Basics of Networking 2: Transport

Daniel STAN

May 27, 2024

Original content by Eric Milard *Éléments de réseau 2* Contrib and Speakers: Daniel Stan, Nidà Meddouri, Elloh Adja, Suzana Dedefa, Christian Diaconu
Version: v1.0, feedback and remarks to: `daniel.stan@epita.fr`

Outline

- Transport: TCP vs UDP
- TCP in practice
- Example of TCP protocol: HTTP

OSI vs TCP/IP

OSI Model

Layer	Function	Usage
Application	User Interface	Data
Presentation	Data Encoding	Data
Session	Session between applications	Data
Transport	Session between terminals	Segment
Network	Global Addressing	Packet
Link	Local Addressing, Medium Access	Frame
Physical	Physical Signal Encoding	bit

TCP/IP Model

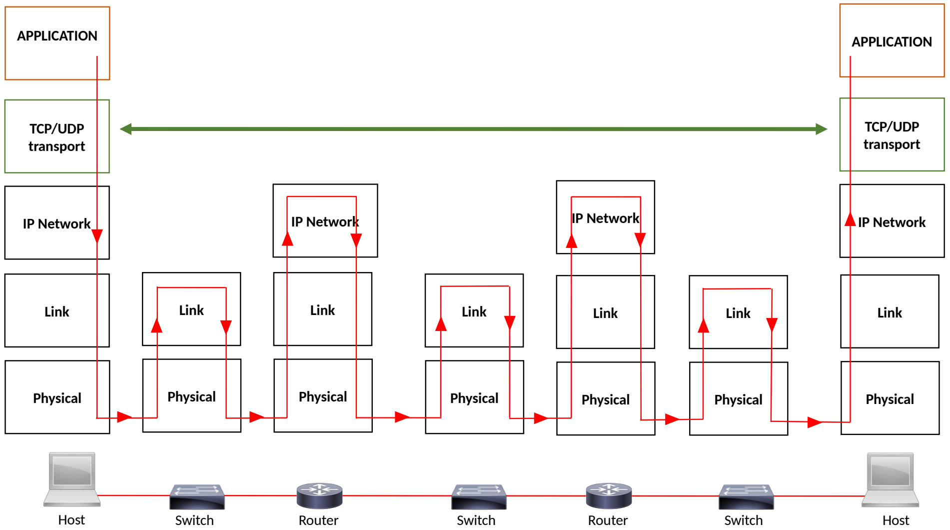
	exemple
Application	HTTP(s) DNS FTP TFTP TELNET ...
Transport	TCP, UDP
Network	IPv4, IPv6
Link	Ethernet
Physical	100BT 1000BT

Hence the joke/memes about "layer 8" problems.

The Transport Layer

- Transport layer is essential for communication between stations on different networks. It enables **reliable** and **efficient** communication between softwares, **independently of the underlying network protocol**.
- Transport is for transmitting data, segmentation, and re-assembly, flow control and congestion, as well as error correction.

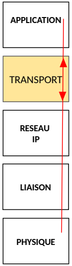
Transport: the global picture



Transport: Receive and Send

- When **sending**, the transport layer receive data from the application layer (TCP/IP model) or session (OSI) and transmit them to the network layer as segments, or packets, respectively.
- When **receiving**, the transport layer receives packet from the network layer, and transmit them to the session layer (or application in TCP/IP),

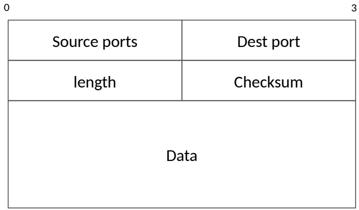
données de la
sion (OSI) et
de segments
paquets de la
session (ou
segments



Transport: Two main protocols

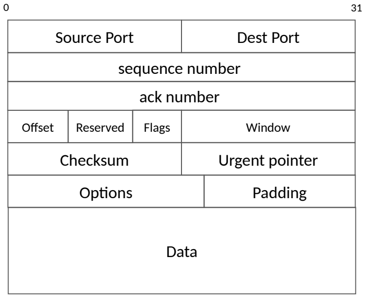
- Transmission Control Protocol (TCP)
  - Session oriented
  - Guarantees delivery
  - ...and ordering
- User Datagram Protocol (UDP)
  - No connection: brief interactions
  - No delivery guarantee
  - No in-order delivery

UDP Header



- **Source port**: a number
- **Destination**: another number
- **Length**: total length of the UDP datagram (8 to 66,535B)
- **Checksum**: some correction code

TCP Header



- **Source port**: a number
- **Destination**: another number
- **Ack number** (acknowledge): flow control
- **Offset**: size of the TCP header
- **Flags**: connection control (Established, Reset, Finish, Sync...)
- **Checksum**: some correction code
- **Urgent point** (if flag = urg): pointer to urgent data.

Source and Destination ports

The **destination port** is used to address what process on the destination machine is this packet for. 16b field, ranging from 0 to 65535. There are three categories of ports:

- 0 to 1023: well known protocols. Example: HTTP: 80, HTTPS: 443, DNS:53, etc
- 1024 to 49151: Registered ports. Example: NFS:2049, MQTT: 1883 (Managed by IANA).
- 49151-65535: dynamically chosen

NB: On most systems, only process running with root privileges can take messages from the first 1024 ports.

Source and Destination ports

The **destination port** is used to address what process on the destination machine is this packet for. 16b field, ranging from 0 to 65535. There are three categories of ports:

- 0 to 1023: well known protocols. Example: HTTP: 80, HTTPS: 443, DNS:53, etc
- 1024 to 49151: Registered ports. Example: NFS:2049, MQTT: 1883 (Managed by IANA).
- 49151-65535: dynamically chosen

NB: On most systems, only process running with root privileges can take messages from the first 1024 ports.

The **source port** is also 16b, and designates the **emitting** process.

- When **initiating** connection, source port is **randomly chosen** by the Operating System.
- When **answering** to a previous message, reuse of the destination port as the source port.

Source and Destination ports

The **destination port** is used to address what process on the destination machine is this packet for. 16b field, ranging from 0 to 65535. There are three categories of ports:

- 0 to 1023: well known protocols. Example: HTTP: 80, HTTPS: 443, DNS:53, etc
- 1024 to 49151: Registered ports. Example: NFS:2049, MQTT: 1883 (Managed by IANA).
- 49151-65535: dynamically chosen

NB: On most systems, only process running with root privileges can take messages from the first 1024 ports.

The **source port** is also 16b, and designates the **emitting** process.

- When **initiating** connection, source port is **randomly chosen** by the Operating System.
- When **answering** to a previous message, reuse of the destination port as the source port.

Other transport protocols don't have ports: ARP, OSPF, etc.

Well known ports

UDP port and TCP are not the same.

TCP

20 - FTP (File Transfer Protocol) - Data
21 - FTP (File Transfer Protocol) - Control
22 - SSH (Secure Shell)
23 - Telnet
25 - SMTP (Simple Mail Transfer Protocol)
53 - DNS (Domain Name System)
80 - HTTP (Hypertext Transfer Protocol)
110 - POP3 (Post Office Protocol 3)
143 - IMAP (Internet Message Access Protocol)
443 - HTTPS (Secure HTTP)

UDP

53 - DNS (Domain Name System)
67 - DHCP (Dynamic Host Configuration Protocol) - Server
68 - DHCP (Dynamic Host Configuration Protocol) - Client
69 - TFTP (Trivial File Transfer Prtocol) - Client
123 - NTP (Network Time Protocol)
161 - SNMP (Simple Network Management Protocol)
162 - SNMP Trap
514 - Syslog (System Logging)
520 - RIP (Routing INformation Protocol)

Well known ports

UDP port and TCP are not the same.

TCP

20 - FTP (File Transfer Protocol) - Data
21 - FTP (File Transfer Protocol) - Control
22 - SSH (Secure Shell)
23 - Telnet
25 - SMTP (Simple Mail Transfer Protocol)
53 - DNS (Domain Name System)
80 - HTTP (Hypertext Transfer Protocol)
110 - POP3 (Post Office Protocol 3)
143 - IMAP (Internet Message Access Protocol)
443 - HTTPS (Secure HTTP)

UDP

53 - DNS (Domain Name System)
67 - DHCP (Dynamic Host Configuration Protocol) - Server
68 - DHCP (Dynamic Host Configuration Protocol) - Client
69 - TFTP (Trivial File Transfer Prtocol) - Client
123 - NTP (Network Time Protocol)
161 - SNMP (Simple Network Management Protocol)
162 - SNMP Trap
514 - Syslog (System Logging)
520 - RIP (Routing INformation Protocol)

NB: DNS uses 53/udp but can fallback to 53/tcp.

The Programmatic Way: Sockets

On most systems: one session $= (IP, Port) \times (SRC, DST) =$ one socket.

```
import socket

# Client: Let's connect on SSH, a protocol on tcp/22
s = socket.create_connection(('ssh.lre.epita.fr', 22))
print(repr(s.recv(4096)))
# b'SSH-2.0-OpenSSH_8.4p1 Debian-5+deb11u1\r\n'
# Let's talk
s.send(b"Hello ? I don't know how to speak SSH actually\n")
# Answer from SSH server:
print(repr(s.recv(4096)))
# b'Invalid SSH identification string.'
s.close()

# Server: '' = all IP addresses
server = socket.create_server(('', 2222))
server.listen()
while True:
    s, client_addr = server.accept()
    s.send(b"Won't talk to you, %r" % (client_addr,))
    s.close()
```

The Unix Way: Netcat

On Linux, we like `cats`, there is a network version of it: `nc`.

```
dstan@flan:~$ echo "Hello ?" | nc ssh.lre.epita.fr 22
SSH-2.0-OpenSSH_8.4p1 Debian-5+deb11u1
Invalid SSH identification string.
```

[1] Carriage Return, Line Feed

The Unix Way: Netcat

On Linux, we like `cats`, there is a network version of it: `nc`.

```
dstan@flan:~$ echo "Hello ?" | nc ssh.lre.epita.fr 22
SSH-2.0-OpenSSH_8.4p1 Debian-5+deb11u1
Invalid SSH identification string.

dstan@flan:~$ nc localhost 2222
Won't talk to you, ('127.0.0.1', 34560)
dstan@flan:~$ nc 10.117.255.88 2222
Won't talk to you, ('10.117.255.88', 58282)
```

[1] Carriage Return, Line Feed

The Unix Way: Netcat

On Linux, we like `cat`s, there is a network version of it: `nc`.

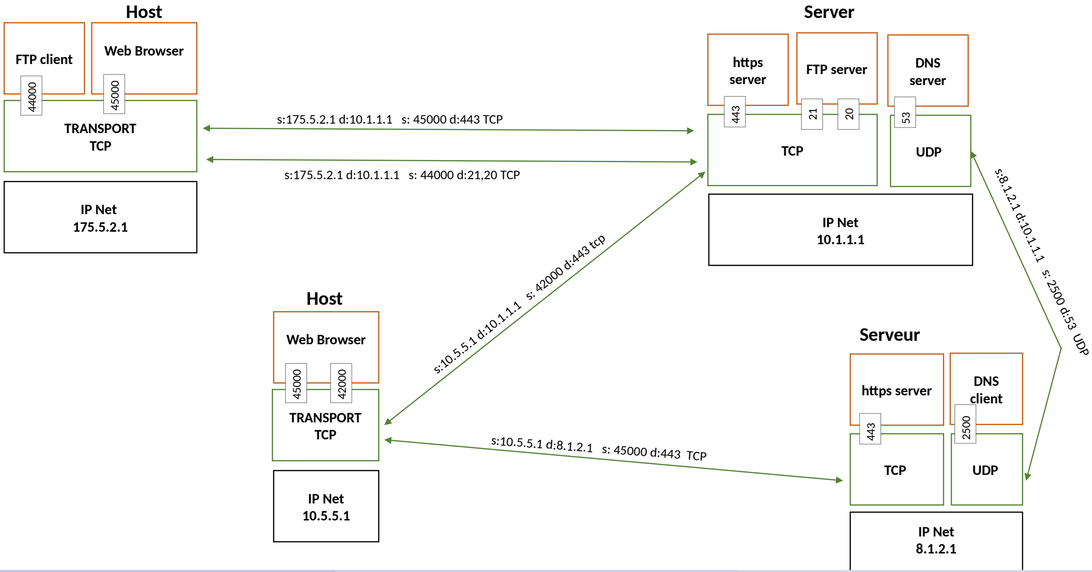
```
dstan@flan:~$ echo "Hello ?" | nc ssh.lre.epita.fr 22
SSH-2.0-OpenSSH_8.4p1 Debian-5+deb11u1
Invalid SSH identification string.

dstan@flan:~$ nc localhost 2222
Won't talk to you, ('127.0.0.1', 34560)
dstan@flan:~$ nc 10.117.255.88 2222
Won't talk to you, ('10.117.255.88', 58282)
```

- `-l` option: **l**isten for incoming connections
- `-u` option for **U**DP
- `-C`: Some protocols expect lines to end with `"`$\backslash r\backslash n$`"` (CRLF[1]), but Linux uses `"`$\backslash n$`"`, this replaces them by `"`$\backslash r\backslash n$`"`.

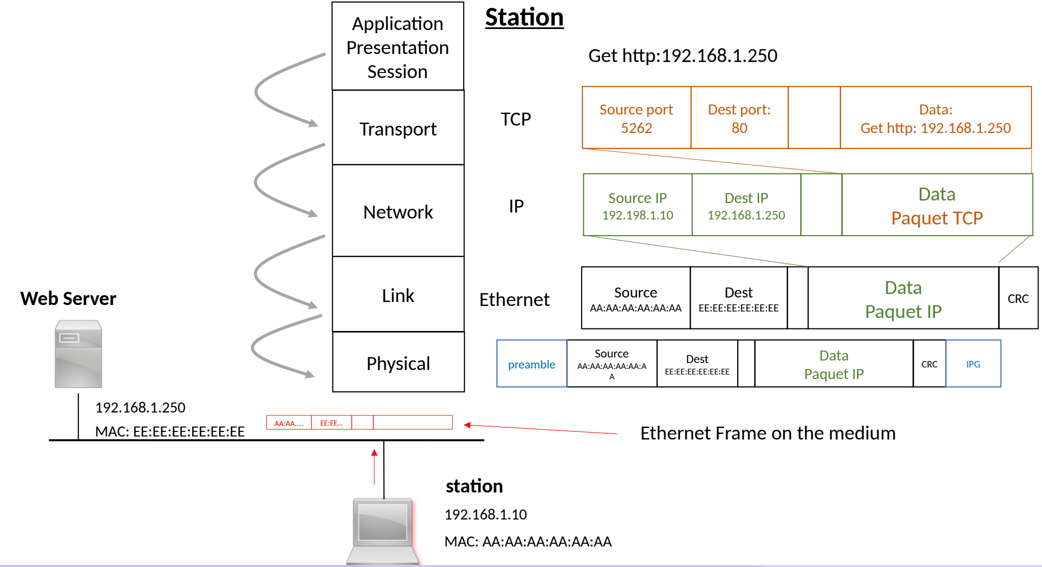
Alternative on any OS: `telnet`, but it's not a `cat`.

[1] Carriage Return, Line Feed

Session examples



Encapsulation



Encapsulation Example

ENCAPSULATION

AA:AA:AA:AA:AA:AA:EE:EE:EE:EE:EE:EE:08:00:
45:00:00:3C:00:00:40:00:40:06:C0:A8:01:0A:C0:A8:01:FA:
14:8E:00:50:00:00:00:00:A0:02:20:00:5A:11:00:00:00:00:01:03:03:07:
67:65:74:20:68:74:74:70:3A:31:39:32:2E:31:36:38:2E:31:2E:32:35:30

Data layer app : get http:192.168.1.250
Header layer 4 : source port 14:8E -> 5262 (dec)
dest port: 00:50 -> 80 ->http
Header layer 3 : source: C0:A8:01:0A -> 192.168.1.10
destination: C0:A8:01:FA -> 192.168.1.250
Protocol: 06 - > TCP
Header layer 2 : source mac: AA:AA:AA:AA:AA:AA
dest mac: EE:EE:EE:EE:EE:EE
Protocol: 0800 -> ipv4

Exercice: Encapsulation

A1:B2:C3:D4:E5:F6:FF:EE:EE:EE:EE:FE:08:00:
45:00:00:3C:00:00:00:00:40:11:C0:A8:01:0A:08:08:08:08:
15:40:00:35:00:33:00:37:00:30:00:30:00:36:
00:37:00:32:00:65:00:70:00:69:00:74:00:61:00:64:
00:6E:00:73:00:00:00:00:00:00:03:77:77:77:06:65:
00:70:00:69:00:74:00:61:00:03:00:66:00:72:00:00:
29:10:00:00:00:00:00:00

- Source IP?
- Destination Port?
- Protocol?
- Source IP?
- Destination IP?
- Source MAC?
- Destination MAC?

Exercice: Encapsulation

A1:B2:C3:D4:E5:F6:FF:EE:EE:EE:EE:FE:08:00:
45:00:00:3C:00:00:00:00:40:11:C0:A8:01:0A:08:08:08:08:
15:40:00:35:00:33:00:37:00:30:00:30:00:36:
00:37:00:32:00:65:00:70:00:69:00:74:00:61:00:64:
00:6E:00:73:00:00:00:00:00:00:03:77:77:77:06:65:
00:70:00:69:00:74:00:61:00:03:00:66:00:72:00:00:
29:10:00:00:00:00:00:00

- Source IP?
- Destination Port?
- Protocol?
- Source IP?
- Destination IP?
- Source MAC?
- Destination MAC?

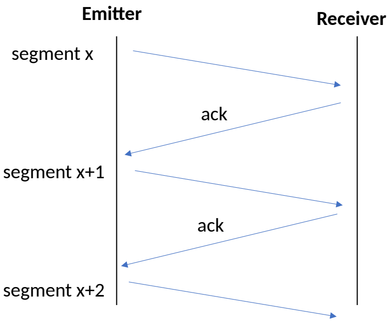
Header layer 4 : source port: 1540 -> 5440 (decimal)
dest port: 0035 -> 53 (decimal) ->dns

Header layer 3 : source: C0:A8:01:0A -> 192.168.1.10
destination: 08.08.08.08 -> 8.8.8.8
Protocol: 11 - > 17 (decimal) ->UDP

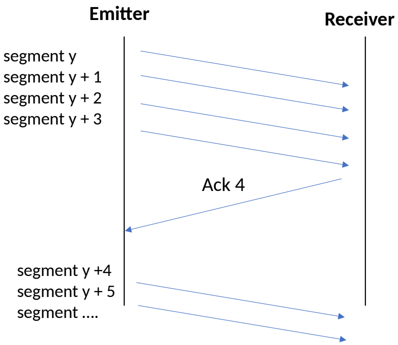
Header layer 2 : source mac: A1:B2:C3:D4:E5:F6
dest mac: FF:EE:EE:EE:EE:FE
Protocol: 0800 -> ipv4

TCP: Flow and Congestion Control

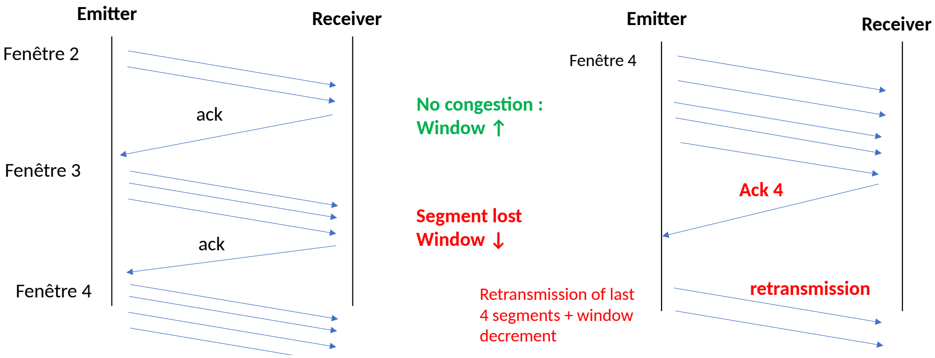
Acknowledgements



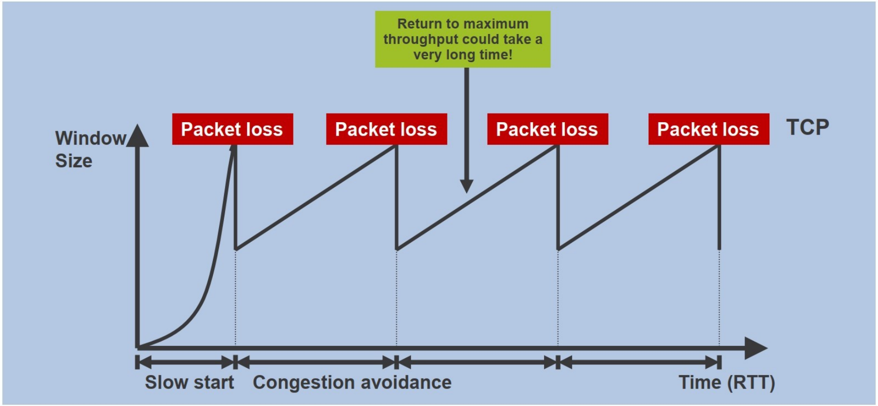
4 segments sent before ack



TCP: Flow and Congestion Control



TCP slow start and bandwidth management



UDP: no congestion mechanism

- “Blind” emission of data
- No retransmission in case of packet losses
- Not a problem for applications dealing with message losses on their own: for example, audio or video streaming:
    - Useless to send past data
    - Encodings have fallbacks, for example keyframes in MPEG.

Quizz

- How does a **HTTP** server differentiate two **HTTP** sessions from the same machine?
- What is faster, UDP or TCP?
- You're working on an app to collect data from remote sensors. These emit messages on a radio network (prone to perturbation and not reliable). What protocol do you use? Why?

An example of an application protocol

Example: netcat HTTP in 5 minutes

HyperText Transport Protocol is a TCP-based protocol on port 80.
Since it's is an old protocol (1990), it uses "$\backslash r \backslash n$".

Let's showcase how to use `nc` to download the webpage:
`http://ftp.debian.org/debian/README.mirrors.txt`.

Example: netcat HTTP in 5 minutes

HyperText Transport Protocol is a TCP-based protocol on port 80.
Since it's is an old protocol (1990), it uses "$\backslash r \backslash n$".

Let's showcase how to use `nc` to download the webpage:
`http://ftp.debian.org/debian/README.mirrors.txt`.

- Protocol: **http** $\rightarrow$ **tcp/80**
- Remote server is **ftp.debian.org** with IP addresses **199.232.170.132** and **2a04:4e42:6a::644**

Example: netcat HTTP in 5 minutes

HyperText Transport Protocol is a TCP-based protocol on port 80.
Since it's is an old protocol (1990), it uses "$\backslash r \backslash n$".

Let's showcase how to use `nc` to download the webpage:
`http://ftp.debian.org/debian/README.mirrors.txt`.

- Protocol: **http** $\rightarrow$ **tcp/80**
- Remote server is **ftp.debian.org** with IP addresses **199.232.170.132** and **2a04:4e42:6a::644**

⚠ Although it's `ftp.debian.org`, we use here **http** (port 80) not **ftp** (port 21). ⚠

Example: netcat HTTP in 5 minutes

HyperText Transport Protocol is a TCP-based protocol on port 80.
Since it's is an old protocol (1990), it uses "$\backslash r \backslash n$".

Let's showcase how to use `nc` to download the webpage:
`http://ftp.debian.org/debian/README.mirrors.txt`.

- Protocol: **http** → **tcp/80**
- Remote server is **ftp.debian.org** with IP addresses **199.232.170.132** and **2a04:4e42:6a::644**

⚠ Although it's `ftp.debian.org`, we use here **http** (port 80) not **ftp** (port 21). ⚠
→ `nc -C 199.232.170.132 80`

Let's `GET` the file `/debian/README.mirrors.txt`...

Example: HTTP Query

A **Query** is done by the client, **upon connection**:

- First line: is a **method** `GET` (or `POST`, ...), then **path**, then `HTTP/1.1`
- Next lines: arbitrary many **headers** of the form `Header-name: value`;
  Only one **compulsory** header: `Host`, its value is the server domain name.
- Last line: empty
- After: possibly some *data* (only if method is `POST` or `PUT`)

Example: HTTP Query

A **Query** is done by the client, **upon connection**:

- First line: is a **method** `GET` (or `POST`, ...), then **path**, then `HTTP/1.1`
- Next lines: arbitrary many **headers** of the form `Header-name: value`;
  Only one **compulsory** header: `Host`, its value is the server domain name.
- Last line: empty
- After: possibly some *data* (only if method is `POST` or `PUT`)

Here: `http://ftp.debian.org/debian/README.mirrors.txt`

```
$ nc -C ftp.debian.org 80
GET /debian/README.mirrors.txt HTTP/1.1
Host: ftp.debian.org
```

Example: HTTP Answer

The **answers** is sent, when the server is sure the query is done, thanks to the **last empty line**[2].

- First line: is `HTTP/1.1`, then a **status code number** then a text for this status (200=OK).
- Next lines: arbitrary many **headers** of the form `Header-name: value`;
- Last line: empty;
- After: possibly some *data*

[2] if some data is sent, a query header is giving its length

Example: HTTP Answer

The **answers** is sent, when the server is sure the query is done, thanks to the **last empty line**[2].

- First line: is `HTTP/1.1`, then a **status code number** then a text for this status (200=OK).
- Next lines: arbitrary many **headers** of the form `Header-name: value`;
- Last line: empty;
- After: possibly some *data*

Here: `http://ftp.debian.org/debian/README.mirrors.txt` after the `GET` query:

```
$ nc ftp.debian.org 80
...
HTTP/1.1 200 OK
Content-Length: 86
...
Vary: Accept-Encoding

The list of Debian mirror sites is available here: https://www.debian.org/mirror/list
```

NB: check that the text is exactly 86 characters long.

[2] if some data is sent, a query header is giving its length

Exercise: HTTP from netcat

Using netcat, download the following files:

- `http://epita.fr`
- `http://ip.lafibre.info/`
- `http://fr.archive.ubuntu.com/ubuntu/pool/`

Notice that the two last websites have the same IP address **51.158.154.169**! The server provides a different answer depending on the `Host` header: **Virtual Host**.

Exercise: HTTP from netcat

Using netcat, download the following files:

- `http://epita.fr`
  `echo -e "GET / HTTP/1.1\nHost:  epita.fr\n" | nc epita.fr 80`
  301 Moved Permanently to `https://epita.fr/`
- `http://ip.lafibre.info/`
- `http://fr.archive.ubuntu.com/ubuntu/pool/`

Notice that the two last websites have the same IP address **51.158.154.169**! The server provides a different answer depending on the `Host` header: **Virtual Host**.

Exercise: HTTP from netcat

Using netcat, download the following files:

- `http://epita.fr`
  `echo -e "GET / HTTP/1.1\nHost:  epita.fr\n" | nc epita.fr 80`
  301 Moved Permanently to `https://epita.fr/`
- `http://ip.lafibre.info/`
- `http://fr.archive.ubuntu.com/ubuntu/pool/`

Notice that the two last websites have the same IP address **51.158.154.169**! The server provides a different answer depending on the `Host` header: **Virtual Host**.

Exercise: HTTP from netcat

Using netcat, download the following files:

- `http://epita.fr`
  `echo -e "GET / HTTP/1.1\nHost: epita.fr\n" | nc epita.fr 80`
  301 Moved Permanently to `https://epita.fr/`
- `http://ip.lafibre.info/`
  `echo -e "GET / HTTP/1.1\nHost: ip.lafibre.info\n" | nc -C ip.lafibre.info 80 | grep "IPv4 publique"`
  `...<strong> : IPv4 publique = 163.5.2.51</strong>...`
- `http://fr.archive.ubuntu.com/ubuntu/pool/`

Notice that the two last websites have the same IP address **51.158.154.169**! The server provides a different answer depending on the `Host` header: **Virtual Host**.

Exercise: HTTP from netcat

Using netcat, download the following files:

- `http://epita.fr`
  `echo -e "GET / HTTP/1.1\nHost: epita.fr\n" | nc epita.fr 80`
  301 Moved Permanently to `https://epita.fr/`
- `http://ip.lafibre.info/`
  `echo -e "GET / HTTP/1.1\nHost: ip.lafibre.info\n" | nc -C ip.lafibre.info 80 | grep "IPv4 publique"`
  `...<strong> : IPv4 publique = 163.5.2.51</strong>...`
- `http://fr.archive.ubuntu.com/ubuntu/pool/`
  `echo -e "GET /ubuntu/pool HTTP/1.1\nHost: fr.archive.ubuntu.com\n" | nc -C fr.archive.ubuntu.com 80 -i`

Notice that the two last websites have the same IP address **51.158.154.169**! The server provides a different answer depending on the `Host` header: **Virtual Host**.

Summary: UDP vs TCP

Differences:

- UDP offers no acknowledgement mechanism and no ordering, faster but unreliable.
- TCP offers a **reliable**, **resilient** and **persistent** connection between two hosts.

Similarities:

- **Transport** layer
- Based on source and destination **ports**
- **Binary encoding** of datagrams on this layer (similar to IP and below)
- First layer to provide a communication stream: **socket**