Computer Networks (part 2)

Rémi Emonet – 2021
Université Jean Monnet – Laboratoire Hubert Curien

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Computer Networks: global overview

1. Introduction to computer networks
2. Networking application layer (HTTP, FTP, DNS, …)
3. Data transfer layer (UDP, TCP, …)
4. Network layer (routing, IP, ICMP, NAT, …)
5. Lower layers, wireless and mobile (Ethernet, ARP, …)
6. Security (SSL, …)

Part 2: Application Layer

- Application layer
  - high level of abstraction
  - « client » of the host to host network
  - interacts with the transport layer
- Part 2
  - application layer and protocols
  - interaction with the transport layer
  - design of protocols
  - programming connected application (socket)

application
transport
network
link
physical

Part 2: Application Layer

- Goal
  - protocols: general principles and existing protocols
  - sockets: programming and services from the transport layer
- Overview
  - Principles of distributed applications
  - HTTP and the web
  - FTP: file transfer
  - Electronic mail
  - DNS: name resolution and more
  - P2P Applications (peer to peer)
  - Network programming: using sockets

application
transport
network
link
physical

Part 2: Application Layer

- Goal
  - protocols: general principles and existing protocols
  - sockets: programming and services from the transport layer
- Overview
  - Principles of distributed applications
  - HTTP and the web
  - FTP: file transfer
  - Electronic mail
  - DNS: name resolution and more
  - P2P Applications (peer to peer)
  - Network programming: using sockets

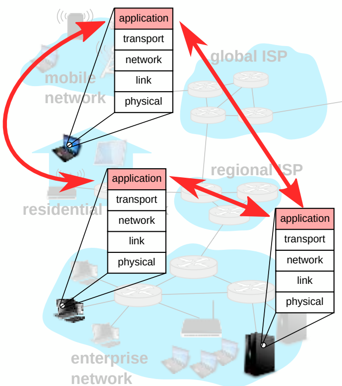
application
transport
network
link
physical

Examples of Distributed Applications

- e-mail
- web
- instant messaging
- multiplayer games
- video and audio streaming
- social networks
- P2P file sharing
- remote connections
- telephone
- real time video-conferencing
- …

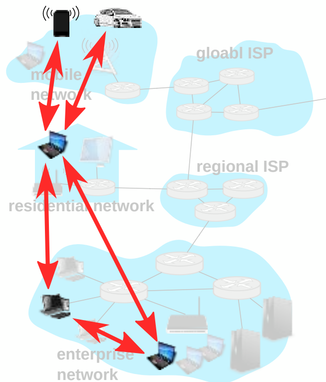
Design of Distributed Applications

- Writing programs
  - executed on different hosts
  - communicating through the network
- Network abstraction and separation
  - the application ignores the numerous details
  - the network core does not execute the application
- Canonical types of architectures
  - client-server
  - P2P (peer to peer)



Architectures for Distributed Applications

- Client-Server
  - Server
    - always on
    - fixed (IP) address
    - server farms
  - Clients
    - intermittent comm.
    - changing address
    - comm. only with the server
- P2P (peer to peer)
  - host = both client and server
  - no central server
  - complicated, dynamic management
  - better scalability

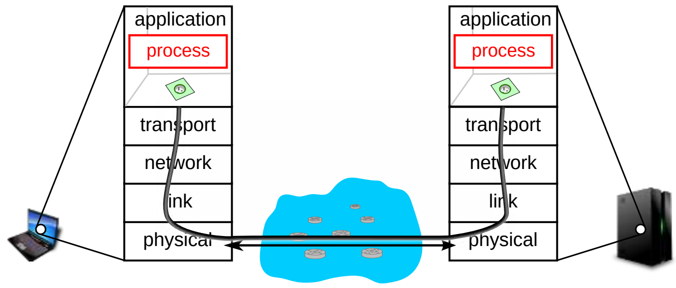


Network Abstraction: interprocess comm.

- Process
  - program running on a host
  - exchanging messages over the network
  - server process
    - waiting to be contacted
  - client process
    - contacting a server
  - P2P: client and server at the same time
- Inter-process communications(IPC)
  - alternative to the network
  - works only on a single host

Network Abstraction: socket

- socket
  - used by a process (application)
  - interface to the rest of the network stack
  - interface to another (remote) process



Network Abstraction: process identification

- Address of an host
  - IP address: 32 bits
  - example: 78.109.84.114
  - but: there could be multiple processes on a host
- Process identifier
  - address of the host
  - port number
  - example: 80
  - ⇒ 78.109.84.114:80

Protocols from the Application Layer

- Types of messages
  - initialization, request, response, …
- Syntax and format of messages
  - structure of messages
  - fields and their size
  - encoding, separators, …
- Semantic of messages
  - meaning of the different message types
  - interpretation of the fields
- Processing rules
  - how to answer the message?
  - when to answer?
- Open protocols (HTTP, ...) vs proprietary protocols (Skype)

Services from the Transport Layer (from the application point of view)

- Transport integrity
  - guaranteed reception of all bits sent
- Latency (delay)
  - reception of messages after a small time interval
  - guarantee on a maximum delay
- Throughput (bandwidth)
  - guarantee on the average data transfer rate
  - guarantee on a (minimal) constant rate
- Security
  - encryption, privacy protection
  - integrity (non-corruption)

application
transport
network
link
physical

How sensitive to these aspects are the following application?

- Aspects: integrity, latency, throughput, security
- Applications
  - file transfer, e-mail, web browsing,
  - real-time audio/video, audio/video streaming,
  - multiplayer games, instant messaging

Options for Transport with Internet

- Transport with TCP
  - connection oriented (stream)
    - handshake at init.
  - transfer integrity
    - from socket to socket
  - flow control
    - prevent "spam"
  - congestion control
    - adaptation to network load
  - missing services
    - guaranteed latency
    - guaranteed rate
    - security
- UDP
  - packet oriented (datagram)
  - transport not guaranteed
  - missing services
    - transfer integrity
    - flow control
    - congestion control
    - guaranteed latency
    - guaranteed rate
    - security
- Question: so, why UDP?

Internet Apps and Transfer Protocols

Application	Application Protocols	Transfer Protocols
e-mail	SMTP (RFC2821)	TCP
Web Browsing	HTTP (RFC2616)	TCP
remote access (terminal)	Telnet (RFC854)	TCP
remote access (terminal)	SSH (RFC4251)	TCP
file transfer	FTP (RFC959)	TCP
Streaming	HTTP, RTP (RFC1889)	TCP, UDP
Voice over IP	SIP, RTP, prop.	TCP or UDP

Absence of Security in TCP and UDP

- TCP and UDP do not propose encryption
  - data sent "as is", including passwords etc
  - possibility for any router to read these
- TLS (Transport Layer Security)
  - evolution/renaming of SSL (Secure Sockets Layer)
  - systematic encryption before sending through TCP
  - authentication/identification of hosts (with "certificates")
- Notes about TLS
  - TLS is an application layer protocol
  - TLS is just a software library
  - the `ssh` command allows users to create secured tunnels

Part 2: Application Layer

- Goal
  - protocols: general principles and existing protocols
  - sockets: programming and services from the transport layer
- Overview
  - Principles of distributed applications
  - HTTP and the web
  - FTP: file transfer
  - Electronic mail
  - DNS: name resolution and more
  - P2P Applications (peer to peer)
  - Network programming: using sockets

application
transport
network
link
physical

A Web Page in a Browser

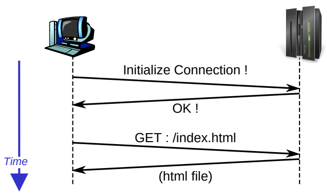
- Concepts
  - web page = set of objects
  - type of objects
    - HTML source, CSS, javascript
    - image: PNG, JPEG, …
    - multimedia file: mp3, avi, webm, …
    - Java applet, flash, …
- URL, uniform resource locator
  - example: `http://wikipedia.fr/Fichiers/LogoWikipedia.png`
    - `http` → protocol
    - `wikipedia.fr` → host
    - `/Fichiers/LogoWikipedia.png` → path
  - « web address » (when `http://`)

HTTP: Introduction

- HTTP: hypertext transfer protocol
- Application layer protocol
- Client-server architecture
  - client process
    - web browser
    - queries, receives and displays web objects
  - server process
    - web server
    - send objects in response to requests
- Use of TCP
  - listen on port 80 (by default)
- Stateless protocol (without state)
  - the HTTP server (the protocol) does not keep information on the client
  - independent requests

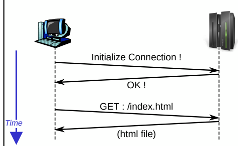
HTTP Connections

- Non-persistent connection
  - a TCP connection per request
  - opening and closing at each request
- Persistent connection
  - opening a TCP connection
  - sending multiple HTTP requests/responses on this connection
  - need to keep the connection open
- Example of HTTP session over TCP



How long does it take to display a web page from a server in Mexico?

- Round-trip time Saint-Étienne ⇆ Mexico: 180ms
- The HTML page references
  - 5 images, 3 javascript files
  - 2 stylesheets (CSS)
- Objects are supposed to be small (~ 1 kB each)



What solution can you imagine to reduce the delay to display the web page from Mexico?

Example HTTP Request

- HTTP defines requests and responses
- Example request

```
GET /index.php HTTP/1.1
Host: wikipedia.fr
User-Agent: Mozilla/5.0 (X11; Ubuntu; …) Gecko/20100101 Firefox/26.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

- Request line: method (GET, POST, HEAD, …), path, version
- Header lines
  - *«name»* : *«value»*
  - content negotiation
  - connection persistence
  - …

Format of an HTTP (pseudo-grammar)

- *«request»* → *«req-line»* *«header-line»** *«newline»* *«body»*
- *«req-line»* → *«method»* *«␣»* *«path»* *«␣»* *«http-version»* *«newline»*
- *«method»* → GET | POST | HEAD | PUT | DELETE
- *«header-line»* → *«name»* : *«value»*
- *«␣»* → (a space)
- *«newline»* → \r \n

Sending More Data in an HTTP Request

- Typical use
  - data to send from the client to the server
  - values from a form (filled by the user)
  - file to upload
- Use of the POST method
  - send data in the *«body»*
- Encoding data in the URL (with GET)
  - for short data only (forms only)
  - adding new elements in the URL
  - `http://en.wikipedia.org/w/index.php?search=kitten&title=Special%3ASearch`
    - `search = kitten`
    - `title = Special:Search`

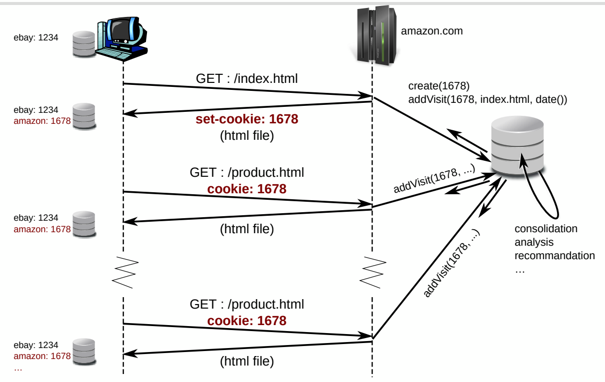
Format of an HTTP Response

- Request (reminder)
  - *«request»* → *«req-line»* *«header-line»** *«newline»* *«body»*
  - *«req-line»* → *«method»* *«␣»* *«path»* *«␣»* *«http-version»* *«newline»*
- *«response»* → *«resp-line»* *«header-line»** *«newline»* *«body»*
- *«resp-line»* → *«http-version»* *«␣»* *«code»* *«␣»* *«message»* *«newline»*
- *«code»* → 200 | 404 | …
- *«message»* → OK | Not Found | …
- HTTP status codes
  - http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
  - 200 OK: object is in the *«body»*
  - 301 Moved Permanently: object changed address
    - a header "Location:" gives the new location
  - 400 Bad Request: the server does not understand the request
  - 404 Not Found: the object is not found on the server
  - 505 HTTP Version Not Supported

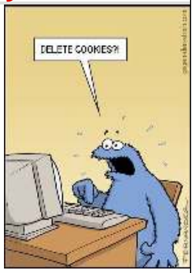
HTTP Cookies

- Store a state
  - inside the client
  - on the server initiative
  - sent back to the server at every request (by the client)
- Usage
  - save preferences (language, skin, …)
  - user tracking
    - the server generates a unique ID
    - the server tracks all page hits from this ID
    - the server profiles the client
    - ⇒ product recommendation, session persistence, …

HTTP Cookies

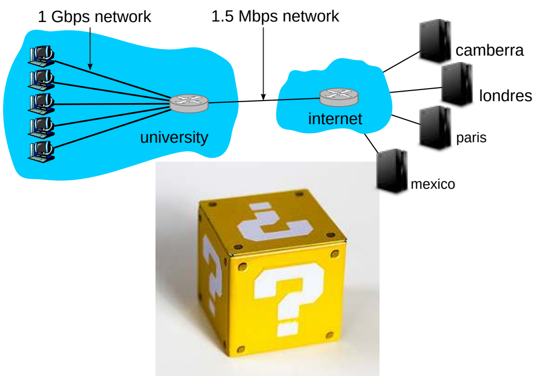


HTTP Cookies: summary

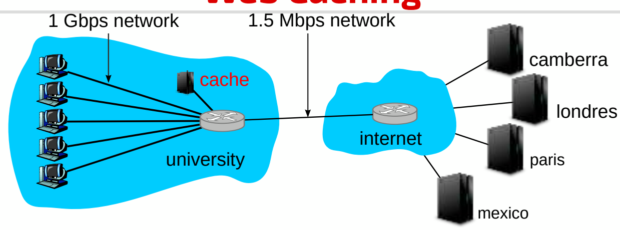


- Uses of cookies
  - user preferences
  - authentication
  - shopping basket (e-commerce)
  - user session
- HTTP is stateless
  - the “cookies protocol” is at the application level
  - clients and server(s) maintains a state together (cookie)
  - the HTTP message contains the state
- Privacy issues?
  - the user is tracked without nowing
  - all the browsing history
  - user profiles get sold
  - price policy
  - information mixing between very different sites
  - widgets (buttons like +1, like, analytics, etc) on so many sites

What problem/solution do you see in the network below?

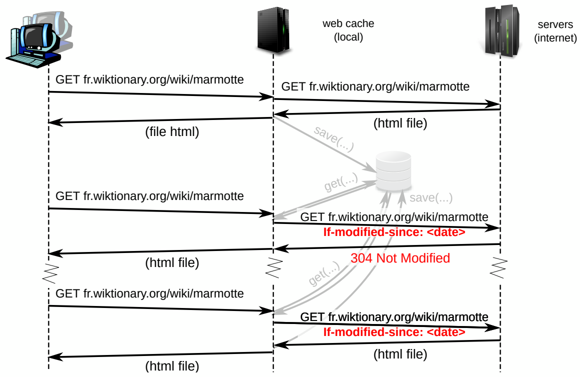


Web Caching



- Local cache Server
  - on the local network
  - “intercepts” web communications to internet
  - saves web objects webs coming from outside
- Advantages
  - reduced data transfers with the outside
  - reduced costs
  - average latency also reduced

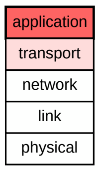
Web Caching: protocol, conditional GET



- HTTP caching protocol
  - header: `If-not-modified-since`
  - response: `304 Not Modified` or `200 OK`

Part 2: Application Layer

- Goal
  - protocols: general principles and existing protocols
  - sockets: programming and services from the transport layer
- Overview
  - Principles of distributed applications
  - HTTP and the web
  - FTP: file transfer
  - Electronic mail
  - DNS: name resolution and more
  - P2P Applications (peer to peer)
  - Network programming: using sockets



FTP: RFC959

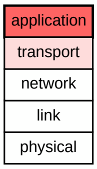
- Goal
  - transferring files to/from a server
  - from hard drive to hard drive
  - bi-directional
- Principles
  - 2 networking ports
    - port 21: control connection
    - port 20: intermittent data connection
    - → out-of-band commands (separate from the data)
  - the server is stateful, it stores for each client
    - the current folder
    - the logical link between data and control connections
  - 2 modes for *data* connections
    - passive (PASV), the client connects to the server
    - active (PORT), the server connects to the client

FTP: Examples of requests and responses

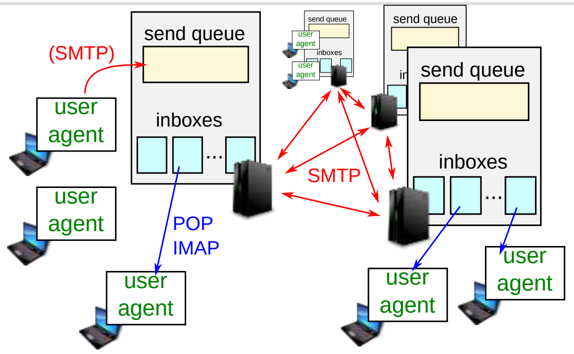
- Requests
  - `USER bob`
  - `PASS secret`
  - `LIST`
  - `RETR file.txt`
  - `STOR new.txt`
  - `PASV`
  - `QUIT`
  - …
- Responses
  - `331 Username OK, password required`
  - `125 Data connection already open; transfer starting`
  - `425 Can't open data connection`
  - `452 Error writing file`
  - `221 Goodbye`
  - …

Part 2: Application Layer

- Goal
  - protocols: general principles and existing protocols
  - sockets: programming and services from the transport layer
- Overview
  - Principles of distributed applications
  - HTTP and the web
  - FTP: file transfer
  - Electronic mail
  - DNS: name resolution and more
  - P2P Applications (peer to peer)
  - Network programming: using sockets

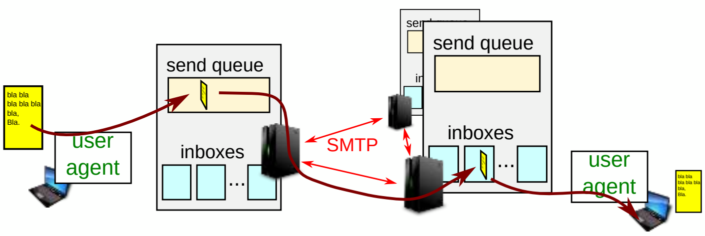


Architecture for Electronic Mails



- SMTP Protocol: sending emails
- POP and IMAP Protocols: reading emails
- User-Agent: mail client, webmail, etc

Sending Electronic Mails



SMTP Protocol: RFC2821

- Goal
  - sending emails
  - asynchronous messages
- Principles and limitations
  - use of TCP
  - use of port 25 (by default)
  - request/response protocol
  - direct transfer to the destination server
  - a server responsible for each domain
  - message
    - sender
    - recipient
    - content
    - headers
  - 7-bits ASCII

Example of SMTP Session

- C -> S: client (crepes.fr) to server (hamburger.edu)

```
S->C: 220 hamburger.edu
C->S: HELO crepes.fr
S->C: 250  Hello crepes.fr, pleased to meet you

C->S: MAIL FROM: <alice@crepes.fr>
S->C: 250 alice@crepes.fr... Sender ok
C->S: RCPT TO: <bob@hamburger.edu>
S->C: 250 bob@hamburger.edu ... Recipient ok
C->S: DATA
S->C: 354 Enter mail, end with "." on a line by itself
C->S: How do you do? Well?
C->S: Networking is amazing!
C->S: .
S->C: 250 Message accepted for delivery

C->S: QUIT
S->C: 221 hamburger.edu closing connection
```

- "HELO", "MAIL FROM", "RCPT TO", "DATA"
- end with a line containing only with "."

SMTP: format for « DATA »

- *«message»* → *«header»* *«newline»* *«body»*
- *«newline»* → \r \n
- *«header»* : RFC 822
  - ex : `To: ...`
  - ex : `From: ...`
  - ex : `Subject: ...`
  - different from MAIL FROM, RCPT TP, etc
- *«body»* : the message in ASCII

Accessing Electronic Mails

- SMTP: sending
- POP3 Post Office Protocol, RFC 1939
  - access to the inbox
  - downloads and deletes, or, downloads and keep
  - minimal protocol (list, get), plain-text protocol
- IMAP Internet Mail Access Protocol, RFC 1730
  - conservation of messages on the server
  - creation of folders (and sub-folders)
  - organization of messages: tags, etc
  - plain-text protocol
- Webmail over HTTP, diverse range of features
- Common properties
  - require authentication
  - versions with secured transport (SMTPS, IMAPS, …)

Part 2: Application Layer

- Goal
  - protocols: general principles and existing protocols
  - sockets: programming and services from the transport layer
- Overview
  - Principles of distributed applications
  - HTTP and the web
  - FTP: file transfer
  - Electronic mail
  - DNS: name resolution and more
  - P2P Applications (peer to peer)
  - Network programming: using sockets

application
transport
network
link
physical

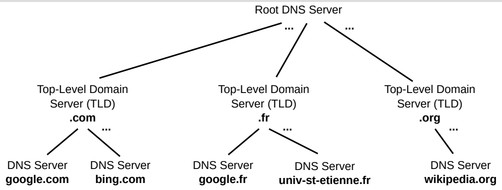
DNS: Global View

- Goal:
  wikipedia.fr ⇄ 78.109.84.114
- Us, humans
  - first name, last name
  - passport number
  - social security number
  - driving license number
  - …
- Internet hosts
  - IP address
    - for the network layer
    - ex: 78.109.84.114
  - "name"
    - for the human users
    - ex: wikipedia.fr
- DNS
  - Domain Name System
    - name resolution
    - IP(s) ⇄ name(s)
  - acts as a distributed database
    - many servers
    - hierarchical organization
  - at the core of internet
    - very useful
    - implemented in the "application" layer

DNS: Services and Distributed Architecture

- DNS Services
  - name resolution (name → IP)
  - name aliases (pseudonym)
    - canonical name
    - list of aliases
  - mail server alias
    - ex: message for `bob@blabla.com` sent to `mymailserv.blabla.com`
  - load balancing
    - `www.google.fr` → `74.125.132.99, 74.125.132.147, 74.125.132.104, …`
- Distributed architecture with caching
  - multiple distributed servers
  - fault resistance
  - maintainability
  - scalability (traffic volume)
  - reduces latency

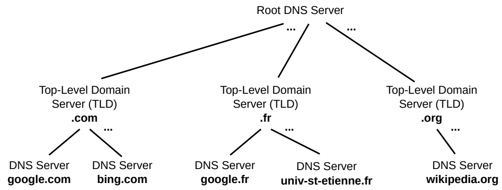
DNS: Hierarchical Organization



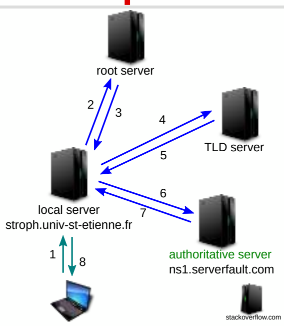
- Hierarchy
  - Root DNS servers (global)
  - TLD (Top-Level Domain) servers (ex of TLD: `.fr`)
  - Authoritative DNS servers
    - gives authoritative answers for a given domain
    - server in a company, university, ISP, ...
- List of TLDs : https://www.iana.org/domains/root/db/
  - each TLD has its own rules

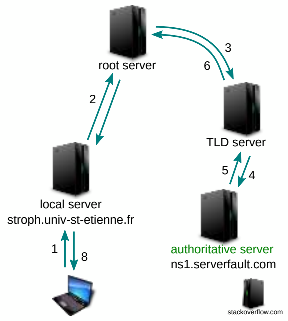
DNS: Resolution Principle

- Resolving `fr.wikipedia.org`
  - asks a root server
    what is an address for a TLD server `.org`
  - ask the TLD server
    what is the address of an authoritative server for `wikipedia.org`
  - ask the authoritative server
    what is the address for `fr.wikipedia.org`



DNS Request: recursive vs/and iterative





- Typical set of requests
  - "recursive" request to a local server
  - "iterative" requests by the local server
- Requests in all-recursive (seldom used)

DNS and Caching

- Caching done by servers
  - as soon as a server discovers a DNS record (e.g., name ⇄ IP)
  - this record is cached
  - ex: addresses of TLD servers cached by local servers
  - fixed caching duration (set by the authority): TTL (Time To Live)
  - update delay, possibility of "errors", due to caching
- Protocol to notify of updates
  - to limit this update delay
  - RFC 2136

4 Different Types DNS Records

- DNS = database of DNS Resource Record (RR)
  - format of a RR: (type, name, value, ttl)
  - TTL: Time To Live
- Messages in the DNS protocol: binary format
- Port UDP 53 (+ TCP in case of need)
- type = A
  - name: host name
  - value: IP address
- type = NS
  - name: domain name (e.g., wikipedia.org)
  - value: name of the authoritative server for this domain
- type = MX
  - name: email domain name
  - value: name of the mail server
- type = CNAME
  - name: name alias
  - value: canonical name

Adding DNS Records: say hi to the world

- Example
  - creation of the (big) SuperProject project
  - wants to have a domain name `superprojet.fr`
- Buying the domain name to a domain name registrar
  - example of registrar: gandi.net, ovh.com
  - creation of two DNS records
    - `(superprojet.fr, ns1.superprojet.fr, NS)`
    - `(ns1.superprojet.fr, 222.222.222.1 , A)`
- On our DNS server (ns1.superprojet.fr, IP 222.222.222.1)
  - creation of other DNS records
    - `(www.superprojet.fr, 222.222.222.33, A)`
    - `(superprojet.fr, mail.superprojet.fr, MX)`
    - ...
- We can also put everything on the registrar server (and not have our DNS server)

Attack on DNS Infrastructure

- DDoS, Distributed denial of service
  - flooding root servers
    - traffic filtering
    - caching in local DNS
    - ⇒ failure up to now
  - flooding TLD servers
    - more chances of success
- Man-in-the-middle: interception of requests
- DNS cache poisoning, DNS spoofing
  - sending fake records to servers
  - caching of fake records in the (clean) servers
- Using DNS servers for DDoS
  - using DNS servers
  - to create DDoS

Part 2: Application Layer

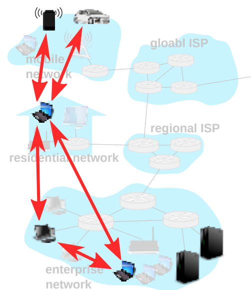
- Goal
  - protocols: general principles and existing protocols
  - sockets: programming and services from the transport layer
- Overview
  - Principles of distributed applications
  - HTTP and the web
  - FTP: file transfer
  - Electronic mail
  - DNS: name resolution and more
  - P2P Applications (peer to peer)
  - Network programming: using sockets

application
transport
network
link
physical

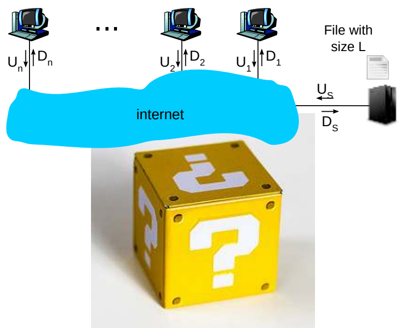
Peer-to-Peer (P2P) and the BitTorrent Example

P2P Architectures

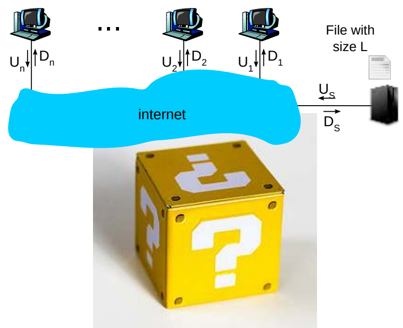
- Pure P2P
  - No server (that need to be always on)
  - Any host
    - peer to peer communications
    - connection, disconnection
    - changing IP
- In practice
  - sometimes, some support/organization servers
  - or some peers that are more important



How long does it take to distribute the file to all clients using a client/server architecture?



How long does it take to distribute the file to all clients using a peer-to-peer architecture?



BitTorrent: Principles and Architecture

- File (or group of files) split in chunks of 256kb (16kB)
- Torrent
  - a group of machines (processes)
  - exchanging the chunks of a file
- Tracker
  - server maintaining a list of the torrent participants
  - useful for new peers
  - relatively low processing charge, easily replaceable
- What a new peer does
  - asks the tracker for a (sub)list of peers
  - connects to each peer asking chunks
  - receives chunks
  - then, can simultaneously send/receive chunks
  - reconsiders connections
  - when all chunks are obtained → stay to upload

BitTorrent: Distributed Optimization

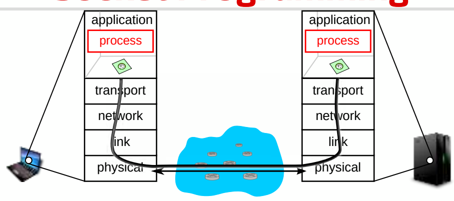
- Selecting the chunk to download
  - periodically obtain a list of chunks from the neighbors
  - query missing chunks, starting with the most rare
- Choosing which peer to send chunks
  - send to the 4 peers that upload the fastest (to me)
    - ignore other requests
    - re-evaluate every 10 seconds
  - selecting a peer randomly
    - every 30 seconds
    - sending to a new peer
    - goal: test the speed of new peers
  - if not enough peers
    - ask the tracker again
- Global result
  - replication of the rarest chunks
  - optimized communication with high-throughput peers

Part 2: Application Layer

- Goal
  - protocols: general principles and existing protocols
  - sockets: programming and services from the transport layer
- Overview
  - Principles of distributed applications
  - HTTP and the web
  - FTP: file transfer
  - Electronic mail
  - DNS: name resolution and more
  - P2P Applications (peer to peer)
  - Network programming: using sockets

application
transport
network
link
physical

Socket Programming



- socket
  - interface to the “transport” layer
  - interface to a remote process
  - abstraction of everything else
- Objective
  - programming distributed applications
  - with a minimal amount of knowledge on the network
- 2 types of sockets: UDP, TCP

UDP Programming

- No connection between clients and servers
  - no initialization
  - directly sending packets (to IP:port)
  - address of the sender (IP:port) in the paquet
- Unreliable transport
  - possibility of loss
  - possibility of re-ordering
- For the applications
  - need to know the IP:port of the destination
  - ex: a known server with a know port
  - Java `DatagramPacket` class
    - UDP “packet” (segment)
  - Java `DatagramSocket` class
    - interface to exchange packets
    - bi-directional: send/receive

TCP Programming

- Reliable bi-directional connection between two processes
  - connection IP:port ⇄ IP:port
  - connection seen as a two streams
- Connection establishment
  - a “server socket” listens for new connection (a port)
  - a client contacts this server (from a local port)
  - a new socket is created on both sides
- For the applications
  - the server can wait for connections
  - the client need to know the server's IP:port
  - connection = two stream objects (each direction)
  - Java `Socket` class
    - a TCP communication to a remote process
    - bi-directional: send/receive
    - getOutputStream / getInputStream
  - Java `ServerSocket` class
    - to accept TCP connections from clients
    - *factory* (design pattern) of `Socket`

Threads (and Java)

- Multiple execution threads in a program
- Code is (virtually or really) executed in “parallel”
- Problems of concurrent accesses
  - shared variables (memory)
  - modification of a container while iterating over it
- Solutions
  - use only one thread
  - use mutexes
  - used *synchronized* methods
  - synchronization using robust concurrent-access message queues (`BlockingQueue`)

Handling Concurrent Streams (network…)

- Solution 1 (not recommended)
  - one "accepting" thread
  - one thread per client
  - synchronization via `synchronized` or ad'hoc things
- Solution 2
  - one "accepting" thread
  - one thread per client
  - one (or a few main) thread(s) for the logic
  - communication between thread via message queues
- Solution 3
  - using `select` (or `Selector`)
  - single thread approach

Key Points

save