Transport and Application Layer Principles

COMPUTER NETWORKS A.A. 24/25

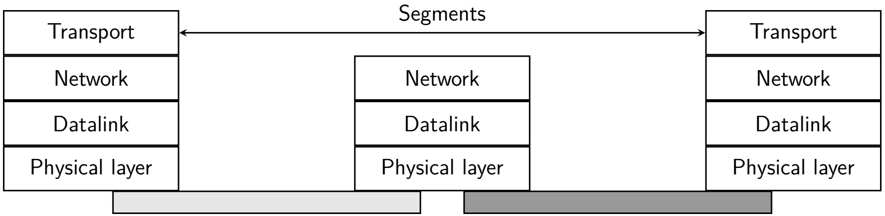
Leonardo Maccari, DAIS: Ca' Foscari University of Venice,
leonardo.maccari@unive.it

Venice, fall 2024

License

Sect. 1 The Transport Layer

Back to the TCP/IP stack



Layers and Services

- At this point we should know that every layer is independent, and every layer offer services to the layer below.
- To introduce the transport layer we need to make some assumptions on the network layer, and to understand the needs of the application layer

The Assumptions on Network Layer Service

- The Network Layer that we use on the Internet offers an SDU delivery service that is: *connectionless* and *unreliable*.
- It means that it is datagram oriented (not circuit-based)
- and it does not guarantee any reliability.

Summarising: Limits of the Network Layer

1. the connectionless network layer service can only transfer SDUs of limited size
2. the connectionless network layer service may discard SDUs
3. the connectionless network layer service may corrupt SDUs
4. the connectionless network layer service may delay, reorder or even duplicate SDUs

Requirements Given by The Applications

One general requirement from the application layer is easy to understand:

Make the delivery reliable

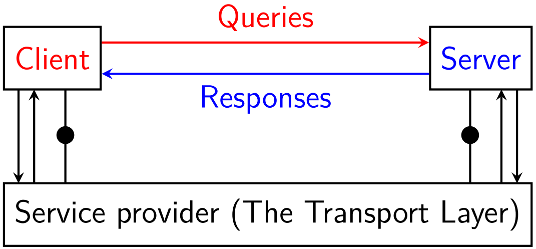
This means:

- Ensure all data arrive correctly
- Ensure the receiver can reorder them
- Ensure duplicates are detected

For a second requirement we need to have an abstract model for the applications...

Application Models: Client-Server Model

- It is highly asymmetrical and identifies two different roles:
  - The server: the software offering the service
  - The client: the software accessing the service
- It is reasonable to assume that Internet hosts have more than one piece of client/server software running at the same time.



Multiplexing Applications

- Multiplexing roughly means "*sharing a single channel for multiple applications*"

Multiplexing

The transport layer must make it possible to distinguish the data flow that belongs to two different client-server couples.

Transport Layer Terminology

- The applications do not know anything about packets, they share a memory **buffer** with the transport layer that contains a stream of data
- The transport layer **segments** that stream and deliver it to the network layer, that will put them in packets.
- So at the transport layer we refer to **segments** as the unit of data, while we refer to **packets** when we consider network-layer units.

Transport Layer Services

We deal with two models

- The connectionless transport layer service (offering *multiplexing* only)
- The connection-oriented transport layer service (offering *multiplexing* **and** *reliability*)

The Transport Layer

↳ 1.1 Connectionless Service

Connectionless Service

The connectionless transport layer service provides:

- Unreliable delivery of data, usually with error detection
- Multiplexing at the two endpoints.

! **It is normally stateless: every segment travels independently from the previous one. The transport layer endpoints do not maintain any state variable**

Typical Applications: Interactive Applications

- Video streaming, audio streaming, gaming
- These are applications that accept the loss of some data, and do not care about retransmitting
- Indeed, if the video conference has arrived at second t_1, it is not useful to ask for the retransmission of a lost segment that contains information for time $t < t_1$.

Transport Layer Headers

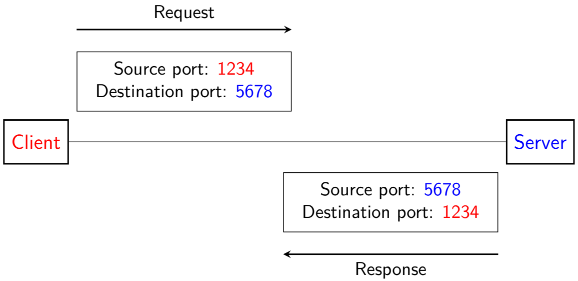
- To guarantee error detection, the transport layer adds a header that contains a checksum of the data
- When receiving a packet, the receiver will read the header and compute the checksum on the segment

Multiplexing

- It answers the question: If the server serves many client at the same time, how does it distinguish the traffic that goes to each of them?
- The transport layer solves this with the concept of ports.

Port Numbers

- The transport layer header contains two ports
- One identifies the application on the client, one on the server
- When the transport layer receives the segment it delivers it to the correct application based on the port number



The Transport Layer

↳1.2 Connection-oriented Service

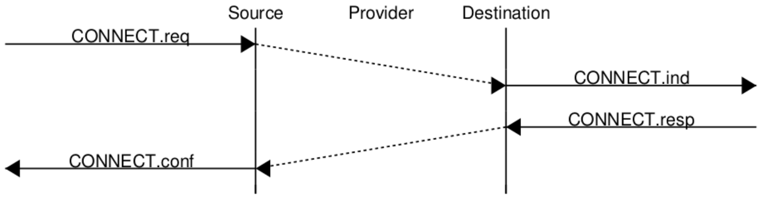
Connection-oriented Transport Service

- The service creates a connection between the two entities (normally, a client and a server)
- The connection has a **state**: some internal variables that are used to track the evolution of the connection
- As such, before sending data, there is the need to set-up the connection
- And when the connection is not needed anymore, it will be tore down.
- Multiplexing using port numbers is the same as in connectionless services

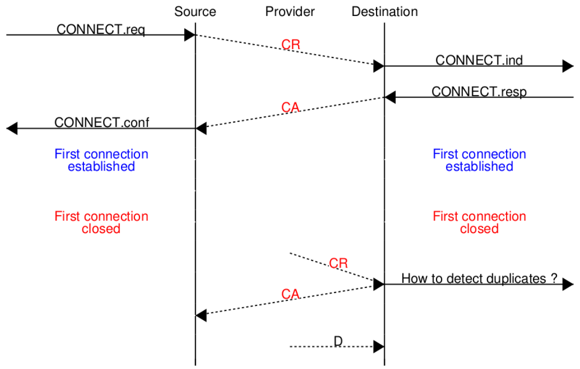
! **Connections are stateful: the endpoints maintain a state, and can packets/segments are related one with the other.**

Connection set-up

- Set-up could be as easy as the exchange of two packets
- However, we need to remember that the network layer is unreliable.



Example: packet duplication



- The connection request is sent once, but it arrives twice (it is duplicated), after some time
- How can we detect that it is duplicate and not another connection?

Unique identifiers?

- One idea would be: add to all segments one unique identifier
- If two segments with the same identifier are received, drop the last one
- However, this forces the server to maintain a queue all received identifiers.
- This is clearly unfeasible.
- We need another mean.

Maximum Segment Lifetime (MSL)

- Even if the network layer is unreliable, we need to make assumptions: packets duplication may depend on loops. Loops are temporary so we define:

> **Maximum Segment Life:** The maximum estimated time a packet (and eventually the acknowledgment related to the packet) can persist in the network.

For this specification the MSL is taken to be 2 minutes. This is an engineering choice, and may be changed if experience indicates it is desirable to do so. [RFC 793]

- So on the Internet we assume MSL < 120s, and the problem is easier to solve: how to identify a sequence number not older than 120s

A Transport Clock

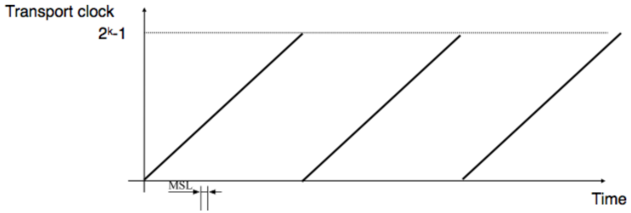
- First of all, we need a variable that increments with time.
- Every transport layer entity maintains a *transport clock*
- It is a counter that is gradually incremented with time, and deterministically at every new connection.
- The clock does not need to be synchronized among the connection endpoints

Initial Sequence Number (ISN)

- Every connection request includes an Initial Sequence Number (ISN)
- This number is a function of the timer, if the timer increases, the ISN increases, as a first simplification *assume that ISN is just the counter*
- The server returns the sequence number in the CA message
- When the client receives CA, it checks the sequence number and knows if it is related to an ongoing connection.
- In practice the ISN is used to correlate the CR with the CA

Initial Sequence Number (ISN)

- However, the ISN must be included in a header, so it size is limited
- With time, the ISN wraps



! **It is important that the wrap up timer of the counter is way larger than the MSL, let's see why.**

Relationship between ISN and MSL

Consider the following scenario:

- t_0 The client opens a connection with ISN=X,
- t_1 The connection ends
- t_2 The client opens a new one with ISN=$X + \Delta$ towards the same server. in modulo n operations however the ISN wrapped and $(X + \Delta)\%n = X$
- t_3 The client receives CA acknowledging X, is it referring to the new connection or is it a copy referring to the old one?

Initial Sequence Number (ISN)

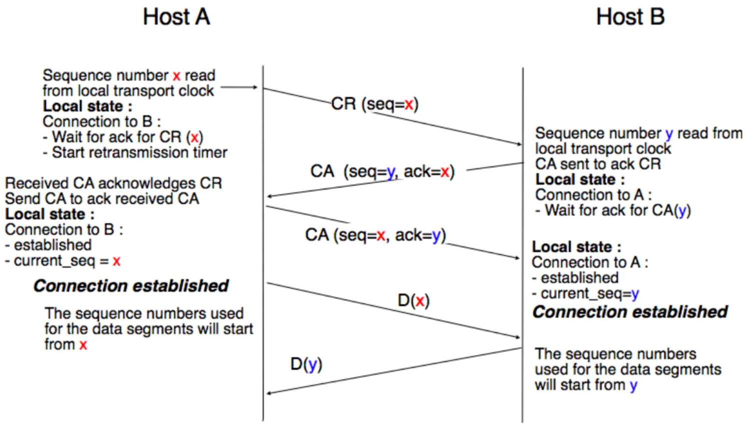
- Case 1: We can assume the wrap time of the ISN is very large compared to MSL, then, the CA refers to the new connection
- Case 2: The wrap time of the counter is comparable to MSL, then it could be that the CA is a copy of the one of the old connection.

This must not happen, even in case of a reboot, so the MSL must be way smaller than the wrap up time: **we need the ISN to wrap in a time interval much larger than MSL.**

Two-way communication

- One sequence number is actually sufficient if we want to reconstruct the stream of segments from the client to the server.
- In general we want a bi-directional connection: we need two sequence numbers, one for every communication direction.
- Also the server will generate an ISN and send it to the client.

The Three Way Handshake


Host A
Host B
Sequence number x read from local transport clock
Local state :
Connection to B :
- Wait for ack for CR (x)
- Start retransmission timer
CR (seq=x)
Sequence number y read from local transport clock
CA sent to ack CR
Local state :
Connection to A :
- Wait for ack for CA(y)
CA (seq=y, ack=x)
Received CA acknowledges CR
Send CA to ack received CA
Local state :
Connection to B :
- established
- current_seq = x
CA (seq=x, ack=y)
Local state :
Connection to A :
- established
- current_seq=y
Connection established
Connection established
D(x)
The sequence numbers used for the data segments will start from x
D(y)
The sequence numbers used for the data segments will start from y

The three-way handshake

1. the CR contains a port number and a sequence number (seq=x in the figure) whose value is extracted from the transport clock.
2. The server processes the CR segment and creates state for the connection attempt. It does not yet know whether this is a new connection attempt or a duplicated packet.
3. It returns a CA segment that contains an acknowledgment number to confirm the reception of the CR segment (ack=x) and a sequence number (seq=y) whose value is extracted from its transport clock. At this stage, the connection is not yet established.

The three-way handshake

4. The client receives the CA segment. The acknowledgment number of this segment confirms that the remote entity has correctly received the CR segment.
5. The transport connection is considered to be established by the client and the numbering of the data segments starts at sequence number x.
6. Before sending data segments, the initiating entity must acknowledge the received CA segments by sending another CA segment.
7. The server considers the transport connection to be established after having received the segment that acknowledges its CA segment (now it is sure the initial CR was not a duplicate). The numbering of the data segments sent by the remote entity starts at sequence number y.

TCP 3-way handshake

- This is the model used in the 3-way handshake by the TCP protocol, that we will see later on.
- In the book you have examples of how this prevents errors due to lost/duplicated/reordered packets
- There are some important modifications adopted in TCP, for instance the initial sequence number must not be predictable for security reasons.
- Recent implementations use random initial sequence number (see this RFC for the specs, and this code for the Linux implementation.)

The Transport Layer

↳1.3 Reliable Data Transfer

Data Transfer

- Once we have sequence numbers, we can use checksums and go-back-n/selective repeat to enforce correct delivery of data
- There are some key differences when these techniques are used at the transport layer

Sequence Numbers are for Bytes, not for Frames

- The transport layer receives a stream of bytes, and sequence numbers refer to the position of the byte in the stream
- This means that the receiver must have a buffer, even with simple go-back-n, as the quantity of data that arrives is not known in advance (i.e. there is not a frame size like at layer 2)
- As long as the buffer has space, the receiver can fill it with data coming from the lower layer.
- This happens only if `ACK`s are sent immediately
- The `Data.ind()` call is not blocking anymore: , the receiver can receive data while the application processes them.

Sliding Window Size

- However the receiving application may not be fast enough to process all the data that is sent.
- When this happens, the receiving buffer may be overloaded by data
- Since the call is not blocking, the `ACKs` will be instantaneously sent.
- There must be a way for the receiver to communicate to the sender that it must slow down the sending rate, that is not just blocking the `ACKs`.

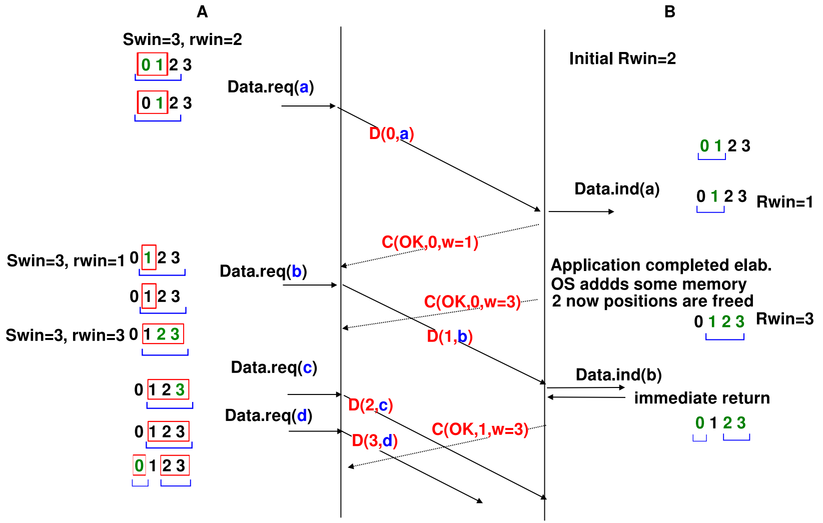
Window Advertising

- To deal with this issue, transport protocols allow the receiver to advertise the current size of its receiving window in all the acknowledgments that it sends.
- The receiving window advertised by the receiver bounds the size of the sending buffer used by the sender. This is piggybacked to the acknowledgment packets.
- In practice, the sender maintains two state variables : $swin$, the size of (the empty space in) its sending window and $rwin$, the size of (empty spaces in) the receiving window advertised by the receiver.
- At any time, the number of unacknowledged segments cannot be larger than $min(swin, rwin)$

Example of the Window Evolution

- The next picture is similar to the one in the book. It shows a simpler case of data transmission from A to B, 2 bits are used for the sequence numbers.
- The convention for the notation is:
  - The red box at the sender is the number of unacked frames. It must always be limited by $min(swin, rwin)$
  - The blue line is the sender window, whose length not change, it only shifts. We report it only when it changes.
  - Green positions are free, black ones are occupied
  - Caps names (as in **S**win) refer to the window of the host, lowercase names (as in **r**win) refer to the window communicated by the other.

Dynamic Receiving Window



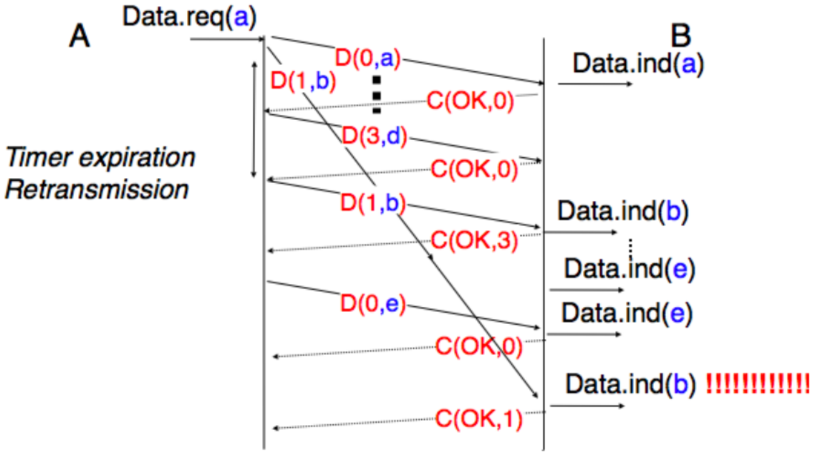
Zero size Window

- Windows can be even zero, in case the receiver is completely overloaded
- In that case the sender pauses and waits for a gratuitous ack from the receiver
- It periodically re-sends old data, to give the receiver the chance to send another ack, in case an old one gets lost.

Sequence Number and Throughput

- Consider as an example a sequence number of only 2 bits, it will wrap up when counter reaches 3
- B is using selective repeat (it accepts out-of-order data, bust still sends only cumulative ACKs)
- What happens if a packet is delayed?

Sequence Number and Throughput



Step by step

1. A sends D(0,a), this is received and acknowledged by B with C(OK,0)
2. A sends D(1,b), this is largely delayed
3. In the meantime A sends data up to D(3,d). However B did not receive D(1,b) so it sends again C(OK,0)
4. then A resends D(1,b), this time it arrives. B moves its window forward, it acks C(OK,3)

Step by step

5. The counter wraps, so A sends more data with the same sequence number: D(0,e)
6. The data is received correctly, and B acks C(OK,0).
7. At this point in time B is expecting to receive sequence number 1
8. And there the old D(1,b) is received. This is not a new packet, it is a a replica of an old one. Note that this must happen before SML seconds from point 2 (by definition of MSL)
9. However, its sequence number is in the valid range, so it will be accepted and the data stream is corrupted.

Sequence number Wrap

- We have seen that the sequence number should not repeat in a time interval lower than MSL, or the client may not distinguish two connections that he started.
- This is another reason why this should never happen, because sequence numbers acknowledge bytes, and if the wrap up time is lower than MSL, the receiver may be tricked in acknowledging an old byte.

MSL and Sequence number Wrap

- However, this implies a restriction on the maximum throughput
- Assume that the network is very fast: 30 Gb/s and sequence numbers are 32 bit numbers, each sequence number addresses a byte
- $2^{32} \simeq 4 \times 10^{9}$ so the sequence number wrap every 4GB transmitted
- This takes $\frac{2^{32} \times 8}{30 * 10^{9}} \simeq 1s << 120s = MSL$!
- With such a fast connection, the sequence numbers wrap way before MSL.
- The important parameter to monitor is $capacity * delay$

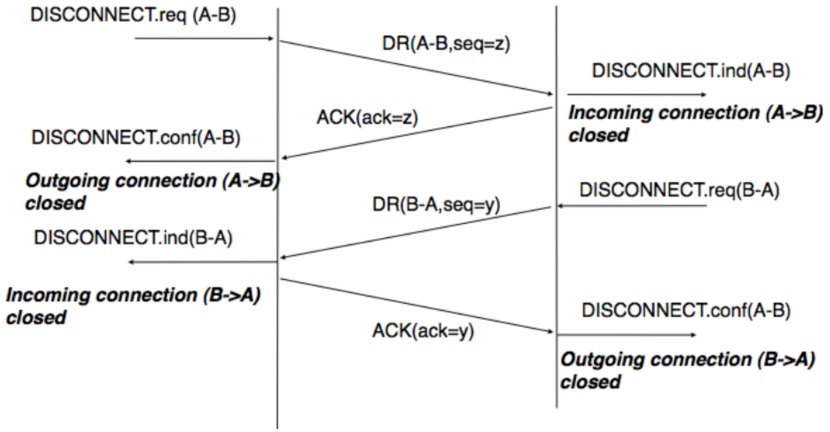
Hi $capacity * delay$

- When the capacity is very high like in the previous example, normally the packets travel in local networks, so delay is also small
- Then, the real MSL $<<$ 120s, and the problem does not happen
- However, recently we started to have networks in which the $capacity * delay$ parameter is high, that is we have gigabit-link on long paths.
- Then we had to update TCP with various modifications (See RFC 1323)

Connection Release

- Once all traffic has been sent, the connection must be closed
- Notice however that data flow in both directions, so both endpoints need to be aware that the connection was closed

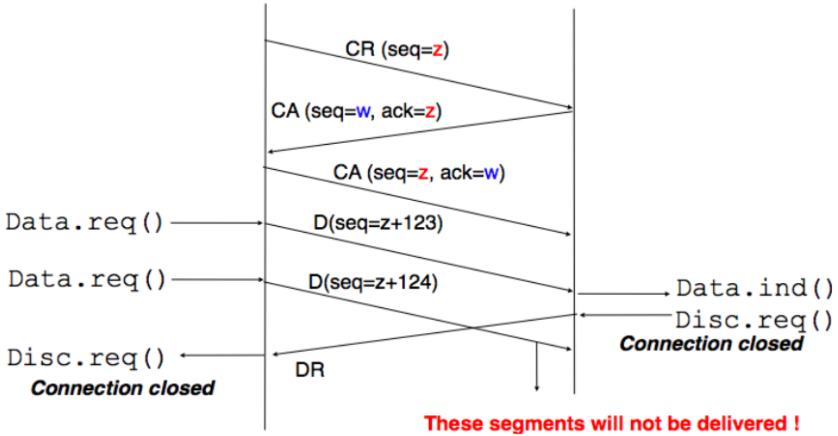
Graceful release



Abrupt Release

- There are situations in which one of the two just needs to close the connection, NOW (for instance, it is switching off)
- In this case it sends a disconnect request to the other, but this may produce impossibility of delivering data from the other side
- The only way of ensuring the delivery of data is with a graceful release.

Abrupt release



The Transport Layer

↳1.4 Exercises

Exercises

1. Assume MSL $= 120$s and seq. numbers of 32 bits, what is the maximum throughput S to ensure no errors are not possible?
2. Is it possible to have two (or more) ongoing communications between the same pair of hosts?
3. What would have more impact on the Internet, changing the transport layer or the network layer? With impact I mean, how many machines we should update to do to make it possible

1) Speed limitation

- Assume MSL $= 120$s and seq. numbers of 32 bits, what is the maximum throughput S to ensure no errors are not possible?
- We said the throughput is limited by $\frac{2^{32} * 8}{120} \simeq 286 Mb/s$
- This is a very high speed if you consider end-to-end throughput, but not impossible.
- In fact, MSL$=$120 is a very conservative assumption, in the real world this is generally much lower, so the throughput increases
- So even with higher speeds errors are rare

2) Transport Layer Ports

- Yes, it is possible. Every connection is identified by the numeric addresses of the endpoints and **two** port numbers.
- Every time a connection is started by a client, a new source port is chosen, so there can be multiple ongoing connections because the endpoints can distinguish packets based on 4 parameters (2 addresses and 2 ports)

3) Changing the Transport Layer

- The transport layer is used by two endpoints of a communication
- If these two endpoints agree on their own transport layer, the rest of the Internet does not need to be touched
- If one instead wants to modify the network layer, it needs to change all the routers of the Internet to make them compatible.

Sect. 2 The Application Layer

Some Application Constraints

- Applications are different one from the other, so it is not easy to standardize their behaviour.
- In general, they tend to use a request/response model, different for each specific application.
- An application layer protocol needs to specify the syntax and the semantics of the way information is requested, and it is provided.

Exchanging Text

- Text is generally encoded in ASCII or, when necessary in UTF-8 encoding.
- These are standard encoding of text chars, that convert a letter into a binary code of 8 bit (for ASCII) or more (for UTF-8)
- Basic applications like remote shells or even e-mail can work with the exchange of text messages
- Application level protocols need to define what are the allowed encodings

Exchanging Raw Data

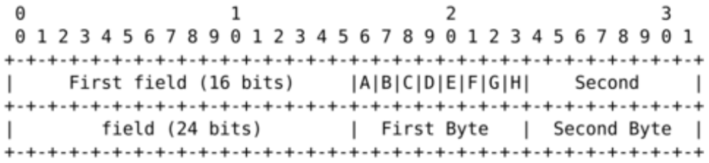
- However, binary data need to be exchanged too, for instance, in an e-mail attachment or from a web page
- Two main computer architectures exist:
  - *big-endian*: the most significant byte precedes the the least significant byte
  - *little-endian*: the least significant byte precedes the the most significant byte
- This means that the number 256, hexadecimal 0x0100 is represented as:
  - 01 00: big-endian
  - 00 01: little-endian
- nowadays all processors are little-endian or support both modes, but back at the time there was more division.
- Eventually, the Internet decided to go big-endian, so all binary data **must** be transferred in big-endian order

`htonl()` and `ntohl()`

- standard C libraries have functions that are used to convert data from the host endiannes to the Internet endianness
- These must be always used to avoid a mismatch in the representation between the sender and the receiver

Header Representation

- Often, when a protocol requires some header, the RFCs specify it using a textual notation, like the following.
- Fields are split in words of 32 bits, aligned to the byte.
- Bytes are transmitted from top to bottom and from left to right.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     First field (16 bits)     |A|B|C|D|E|F|G|H|     Second    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        field (24 bits)        |  First Byte   |  Second Byte  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Sect. 3 Naming System

Names Vs Addresses

- We mentioned that Internet hosts must have an address, that identifies the network interface of the computer
- We learned how to multiplex different applications in the same computer
- How do we address different resources in the same application?
- With names. . .

Domain Names

- Names are way more flexible and easy to use than numbers
- We can see the naming system as a system that provides the matching between a name, and a network layer address
- However assuming i know a name (like *www.unive.it*) at the network layer, **packet headers will use the numeric address**, so there must be a way to map a name to an address
- Initially this was a static list maintained by one entity, then we had to make it scale...

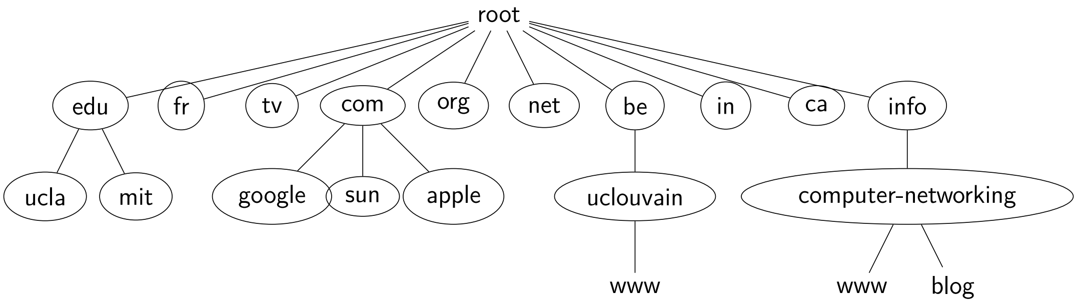
Domain names

- Domain names are strings that contain lowercase letters, dashes ('-') and dots.
- A domain name is hierarchical, with the hierarchy growing from left to right
- in *www.unive.it* the rightmost part (*.it*) is the *top-level* domain (TLD), then we have a second and a third level of the name.
- top-level domains are known, one can't simply create its own TLD.
- lower level domains are leased by entities, for instance the University of Venice owns *unive.it*

Name Servers

- Every entity that needs to be reachable configures a **reachable domain name server** with a valid numeric address
- The server answers to queries of the kind: *provide me the numeric address of the sub-domain "www.unive.it"*
- A number of *root servers* exist that are responsible for the TLDs. Their numeric address is fixed and known.
- Every domain name server needs to register itself to the server of the father domain, so the father domain knows the numeric addresses of the name servers that are responsible for its sub-domain.

DNS Hierarchy



Example

- The root server for *.it* has address X
- the university of Venice leases *unive.it*
- The network manager of UniVe set-up a name server, with numeric address Y
- there is a server whose name is *www.unive.it* that has address Z
- The network manager contacts the managers of the root server and notify that *unive.it* is managed by the server at address Y

Example

- The client with address K needs to reach *www.unive.it*
- It knows the addresses of all the root servers, so contacts server at X (responsible for the *.it* domain) and asks: who is responsible for *unive.it*?
- X responds: *Y is responsible*
- Then K contacts Y and asks: *give me the address of www.unive.it*
- Y responds: *www.unive.it is at address Z*
- Now the client can send network packets with source address K and destination address Z

DNS

- The DNS hierarchy can be as deep as one wants
- It helps in many situations that we will better explain in a dedicated lesson.