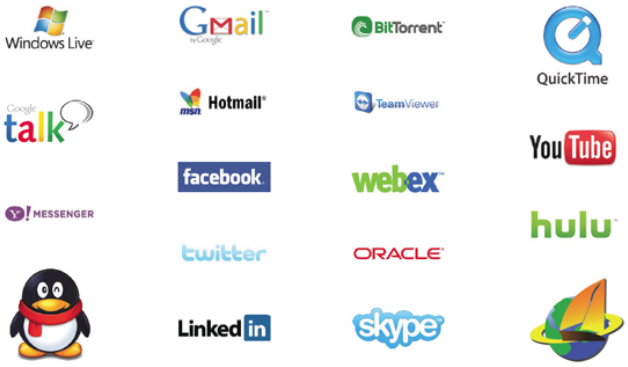
UE PRIP
Principes des réseaux informatiques par la pratique
Application Layer

Isabel Amigo

2022

Notes:



Notes:

Today's objective

- overview of the application layer
- understand the basics of app layer protocols
- app layer protocol example: HTTP (in this course)
- focus on the domain names system (in Lab DNS)

Approach:

- overview of the application layer, course and your questions (please!)
- analyze more in depth the functioning of the DNS (Lab session)

Notes:

1. Network applications

Notes:

Some network applications

you name them!

Notes:

Creating a network app

write programs that:

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

no need to write software for network-core devices

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

Notes:

Application architectures

possible structure of applications:

- client-server
- peer-to-peer (P2P)

Notes:

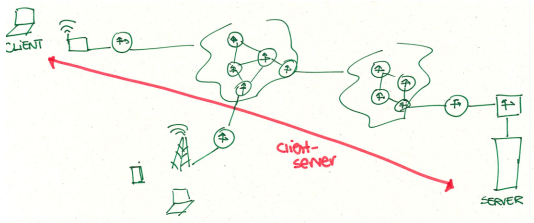
Client-server architecture

server:

- always-on host
- permanent IP address
- data centers for scaling

clients:

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



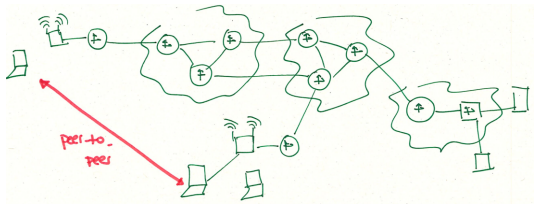
Notes:

P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers

  ! self scalability – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses

  ! complex management

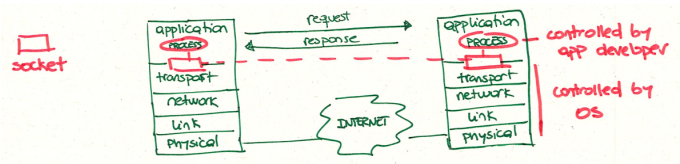


Notes:

Some vocabulary: process

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging *messages* (cf 1st course)



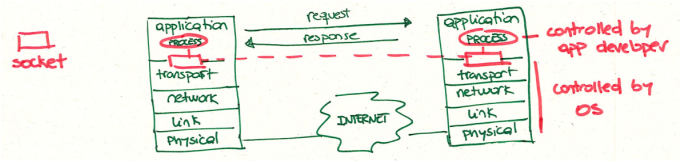
Notes:

Some vocabulary: sockets

process sends/receives messages to/from its socket

- socket analogous to door
  - sending process pushes message out the door
  - sending process relies on transport infrastructure on other side of the door to deliver message to socket at receiving process

More about sockets in Transport layer course!



Notes:

Addressing processes

Q Consider an incoming message to the host, how to know to which process is it destined?

Q Given that host is identified by an IP address, does an IP address suffice to identify a process?

! No, several process running on same host!

ports In the TCP/IP model, 16-bit numbers that are used along with IP addresses for identifying a process

Examples of well-known ports: HTTP server 80, mail server 25, DNS server 53

Notes:

2. Application protocols

Notes:

Application protocols are similar to a human conversation

```
Alice : Hello

Bob : Hello

Alice : What time is it ?

Bob : 11:55

Alice : Thank you

Bob : You're welcome
```

! Works if both speak same language!

Notes:

Application protocols

A set of rules that specify:

- types of messages exchanged
  e.g. request, response
- message syntax
  what fields in messages and how fields are delineated
- message semantics
  meaning of information in fields
- message ordering
  rules for when and how processes send and respond to messages

Open protocols : defined by IETF, allow interoperability e.g. : HTTP, SMTP
Proprietary protocols: e.g. skype (no interoperability possible!)

Notes:

Two types of messages

App protocols can be defined using either:

- Strings or lines of characters
- Bits

! But transport layer allows to transfer bits, not Strings ⇒ need of common representation e.g. usage of ASCII for characters

E.g. of characters defined on the ASCII table
A : 1000011b
0 : 0110000b
carriage return (CR) : 0001101b
line feed (LF) : 0001010b

Notes:

An example of an open application layer protocol: HTTP

- HTTP: Hypertext transfer protocol
- Open protocol standardized by IETF

Q Do you use this protocol? What for?

Versions :

`HTTP`	$\sim$ 1989
`HTTP/1.1`	RFC 2068 (1997), RFC 2616 (1999), RFC 7230 (2014)
`HTTP/2`	RFC 7540 (2015), adds encryption (among other diffs)
`HTTP/3`	draft, implemented in some browsers (e.g. Chrome sep.19)

! More on RFCs soon!

Notes:

(An RFC example)

`https://tools.ietf.org/html/rfc7230`

Notes:

Some other application layer protocols

- DNS
- SMTP
- POP3
- IMAP
- SNMP
- FTP
- XMPP

... among many others

Q Is it really OK to consider the DNS as an application-layer protocol? to be discussed further after the DNS lab!

Notes:

3. Services from lower layers

Notes:

Application layer can rely on services from the transport layer

Which services are needed form an app point of view?

Data integrity
- 100% needed for e.g. file transfer
- Some loss tolerated for e.g. voice

Delay
- Low delay important for some apps (e.g. online gaming)
- Delay-tolerant apps also exist (e.g. file transfer)

Throughput
- Apps needing large bandwidth (e.g. video)
- "elastic" apps which use whatever "remaining" capacity

Q Can you think of any other?

Notes:

Internet transport protocols services at a glance

TCP

- reliable transport between sending and receiving process: no losses, not disordered messages
- flow control: not overwhelming receiver
- congestion control: adapt sending rate when network overloaded
- connection-oriented: setup required between client and server processes
- no guarantees on delay, throughput, security (though security extension exists)

UDP

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

Q Why using UDP then?
Q If an app is relying on UDP but needs some of the not provided guarantees, which solution?

Notes:

4. Summary

Notes:

Summary

Be sure you understand the following aspects:

- Application layer
- Application protocol
- Client-server architecture
- P2P architecture
- Services from the transport layer

Notes:

Appendix: An example of an Internet application protocol: HTTP

Notes:

HTTP overview I

- HTTP application protocol for distributed, collaborative, hypermedia information systems
- foundation of data communication for the World Wide Web
- request–response protocol in the client–server architecture

Q example of client and server?

- client submits HTTP request to server
- server:
  -provides resources (HTML files or other content) or
  -performs other functions on behalf of client
  -returns a response message to client.
- response contains:
  -completion status information about the request
  -requested content (if OK)

Notes:

Recall: web pages

- web page consists of objects
- object can be HTML file, JPEG image, Java applet, audio file,...
- web page consists of base HTML-file which includes several referenced objects
- each object is addressable by a *URL*
  e.g. http://www.example.com/index.html, which indicates a protocol (http), a hostname (www.example.com), and a file name (index.html)

Uniform Resource Locator (URL) references a web resource by specifying its location on a computer network and a mechanism for retrieving it

Notes:

HTTP overview II

Uses TCP:

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

Notes:

HTTP non-persistent and persistent

Non-persistent

- at most one object sent over TCP connection connection then closed
- downloading multiple objects required multiple connections

Persistent

- multiple objects can be sent over single TCP connection

Q Can you think of advantages and disadvantages of each one?

Notes:

HTTP Request message

Text protocol, containing ASCII (human-readable format) characters, formated as follows:

- a request line (e.g., GET /images/logo.png HTTP/1.1)
- request header fields (e.g., Accept-Language: en).
- an empty line
- an optional message body

Notes:

HTTP Request message example

Notes:

```
GET / HTTP/1.1\r\n
Host: asdf.com\r\n
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3\r\n
Accept-Encoding: gzip, deflate\r\n
DNT: 1\r\n
Connection: keep-alive\r\n
Cookie: _ga=GA1.2.156074956.1559831863\r\n
Upgrade-Insecure-Requests: 1\r\n
\r\n
```


Response message

Notes:

- status line: status code and reason (e.g., HTTP/1.1 200 OK)
- response header fields (e.g., Content-Type: text/html)
- an empty line
- an optional message body
- status line and header fields must end with $< CR >< LF >$

Response message example

Notes:

```
HTTP/1.1 200 OK\r\n
Date: Thu, 23 Jan 2020 13:11:43 GMT\r\n
Server: Apache\r\n
Upgrade: h2\r\n
Connection: Upgrade, Keep-Alive\r\n
Last-Modified: Fri, 25 May 2018 14:29:40 GMT\r\n
ETag: "53f-56d089932ee03-gzip"\r\n
Accept-Ranges: bytes\r\n
Vary: Accept-Encoding\r\n
Content-Encoding: gzip\r\n
Content-Length: 683\r\n
Keep-Alive: timeout=2, max=100\r\n
Content-Type: text/html\r\n
\r\n
```

body follows

More on HTTP

Other important aspects we won't have time to discuss in class

- HTTP is a stateless protocol (server maintains no information about past client requests)
- cookies have been created to have some state
  - small piece of data sent from a website and stored on the user's computer by the user's web browser
  - designed for websites to remember stateful information
- web cashing or proxy
- security
  - **Q** Any security concern in the example messages we have seen?
  - alternatives HTTPS, HTTP/2

Notes:

Acknowledgements

The contents of these slides are partially taken from Computer Networking a Top Down approach, J. KUROSE and K. ROSS and K. KUROSE's networking course `http://www-net.cs.umass.edu/cs453_fall_2013/` and from e-book Computer Networking : Principles, Protocols and Practice, third edition `http://beta.computer-networking.info/syllabus/default/index.html`

Notes: