Networking v0.9

2012

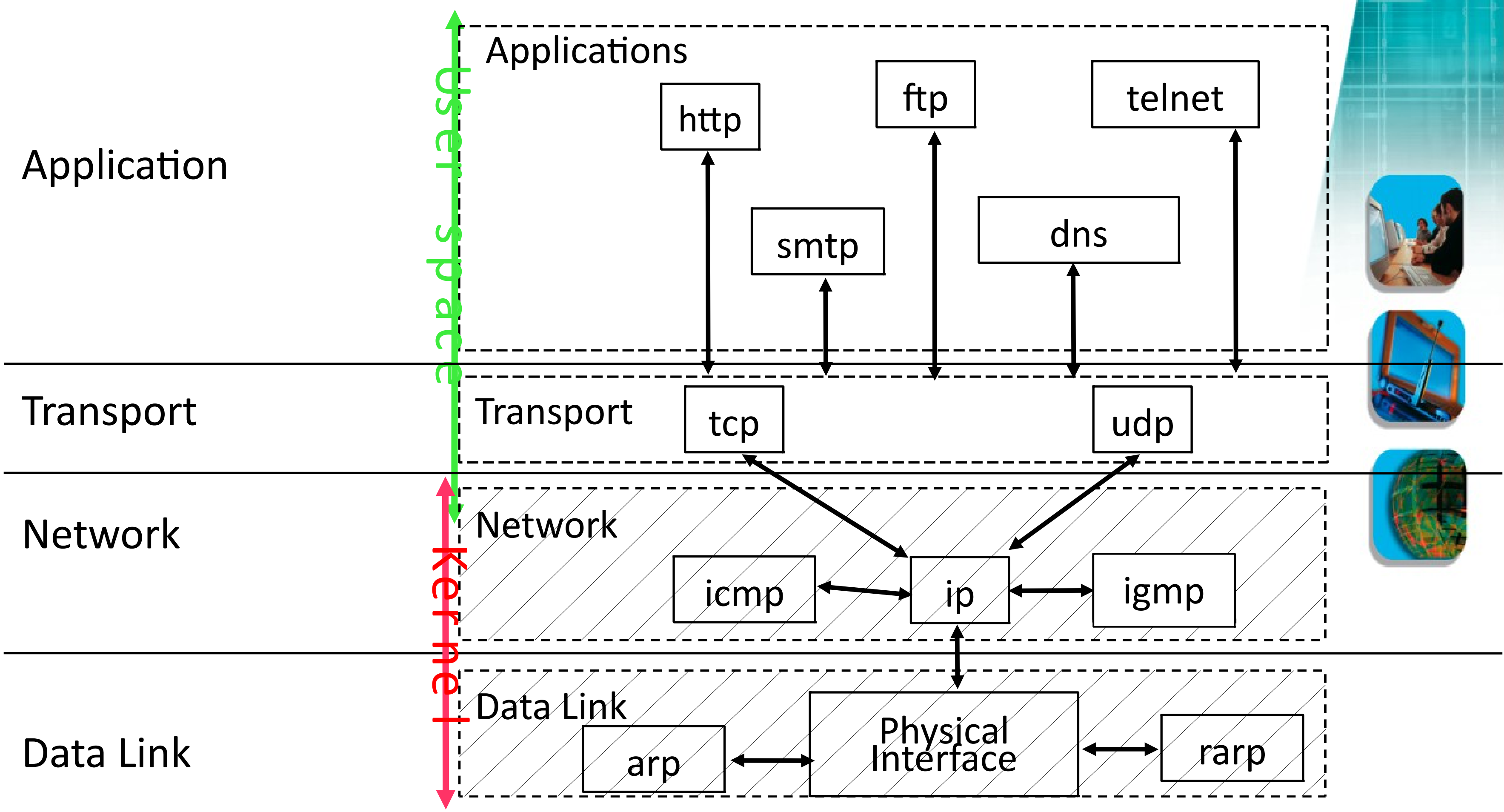
Fabrice Valois, fabrice.valois@insa-lyon.fr

membre de Université de Lyon

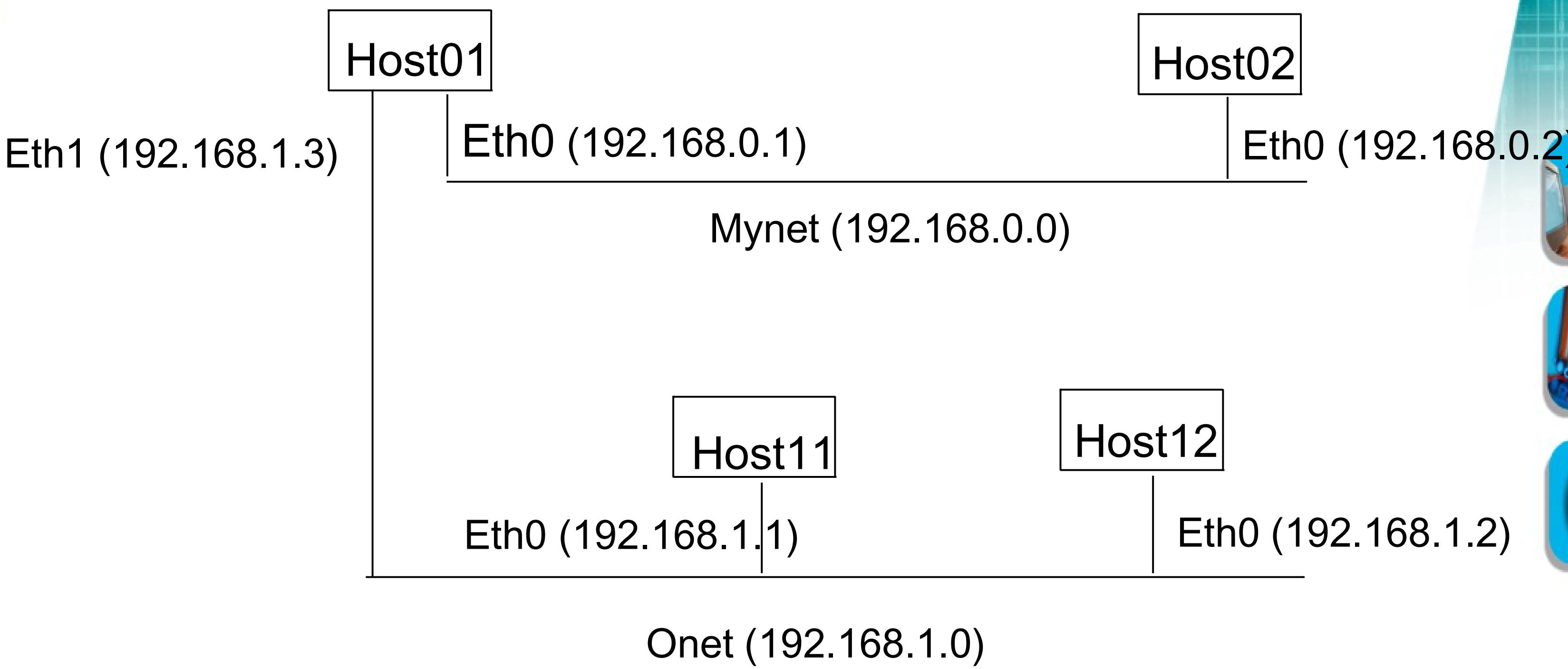
Département Télécommunications
Services & Usages

INSA LYON

Where are we?


Application
Transport
Network
Data Link
user space
kernel
Applications
http
ftp
telnet
smtp
dns
Transport
tcp
udp
Network
icmp
ip
igmp
Data Link
arp
Physical Interface
rarp

Back to your memory :-)



Chapter 6
Transport Protocols (TCP/UDP) : headers, mechanisms and algorithms

membre de UNIVERSITÉ DE LYON

Département Télécommunications
Services & Usages

INSA LYON

Agenda

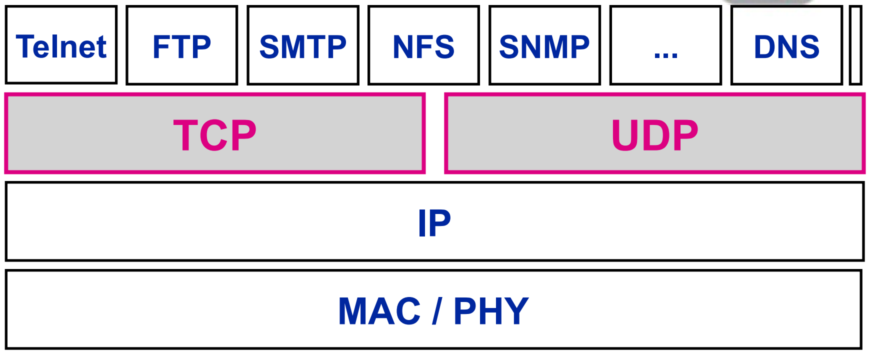
- General overview of transport protocols in IP
- UDP
- TCP
- TCP Connection management
- Congestion management, flow control

Transport Protocols

- End-to-end segments management
- Transport of application datas
- IP is always used to route the packets

- TCP – *Transport Control Protocol*
  - Reliable transport protocol
  - Connected mode

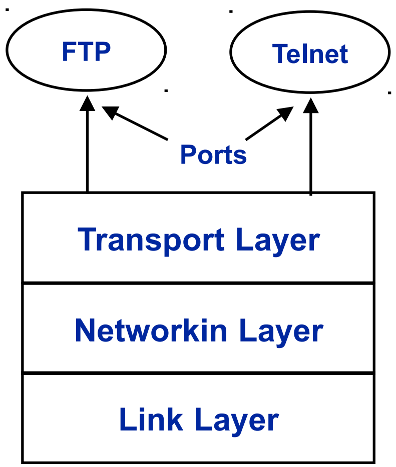
- UDP – *User Datagram Protocol*
  - Non reliable protocol
  - Non connected



What is a `port`?

- Provide an access to a service (smtp, *e.g.)* and to an application (mail)

- A port is defined by a unique number and is used to identifiy an application for the transport layer (TCP, UDP, ...)
- RFC 1700 : port 1 → 1023 are standardized ports
  but ports > 1024 are free of use
- /etc/services : is a list of all the ports and their use



- Examples :

Application	Port
FTP	20
Telnet	23
SNMP	161

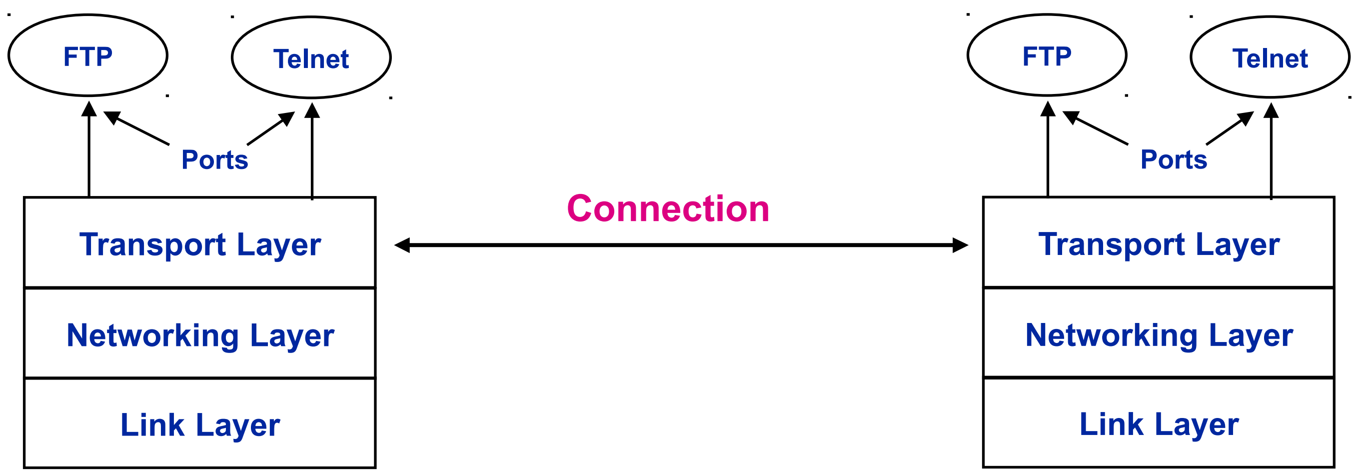
Connection?

→ end-to-end establishment for client–server information exchange

- @IP_{source} and @$IP_{destination}$ are used to identify the hosts
- applications are identified by $port_{source}$ and $port_{destination}$

→ It is a `socket`!

- Example : (18.26.0.36, 1069) et (128.10.2.3, 25)



UDP

User Datagram Protocol (RFC 768)

- Basic mechanism for the end-to-end transport
- Non reliable service
- Non-connected mode
- Based on IP (`Protocol` field: 17)
  → Allow the use of the ports only

- Simple header of 8 bytes

0 … 15	16 … 32
Port UDP source	Port UDP destination
Total Length (bytes)	CRC (optionnal)
Data (Application and associated header)	

UDP

- How it works?

- **- IP is only focused on the routing**
- **- UDP allows the end-to-end transport and the use of ports**
- **- UDP allows the IP fragmentation but without any guarantee that the destination will be able to reassemble the packets**
- **- UDP does not provide additionnal mechanisms for : retransmission, timeout, acknowledgement, application data fragmentation (max. 64 bytes), congestion, etc.**
  - **→ If reliability is required, the mechanisms are provided by the application**



- Examples :

- **- DNS, TFTP, traceroute, ...**
- **- Video Streaming, Network gaming, ...**

TCP

Transmission Control Protocol (RFC 793)

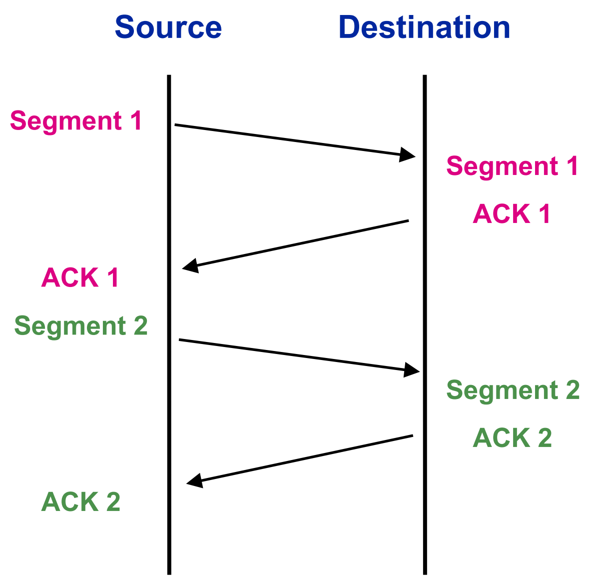
- TCP is based on a connection: to allow an end-to-end segment exchange, the 2 hosts should open a connection
- Reliable transport (using additionnal mechanisms like: retransmission, duplication management, timeout, ...)
- Mechanisms to improve performances: flow control management, sliding window, Naggle, Clark, ...
- IP is used to route the segments to the sourche (`Protocol` field: 6)

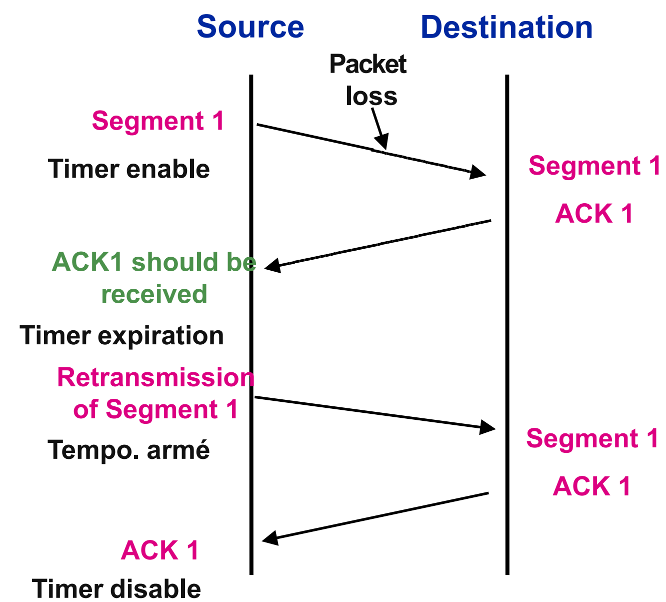
TCP

- **How to be reliable?**
  - **TCP can do the fragmentation for the application data. The size of the fragments are managed by TCP. Segments are transmitted successively.**
  - **When a segment is sent, TCP used a timer to wait an acknowledgement from the destination. When the timer reaches 0 and there is no ACK: the packet is lost → retransmission.**
  - **Each time TCP receives a segment, it send an ACK**
  - **The header and the data of a TCP segment are protected using a CRC**
  - **TCP puts in order the segments received before to transmit it to the application (through the use of the ports)**
  - **TCP provides a flow control (using local buffer)**

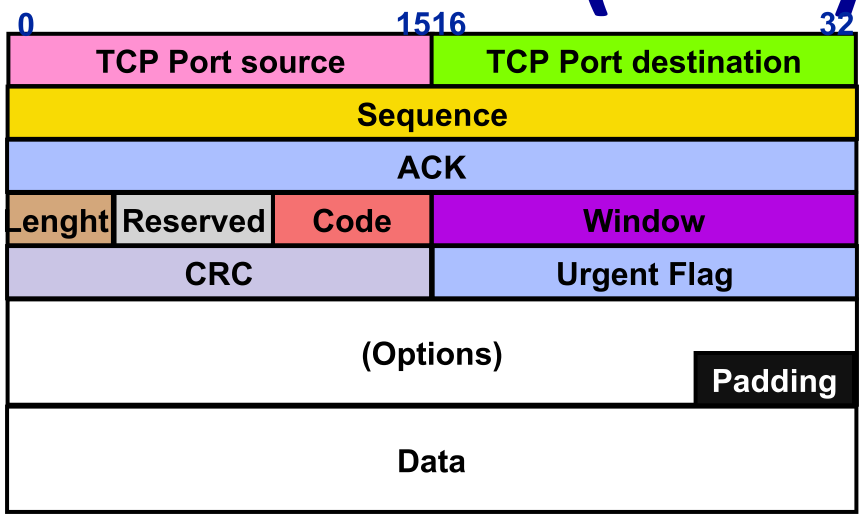
Reliability of TCP

- **Basic mechanisms:**
  - For each segment, TCP uses an ACK
  - Explicit ACK for the last *k* bytes received, waiting for the *k+1*
  - Using a timer to detect loss, congestion





TCP header (20 bytes)



Sequence **: Provide the position of the current byte in the flow of bytes transmitted from this host to the destination**

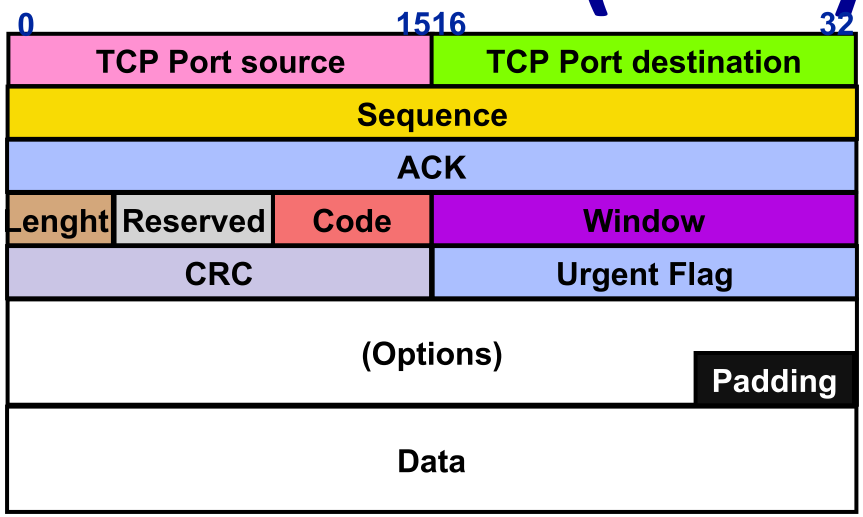
Ack **: the next byte waited (then, acknowledgement of the previous bytes)**

Length (4 bits) **: Header length because of options (32 bit-words)**

Window **: Number of bytes that the host can receive (flow control)**

CRC : Security for the header + the data

TCP header (20 bytes)



***Code (6 bits)* :**

- **URG** : the 'urgent flag' is used
- **ACK** : to declare the use of the ACK field
- **PSH** : the application data should be deliver as soon as possible
- **RST** : Connection restart
- **SYN** : During the establishment connection phase, to declare the initial value of the Sequence field
- **END** : End of the segments transmission (closing the connection)

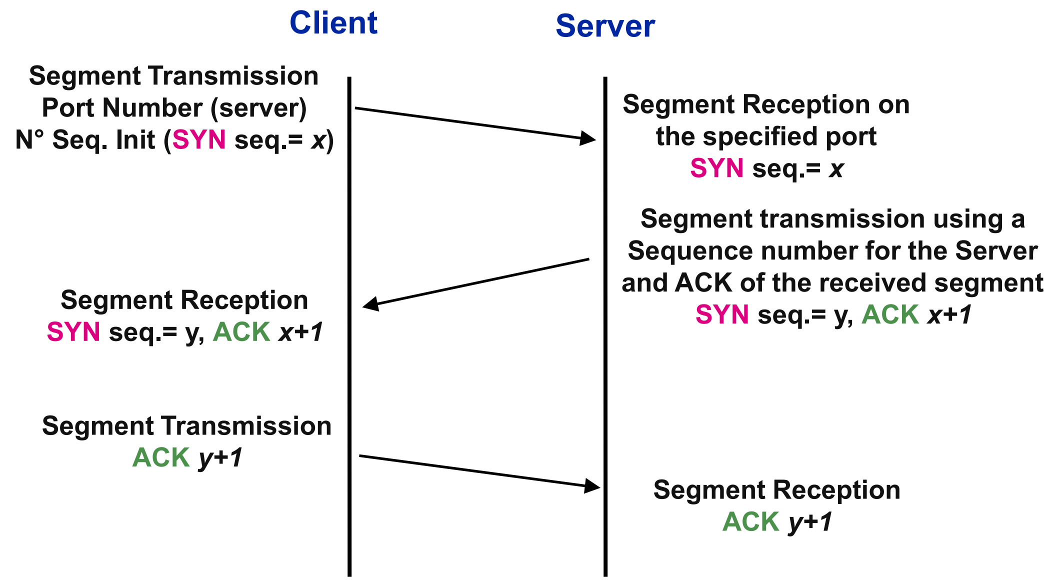
***Urgent flag*: Segment should be transported as urgent**

***Options* : Mainly the MSS (Maximum Segment Size), used by the sender to declare to the remote host the maximum segment size (in bytes), he is able to receive**

Connection management

- TCP is a connected-based transport protocol ⇒ Before to send and/or to receive segments, it is requested to open a connection...and to close it at the end of the segments exchange:
    - 3 steps for establishment
    - 4 steps for closing
- The establishment phase allows the 2 hosts to declare the initial values for the `Sequence` field

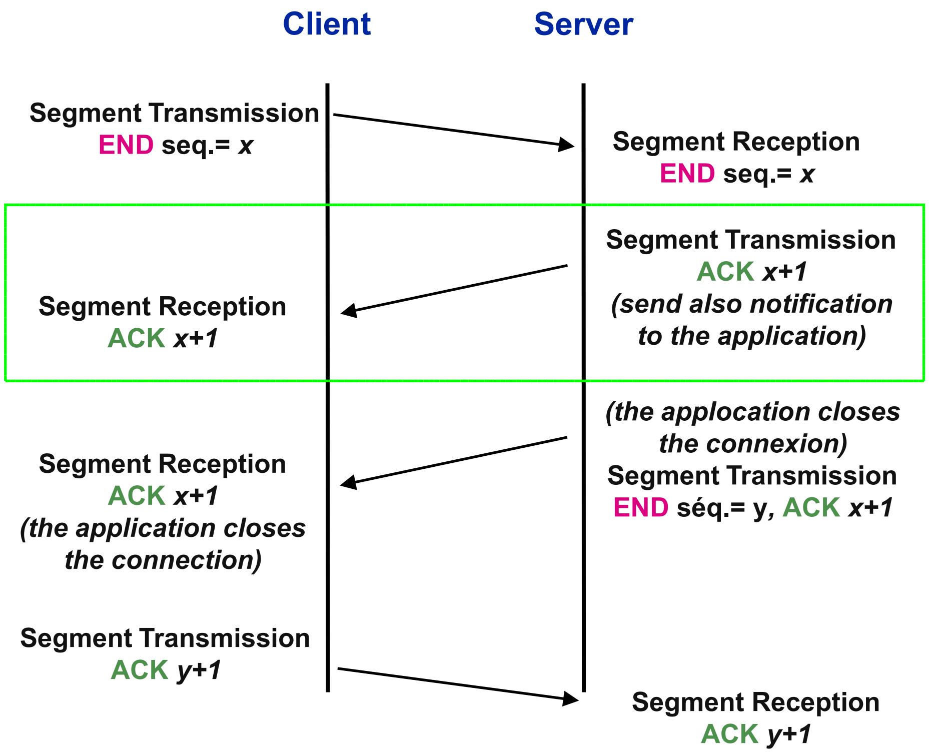
Connection establishment



- If there is no response from the server?

→ a *timer* is used, then several connection establishment requests are sent

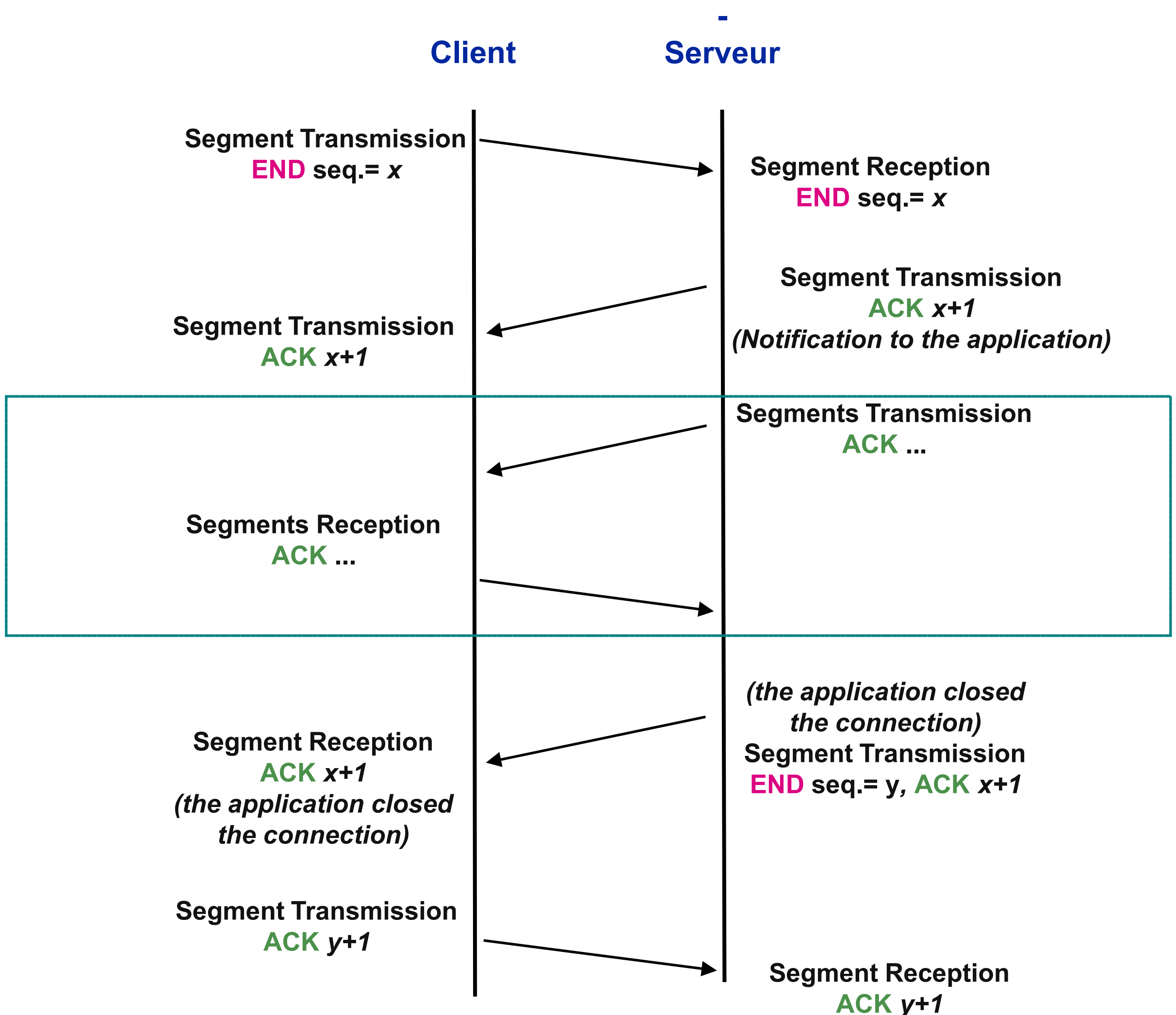
To close a connection



To confirm to the client that the connection is closed, and...
To avoid the remote application to send another END request.

- *full-duplex* transmission (4 steps)
- END: end of the segments transmission from the sender

Half-closed connection



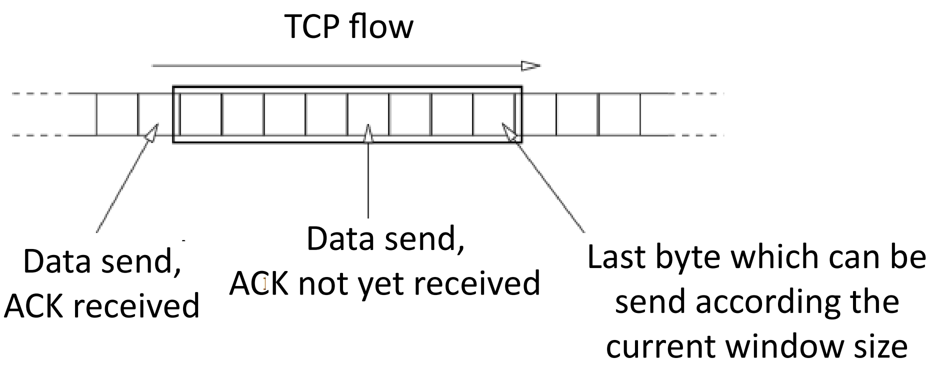
Segment Size and the MSS option

- *Default value* of the data length:
    - locally: 1460 bytes
    - if the segment is routed to a different subnetwork
      → max segment size=536 bytes
      (packet size = 20 (IP) + 20 (TCP) + MSS (Data))
- During the connection establishment, SYN can be used to notify a desired segment size in reception

- Note that the optimal MSS value is the MTU value

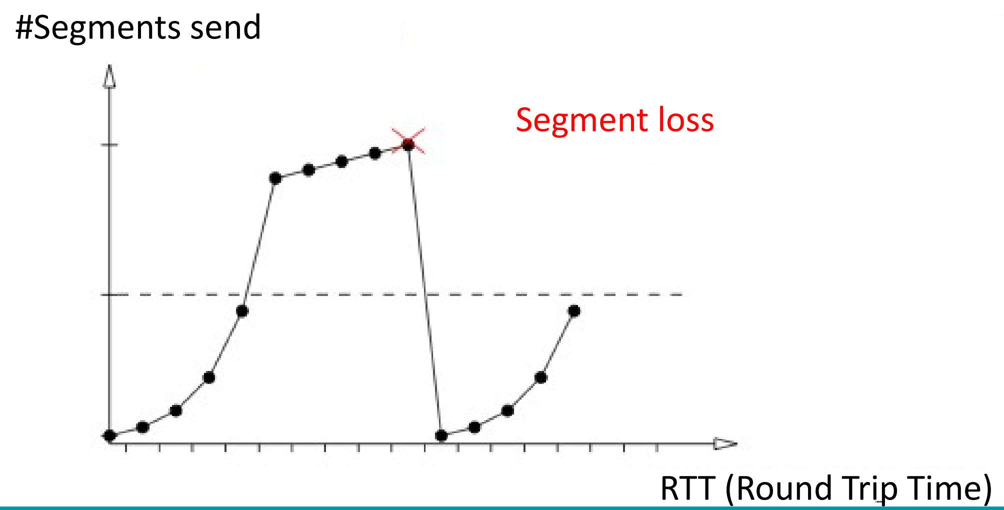
Sliding window

- **Allow the transmission of several segments before to receive ACKs**
- **The size of the window is dynamically adapted according to the host capacity**
- **Can be used to freeze a transmission (Field `window=0`)**
- **Basic idea of the *sliding window* mechanism:**
  - **Window size ↓ when the receiver is congested (no ACK)**
  - **Window size ↑ when the receiver acknowledges segments**



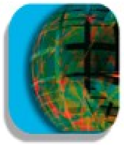
Congestion avoidance

- Based on the work of Van Jacobson (1988)
- Without knowledge of the network state, without information of the destination load, based only on the ACK received, the throughput is dynamically adapted to :
  - the network congestion
  - the packet loss
  - the load of the destination
- *slow-start* mechanism:



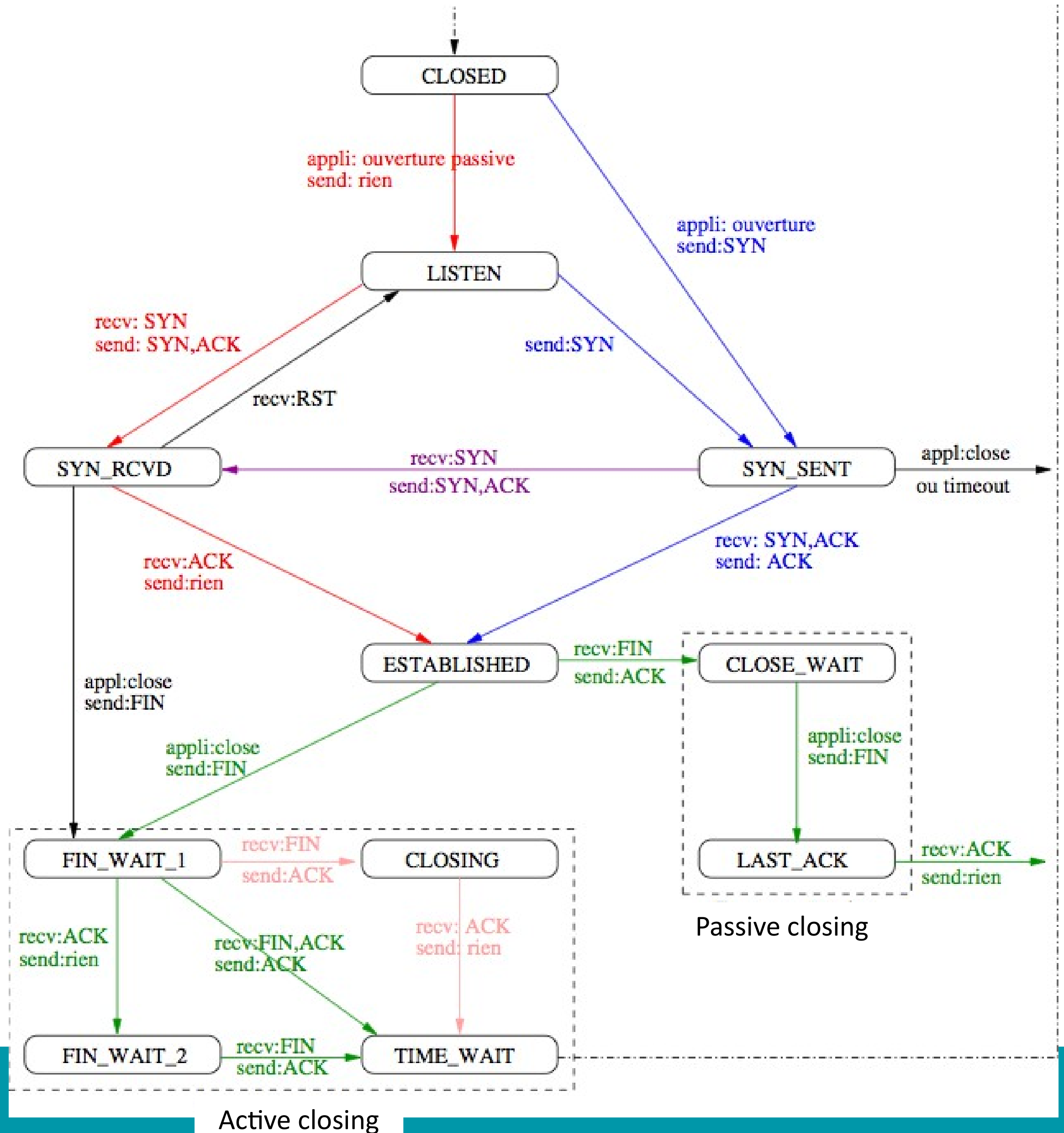
Congestion avoidance (cont'd)

- **Local management of the window size (never transmitted)**
- **Self-adaptation of the slow-start mechanism allowing to find the optimal value of the window according to the network congestion:**



  - **Initially:**
    - **Transmission of 1 Segment / Waiting for ACK**
  - **Exponential increase of the window size**
    - **Transmission of 2 Segments / Waiting for ACK**
  - **…**
  - **No ACK → the window is set to the initial value**
  - **Then, new transmission of segments following an exponential increase of the window size (current value of the congestion), then linear increase of the window size to determine a new congeston value**
  - **…**

TCP : Finite State Machine

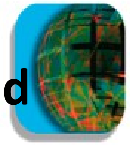


Legend
Server
Client
Connection establishment
Active/passive closing
Symmetric closing
Closing without exchange

General remarks

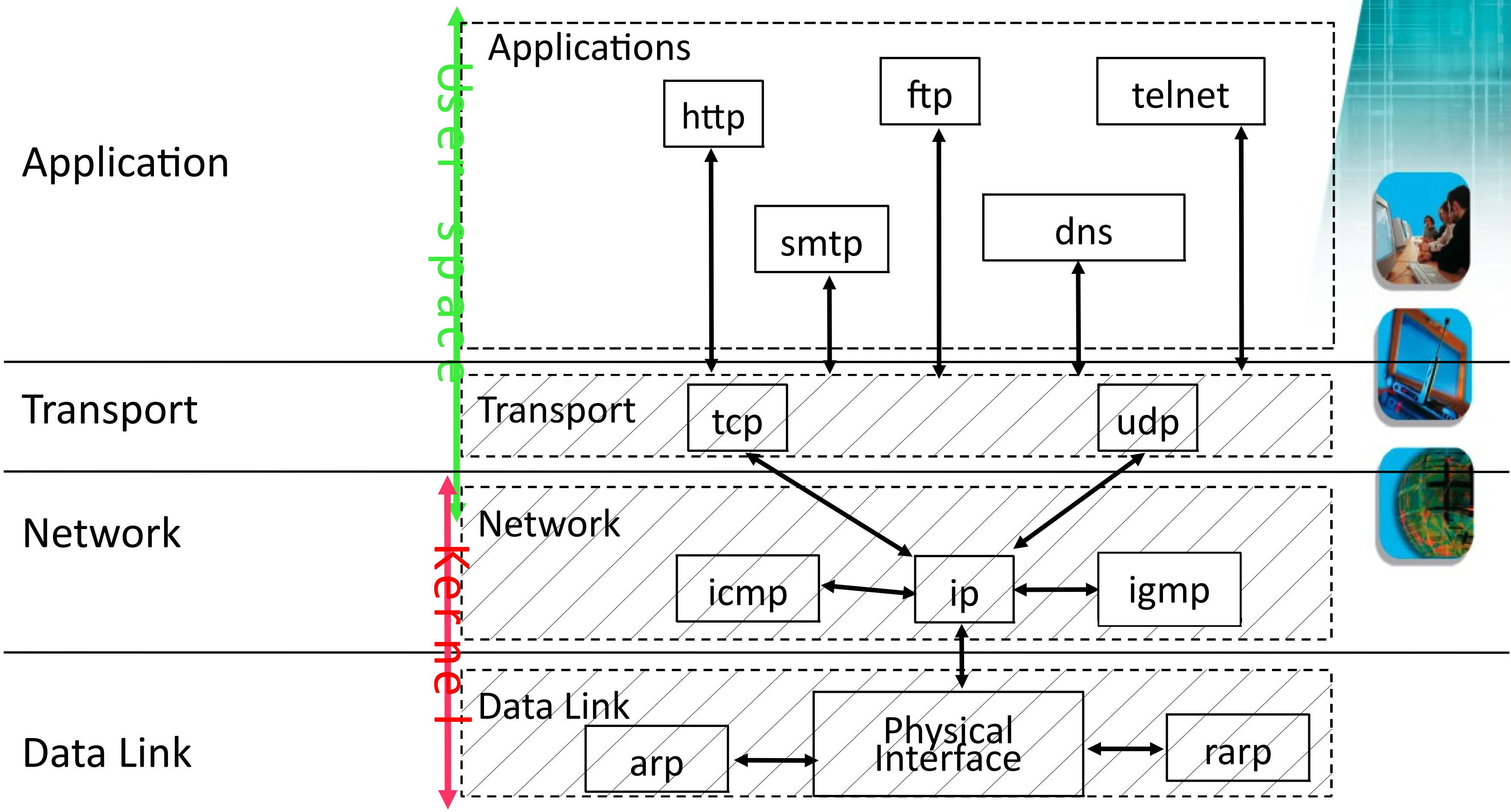
- A TCP connection is active until the use of an explicit END segment (or the client/server reboots)
- Warning: Neither link failure nor a route failure closes a TCP connection !
- In the point of view of performance, there is a dedicated timer for connection: the *keepalive timer.*

General remarks (cont'd)



- Several implementations of TCP are available:
  - TCP Reno: exponantial *slow-start*, management of duplicate ACK's (segments loss)
  - TCP Vegas: linear increase of the sliding window, RTT evaluation for all the tranmistted segments for timer adaptation
  - TCP New Reno (used in Linux > 2.6.8): if duplicate ACKs are received then retransmission of the segments without to wait the timeout, introduction to a Selective ACK

Where are we?



Application
Transport
Network
Data Link
user space
kernel
Applications
http
ftp
telnet
smtp
dns
Transport
tcp
udp
Network
icmp
ip
igmp
Data Link
arp
Physical Interface
rarp