*;

/**
 * @author Kurose and Ross page 144
 *
 * Client side of the application
 */
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("localhost");
        byte[] sendData = new byte[28];
        byte[] receiveData = new byte[28];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
        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();
    }
```

Explicit DNS look-up

Building packet

Receiving packet back

Socket programming in UDP

UDPServer.java

Eclipse Workspace - Computer Networking/_udp/UDPServer.java - Eclipse Platform

```java
package _udp;

import java.io.*;

/**
 * @author Kurose and Ross page 148
 *
 * Server side of the application
 */
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);
            String sentence = new String(receivePacket.getData());
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
```

IP address extraction

Building packet

Outline

Principles of application layer protocols

Web browsing and HTTP

Domain name resolution (DNS)

File transfer (FTP)

Electronic Mail: SMTP, POP3, IMAP

Content distribution: Web caching, CDN, P2P

Socket programming with TCP

Socket programming with UDP

Building a simple Web server

Building a simple Web server

- Handles one HTTP request
- Accepts the request
- Parses header
- Obtains requested file from server’s file system
- Creates HTTP response message: header + file
- Sends response to client
- Self-test
    - IPv4
        - http://127.0.0.1:6789/testWebServer.html
        - http://127.0.0.1:6789/kurose.jpg
    - IPv6
        - http://[::1]:6789/testWebServer.html
        - http://[::1]:6789/kurose.jpg

Building a simple Web server

WebServer.java

Eclipse Workspace - Computer Networking/_http/WebServer.java - Eclipse Platform

```java
    public static void main(String[] args) throws Exception
    {
        String requestMessageLine;
        String fileName;
        ServerSocket listenSocket = new ServerSocket(6789);
        Socket connectionSocket = listenSocket.accept();
        BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
        DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
        requestMessageLine = inFromClient.readLine();
        /* Additional II.10 exercice p. 173 */
        System.out.println(requestMessageLine);
        StringTokenizer tokenizedLine = new StringTokenizer(requestMessageLine);
        if (tokenizedLine.nextToken().equals("GET")){
            fileName = tokenizedLine.nextToken();
            if (fileName.startsWith("/") == true)
            fileName = fileName.substring(1);
            File file = new File(fileName);
            int numOfBytes = (int) file.length();
            FileInputStream inFile = new FileInputStream(fileName);
            byte[] fileInBytes = new byte[numOfBytes];
            inFile.read(fileInBytes);
            outToClient.writeBytes("HTTP/1.0 200 Document Follow\r\n");
            if (fileName.endsWith(".jpg"))
            outToClient.writeBytes("Content-Type: image/jpeg\r\n");
            if (fileName.endsWith(".gif"))
            outToClient.writeBytes("Content-Type: image/gif \r\n");
            outToClient.writeBytes("\r\n");
            outToClient.write(fileInBytes, 0, numOfBytes);
            connectionSocket.close();
        }
        else System.out.println("Bad Request Message");
    }
}
```

Two TCP sockets

Parsing HTTP request

Building HTTP response

Summary

- Study of network applications now complete!
  - Application service requirements: reliability, bandwidth, delay
  - Client-server paradigm
  - Internet transport service model
    - TCP – Connection-oriented, reliable
    - UDP – Connectionless, unreliable
  - Specific protocols
    - Basic: HTTP, DNS, FTP, SMTP, POP, IMAP
    - Secure: HTTPS, DNSSEC, FTPS, SMTPS
  - Content distribution: caches, CDNs, P2P
  - Socket programming
- Learned about protocols
  - Control vs. data messages
  - In-band vs. out-of-band
  - Stateless vs. stateful
  - Reliable vs. unreliable message transfer
  - Centralised vs. decentralised
  - Security
  - “Complexity at network edge”

Review questions

- Can you conceive of an application that requires no data loss and that is also highly time-sensitive?
- Suppose you send an e-mail message whose only data is a Microsoft Excel attachment. What might the header lines (including MIME lines) look like?
- What are the respective roles of the local and of the authoritative name server of a host?
- What is an overlay network in P2P file sharing?
- The UDP server needed only one socket, whereas the TCP server needed two. Why? How many sockets do you need in TCP to support N simultaneous connections?