



## Network Applications and the Web

Lecture given by Emmanuel Lochin

ISAE-SUPAERO

Original slides from A. Carzaniga (Univ. Lugano)  
Extended/modified by E. Lochin (ISAE-SUPAERO) with author permission

Textbook Chap. #2 Sections 2.1 to 2.2.3

## Outline

- General concepts for network applications
- Client/server architecture
- The world-wide web
- Basics of the HTTP protocol

Lecture given by Emmanuel Lochin

Network Applications and the Web

ISAE-SUPAERO 1 / 26

## Examples of Network Applications

- The world-wide web
- Electronic mail
- Instant messaging
- Peer-to-peer file sharing
- Video streaming
- Multi-user networked games
- ...
- Remote login
- ...
- Remote on-line banking
- Network telephony
- ...

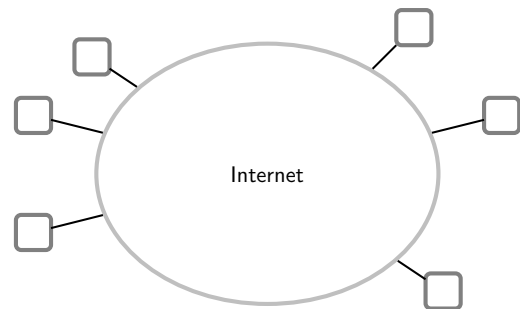
Lecture given by Emmanuel Lochin

Network Applications and the Web

ISAE-SUPAERO 2 / 26

## End System Applications

Internet applications are **end system** applications



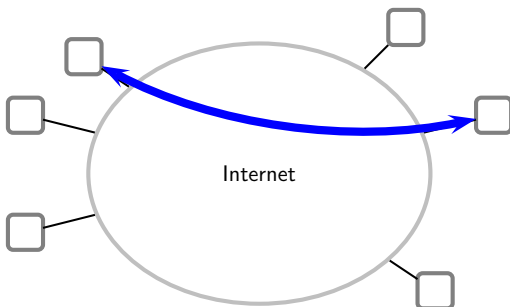
Lecture given by Emmanuel Lochin

Network Applications and the Web

ISAE-SUPAERO 3 / 26

## End System Applications

Internet applications are **end system** applications



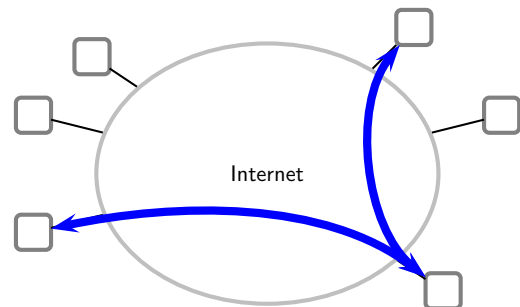
Lecture given by Emmanuel Lochin

Network Applications and the Web

ISAE-SUPAERO 4 / 26

## End System Applications

Internet applications are **end system** applications



Lecture given by Emmanuel Lochin

Network Applications and the Web

ISAE-SUPAERO 4 / 26

## Processes

- A **process** is an execution of a program
- A single **sequential** program
  - ▶ i.e., a single thread
- Processes may exchange messages
  - ▶ obviously, received messages can be considered as input to a process (program)
- Different processes may be running on different end systems
  - ▶ possibly on different computers
  - ▶ running different operating systems
  - ▶ a process must be able to **address** another specific process

## Example

```
while(browsing) {  
    url = read_url(keyboard);  
    page = get_web_page(url);  
    display_web_page(page);  
}
```

```
while(serving_pages) {  
    page_name = read_web_request(network);  
    page = read_file(page_name, disk);  
    write_page(page, network);  
}
```

Lecture given by Emmanuel Lochin

Network Applications and the Web

ISAE-SUPAERO 5 / 26

Lecture given by Emmanuel Lochin

Network Applications and the Web

ISAE-SUPAERO 6 / 26

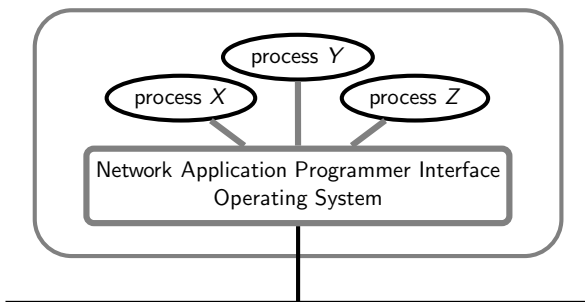
```
while(chatting) {
    msg = read_message(keyboard);
    write_message(msg, network);
    msg = read_message(network);
    write_message(msg, screen);
}
```

```
while(chatting) {
    msg = read_message(network);
    write_message(msg, screen);
    msg = read_message(keyboard);
    write_message(msg, network);
}
```

- For each pair of communicating processes, it makes sense to distinguish two **roles**
- Client** : process that **initiates the communication**
  - specifically, if the communication is carried over a connection-oriented service, then the client is the process that establishes the connection
- Server** : process that **waits to be contacted**
  - specifically, if the communication is carried over a connection-oriented service, then the server is the process that passively accepts the connection
- Some applications have processes that act both as clients and servers. This is often called **peer-to-peer** architecture
- Warning : this classification is useful, but it is little more than nomenclature. Some applications and protocols mix and confuse those terms (e.g., FTP)

## Processes and Hosts

- An end system (host) may run multiple processes



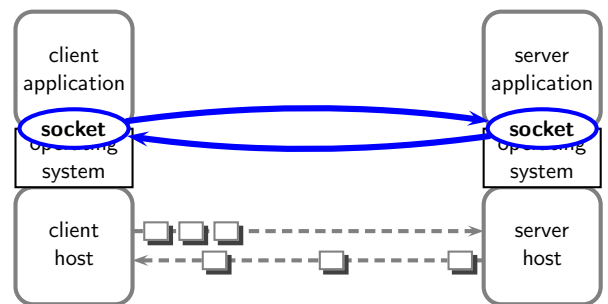
- A process is addressed (within its host) by its **port number**

## Application Programs

- Client application
  - create a socket *C* by “connecting” to the server application
    - i.e., connect to host *H* on port *P*
  - use socket *C* by reading and writing data into it
    - this is the body of the client application protocol
  - disconnect and destroy *C*
- Server application (running on host *H*)
  - create a socket *S* by “accepting” a connection on port *P*
    - a port is often called a “server socket”
  - use socket *S* by reading and writing data into it
    - this is the body of the server application protocol
  - disconnect and destroy *S*

## Sockets

- The **operating system** manages the network interfaces
- Applications use the network through **sockets**



## Example 3 (HTTP)

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply);
}
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket);
}
```

## Example 3 (HTTP)

## Example 3 (HTTP)

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply);
}
```

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply);
}
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket);
}
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket);
}
```

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply); }
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket); }
```

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply); }
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket); }
```

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply); }
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket); }
```

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply); }
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket); }
```

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply); }
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket); }
```

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply); }
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket); }
```

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply); }
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket); }
```

```
while(browsing) {
    url = read_url(keyboard);
    socket = open_connection(url);
    request = compose_http_request(url);
    write_message(request, socket);
    reply = read_message(socket);
    display_web_page(reply); }
```

```
while(serving_http) {
    socket = accept_connection();
    request = read_message(socket);
    reply = serve_http_request(request);
    write_message(reply, socket); }
```

- Developed in the early 1990s
- Based on the idea of **hypertext** and **links**
- Extremely successful, even though...
  - ▶ the **HyperText Transfer Protocol (HTTP)** is just a glorified file transfer protocol
  - ▶ the idea of **hypertext** and **links** was already quite old at the time HTTP was developed
- Success factors
  - ▶ simplicity (openness) of the HTML language and
  - ▶ simplicity of HTTP (a stateless protocol)
  - ▶ low entry barrier for “publishers”
  - ▶ GUI browsers (remember Netscape?), search engines, etc.

- **document**—a web page is also called a **document**
- **objects**—a document may contain several objects (images, applets, etc.). An **object** is simply a file
- **URL**—or **Uniform Resource Locator** specifies the address of an object
- **browser**—also called **user agent** is the program that users run to get and display documents
- **Web server**—is an application that houses objects, and makes them available through the HTTP protocol

## Overview HTTP

## Example : Request

- The main purpose of HTTP is to provide access to Web objects
- Uses a connection-oriented transport mechanism (i.e., TCP)
  - ▶ although it can also work on UDP
- Consists of a **sequence of requests** issued by the client, and **responses** issued by the server, each one in response to a single request
- HTTP is **stateless**
  - ▶ the behavior (semantics) of an HTTP request does not depend on any previous request

- Client request

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr
```

## Example : Reply

## Protocol Features

- Server reply

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 15 Mar 2005 10:00:01 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Tue, 8 Mar 2005 16:44:00 GMT
Content-Length: 2557
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
...
```

- Request
  - ▶ protocol version
  - ▶ URL specification
  - ▶ connection attributes
  - ▶ content/feature negotiation
- Reply
  - ▶ protocol version
  - ▶ reply status/value
  - ▶ connection attributes
  - ▶ object attributes
  - ▶ content specification (type, length)
  - ▶ content

## Protocol Version

## Protocol Version

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr

HTTP/1.1 200 OK
Connection: close
Date: Tue, 15 Mar 2005 10:00:01 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Tue, 8 Mar 2005 16:44:00 GMT
Content-Length: 2557
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
...
```

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr

HTTP/1.1 200 OK
Connection: close
Date: Tue, 15 Mar 2005 10:00:01 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Tue, 8 Mar 2005 16:44:00 GMT
Content-Length: 2557
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
...
```

- **Principle : a protocol should always include a version number**
  - ▶ usually in the very first bits of the protocol (negotiation messages)
- A mechanism to negotiate the protocol version allows the protocol design to change
  - ▶ **design for change**

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr

HTTP/1.1 200 OK
Connection: close
Date: Tue, 15 Mar 2005 10:00:01 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Tue, 8 Mar 2005 16:44:00 GMT
Content-Length: 2557
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
...
```

## URL

## The Host Header

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr

HTTP/1.1 200 OK
Connection: close
Date: Tue, 15 Mar 2005 10:00:01 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Tue, 8 Mar 2005 16:44:00 GMT
Content-Length: 2557
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
...
```

- `http ://www.isae.fr/elochin/index.html`

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr
```

## The Host Header

## The Host Header

- `http ://www.isae.fr/elochin/index.html`

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr
```

- The **host name** in the URL determines where the request goes
  - ▶ host name maps to a network address

- `http ://www.isae.fr/elochin/index.html`

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr
```

- The **host name** in the URL determines where the request goes
  - ▶ host name maps to a network address
- The **host name** is also passed as a parameter within the request, so that the server knows the full URL

## The Host Header

## Connection

- `http ://www.isae.fr/elochin/index.html`

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr
```

- The **host name** in the URL determines where the request goes
  - ▶ host name maps to a network address
- The **host name** is also passed as a parameter within the request, so that the server knows the full URL
  - ▶ this is to allow a single server to serve multiple "virtual" sites (e.g., atelier.isae.fr and www.isae.fr)

```
GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr

HTTP/1.1 200 OK
Connection: close
Date: Tue, 15 Mar 2005 10:00:01 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Tue, 8 Mar 2005 16:44:00 GMT
Content-Length: 2557
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
...
```

```

GET /elochin/index.html HTTP/1.1
Host: www.isae.fr
Connection: close
User-agent: Mozilla/4.0
Accept-Language: fr

HTTP/1.1 200 OK
Connection: close
Date: Tue, 15 Mar 2005 10:00:01 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Tue, 8 Mar 2005 16:44:00 GMT
Content-Length: 2557
Content-Type: text/html

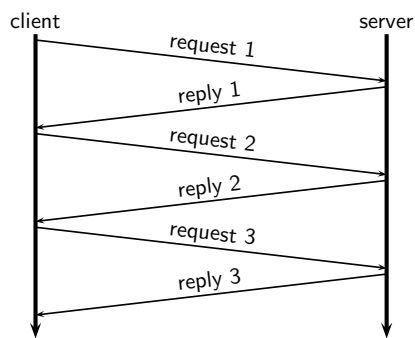
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
...

```

- The first version of HTTP used one (TCP) connection per object
  - ▶ inefficient use of the network
  - ▶ inefficient use of the operating system
- HTTP/1.1 introduces **persistent** connections
  - ▶ the same (TCP) connection can be used by the client to issue multiple request, and by the server to return multiple replies, and possibly multiple objects
  - ▶ the default behavior is to use persistent connections
  - ▶ “Connection: close” in the request and response indicates the intention, of the client and server, respectively, to **not** use a persistent connection while “Connection: keep-alive” enables it

## How HTTP Uses Persistent Connections

- A persistent connection can be used to request and transfer two or more objects



## Persistent Connections With Pipelining

- A more efficient use of a connection is by **pipelining** requests

