Lecture 1 - The TCP/IP protocol stack

- What's the internet?

The internet is a global network of millions of connected computing devices.

- What is a network?

A network is an infrastructure that makes possible the communication between end-systems.

There are two kind of devices we're going to find on the internet:

- **Host/End-systems** → a device on which an application is running (e.g. pc, mobile, phones…), found at the end of the network;
- **Network devices** → devices (e.g. router, switches …) connected together though **communications links** (i.e. any physical connection between devices → fiber, radio, satellite…).

The network can be divided into two parts:

- **Access** → the part of a network that gives the user access to the services (hosts + network devices allowing communication);
- **Backbone** → the core network (or backbone) is the part of a network that connects the different parts of the access network (routers → can be thought as an highway that allows maximum speed transmission of data)

While the architecture of the network is quite simple, its complexity comes from the rules that regulate communication → **protocols**, each implementing a different set of functions.

Protocols:

- Official definition → network protocols are a set of rules outlining how connected devices communicate across a network to exchange information easily and safely. Protocols serve as a common language for devices to enable communication irrespective of differences in software, hardware, or internal processes.
- Roughly speaking → **rules of communications** → define format, order of messages sent and received among network entities and actions taken on message transmission.

- What is internet standard?

The internet standard is a document describing in detail each protocol, named RFC (Request for Comments), written by the IETF (Internet Engineering Task Force) → more specifically, remember that the IETF standard documents are called RFC.
RFC means Request For comments, that describes the behavior, methods or innovations applicable to the working of the internet.

Since networks are complex (because we have many devices and many functions to be performed) we need a **layered reference model** (think about a network such as a graph with nodes and links structured in different layers) → in this model, functions are organized in a hierarchical way (in layers).

The layered reference model defines an organization of the functions implemented in the protocol and it has two main characteristics:

- **Distributed**: all the functions can be executed in all devices;
- **Modular**: it allows functions to be easily updated overtime

We will focus on the **Internet (TCP/IP) protocol stack**.

The Internet protocol stack, commonly known as TCP/IP, is a framework for organizing the set of communication protocols used in the Internet and similar computer networks (basically a family of protocols currently used in the internet).

The Internet (TCP/IP) protocol stack architecture consists of five layers:

- **Application** → The application protocols are often described in terms of the client-server model.
  They provide the internet services, such as file sharing, mail transfer, and remote terminal login.
  These protocols transmit the user's request to the transport layer.
  It deals with data in the form of **message**.
  Protocols used → FTP,SMTP,HTTP.

- **Transport** → The transport layer protocols provide the data transport services to the application layers.
  A primary function of the transport layer protocols is **multiplexing**. This term refers to directing message traffic to abstract delivery points called **ports**.
  When a host is running several application programs at once, the transport layer protocols direct incoming datagrams to the appropriate port so that they can reach the correct application program. Multiplexing allows many applications to access the internet at the same time.
  It deals with data in the form of **segments**.
  Protocol used → TCP,UDP → the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) form the bridge between the user or application programs (such as an electronic mail facility) and the lower level protocols (such as IP).

- **Network** → The network layer protocols provide a connectionless packet delivery service.
  A **packet** is the unit of transfer for a physical network. “Connectionless” describes the packet treatment: all packets are treated as separate entities.
  Network layer protocols provide efficient packet delivery by managing the addressing and routing of packets.
  Protocol used → IP, routing protocols → the Internet Protocol (IP) is the foundation of TCP/IP. It sends and receives packets of information over the physical network.
  IP calls the packet a **datagram** (It deals with data in the form of datagram).
  Each datagram includes its source and destination addresses, control information, and any actual data passed from or to the host layer. The IP datagram is the unit of transfer of an internet.

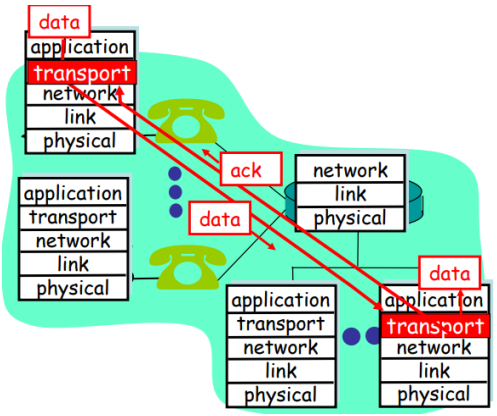
- **Link** → Data transfer between neighboring network elements.
  How do we transport data across one link?
  The data to be transported comes from the sender’s network layer.
  The data must be encapsulated (we will see that the encapsulation process will consist of adding a **header** of information) in a **frame** (it deals with data in the form of frame), converted into the appropriate electrical, wireless, or optical signal for the type of transmission media, and finally transmitted via the media.
  When the data is received, this process is reversed: the signal is decoded, and the data is decapsulated from the frame and passed to the receivers’ network layer.

A modern computer typically has three separate data link layers for Ethernet, wireless, and Bluetooth, and switches in the appropriate layer as needed.

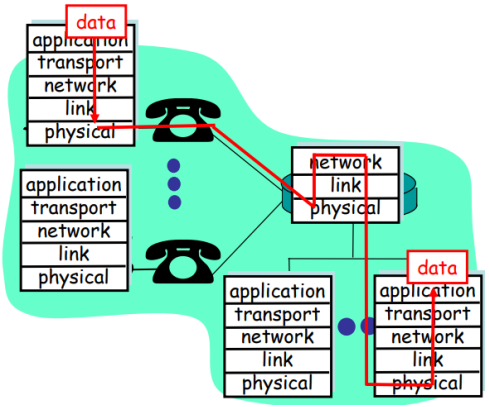
- **Physical** → the Physical Layer is the lowest layer of the TCP/IP model.
  This layer mainly handles the host to host communication in the network. It defines the transmission medium and mode of communication between two devices.
  It deals with data in the form of **bits**.

Layering communication:

- **Logical** → "logical" communication means that, although the communicating application processes are not physically connected to each other (they may in fact be on different sides of the planet, connected via numerous routers and a wide range of connection types), from the applications' point of view it is as if they were physically connected.
  Application processes use the logical communication provided by the transport layer to send messages to each other, without worrying about the details of the physical infrastructure used to transport these messages.



- **Physical** → the data to go from application to application has to go through the lower layers to be sent through the network from the source, and then goes down-up and up-down in any receiving device, with each layer running its own functions.



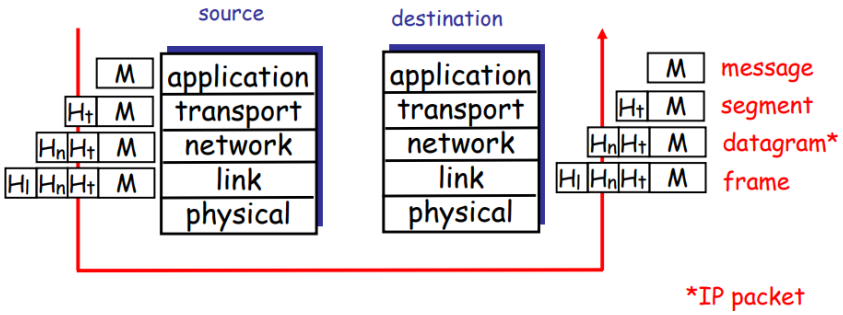
We said that each layer takes data from the previous one.

When data enters a layer, this adds an **header** of information to it (**encapsulation**) and the new data unit is passed to the next layer. At the destination device each layer only reads the header corresponding to the same layer of the source (logical communication)

Header → refers to supplemental data placed at the beginning of a block of data being stored or transmitted.
Why do we need it?

- marks the start of the data
- holds the important data such as the source and destination address, packet type and size, and related details.

Roughly speaking, we need the header because the information it contains allows the message to be read correctly by each layer.



Lecture 1 - Communication services

We have said that a network can be divided into:

- Network **edge** → application and hosts
    - End systems (hosts) → run application programs (www, email);
    - Client/server model → client host request, receives service from server (www client (browser) / server; email client/server);
    - Peer - to - Peer model → host interaction symmetric (file sharing).

- Network **core** → **routers** → a router is a device that connects two or more packet-switched networks or subnetworks. It serves two primary functions: managing traffic between these networks by forwarding data packets to their intended IP addresses, and allowing multiple devices to use the same Internet connection.

OK so in the network core there is the actual passing of data which is allowed due to the presence of routers.

So let's start here to explain what happens in the network core.

- The **traffic** of a network is the amount of information that is flowing through the network at any given time.
- A **flow** is an end-to-end data stream expressing the relationship between source host and the destination host.

As we have also said before, the network can be modeled as a graph composed of:

- **Nodes** (= routers) → are responsible for:
  - **processing capability** → defining its power to compute the information;
  - **switching** → forward data from the incoming interface ti the proper outgoing interface;
- **Links**:
  - **bandwidth** (bit/sec) → the amount of information that a link can transfer in a period of time.
  - **multiplexing** → how different flows can share the same link.

Let us briefly explain this concept of link multiplexing.

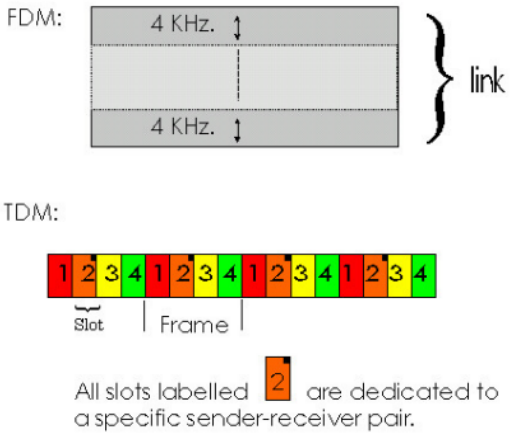
Multiplexing is a method used by networks to consolidate multiple signals into a single composite signal that is transported over a common medium.

Basically it consists of gathering data from multiple app processes contained within different layers and enveloping data with header → headers will then be needed to allow **demultiplexing** (→ demultiplexing is a process in which one input data signal is divided into various output signals and allow the delivery of received segments to the correct application layer processes).

Multiplexing can be both:
- **static**
- **dynamic**

Static multiplexing → it can be achieved in two different ways:
- Frequency Division Multiplexing (**FDM**) → link bandwidth divided into “pieces” (**channels**) and each channel is allocated to a single flow;
- Time Division Multiplexing (TDM) → divide time period into pieces and allocate to each one a specific flow, periodically transmitting them.

The **node forwarding** (= azione di instradare il pacchetto attraverso i routers) in this case is **direct**: the forwarding action is pre-defined for each flow → no processing of data at nodes. The delay is fixed.



Problem → both approaches have the constraints of knowing upfront how many flows there will be at a given time, something difficult to know in practice.

Dynamic multiplexing → Links are not divided and at each time the whole bandwidth is used by a flow that is generating data → need to manage contentions between flows.
This approach requires adding the flow into a data packet to enable streams to be identified (NB: this was not necessary with TDMas each position was allocated to a specific flow).

In this second case, the type of node forwarding is "**store and forward**" → using dynamic multiplexing, the node has to process the incoming packets to determine the outgoing path, moreover **buffering** is needed to manage contentions resulting in variable delay.

MEMO: Buffering refers to the process of temporarily storing data in memory (a buffer) before it is sent or received over a network. The buffer helps to smooth out any variations in the rate at which data can be sent or received, and ensures that the sender and receiver can work at their own pace without interruption.

In the end there are two different switching mechanisms to transfer data through the network:

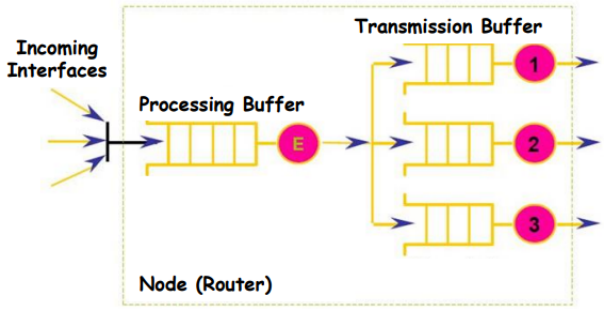
- **Circuit switching** (es: telephone):
  - Static Multiplexing → the Connection will last for the whole session
  - Direct Forwarding → no processing of the data during the forwarding
  - Reservation Procedure → you have to check whether the connection is available before!

- **Packet switching** (es: internet) → allow more users to use the network:
  - Dynamic Multiplexing → the connection will not last for the whole session, but all the bandwidth will be used for that moment.
  - Store and Forward → the node has to process all the packages, this implies some management of traffic and **congestion** (= network congestion is the reduced quality of service that occurs when a network node or link is carrying more data than it can handle → packets queue waiting for link to use) → Buffer.
  - Packet header needed → since the connection won't be stable, the communication will be divided into packets.

Let us explain this concept of **delay** in more detail.

We said that in the "store and forward" node forwarding packets move one hop at a time → transmit over a link and then wait the turn for the next link.
This can lead to traffic and congestion.
Moreover, there may be so-called resource contention → the aggregate demand for resources may exceed the available resources, so a queue will be created.

All this leads to delay.

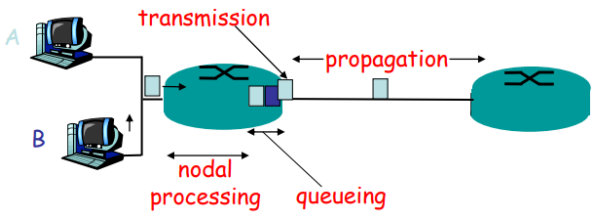


In the example shown in figure we can see a node that produces a single packet at a time and forwards it to one of the three possible interfaces.
A transmission buffer is needed because there might be what is called transmission delay of the previous packet → this is due to the fact that transmission times depend on (1) the length of the packet and (2) the link bandwidth.

In case there is no space in one of the buffers (**overload**) the packet has to be dropped.

Anyway, in total, there are four types of delay contributing to the overall delay of packet delivery:

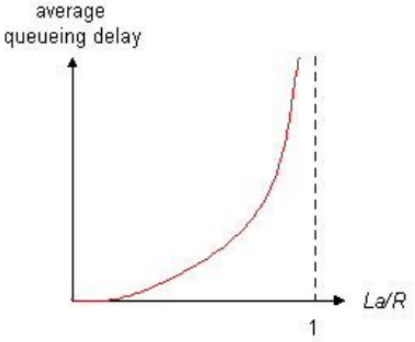
- **Nodal Processing** → it's the process done by the nodes, to read and analyze the data of the packets (tempo che ci mette il router per leggere il contenuto del pacchetto);
- **Queueing delay**→ time that a packets has to wait to be put in the output link (tempo che ci mette il pacchetto prima di essere reindirizzato correttamente nell'uscita (coda)).
- **Transmission** → it's the time that it takes for the whole packet to be transmitted into the link (not the same as Queueing)
    - Transmission delay = packet length (bits) / link bandwidth (bps) → time to send bits into link;
- **Propagation** → the time that it takes for the packet to go from the starting point of the link to the endpoint (tempo che ci mette per andare da un router a un altro).
    - Propagation delay = physical link length / propagation speed (~ 2*10^8 m/sec)



The queuing delay is related to the concept of **traffic intensity** → a measure of the average occupancy of a server or resource during a specified period of time.

- R = link bandwidth (bps)
- L = packet length (bits)
- a = average packet arrival rate
- Traffic intensity = (L*a)/R

Remember that:
- T.I. ~ 0 → average queuing delay small;
- T.I. = 1 → delays becomes large;
- T.I. > 1 → more "work" arriving that can be serviced → average delay infinite!



Lecture 2 + 3

The Application Layer (quick parenthesis)

The layer where all the applications are found is called Application layer.
Typical network app has two pieces: client and server.
The traditional paradigm is called the client-server paradigm → in this paradigm, the service provider is an application program, called the **server process**; it runs continuously, waiting for another application program, called the **client process**, to make a connection through the Internet and ask for service.

So basically what happens is:

- Client → initiates contact with server ("speaks first") and typically requests service from server;
- Server → provides requests to clients.

MEMO:

- **Process** → a program that is running within a host.
  Within the same host, two processes communicate with **interprocess communication.**
  Processes running in different hosts communicate with an **application layer protocol**.
- **Logical communication** → the different hosts will communicate and compute the process as if they were communicating directly with one another.
- **User agent** → interface between the user and the network application (ex. web: browser; email: mail reader ecc…)

The Transport Layer

The transport protocol provides a **logical end-to-end communication** between app processes running on different hosts (remember that is an end-to-end communication because the transport protocols run in end systems).

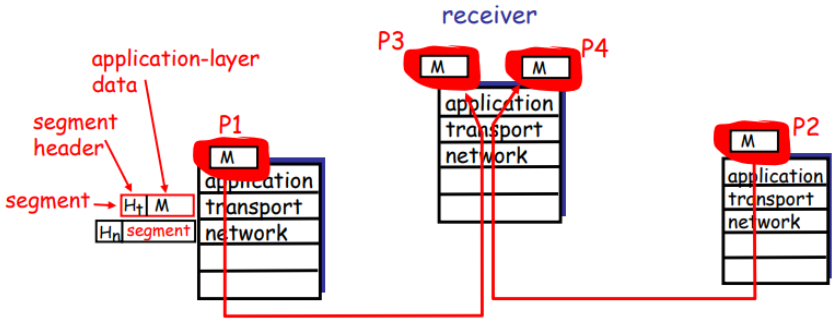
MEMO:

- Transport layer → data transfer between processes;
- Network layer → data transfer between end systems.

We have two different protocol at the transport layer:

- **UDP** → User Datagram Protocol → unreliable and unordered data delivery;
- **TCP** → Transmission Control Protocol → reliable, in order data delivery (congestion, flow control, connection setup).

Each application can choose whichever protocol it prefers BUT an application using UDP should take into account the scenarios that the protocol doesn't manage (e.g. loss of packets).
This layer has to gather the data received from different app processes and eveloping data with header (multiplexing) as well as deliver received segments coming from the network layer to the correct app layer processes (demultiplexing).



To have a correct transportation layer, the multiplexing and demultiplexing need to be done in the correct way, this means that the packets need to contain certain info useful when they have to be read.

Multiplexing/Demultiplexing is based on **sender** and **receiver port numbers** identifying the process (16 bit strings) and **IP addresses**.

For certain applications all servers must use a specific port number to allow clients to know this upfront (e.g. HTTP → port 80), in particular:

- 0 - 1023 → well-known apps
- 1024 - 65535 → available

Remember also that communication can be **connection oriented** or **connectionless** depending on whether the client sends the message with or without establishing a connection with the server first.

Now that we understand what the transport layer looks like, let's go and analyze the two types of protocols that make up this layer.

UDP → a very simple and fast protocol with the following characteristics:

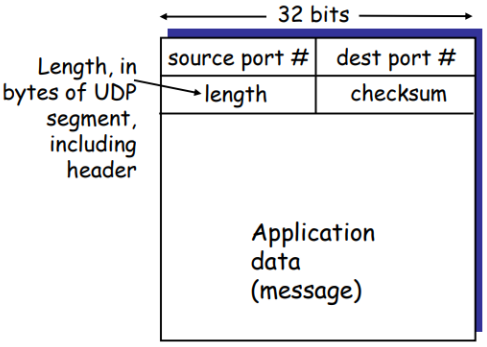
- "Best effort" service → it sacrifice congestion (no congestion control = the segment transfer will go as fast as it can) and any kind of control for a fastest procedure → packet can be lost + unordered delivery → unreliable;

- Connectionless → no connection establishment = no "handshaking" between server and receiver, which means that it doesn't need to let the other part of the communication know that there are some packets coming.
  Each packet (segment) is independent, which means that the packets can arrive in a different order than the one they had when originally "shipped".

- Small segments header → UDP headers are limited to 8 bytes in size

Can we do something about this unreliability?
The protocol itself can't be touched, but we can do something about it in the application layer → adding a **timeout** can be one solution to add some reliability.

Checksum procedure → used to detect "errors" in transmitted segments → is computed by taking the sum of the other 16 bits strings in the UDP format.
The receiver re-computes the checksum and compares it with the one received to check if the received message is complete.

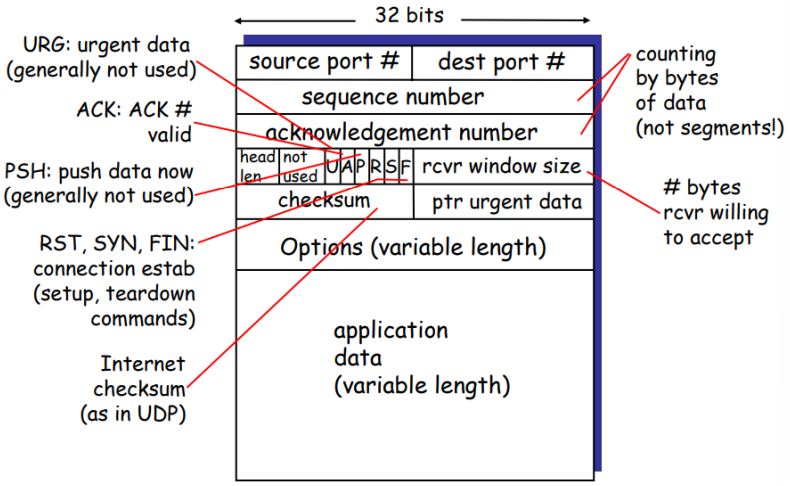


UDP segment format

When is the UDP protocol used?
It is often used for streaming media because of its loss tolerance and rate sensitivity.

TCP → the goal is to build a **reliable** communication between processes (notice that both sender and receiver play each other's role since the communication is bidirectional).

The TCP segment structure is something like:



The TCP, in general, is a reliable data transfer → the protocol itself is reliable.
However, there might be some unreliability in other parts, for example the channel for communication might be unreliable → this will determine:

- Errors
- Delays
- Packet loss

These 3 characteristics of an unreliable channel will determine complexity of reliable data transfer protocol (rdt).

Question: how does the sender know that an error occurs?

Answer: let's introduce the concept of **acknowledgement** (**ACK**).

Definition: An acknowledgment refers to a message sent by the receiver to the sender indicating that a segment of data has been received successfully.

So, the receiver sends back either an ack (receiver explicitly tells sender that pkt received) or a negative acknowledgement (NACK → receiver explicitly tells sender that pkt had errors).

Notice the receiver has no way to know if its last (N)ACK has been received correctly by the sender.

The ack is the main function added for reliability, but we still have few cases we need to handle:

- **corrupted** (N)ACK → the sender will just retransmit the packet BUT remember that the sender doesn't know what happened at the receiver so this retransmission might cause duplicate packets sent;
- handling **duplicates** → senders add a **sequence number** to each packet so that the receiver can drop duplicate packets;
- **packet loss** → we know that as 'safety mechanisms' we already have checksum, seq.numbers, ACKs and retransmission BUT it is still not enough because a packet loss might happen.
  In this case the approach is: the senders waits a "reasonable" amount of time for ACK and then we can be faced with two situations:
  - retransmission if no ACK received in time;
  - if pkt (or ACK) just delayed (not lost) → there will however be a retransmission that will be duplicate but the use of seq.number already handle this (so the receiver must specify the seq.number of the pkt being ACKed).

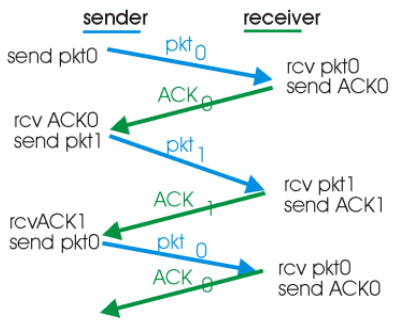
This first kind of **flow control** for packet loss is called "**stop and wait"** → the sender sends one packet then waits for the receiver's response. This process does not require a buffer at the sender or at the receiver.

MEMO:

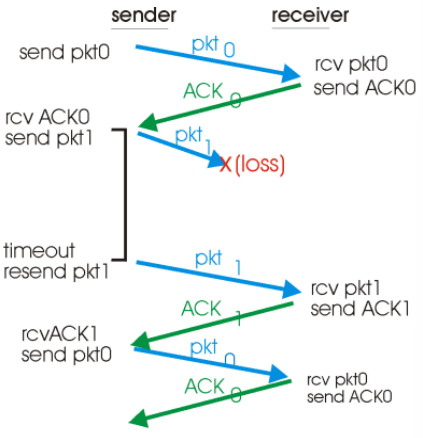
- The ***flow control*** is another way of ensuring reliability → it relates the rate at which data is transmitted between the sender and the receiver to prevent the receiver from being overwhelmed with too much data.
- We can simplify the model by dropping the NACK → rather than sending a NACK for packet *n* the receiver sends an ACK for packet *n-1*.

RECAP: Basically, the ack confirms that the data has been received without errors; if it is not received within a certain time period then the lost packet will be sent again.

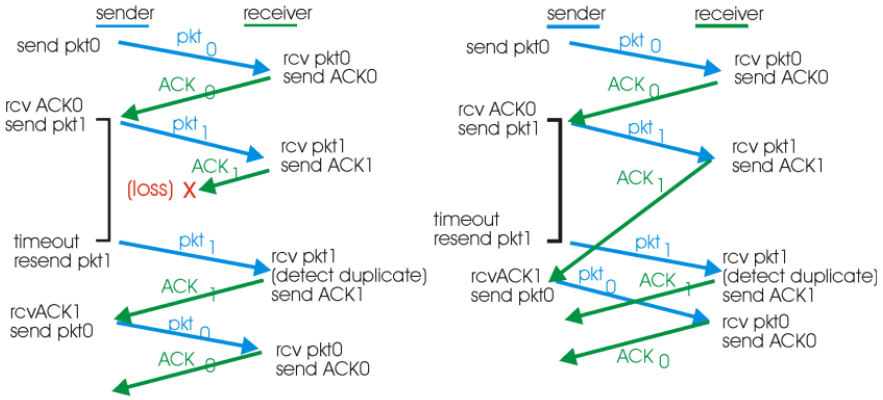
We can be faced with four possible scenarios:



(a) operation with no loss



(b) lost packet



We have a <u>second</u> kind of flow control which evolves the **pipelined protocols** (*sliding window protocols*).

- **Pipelining** → is a technique used in network protocols that **allows multiple packets** or segments to be transmitted at the same time without single-time acknowledgment.

Basically, we now want to relax the model to allow the sender to transmit a sequence of packets (pipelining) without waiting for each to be ACKed.

To handle this scenario there are two kind of pipelined protocols where the main difference from one another is how they handle the retransmission of packet loss:

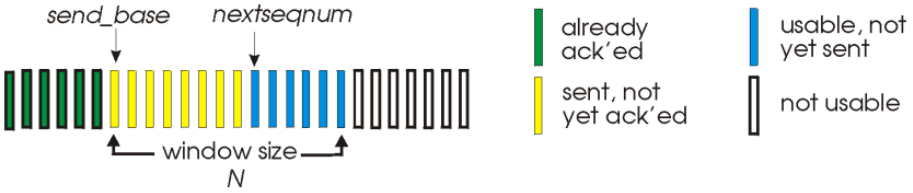
- **go-Back-N** protocols → cumulative ACK, discard out-of-order packets;
- **selective repeat** protocol → selective ACK, accept out-of-order packets → a buffer is needed to control the flow of pkts.

Let us analyze them separately.

<u>GO-BACK-N</u> → We define a **transmission window** of a certain size of packets that can be sent without waiting for an ACK. This window slides over the sequence of packets to be sent that moves as soon as ACK is received.
If a pkt is lost or corrupted, all the pkts that were sent after that need to be retransmitted.

<u>Sender</u>:

- **cumulative ACK** → if an ACk with a sequence number *n* is received then the window can slide starting at packet *n+1* .
- if a **timeout** happens then all pkts within the window are retransmitted.



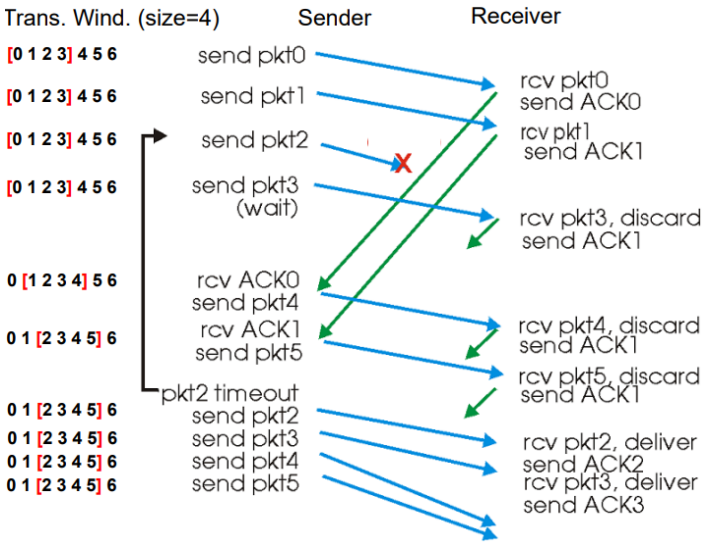
<u>Receiver</u>:

- ACK-only → always send ACK for correctly-received packet with higher-in-order sequence number;

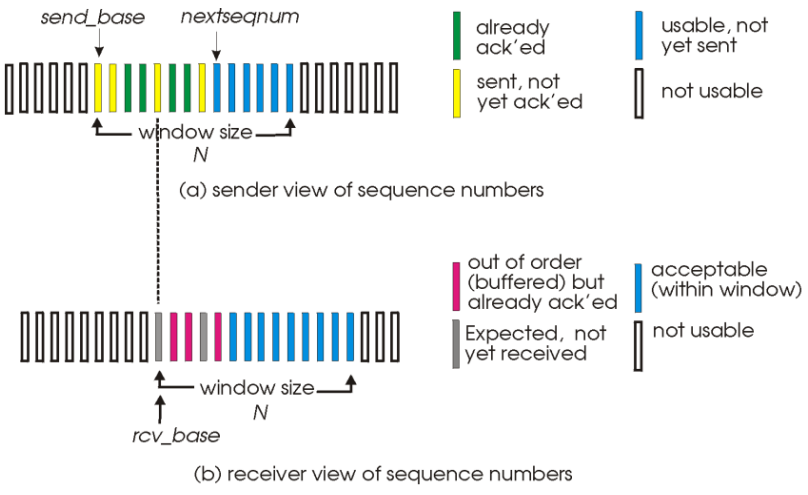
- Need only to remember the expected sequence number;
- Discard out-of-order packet and ACK packet with highest in-order sequence
- number → because there's not a receiver buffer;
- Problem → may be generate duplicate ACKs

Go-Back-N in action:

- When receiver gets pkt3 it sends back ACK1 (remember this is a equivalent to a NACK2);
- When sender receives ACK0 (and 1) slides window forward;
- The sender keeps sending segments in the window without waiting for an ACK;
- Upon timeout for pkt2 it sends again all packets in the window.



SELECTIVE REPEAT → What basically changes from go back n is that selective repeat only resends the lost packet, not all those in the sliding window.



Receiver:

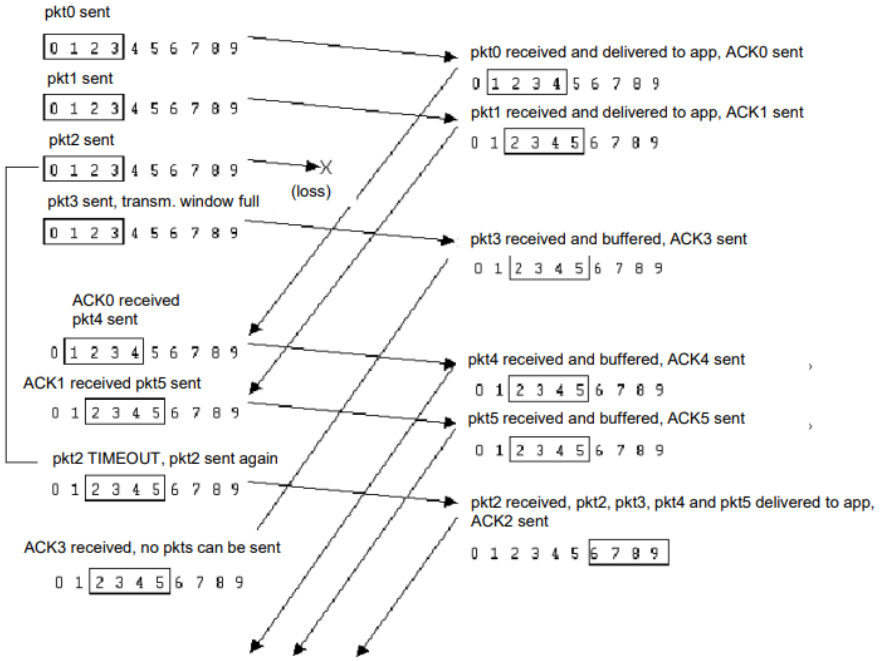
- Individually acknowledges all correctly received pkts;
- Buffers packets, as needed, to provide in-order delivery for the upper layer.

Sender:

- Resend only the pkts for which ACK was not received.

Selective repeat in action:

- Sender and receiver maintain a different window;
- Receiver buffers the packet 4 and 5 since 2 has been lost;
- Upon timeout for pkt 2 only that is resent;
- No more packets can be sent until ACK2 is received (the window doesn't slide forward)



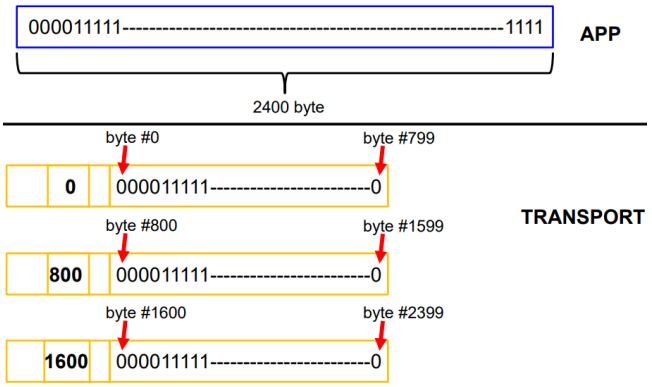
MEMO: Transmission window size can't be greater than **half** of the sequence numbers space cardinality.
If the window size is too small it's possible the receiver mistakes a duplicate pkt for a new one (see picture).

Sequence numbers and ACKs in TCP protocol

A couple of important differences of the TCP wrt to the solution of the previous paragraph are:

- It assigns sequence numbers using the number of the byte from which the segment starts, in the picture on the right would be 0, 800, 1600 → in TCP we're counting bytes, not segments: if we had counted the segments, the seq.number for the second segment would have been 1 and not 800.

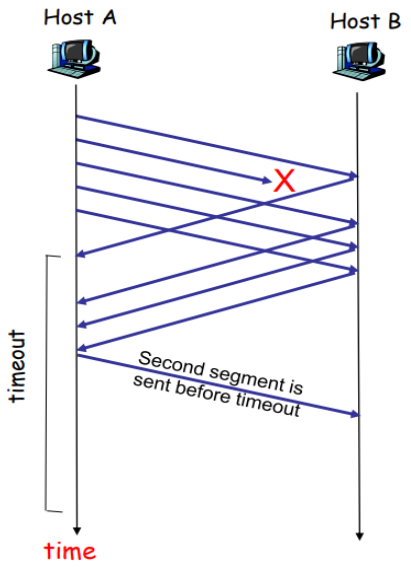
- When sending back an ACK it sends the number of the next segment it expects (i.e. the first byte number of the next segment), in the picture below upon receiving segment 0 it would send ACK 800.



MEMO: Generally TCP does not define a rule for out-of-order packets, but in the actual implementation packets are stored waiting for the missing ones.

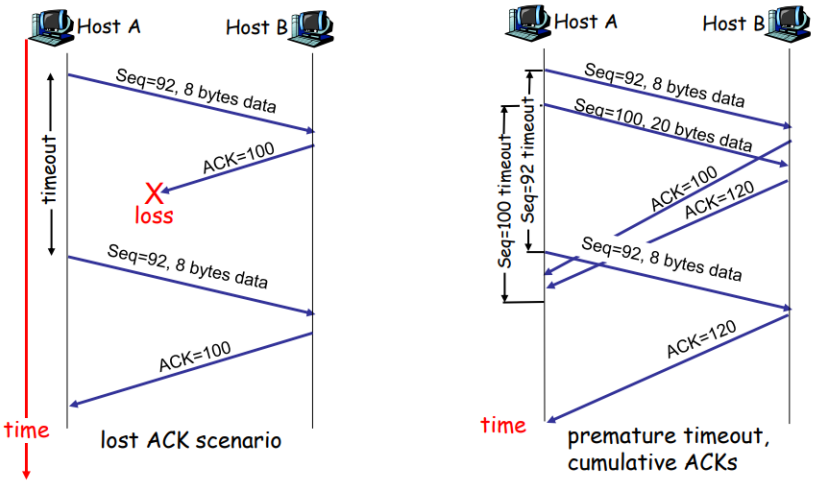
Retransmission scenarios in TCP → 3 possible retransmission scenarios:

1) **Duplicates ACKs** → In case the sender receives three consecutive duplicates ACK it re-sends the missing packet before the timeout.
   When Host B receives the missing PKT 2 since it stored the PKT 3-4-5 it will send back ACK 6.
   Note: in practice these numbers will actually be byte numbers rather than consecutive integers → we're counting not the segments but the bytes.



2) **Lost ACK** → When an ACK is lost, the sender waits the timeout and then sends again the packet assuming it was lost while the receiver will send back the same ACK since it already received that packet.

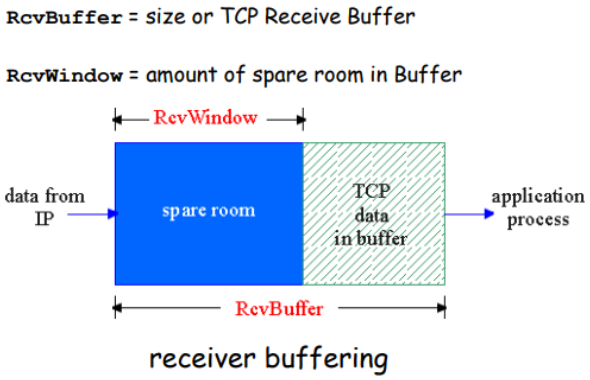
3) **Premature timeout** (cumulative ACKs) → When the timeout expires before the ACK is received (timeout too short) then the packet is sent again and the receiver will just send back the same ACK as it already received the packet.



TCP Flow Control → The goal of this functionality is to avoid sending too much data too fast to the destination.
The destination has a buffer to store the receiving data, but if its reading speed is slower than the arrival speed of data there will be an overflow of data and hence discarded.

Basically → It adjusts the rate of data sent so that the receiver is not saturated with data at that time.

We must define another window that allows flow control → **receiving window**.



The way in which this overflow is avoided:

- Receiver: informs the sender of (dynamically changing) amount of free buffer space (RcvWindow field of TCP segment)
- Sender: keeps amount of transmitted but not ACKed data less than most recently RcvWindow.

Principles of Congestion Control → informally: “too many sources sending too much data too fast for the network to handle”.

Remember that is different from flow control → Flow Control and Congestion Control are traffic controlling methods for different situations.

- **Flow** control → technique used to regulate the flow of data between different nodes in a network. It ensures that a sender does not overwhelm a receiver with too much data too quickly. The goal of flow control is to prevent buffer overflow, which can lead to dropped packets and poor network performance.
- **Congestion** control → is a technique used to prevent congestion in a network. Congestion occurs when too much data is being sent over a network, and the network becomes overloaded, leading to dropped packets (lost pkts) and poor network performance (long delays).

We need a way to understand implicitly from events if the network is congested → TCP detects congestion when it starts to see pkts losses (buffer overflow at routers) and long delays (queueing in routers buffers) → it implies that a reduction of transmission rate is needed.

How to reduce this rate? **Congestion window** → transmission window related to the congested scenario.

How do the TCP senders determine then sending rates such that they don’t congest the networks but at the same time make use of all the available bandwidth?

In order to use the bandwidth as much as possible, the Congwin is adjusted dynamically → **bandwidth probing** → what we would ideally like is to transmit as fast as possible (congwin as large as possible) without loss.
So, what we do is to:

- probing → try to transmit as fast as possible without loss;
- increasing → increase congwin until loss (congestion);
- decreasing → decrease the congwin and try probing again.

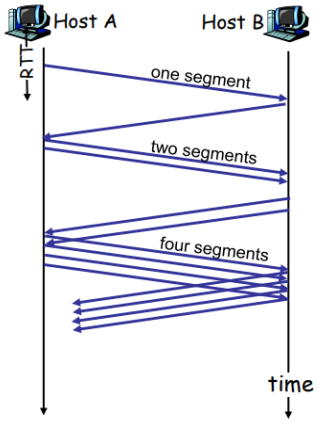
We will see two congestion control procedures: **TCP Tahoe** and **TCP Reno**, which is nothing more than an improved version of TCP Tahoe.

The **TCP Tahoe** congestion procedure has 2 phases:

1) Slow-Start
2) Congestion avoidance

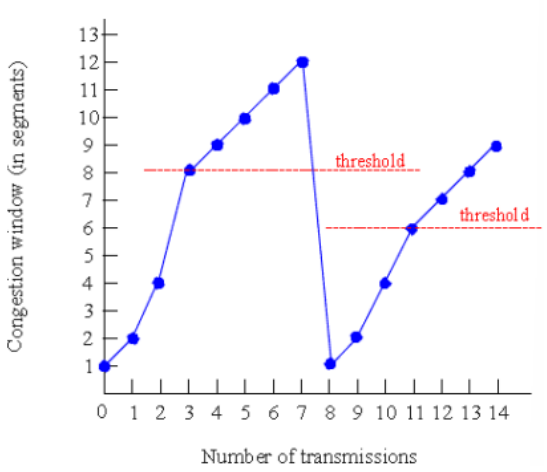
MEMO: by **loss event** we mean either a timeout (an ACK took too long to arrive) or three duplicate ACKs because if I receive three duplicate ACKs I automatically consider that one packet has been lost.

1) **SLOW-START** → exponential increase
   initialize Congwin = 1
   for (each segment ACKed):
   Congwin = 2*Congwin
   until (loss event OR Congwin > threshold



2) **CONGESTION AVOIDANCE** → once the threshold is reached, we move from the slow-start phase to the congestion avoidance phase → linear increase.

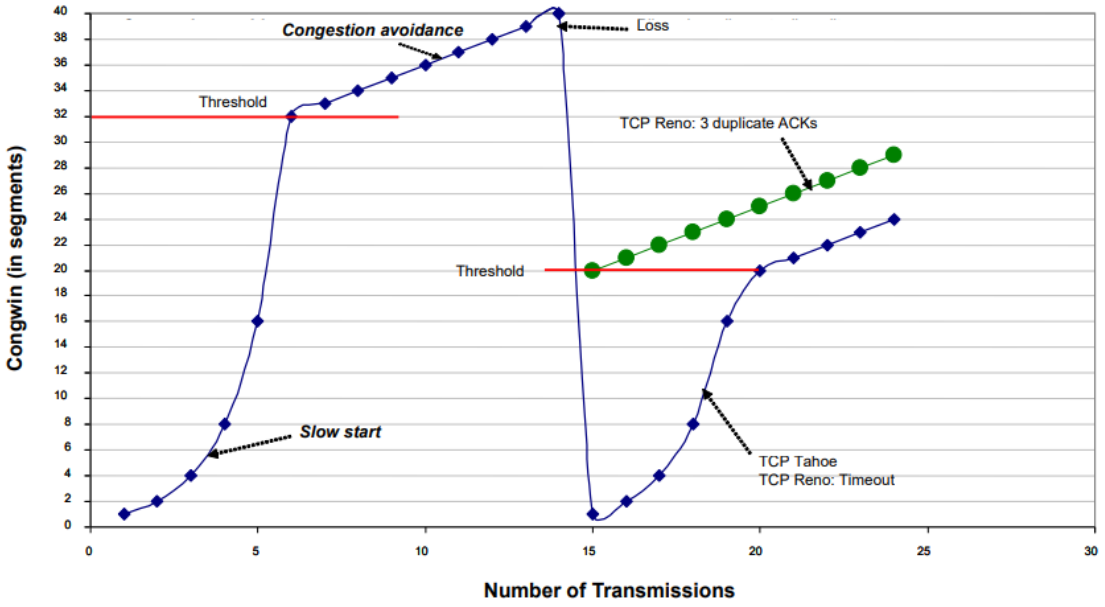
   /* slow-start is over + congwin > threshold */
   Until (loss event) {
   Congwin ++ }
   threshold = Congwin / 2
   Congwin = 1
   perform SlowStart



The **TCP Reno** is nothing more than the improved (in some respects) TCP Tahoe procedure.

In fact, if the loss event is a timeout, the procedure remains exactly the same as in TCP Tahoe, whereas if the loss event is the three duplicate ACKs, things change slightly.

In this second case (3 ACKs) what happens is simply that you start from the threshold directly with congestion avoidance.



Throughout the communication process we have two different windows:

- **RcvWindow** → computed by the flow control procedure;
- **Congwin** → computed by the congestion control procedure.

Remember that **Transmission Window** = min [RcvWin; CongWin]

TCP Connection Management

TCP senders and receivers establish a connection before exchanging data segments and initialize TCP variables (seq.numbers, buffer, flow control info such as RcvWin ecc…)

In particular, there are three “handshake steps”:

1) client end system sends TCP SYN control segment to server, specifying initial seq.number;
2) server end system receives SYN, replies with SYNACK control segment: ACKs received SYN + allocates buffers + specifies server (specifies receiver initial seq.number)
3) client replies with ACK → possible to start sending data here.

To close the connection a segment is sent using the FIN (F) field in the TCP segment.

LECTURE 4 - The Network Layer

The network layer plays a critical role in packers transportation → it ensures that data pkts are delivered efficiently and reliably from sending to receiving hosts.

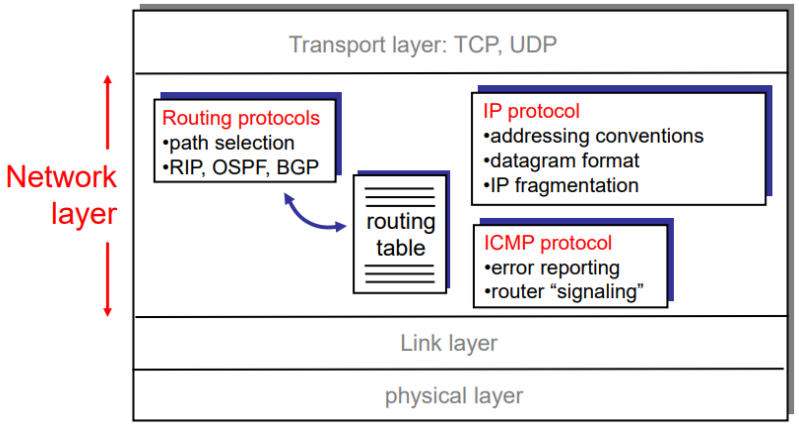
It implements 3 important functions:

- **Logical addressing** → the network layer assigns a unique IP address to each device on a network; these addresses are used to identify the source and destination of data pkts.
  This is the most important property → every device in the network must have a unique address associated with it so that packets can be correctly routed to their destination.

- **Path determination (routing)** → process of selecting the best path for packages to get from the sender to the receiver (performed by routing algorithms).

- **Fragmentation** → splitting network packets (and successive merging once these fragments arrive at destination) into smaller packets, when these are too large to be transmitted over the network efficiently.
  This process may be necessary due to constraints imposed by the link layer.

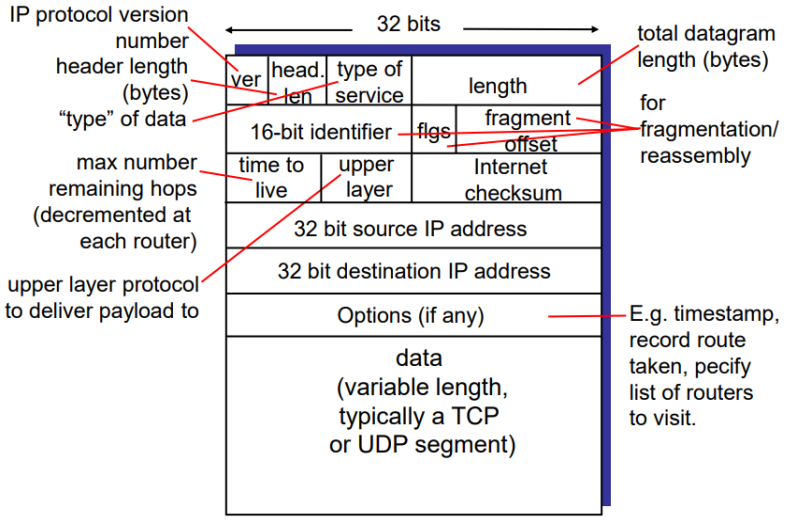
This layer has some routing protocols, an ICMP protocol and an IP protocol (which is the most important).



- Routing protocols → set of rules that allow network devices to exchange information on network topology;

- Routing table → table that is used by network devices (such as routers) to determine the best route through which to route data packets.
  In a routing table, each row represents a possible network destination and includes information such as the network address, subnet mask and default gateway address.
  Based on this information, the network device can determine which network interface to use to send data packets to their destination.
- ICMP protocol → signaling protocol that is used to report errors and to manage communication between network devices.
- IP protocol → protocollo fondamentale su cui si basa Internet e definisce il formato dei pacchetti di dati utilizzati per trasmettere informazioni attraverso la rete.

The IP datagram format → IPv4 header



MEMO: What is the IPv4 address?
IPv4 (Internet Protocol version 4) is the standard address format that allows all computers on the Internet to communicate with each other.
An IPv4 address is written as a 32-bit string of digits and consists of four numbers, each between 0 and 255, separated by full stops.

The IPv4 header shown above was defined ~40 years ago.
The main drawback is the number of bits used to define an IP which led to the introduction of the IPv6, but is still the most used due to the cost of updating the network devices.

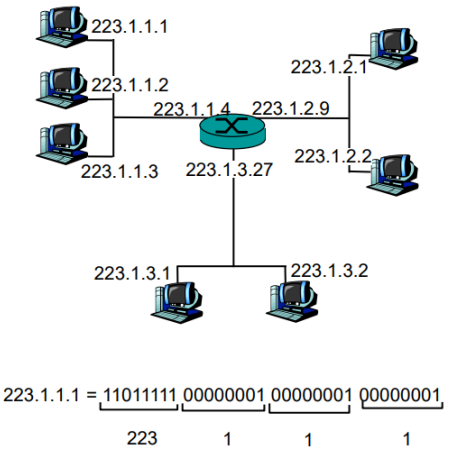
Fields:
- ver (= version field) → represent the IP protocol version (IPv4 or IPv6)
- 16-bit identifier (and in general all the second row) → used to manage fragmentation;
- Time to live → used to avoid a packet remains in the network an undefined amount of time (decreased by 1 each time it passes through a router) → related to ICMP protocols;
- Upper layer → defines if the data we have is a TCP or UDP segment;
- 32-bit Destination IP address → when a device sends data over a network, it includes the IP address of the destination device in the data packet, which is used by routers and other networking devices to determine the best path for the packet to take through the network.
- Options (if any) → e.g. timestamp, record route taken, specify list of routers to visit.

1) IP addressing

An IP address is a **unique 32-bit** number (**identifier**) assigned to each device (host, router, interface) on a network that uses the IP protocol.

Since each IP address is 32-bit long, there are a total of 2^{32} possible IP addresses.

Let's start by analyzing this image to arrive at a definition of the IP address concept.



In the picture above we have:

- on the left → three directly connected hosts which are connected with one interface of the router;
- on the right → two directly connected hosts which are connected with another one interface of the router;
- at the bottom → two directly connected hosts which are connected with a third interface of the router.

Imagine as if we had three different rooms each connected with a router interface.

Now, the set of devices physically connected between them + the router interface to which they are connected form a **subnetwork.**
Basically, we say that two hosts are in the same subnetwork if they can communicate (i.e. send packets to each other) without passing through a router.

How are hosts and routers connected into the network? → **Interface** → physical connection between a host, or a router, and a subnetwork.

Remember that:

- Routers typically have multiple interfaces;
- Host may have multiple interfaces (and a single link to send datagram);
- IP addresses are associated with the interface (not the host/router) → because routers and hosts could have more than one interface (es: In the case of the figure above, each host has only one interface while the router has three).

To be more precise → we know that each host has to be identified by a number, called IP address BUT in reality, the IP address is not associated with the host but with the interface → we assign an IP

address to the interface of the hosts and of the routers (in fact, in the case of the example pictured above, the router has 3 IP addresses).

Another thing to add: we know that the ip address is a 32 bit number made up of 0 1 but there is also a 'human' representation of this → we take the 32 bits and divide them into blocks of 8 bits (or 1 byte) and make a kind of translation from binary representation to a decimal representation (look at the bottom of the photo as an example).

IPs are organized **hierarchically** → two interfaces directly connected have to share part of their IP.

In a network from the IP perspective:

- device interfaces share part of the IP address;
- devices can physically reach each other without a router.

RECAP: two rules for generating an ip address:

1) is unique;
2) must respect a hierarchical structure.

The IP address consists of two parts:

- The **network** part → consists of the **high-order bits** of the IP address and identifies the subnetwork to which the device belongs. In other words, all devices within the same subnetwork share the same network part bits.

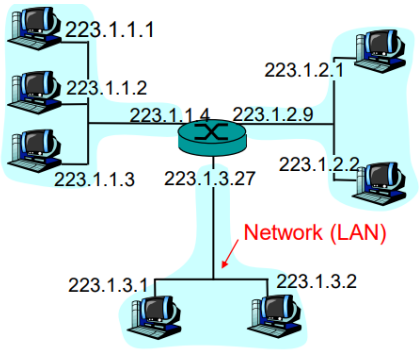
- The **host** part → consists of the **lower order bits** of the IP address and identifies a specific device within the subnetwork.

The length of these two parts can vary depending on the class of IP address being used.

Let us take the same figure as an example and see how the ip addresses were constructed.

The first 24 bits were used to identify the network part (the first 3 bytes) and the last 8 bits (the last byte) to identify the hosts inside the network.

- Left subnetwork:
  - 223.1.1 → network part
  - 1,2,3 for the host identification
- Right subnetwork:
  - 223.1.2 → network part
  - 1,2,9 → host part
- Bottom subnetwork:
  - 223.1.3 → network part
  - 1,2,27 → host part



network consisting of 3 IP networks
(for IP addresses starting with 223,
first 24 bits are network address)

All this to say that ip addresses cannot be assigned at random.

Question: How do we know which part of the ip address relates to network identifying and which to host identifying?

Answer → if you only look at the ip address it is not something you can know: you need one more piece of information to answer this question.
You need an extra identifier, called netmask, which gives us a simple piece of information: how many bits are in the network part?

How is the netmask composed?

Netmask → is a **string of 32 bits** used to identify the network part of an IP address (in binary → 1's in the network part and 0's for the host part).

IP_Address= Net_Id . Host_Id
Netmask = 1111….11 . 00….000

Thus, in association with each ip address, we also have the netmask identifier.

Obviously, the number of bits dedicated to the network part can be chosen independently.

What is the criterion for the choice? It depends on how big your network is, how many IP addresses you need → few addresses = small host part and vice versa .
It also defines the number of IP addresses available for a given network.

As done for ip addresses, the netmask can also be translated according to the decimal system.

- Example: first 24 bits are network address

Netmask = 1111111.11111111.11111111.00000000
255.255.255.0
/24
n° of IP addresses = 2^8 = 256

- The notation /24 indicates that the first 24 bits of the IP address are the network address.

- There are 8 bits for the host address, which means that there are 2^8 = 256 IP addresses available.
  If 256 Ip addresses were not enough I can reduce the network part.

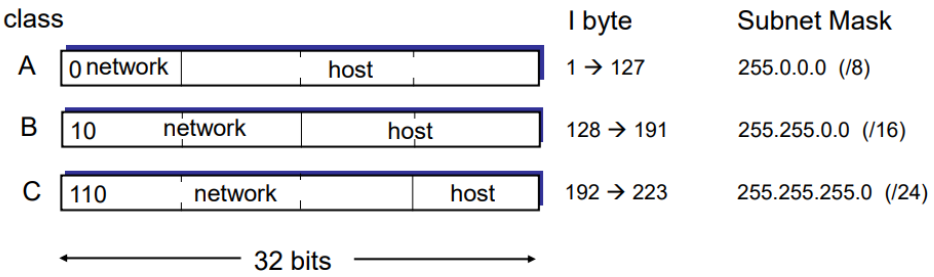
However, 256 is the maximum number of IP addresses but actually I cannot use them all → in fact, 2 of these 256 are reserved for specific purposes but we will see in a moment.

Classfull addressing

We open and close this chapter on classfull addressing only to explain how it worked before and how it works now with regard to the definition of the netmask in IP addresses.

When ip addresses were invented, many years ago, internet consisted of only a few devices, so IP addresses consisting of 32 bits were considered excessively large → there were many more IP addresses than were actually needed (now not more true and that is why IPv6 was introduced which consists of 128 bits).

However, when IPv4 was invented, there were precise rules for defining the network part → only 3 possible sizes for this part → 8, 16, 24.
No other options were possible → ex: 8 bits → class A IP addresses and so on.



What was the idea behind all these rules?

Basically, they had considered that all networks in the world could be divided into these three classes according to their size.

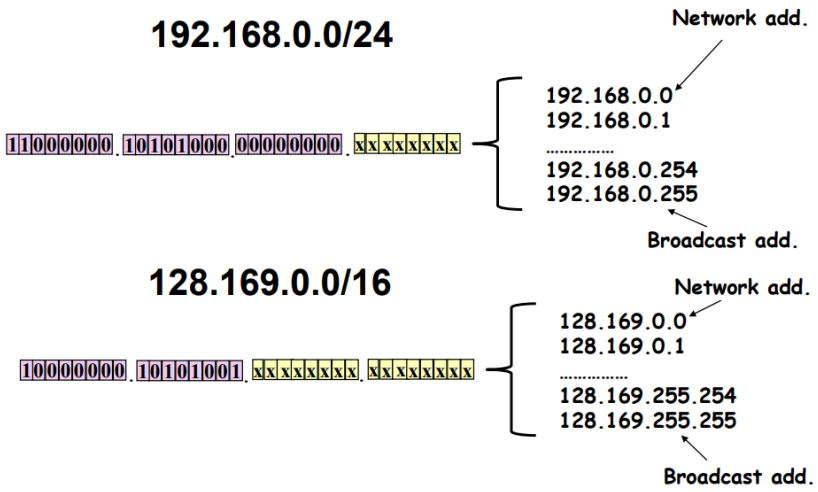
This was not a very flexible solution, but it was a very simple solution, especially from a certain point of view → they had decided to identify the netmask information directly within the IP address.

How did it work?

If an IP address had as its first bit a number between 0-27 then it belonged to the first class and so on for the others.
This basically meant that it was not necessary to associate the netmask with all IP addresses → what they did was to "code" the information of the netmask inside the IP address → 3 subnet mask sizes.

In any case, this subdivision is no longer used (the reason we introduced it is that the terminology continues to be used today).

Example



In the example above, we defined 24 bits for the network part and 8 bits for the host part: these 8 bits can vary and there are 256 possible combinations.
But, as we said before, we had to explain why we cannot actually use all 256 possible addresses but only 254 → in particular we can use all of them except the first and last.

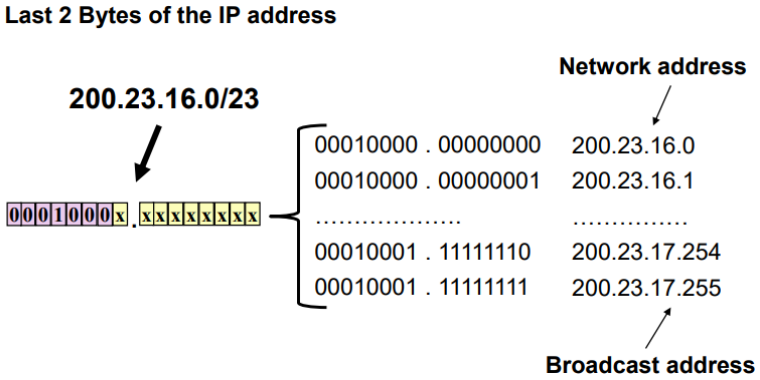
In the example:

- 192.168.0.0 → the first is also known as **network address** → all 0's in the host part → used to identify not only a single host but the whole subnetwork.
- 192.168.0.255 → the last one → all 1's in the host part → also known as **broadcast address** → an ip address used when I want to send a pkt in my network and I want that my pkt is received from all the devices in my network → we'll better understand this when we talk about routers.

So, we said that 'classfull addressing' is now no longer in use, hence how do things work now?

To overcome the limitations of classfull addressing, **CIDR** (Classless InterDomain Routing) was introduced → consists of using address blocks of varying sizes (any number between 0 and 30 may be introduced to indicate the network part) but the netmask information must always be specified.

Example:



- In that case, how many free IP addresses do I have? 2^9

- We also note that the third block of the last addresses is 17 → comes from putting a 1 as the last digit of the third block (the pink one).

IP addressing and host configuration

How, in practice, are the IP addresses assigned to the Internet users?

To obtain an IP address block, an **ISP** (Internet Service Provider) must apply to **ICANN** (Internet Corporation for Assigned Names and Numbers).
ICANN is the international organization that manages IP addresses and domain names on the Internet and is responsible for assigning IP addresses to network operators (Wind, Vodafone etc…)

Now that we have the unique IP address, we need to figure out how to divide it among the various hosts in the network.

To better understand this, we will take a very simple example.

We are Sapienza's network administrator and our ISP provides us with a block of IP addresses.
Once we have been given the IP address we have to assign a specific and unique ip add to each host in our network.
Now, if I know I have 100 PCs, it is easy to assign them a unique ip address.
The problem arises when, for example, a student comes, connects his device, stays connected for an hour then leaves and in his place another student comes and connects another device.

We need to have a very smart way to assign unique IP addresses even when hosts change so frequently because doing it by hand would be impossible → there is a dedicated protocol → **DHCP** → assigns IP addresses in a dynamic way.

How does it work? Let us always consider the case of the Sapienza network.

In the Sapienza network there is a DHCP server which is responsible for assigning IP addresses to hosts.

The Sapienza network administrator has entered the block of IP addresses assigned to us by the ISP into that server and from now on, dynamically, the server will temporarily assign IP addresses to hosts.

What happens in practice?

- DHCP discovery phase → the first time you click on a new wifi your device doesn't know about it so it looks for the DHCP server on that network.
  In particular, your device sends a broadcast message to find out who the DHCP server is on the network you are on;
- DHCP offer phase → the DHCP server replies by sending a so called DHCP offer in which it offers an IP address to the host;
- DHCP request phase → the host, after receiving the offer, responds with a request saying "ok I want the IP address";
- DHCP ack phase → finally after receiving the request message the server replies with an ack and says "ok from this point onwards for a certain time" you are using this IP address.

Thanks to this protocol, this whole procedure is hidden from users who may not know anything about IP addresses.

MEMO: it may happen that there is more than one server and that when you send the broadcast you receive more than one offer so then the device chooses one of them.

Private IP addresses + NAT Router

The unicity of the ip address is a very tight constraint for the evolution of the internet.

Problem → the maximum number we can have is 2^8 which is a very low value for the amount of devices that the internet now counts

Solutions:

- Make ip addresses bigger (ipv6) → not immediately implementable because changing the protocol version is not exactly straightforward;
- allow there to be no single IP address → make it possible for 2 different hosts to use the same IP addresses while trying to resolve any ambiguities that may arise (will be resolved thanks to the NAT router)

Before understanding how NAT works, one needs to understand the concept of **private IP address** → is an IP address (still a 32-bits string) that can be assigned to different hosts → more than one host with the same IP address (although this too has its constraints).

Question: How do I know whether an IP address is public (i.e. unique) or private?

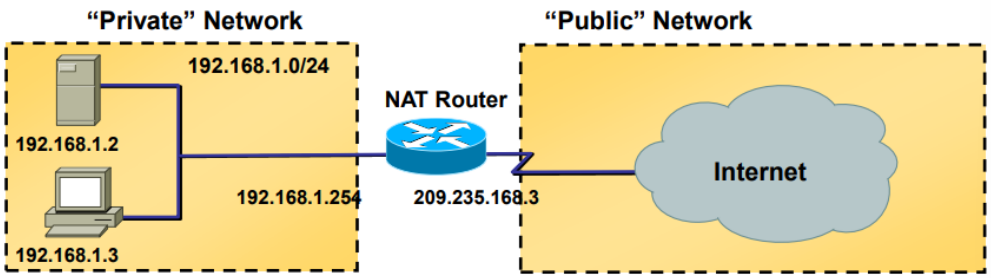
Answer: private is defined in a static way → a group of IP address blocks have been defined as private so basically you know immediately if an IP address is private just by looking at it.

- first block → defined as 10.0.0.0/8 (class A)
  ex: 10.0.0.3 sai che è un indirizzo privato.

Address Class	Number of Network Numbers Reserved	Network Addresses
A	1	10.0.0.0
B	16	172.16.0.0 - 172.31.0.0
C	256	192.168.0.0 - 192.168.255.0

The only constraint is that if you need to send a message from your private network to the 'public internet' that private IP address must be removed → you cannot use it outside the network where you are because otherwise it would become public.

Let's better explain the concept of 'no one can see it' and 'someone has to translate your private IP address when you go outside your private network'.



Suppose I have a private network with IP addresses that I use without asking anyone's permission precisely because they are private.

In my private network I do what I want without any problems → I send packets in the classic way from a source IP to a destination IP.

But what happens if I want to send a pkt to a host in the public network?
The host puts its ip address in the source add and the packet is sent to the router.
The router should forward the pkt to the internet but is not possible to cross the router with a private IP because the private IP has a local meaning only inside the private network.

The router should be able to do some sort of translation, removing the private IP and inserting the public one, and this is exactly the idea that is behind the **NAT Router** (Network Address Translation Protocol) → allows communication between private and public networks.

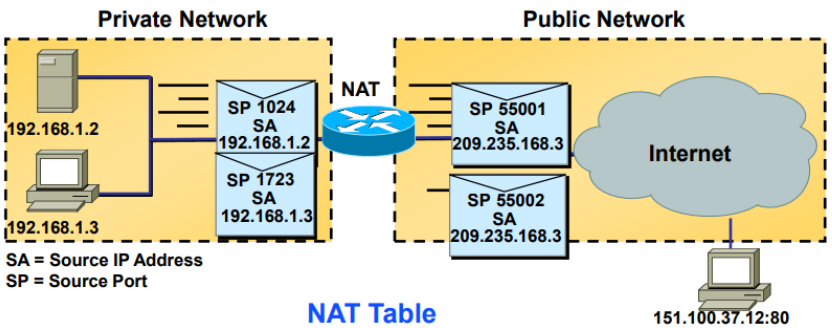
A NAT router has two interfaces:

- **local** interface that is connected to the private network devices
- **public** interface: has a public IP address assigned by the ISP and allows all private hosts to connect to the Internet.

Now the problem is: if there are multiple hosts that want to send messages to the public internet at the same time, using the same IP address provided by the NAT, how are they differentiated?

Basically what NAT does is break the rules of the TCP/IP protocol previously defined → in particular the rule that is broken is that of independence between layers.

Indeed, we need the NAT to be able to act on the transport header, in particular to read and modify a specific field of the transport header → the field dedicated to the **port number**.



NAT Table

Inside Local IP Address & port	Inside Global IP Address & port
192.168.1.2:1024	209.235.168.3:55001
192.168.1.3:1723	209.235.168.3:55002

MEMO: port number = local identifier of the application process that has generated data.

Example (see photo above)

We want to send a message to that device which has IP add + port number = 80.

The NAT not only changes the private add IP with the generic public one but also acts on the port number and chooses a new random number to assign to this field, making sure it is different from all the others it has already assigned so that it is that host.

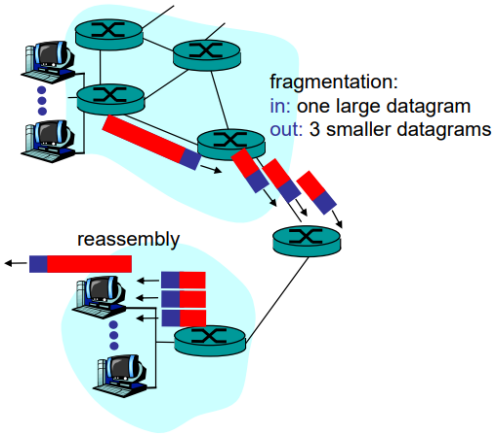
The generic public IP add and the new port number assigned by the NAT are both the same that the receiver will enter in the "Destination IP address" and "Destination Port" fields when it replies to the message that has arrived.
NB: the receiver enters the numbers that have arrived already modified as the pkts have already passed through the NAT: to ensure that they reach the right senders, the NAT must translate everything again.

Question: How many possible port numbers are there?
Answer: 2^{16} because a port number is a 16 bits string

Question: What happens if I don't have enough port numbers?
Answer: I have to ask for one more public IP address → with one public IP address I can have up to 2^{16} port numbers → if it's not enough I need another public IP address.

MEMO: the NAT has 4 layers → physical, link, network and part of the transport, while all the others have 3.

2) IP Fragmentation → process used by the Internet Protocol (IP) to enable the transmission of data packets through a communication network that imposes a maximum size limit (**MTU = Max Transfer Unit**) for the packets that can be transmitted.



Network links have different max transfer units (MTU) and so larger packets can be fragmented in individual datagrams which are then reassembled at the destination using the IP header bits to be identified and ordered.
The problem is not so much when the packages are split up but when they have to be reassembled because you have to consider the order in which they have to be put back together.

The information needed to put the packages together is contained in the second line of the header. In particular, three fields are used to identify the datagrams:

- ID: identifier of original IP packet (16-bits identifier) → all the fragments obtained by the same original datagram will have the same identifier;
- Fragment Flag: used to identify the last fragment obtained → 0 if is the last fragment, 1 otherwise (just to be sore that all fragments have been received);
- Fragment Offset: used to retrieve the correct order of the fragment → the fragments are numbered on the basis of the position of their first data byte in the original packet and then counted in blocks of 8 bytes (fragment length must be a multiple of 8) → we're counting the bytes again instead of the fragments but to have a small number we count bytes in block of 8's.

3) Routing (or forwarding)

The goal of each router is to identify the so-called 'output interface'.

IP router actions to be performed for each incoming packet:

- Reading, in the header, the IP destination field of the incoming packet;
- Looking at the IP routing table to find the longest prefix matching for the IP destination address;
- Detecting the next-hop router toward the destination;
- Forwarding the packet toward the proper outgoing interface.

Routing Table → is a set of rules, often viewed in table format, that's used to determine where data packets traveling over an Internet Protocol (IP) network will be directed.

Each router/host has its own IP routing table containing all routing information about the known destinations networks.

What does a routing table look like?

Of course it is a table, with rows and columns, and the number of rows is equal to the number of destinations that the router knows (= number of subnetworks in the network).

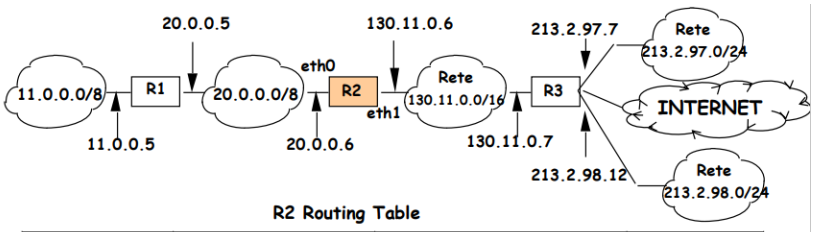
Each row is dedicated to a specific destination network and contains the following fields
A row contains:

- N: destination IP (used to identify the network);
- M: destination netmask (used to identify the network);
- NH: IP of first router (next-hop router) in the path toward the destination;
- I (è una i maiuscola): outgoing interface to reach the destination.

We said that a router, in order to correctly forward a packet, looks at the routing table and uses the information it contains to apply the '**Longest Prefix Matching**' rule.

How does this rule work?
Given D as the IP destination address, for each row i the following operation is performed:
If [D AND M(i)] = N then Matching(i)=M(i).
The rule having the greatest Matching(i) value is used to forward the packet.

Example: First of all, let's build the routing table for the router R2.



R2 Routing Table

Destination N	Netmask M	Next hop NH	Interface I
20.0.0.0	255.0.0.0	d.c. (20.0.0.6)	eth0
130.11.0.0	255.255.0.0	d.c. (130.11.0.6)	eth1
11.0.0.0	255.0.0.0	20.0.0.5	eth0
213.2.97.0	255.255.255.0	130.11.0.7	eth1
213.2.98.0	255.255.255.0	130.11.0.7	eth1
0.0.0.0	0.0.0.0	130.11.0.7	eth1

Now suppose that router 2 receives a packet that has Destination IP address = 130.11.0.7

Two things are done for each row in the routing table:

1) logical operation between Destination IP address and netmask of row i
2) match between the result of this logical operation and the destination IP address contained in the routing table.

I Row: 130.11.0.7 AND 255.0.0.0 = 130.0.0.0 ≠ 20.0.0.0 No matching
II Row: 130.11.0.7 AND 255.255.0.0= 130.11.0.0 = 130.11.0.0 Matching

…… No matching

VI Row: 130.11.0.7 AND 0.0.0.0 = 0.0.0.0 = 0.0.0.0 Matching

NB:

1) The logical operation between netmask and Destination IP address is simply to rewrite the Destination IP address as it is for every 255 (i.e. for every block of 1's) we find in the netmask, and put 0 in everything else.
2) Once a match has been found, the router does not stop, because it could happen that there is more than one match and that one destination address has a better match than another.

How do we know who has the better match?

Every time the router finds a match, it assigns a number to that row → in this case, row 2 was assigned the number 16 because the match is perfect for the first 16 bits.
After analyzing all the destination addresses, the one with the highest number will be the chosen route for the packet to follow.

In this example we have two matches, but the one with the Internet is a match with 0 bits so we still consider the destination in the second line as the destination of our packet.

Static and Dynamic Routing

Routing tables are calculated/configured on the basis of network routes.

There are two ways to configure them:

1) **Static** Routing → configuration performed by network administrator → the network administrator manually configures the paths in the router's routing table (e.g. shortest path algorithm).

The problem with a manual configuration is that it is:

a) not scalable;
b) (very) slow to react to network changes.

2) **Dynamic** Routing → exchange of control packets among routers (Routing Protocols) → routing method in which routers exchange network information using routing protocols such as OSPF, BGP and RIP.
Routers use this information to determine the best routes and update the routing table (e.g. distance vector routing algorithm).

Before going any further, delving into the subject of dynamic routing, we must introduce the concept of an **Autonomous System (AS)**:

- is a portion of the Internet (routers and networks) managed by a single network administrator (es: ISP like Vodafone or Telecom);
- is identified by a unique number, called Autonomous System Number (ASN) → we can consider the Internet as composed by a set of AS;

Each admin of an AS can independently choose a routing protocol for its AS.

Obviously if you send a packet over the secure internet you will cross more than one AS → what is needed is a single 'higher' protocol that regulates the passage of the packet through the different AS (a sort of agreement).

Now that we have introduced the concept of AS, we can move on to define the various types of routing protocols.

Routing protocols can be divided into two categories based on their functioning principles:

- **Distance Vector** routing protocols;
- **Link State** routing protocols.

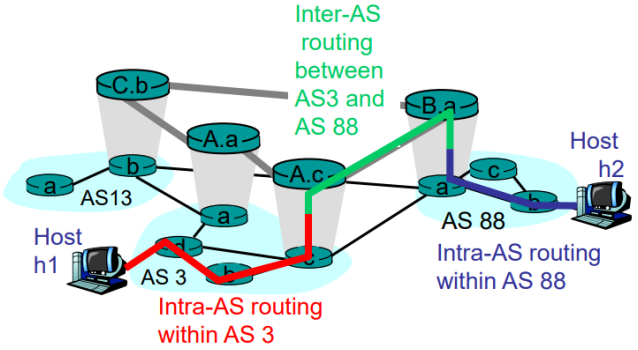
MEMO: However, the objective of a routing protocol in general is always to compute the routing table and calculate the best network path.

We will see that at the basis of each category there will be a specific routing algorithm.

There is also another classification of routing protocols:

- **Intra-AS** routing protocols → those protocols that are running within an autonomous systems.
- **Inter-AS** routing protocols → those protocols that are running among autonomous systems.

Let's start by explaining the difference between Inter-AS and Intra-AS routing protocols.



Suppose we have a network composed of three ASs and each of these ASs is identified by a number, in this case → 3,13,88 (es: AS 3 → composed by 4 routers and few links between these routers) and see how a host in AS 3 can send pkts to a host in AS 88.

What happens?
As long as the pkt we are about to send is within AS 3, the path to traverse AS is defined by the routing protocols used by AS 3 (Intra-AS protocol), which is totally free to choose its own routing protocols.
Now from router C we have to go out of the AS 3 → there will be a different routing protocol responsible for selecting the set of AS to be crossed to reach the destination.

This is where the inter-AS routing protocol comes into play, which is responsible for choosing the best route between the AS.

What is the difference between the red and green part?
They're computed by different routing protocols

When reaching the router B.a the packet enters inside another AS so the admin of AS 88 comes into play and the routing protocol of this AS is responsible for choosing the path that takes the packet to its destination.

The final outcome path will be a collection of the paths chosen by the different routing protocols.

NB: Regarding Intra-AS protocols there is no possibility of choice → they must all agree and choose a single protocol that fits all operations among the different AS.

OK now we can move on to explain the algorithms that the routing protocols use to calculate the shortest paths, which as we said before are based on two different approaches → Link State and Distance Vector.

Let us imagine the network as a graph, where the nodes are the routers and the edges are the physical links.

We define "**link cost**" as the delay, the cost in terms of money or congestion level that following that particular link to send the packet implies → there are several factors to consider when determining the 'good' path, including network latency, available bandwidth, network congestion, data security and router availability.

We then define the "**best path**" as the path with the lowest cost.

How to compute the minimum cost path over a graph? There are two solutions based on the network knowledge that routers have:

- **Global** solution → all routers have **complete topology** (remember what topology is: The Internet Topology is the structure of how hosts, routers or Autonomous. Systems are connected to each other) → all links cost info are known → **Link State** Algorithms;
    - More complete but more computationally demanding;

- **Local** solution → routers only have knowledge about their neighbors → we need an iterative process to make it possible to exchange info between neighbors → **Distance Vector** Algorithms.
    - Simpler computationally speaking but not very robust because you only have local knowledge.

Let's start with the global solution, the one that makes use of Link State Algorithms, such as Dijkstra's algorithm.

Question: How is it possible for a router to have global knowledge?

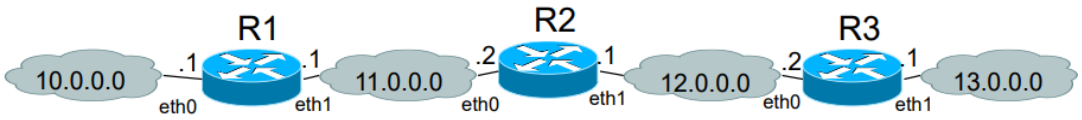
Answer: to have a whole topology the router has to communicate among them, each router describes its topological state (all its interfaces, all its IP addresses etc...), it's going to create a message which is sent to all other network's routers.
At the end of this procedure of message exchange, each router has the description of all other network's routers.

Based on the global knowledge they have acquired, each router can independently run Dijkstra's algorithm, used to compute all the minimum paths to all other destinations.
It's an iterative algorithm and after k iterations the router is able to know the least cost paths to k destinations.

Now we can move on with Distance Vector Algorithms → in this case the router does not have a complete knowledge of the topology.

It's a **distributed** and **asynchronous** algorithm where each node iteratively communicates with its neighbors, stopping when there's no more ongoing communication.
Each node computes its own **distance table** where it contains a row for each possible destination.

Example:



- Step 1 → the router only knows the neighbors' information → add them to the distance table.

R1

Network	Next-hop	Distance
10.0.0.0	eth0	0
11.0.0.0	eth1	0

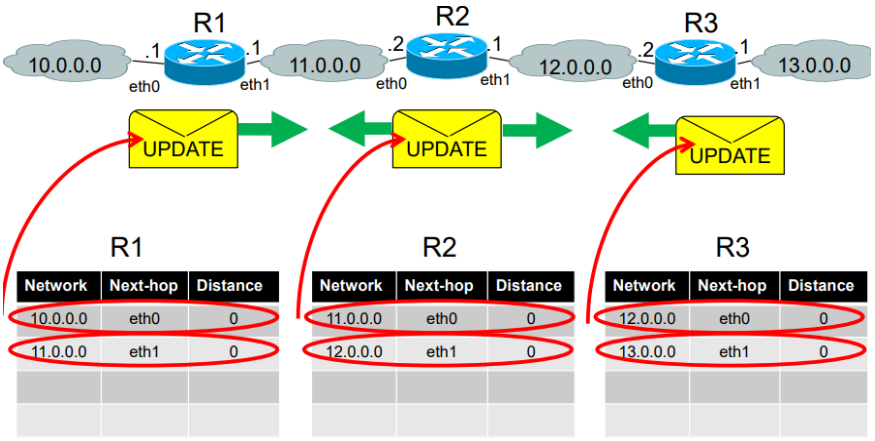
R2

Network	Next-hop	Distance
11.0.0.0	eth0	0
12.0.0.0	eth1	0

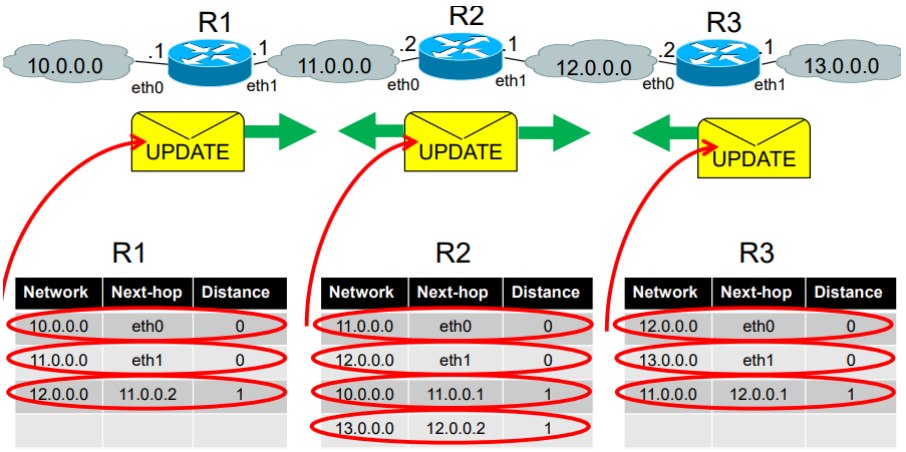
R3

Network	Next-hop	Distance
12.0.0.0	eth0	0
13.0.0.0	eth1	0

- Step 2 → When we say that a router describes itself, what it is actually doing is just a summary of its routing table → so each router sends the update of its routing table to all its neighbors.



- Step 3 → each router takes from the neighboring one only the information it lacks.
  Looking at, for example, R1 and R2 → R2 already knows how to get to 11.0.0 but it lacks how to get to 10.0.0 so it takes this info from routing 1 and writes it down.



And so on until all routing tables are complete.

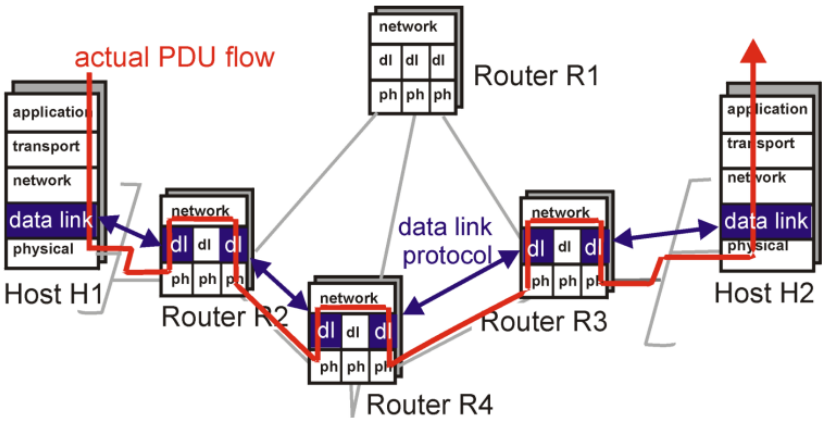
OK, just to summarize all the type of algorithm we could have:

- Intra-AS, also known as Interior Gateway Protocols (IGP)
  - **RIP** → Routing Information Protocol
    - Distance Vector
    - Link cost equal to 1
  - **OSPF** → Open Shortest Path First
    - Link State
    - Link cost inversely proportional to its bandwidth

- Inter-AS, also known as Exterior Gateway Protocols (EGP)
  - **BGP** → Border Gateway Protocol
    - Path Vector protocol → the entire path is notified (path: list of ASes crossed).

LECTURE 5 - The Data Link Layer

Let's set up the context:



We have to imagine as if the path that packets have to take from one device to another could be split into **sub-paths**, where each sub-path is a "link" between neighboring devices (NB: we use the term link even though it is not entirely appropriate because we will see that the link is a more complex thing than point-to-point communication → for example, inside a link could be other kind of devices, like switches, but we'll think about it later).

In this case we have four sub-paths: H1 → R2, R2 → R4, R4 → R3, R3 → H2.

We will see that a whole series of functions (data link layer function) are required to send packets from H1 to R2 → we will go into much more detail regarding communication between two physical devices.

Before going any further, it is important to remember that each physical connection has its own data link layer protocol → so there is not one protocol for everything but each sub-path has its own → doesn't exist a single data link layer protocol because different technologies exist to create a physical network.

MEMO: of course it is always the interfaces (in this case physical interfaces) that are connected and each interface has its own data link layer.

So, before going into detail on the functions of the data link layer, we must bear in mind:

- There is more than one protocol to define a link (so it's not like the network layer that revolves around a single protocol → IP)
- Two physically connected devices can be: host-router, router-router, host-host, host-switch
- The data units in the data link layer are called frames → a frame is an IP packet, so a data unit coming from the network layer + the header of the data link layer.

What are the main **functions** of the data link layer?

- **Framing** → create frames → encapsulate datagram into frame + adding **header** and **trailer** (information is added both at the beginning and at the end → the order is: header + IP packet + trailer).
  What is the information you have in the header and trailer? It depends on the specific data link layer protocol → anyway there is some info that is common also among different protocols and

in particular there is an info which is called **physical address** which is used in frame headers to identify source and destination.

- **Medium access control** → in some specific cases one must implement a mechanism that coordinates the access to the physical layer.

- **Reliable delivery between two physically connected devices** → not all data link layer protocols have it, but this is a mechanism that allows us to detect and correct possible errors (signal attenuation, noise) that may occur in communication between two devices → the mechanism is similar to the checksum but is more reliable.

We will focus specifically on medium access control (why need a sort of coordination to send data over a physical channel) and we're going to talk about the physical aspect, in particular about a specific data link layer protocol known as **Ethernet**.

The last thing it is important to know before starting is that all the functions of this layer are implemented in hardware (inside the physical interface) → not a software implementation but a harder one.
This is important because, with this type of implementation, you cannot have complex solutions → all functions must be simple and easily implemented.

Medium access Links and Protocols

If we think about how physically a link can be realized, we have three different solutions:

- **Point-to-Point** → this is what usually happens between routers → there is a real physical link connecting them.

- **Broadcast** (e.g. wifi network or Ethernet) → there is no physical link but a broadcast link (i.e. a communication channel something on which data can be sent and received shared between several devices → there is a single channel for a huge set of devices, so there is a problem of how to manage all these devices → we will talk about this in a moment).

- **Switched** links (i.e. ATM, switched Ethernet) → here you have to connect some specific devices called **switches**; here the broadcast problem has been solved in a very interesting way (we will also talk about this).

For now, let's consider the most critical case→ our devices are connected through a broadcast network.
Broadcast Channel

What happens if two nodes (i.e. two smartphones) start sending data at the same time?
Since it is not possible to use the same physical channel at the same time, the two signal interfere and one is lost → only one node can send successfully at a time.

Solution: **Multiple Access Control** protocols (**MAC**) to manage communications → these protocols allow multiple nodes or users to access a shared network channel: several data streamed from several nodes are transferred via the multiple transmission channel.

This is a **distributed** algorithm solution (Remember: distributed algorithms are algorithms designed to run on multiple processors, without tight centralized control).
The objectives of multiple access protocols are to optimize transmission time, minimize collisions and avoid interference.

From a theoretical point of view we have 3 ways to solve this problem → 3 classes of the MAC protocol:

- **Channel Partitioning** → divide channel into smaller "pieces" (time slots, frequency) and allocate piece to node for exclusive use (similar to the concept of time division multiplexing) → not easy to be managed because depend on the number of subchannel we have so it's not an efficient solution.

- **Taking Turns** → each device has the use of the channel for a specific period → you can transmit something when it is your turn (you have your turn even if you have not something to transmit → again, not an efficient solution).

- **Random Access** → when a node has something to transmit it tries to transmit the whole frame when possible → this however leads to interference because if I try to transmit when someone else is doing it, this leads to **collisions**, in which everything is lost.
  Above all, we have to be careful about **retransmissions**: after a collision the data has to be transmitted again → we have to understand how to handle collision and retransmission.
  The target properties of this protocol are: efficient, fair, simple and decentralized.

Random Access protocols → A node sends a packet when necessary at full channel data rate with no a priori coordination with the other nodes.

Random access MAC protocol specifies:
- how to detect collisions;
- how to recover from collisions (e.g. via delayed retransmissions).

Notice one can only aim to reduce the probability of collisions, not to completely avoid them.

Examples of random access MAC protocols:
- Slotted ALOHA;
- Pure ALOHA;
- CSMA and CSMA/CD.

Let us analyze these protocols in detail.

Slotted ALOHA

First of all, **synchronized assumption**→ synchronization between nodes is required → every node must know when a timeslot starts and this is a costly operation that needs a dedicated decentralized infrastructure.

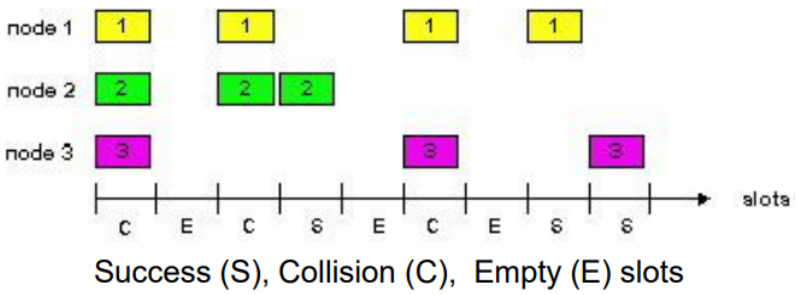
Now, suppose that.
- Time is divided into equal size slots (time = pkts transmission time) and each node know this (because they're synchronized);

- All frames (pkts) have same fixed length → because I can insert exact one frame in one time slot;
- A node can send a pkt only at the beginning of a slot → each node waits for the next time slot to send its pkt;

How does the algorithm work?
A node waits for the start of the time slot and sends its packet, then a collision is checked.
If there was no collision, lucky node, otherwise the collision means loss of the packet.

How is the collision handled?
The node must send the packet back to future slots, and retransmits it with probability *p*.

Example:



Success (S), Collision (C), Empty (E) slots

- Three packets are sent together → collision → all three are lost.
- The three nodes are aware of this (synchronization assumption) so what do they do? With some probability *p* they decide whether or not to send the packet back to the next slot.
- At the second slot nobody decides to send anything → is empty.
- At the third slot, 1 and 2 decide to send it but again collide → both lost.
- At the fourth slot only 2 decide to send it → successful!

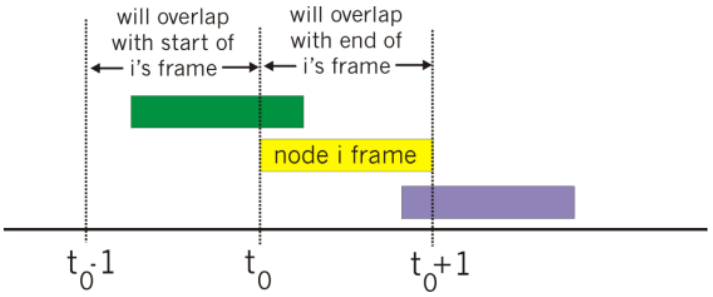
And it goes on like that…
MEMO:
- This transmission is only from one node to another, not from one node directly to the destination.
- There is always a collision at the first slot because each node has something to send.

Probabilistically speaking:
Suppose nodes want to send a packet and let be the probability of transmission:
- The probability of successful transmission for a single node is: $S = p(1 - p)^{N-1}$
- The probability that at least a node successfully sends the packet is: $S = Np(1 - p)^{N-1}$

Choosing optimum *p* as $N \to \infty$ we obtain *S* = 0.37 → this means that 63% of the time the channel is not used because of collisions and empty time slots.

Pure (unslotted) ALOHA → very similar to slotted ALOHA but without synchronization (drop synchronization allows transmission at any time) → it's simpler.

In this case, pkts are sent without waiting for beginning of slot → collision probability increases because a pkt sent at t_0 could collide with all other pkts sent in $[t_0 - 1;\ t_0 + 1]$.



So, now, when we choose p as $N \rightarrow \infty$ we obtain $S = 0.18$ → this means that 82% of the time the channel is not used because of collisions and empty time slots.

CSMA (Carrier Sense Multiple Access)

The functioning principle is to listen to channel before transmit:

- if channel is idle → transmit entire pkt;
- if channel is busy → defer transmission

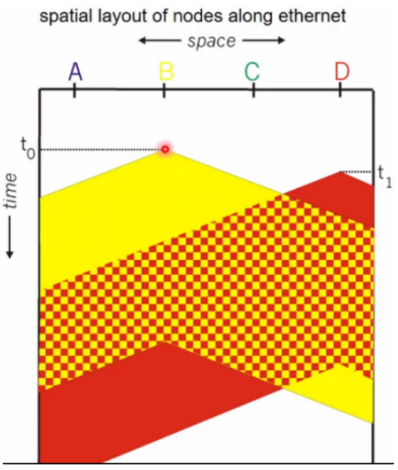
In case of busy channel, there are two ways to handle the pkts transmission:

- **Persistent** CSMA: retry immediately with probability p when channel becomes idle → (may cause instability);

- **Non-persistent** CSMA: retry after random interval.

However, collisions can still occur due to **propagation delay** (= physical distance to reach the destination) → two nodes may not hear each other's transmission → if two nodes are waiting for the channel to become free and as soon as it becomes free they both transmit, there is a collision.

Example:



The propagation delay in the image is shown horizontally.
At time t_0 a packet is sent by *B* and a time t_1 a packet is sent by *D*, who's not yet aware that the channel is busy because there's a propagation delay from *B* to *D*.
In conclusion, the problem we have in the case of using CSMA is collision due to propagation delay → we can reduce the collision probability but we can't avoid it because it depends on the physical distance among nodes.

Now that we have said that collision can happen, we must take two aspects into consideration:

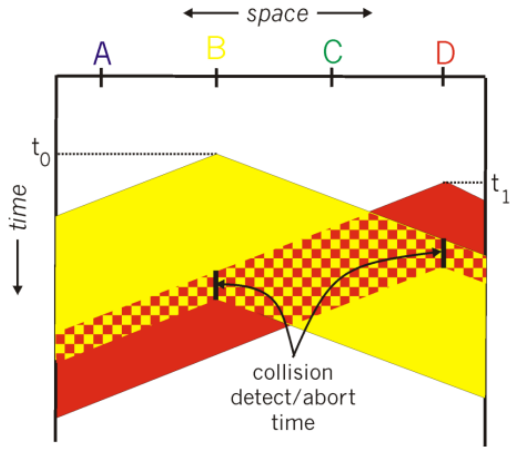
- how to be aware of the collision
- what to do after a collision happens

Let's introduce the CSMA/CD solution.

CSMA/CD (Collision Detection)

Main idea → detection during transmission.

During transmission, a node also detects the medium, so that when a collision is detected it aborts transmission within a short period of time → if during transmission a node receives a packet from another node, it interrupts transmission and detects a collision, reducing channel wastage.



Now, remember that: without sensing the medium I can't be aware of the collision but I'm sensing only while I'm transmitting → merge these two things I can conclude that I'm aware of a collision only if it happens when I'm transmitting

The condition for collision detection to happen is that the minimum length of a frame is at least twice the propagation delay, i.e:

$$d_{trasm} = \frac{L}{C} \geq 2d_{prop} = 2\frac{d}{v}$$

where:

- d_{trasm} = (minimum) transmission time → the time to transmit the smallest frame I can;
- d_{prop} = (maximum) propagation delay → the two most distant nodes must be considered;
- L = minimum frame length
- C = transmission capacity
- d = distance of the network segment
- v = propagation velocity of the medium

How to respect this condition? The are two points of view:

- The maximum propagation delay is a fixed value, so I know which are the furthest nodes and the distance between them → the $2d_{prop}$ is a fixed quantity → in that case I have to compute

the minimum length of a frame (i.e. the minimum transmission time) which is equal to $\frac{length\ of\ the\ shortest\ frame}{capacity}$.

- The minimum length of a frame is a fixed value so I have to compute the maximum distance among two hosts (i.e. the maximum propagation delay) which is equal to $2 * \frac{distance\ of\ the\ network\ segments}{propagation\ velocity}$.

We have concluded the Medium access control function, now let's talk about another function → physical addresses.

Physical Addresses

We said that the main function of this layer is to create frame = add header and trailer to the IP packet, and we also said that in some fields in the header of the data link there are two fields related to physical addresses → we have different iìkind of addresses in the data link layer.

What is a physical address? Is an identifier of an interface, exactly as the IP address → The difference is that the IP is related to the network layer (layer 3) while the Physical address is related to the data link layer (layer 2).

The two most used physical addresses are → **LAN** (Local Area Network) address and **MAC** address.

What does a physical address look like?

- Is a binary number composed of **48 bits** (in the IP case there were 32 bits).
- In this case, instead of translating this string in a decimal version, we use the **Hexadecimal** representation (Example: 1A:23:5B:F8:00:69).
- Is a **fixed** identifier (the most important difference with respect to the IP add) → is something written in the hardware of the interface (for sure is **unique**).
- In this number there are no network's identifier numbers → the interfaces have not a bit in common → Each adapter on LAN has unique LAN address:

Let's try to understand why we need physical addresses.

We need them to make it possible, for a frame generated by a node, to reach another node in the same network.

The transmission of packets in the network layer works through IP addresses → do that routing table thing, find the corresponding destination IP address, and send the packet.

How does this same process work in the data link layer?

We have to insert in the header of the datagram the source MAC address and the destination MAC address.
So we have a double identifier level → to send a packet from one node to another, it is not enough for us to know the IP address but we must also know the physical address.

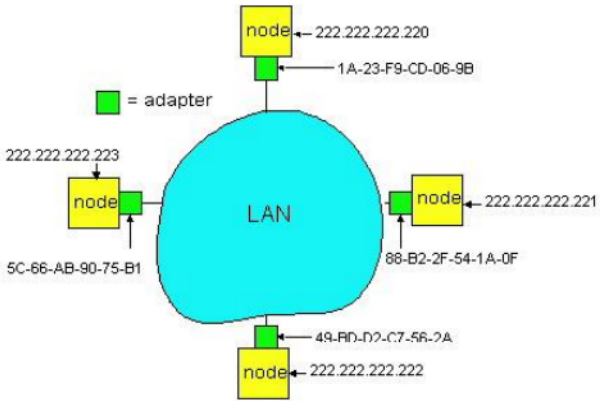
Problem: how to know which is the physical address of a node? This is not easy information to get because, while the IP addresses respect a hierarchy, the physical addresses don't.

Solution: there is a dedicated protocol, known as **ARP** (Address Resolution Protocol), which is responsible for 'asking' the node for its MAC address.

How does this protocol work?

Basically, the ARP allows nodes within the same layer 2 network to find MAC addresses of each other.
Suppose node *A* wants to communicate with node *B*:

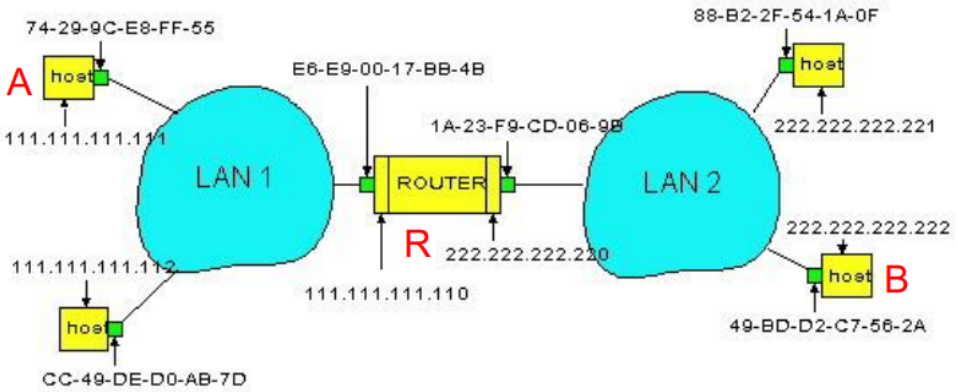
- *A* knows *B*'s IP address and wants to learn physical address of *B*;
- *A* sends broadcast message (FF:FF:FF:FF:FF:FF) as an ARP query packet, containing *B*'s IP address → all machines on LAN receive ARP query;
- *B* receives ARP packet, replies to *A* with its (*B*'s) physical layer address;
- *A* saves in its **ARP table** (each IP node (Host, Router) on LAN has its own ARP table) the IP-to-physical address pairs until information becomes old (times out → typically 20 minutes).



MEMO: ARP Table contains rows with 3 columns: (1) IP, (2) MAC address, and (3) Time to Live (TTL), the time after which the mapping will be forgotten.

Question: What happens instead when I want to send a packet from one node to another, when these two nodes belong to two different LANs?

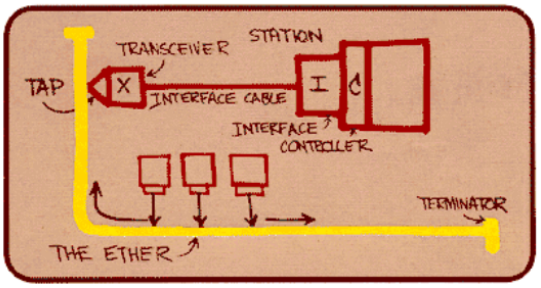
Answer:



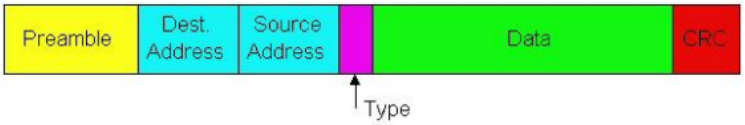
- *A* creates an IP packet with source *A* and destination *B*;
- After looking at its routing table, *A* uses ARP to get *R*'s physical layer address for 111.111.111.110 → we need to go through the router first because the data link layer only deals with what happens inside the Local Area Network (n this case LAN 1);
- *A* creates Ethernet frame with *R*'s physical address as destination → Ethernet frame contains *A*-to-*B* IP datagram;

- *A*'s data link layer sends Ethernet frame;
- *R*'s data link layer receives Ethernet frame;
- *R* removes IP datagram from Ethernet frame, seeing that its destined to *B*;
- After looking at its routing table, *R* uses ARP to get *B*'s physical layer address;
- *R* creates a frame containing *A*-to-*B* IP datagram to be sent to *B*.

Ethernet → Specific data link layer protocol and was created to define broadcast LAN.



Let's start with the Ethernet frame structure → the sending adapter encapsulates the IP datagram inside this frame.



It has 5 fields → four for the header and one for the trailer.

It's composed of:

- **Preamble**: combination of 7 bytes → the first seven bytes follow the pattern 10101010 and are followed by one byte with pattern 10101011 → the aim of having a fixed sequence is to identify when a new frame arrives.
  Used to synchronize receiver and sender about the reception of a frame.
- **Addresses**: 6 bytes → remember that the frame is received by all adapters on a LAN and dropped if address does not match
- **Type**: indicates higher layer protocol (usually IP but others may be supported).
- **CRC**: checked at receiver for error detection → if an error is detected, the frame is dropped.

It has very simple functions to perform compared to, say, IP protocol → addressing packets + error detection.
Ethernet uses the protocol CSMA/CD, we already talked about, to perform medium access control.
Let's analyze more in detail what this protocol does.

```
A: sense channel, if idle
   then {
            transmit and monitor the channel;
            If detect another transmission
              then {
                 abort and send jam signal;
                 update # collisions;
                 delay as required by exponential backoff algorithm;
                 goto A
                 }
              else {done with the frame; set collisions to zero}
        }
   else {wait until ongoing transmission is over and goto A}
```

Suppose we have an Ethernet node, *A*, that wants to send a frame.

The node senses the channel → if the channel is idle the node starts sending the frame and, at the same time, keeps sensing the medium to check if a collision happens.

What happens if it detects a collision?

- It aborts and sends a **jam signal** (= a signal that is used to let to all the other nodes that a collision happened → make sure all other transmitters are aware of collision);
- It updates the number of collisions experienced by this frame.

The problem now is: this frame experienced a collision. How much time do I have to wait until I send it again? We remember that it happens in a random way → let's see how this random way is implemented.

There is an algorithm to compute this time → **exponential backoff algorithm** → used to choose the next time to transmit the frame. Let's see how it works.

The aim of the algorithm is to try to adapt the retransmission time to the estimated network load → if the network is **underloaded** (few transmissions on the medium) and the frame experiences a collision, it must be sent again as soon as possible.
Otherwise, if the network is **overloaded**, I should wait for a longer time.

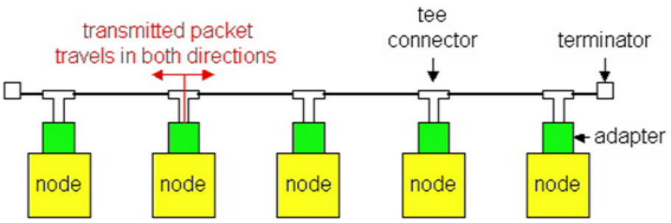
The problem is that I don't know what the network load is.
The idea is to use the number of collisions experienced by the frame to estimate the network load.

- first collision: choose K from {0,1}; delay is K x 512 bit transmission times
- after second collision: choose K from {0,1,2,3}…
- after ten or more collisions, choose K from {0,1,2,3,4,…,1023}

Fix that the value of K is randomly chosen between the value of the set we have (which depends on the number of collisions).

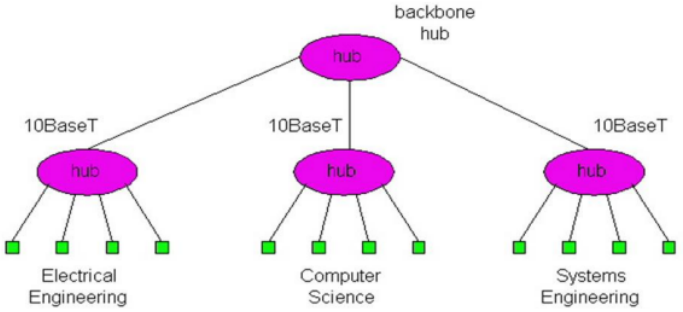
Ethernet technologies

- **10Base2** (bus topology → we have a medium, which is the bus, where all the nodes are directly connected):



  - 10 Mbps (bandwidth)
  - 2 → 200 meters max cable length
  - The reason why it is a broadcast network is because information is sent from one node to all the other nodes.
  - the nodes are connected with a thin **coaxial cable**
  - Is the first version of Ethernet and now it no longer exists.

- 10/100BaseT (star topology → fast ethernet → the ethernet network we use today)



  - 10/100Mbps
  - T refers to the cable type → Twisted Pair cable
  - In this case, we don't have the bus anymore, but some new devices that guarantee connection → the **hubs:**
    - devices providing connection between different nodes) → the nodes are not directly connected through a shared medium but through different devices.
    - In particular, a hub is a layer 1 device → a device which implements only physical layer functions (is not able to understand, for example, what a frame is, or what a packet is).
    - By using hubs, we have a LAN divided into segments.
    - CSMA/CD implemented also at the hub.
    - Hub Advantages → simple (inexpensive) device, extends maximum distance between node pairs (100m per Hub) → max distance from node to Hub is 100 meters.
    - Hubs do not isolate collision domains (= Portion of a network that can lead to collisions) → node may collide with any node residing at any segment in LAN

The hubs are really simple and basic devices → to compensate for all the shortcomings of the hubs, other devices were introduced → **bridges:**

- Link Layer device → the bridge is able to read inside the header of a frame.
- Store and forward device → it buffers frames
- When frame is to be forwarded, bridge uses CSMA/CD to transmit
- Bridge isolates collision domains
- Bridges filter packets: same-LAN-segment frames not forwarded onto other LAN-segments → in this way the frame can reach only the real destination and not also all the other nodes-

Question: How does this filtering procedure work? How to know which LAN segment on which to forward frame?

Answer: We said that the bridge is able to read inside the header of a frame: what is the information we're interested in inside the header? Destination MAC address of the frame.
Based on this information, it is able to choose the right interface to which to send the information.

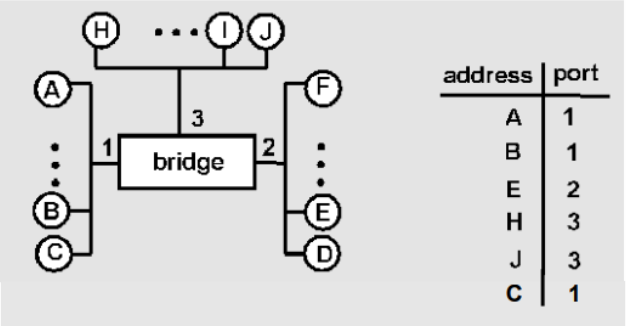
Question: How does the bridge know which hosts it can reach and through which interfaces?

Answer: There is a simple learning procedure through which a **filtering/forwarding table** is created. When a frame is received from the switch, bridge "learns" the location of the sender (incoming LAN segment) and records sender location in the filtering table.

Filtering table entry:

- Node LAN Address
- Bridge Interface
- Time Stamp → stale entries in Filtering Table dropped (TTL can be 60 minutes)

Example:



We have 3 interfaces + a forwarding table of the bridge (remember that in the real world we don't have the letters but the MAC addresses).
Suppose C sends frame to D and D replies back with frame to C.
C sends a frame and the frame reaches the bridge. The bridge reads that the destination is D, but the bridge has no info about D.
The bridge has a function:

- keep learning about the network dynamically → the first thing it does is to write in the forwarding table what is the source interface (→ C in reachable through the interface 1)

We then know that the information reaches all nodes but only node D will be able to read it → the first thing the nodes do is read the destination MAC address, then the moment they read it and realize that they are not the destination, they discard the frame.

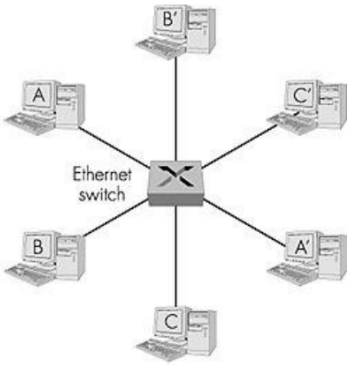
So, the information reached node D, which read it and now has to send a reply back.

D generates a reply to C.
Bridge sees a frame from D and notes that D is on interface 2 → bridge now knows that C is on interface 1, so selectively forwards the frame out via interface 1 → In this second step, the information does not reach all nodes but only the one in the correct interface.

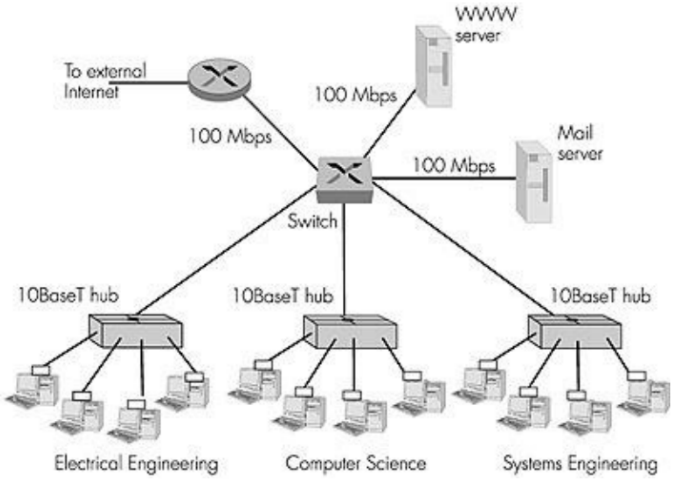
Over time there has also been an evolution of bridges → today's bridges are called Ethernet **switches** (also in this case remember that we're talking about a data link layer device → a device which is able to perform a layer 2 functions → store and forward device, receiving frames, read the destination MAC address, learn MAC forwarding table ecc…):

- Similar to the bridges (more interfaces)
- Full duplex links → represent the real improvement there has been in the transition from bridge to switch.
  Why are they called "full duplex"? There is one part of the link that is responsible for outgoing transmissions and another for returning transmissions → **no collisions!**
- Typical topology: individual hosts, star-connected into switch
- Plug-and-play: no need to be configured (actually, there are more complex cases in which they must be configured, but we will see them later).

Example of Ethernet Switches:



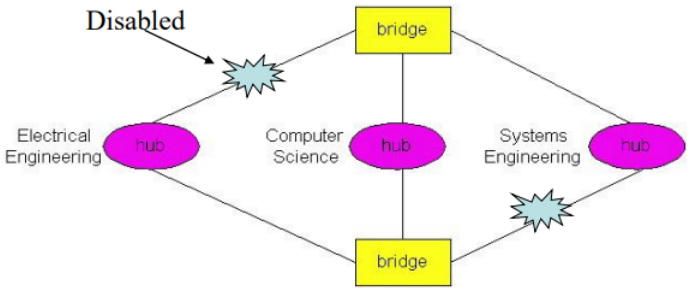
Example of Ethernet LAN (ethernet network)



The Spanning Tree protocol

Usually, in today's LAN network, we have topologies that have a lot of links → different paths to reach the same destination.
For increased reliability, it is desirable to have redundant, alternate paths from source to destinations.

But there are some issues with this solution → cycles (loops) can appear → to avoid this problem, the spanning tree protocol has been created, with the aim of disabling some interfaces in order to avoid loops.



LECTURE 6 - Virtual LAN, VXLAN, LISP

Introduction

Until now we've talked about the fundamentals of networking → all the basic functions of our IP network.
Up to now we're going to talk about new functions → How? Introducing new protocols.

Until now we focused on a specific LAN, which was the Ethernet LAN (memo: LAN is a layer 2 network).

What we're going to do is basically analyze if it's possible to build a data center exploiting the functionalities of Ethernet

We'll see two scenarios:

- **Single** layer 2 data center
- **Multiple** layer 2 data center

and we're going to talk about 3 specific protocols:

- Virtual LAN + VXLAN → related with Ethernet
- LISP → related with IP addressing

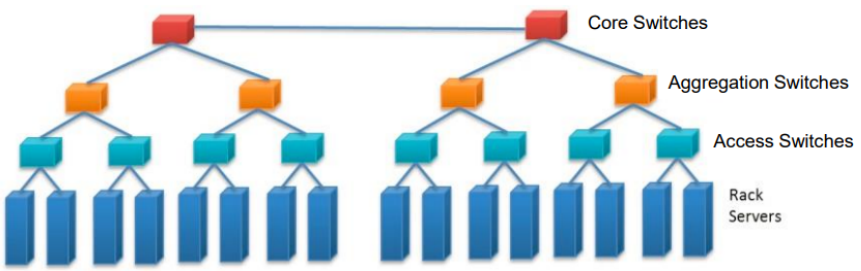
There new protocols exist to support the management of a Data Center providing intensive data processing.

Let's start!

Data Centers

A data center is a LAN (Ethernet network) which contains a collection of high performance servers organized in racks (20-40 servers per rack) that are able to communicate through (many) switches and (a few) routers.

A data center has a **hierarchical** structure (so the **topology** is a classical hierarchical topology but of course also different solutions) with different types of switches connecting the servers.



The situation a data center has to handle is basically a group of users (called **tenants**) asking for dedicated Virtual Machines (VMs) where to execute specific jobs → in a data center an application runs on a Virtual Machine (VM) which is hosted on a physical server so each server has different hosts (VMs).

We remind that the principles of an Ethernet (Layer 2) network are:

- MAC Addresses
- ARP
- Switches → which are Plug and Play device (no configuration required)
- MAC Forwarding Table
- MAC Learning

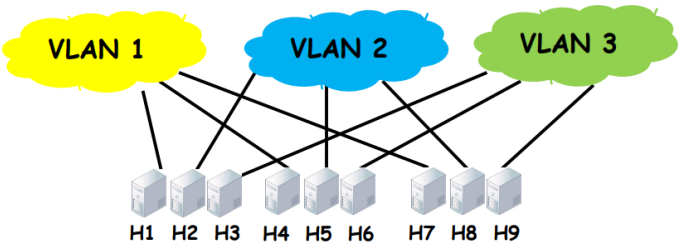
We will need to improve layer 2 protocols to provide a set of new functionalities and this is due two main drawbacks that arise in a DC with a thousand of end devices:

- Isolation of network portions (**security**) → in a layer 2 network any device can send a packet to any other device, but in a DC this is not always necessary → imagine two virtual machines that provide specific services to the outside world and do not need to communicate with each other.
- **Broadcast storm** → in such a huge network if hosts send a broadcast message (a message that reaches all nodes) they will generate a large amount of traffic that might saturate its bandwidth → it is then necessary to divide the network in subnetworks so that a host only sends this messages with it

Virtual LAN

The physical network is separated in different logical networks called VLAN:

- A VLAN is identified by means of a VLAN ID (basically a number → 1,2,3…)
- Each virtual LAN has its own IP address space (because they are different subnetworks)
- Communication possible only inside the VLAN



However, in order to achieve this structure, we have to increase the complexity of the switches → they can't be considered anymore as plug-and-play devices but they need a **configuration** by administrator.

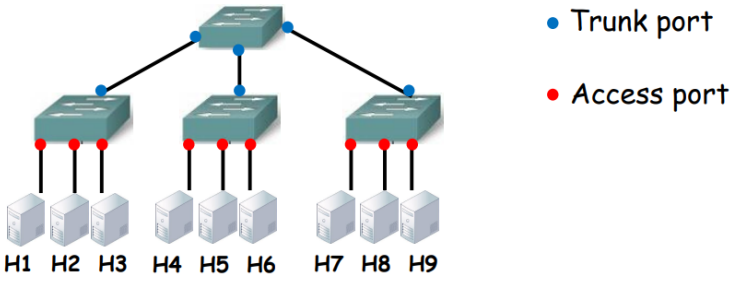
The switches must:

- know the existing VLANs
- associate each interface to the proper VLAN
- identify to which VLAN a frame belongs to
- maintain a MAC forwarding table for each VLAN

The first step to configure a switch so that is able to achieve all of the above is to define the interfaces role:

- Access port (to host) → interface of the switch directly connected with a host
- Trunk port (to switch) → interface of the switch directly connected with an another switch

Notice the term port here is used to refer to an interface which is different from the use of port we used in layer 4.



So, the first thing we have to do while configuring switches is to define which are the access ports and which are the trunk ports → this is something that must be done from an admin.

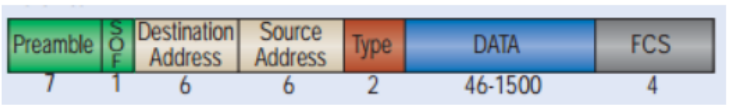
Access ports:

- are directly connected to the hosts so it's easy to identify from which VLAN packets are coming from → no need to change the internet protocol.
- can use two kinds of associations (associated to only ONE Virtual LAN):
    - **Static**:
        - configured by the network administrator
        - location-based → each access interface is statically associated to a specific VLAN (the admin has to specify each port to which VLAN is connected to → he doesn't have to change anything inside the host, only the switch is configured)
    - **Dynamic**:
        - MAC address based
        - each host is associated to a specific VLAN → the switch has to query a static server in the network to know the VLAN of the host (for the first time it sends a frame in the network) → it's a more complex solution because we need a server and a protocol to handle this association function.

Trunk ports:

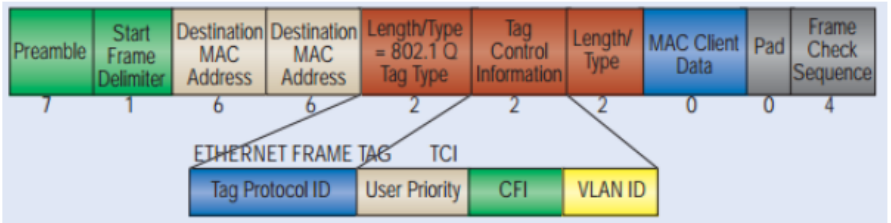
- must be able to detect the VLAN each received frame belongs to → each frame must have a VLAN identifier in its L2 header (while in the access part there is a sort of "space association" → each interface receives packet belong to only one VLAN, in the case on trunk ports this is not true).
- to compute this "detection function", the switches, when receiving a frame from a host (via an access port), augment it by adding a part of the header containing, among other fields, a 12-bit (→ that implies that exists about 4K different ids) VLAN ID → extension on ethernet.

We used to have an Ethernet frame made up like this:



802.3 (Ethernet) frame

We have now implemented a new version of Ethernet, also known as **802.1Q**:

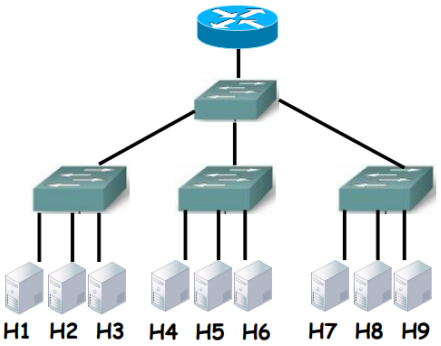


802.1Q frame

MEMO**:** remember that the host expects an Ethernet frame, so once the VLAN has been identified, before the frame arrives at the host, the added part used to specify the VLAN ID must be removed → from the host point of view, nothing changes w.r.t the original ethernet protocol.

Problem: VLANs are isolated.

Solution: To make possible the communication among devices belonging to different VLANs, a router (L3 device) must be inserted → it is possible to “connect” a subset of available VLANs .

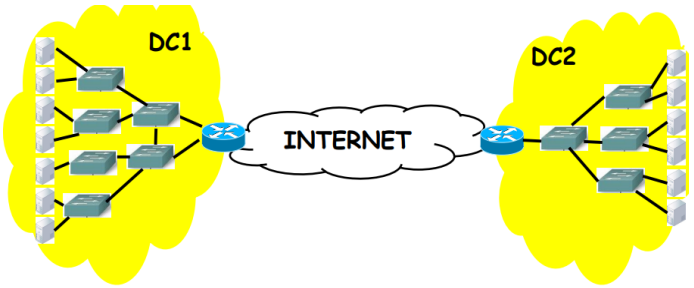


Virtual eXtensible LAN (VXLAN)

Let's now consider the case of multiple data centers located in different places across the world managed by the same organization. The Internet will provide communication among them.

The situation is the following:

- Servers: Virtual Machines (VMs)
- Switches: inside the DC
- Routers: among the DC and Internet

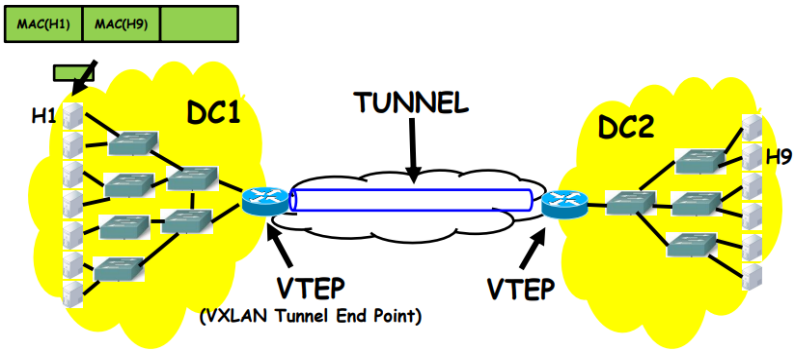


We would like that two servers from two different DCs,working on the same task, view each other as part of the same Layer 2 network.

Idea: create a direct logical connection among the DCs.

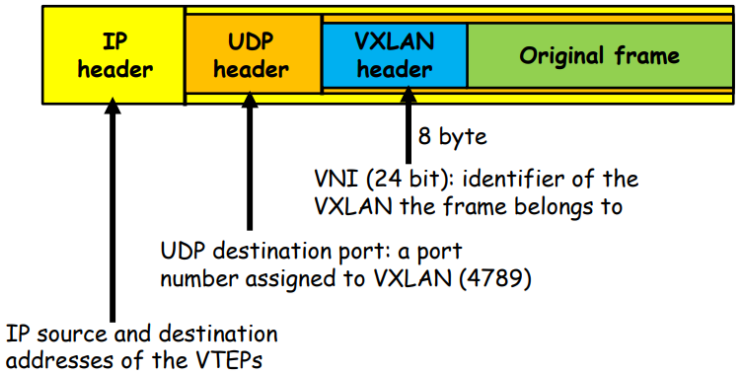
This idea is realized through the use of the Virtual eXtensible LAN (VXLAN) → a host in DC1 will be able to send a frame directly to a host in DC2 as if they were in the same L2 network.
Say the host H1 in DC1 wants to communicate with the host H2 in DC2 then it will create a Layer 2 frame with both MAC addresses → but then we need some mechanism that allows the communication to pass through the internet → something we would do with an IP packet.

The idea is to create a direct connection between the routers of the DCs which we call **VXLAN Tunnel End Point**.



A new IP packet is created containing the IP of the two VTEP (origin and destination) encapsulating the original frame so that it can be directly delivered to the destination VTEP.

The Ethernet frame is encapsulated into an UDP packet from Router A to Router B (**MAC-in-UDP encapsulation**) → the original frame is treated as the output of some application and new headers are added so that this packet can cross the internet and reach the destination VTEP.



NB: typically we have the frame header encapsulate the IP packet, instead here the IP packet is used to deliver a frame.

We have the following headers:

- IP header: source and destination addresses of the VTEPs
- UDP destination port: a number assigned to VXLAN (4789) used to identify the frame is of a VXLAN (we don't use TCP because it is too complex)
- VXLAN header is called VNI (24 bit) and identifies the VXLAN where it belongs

NB: the destination VTEP router has to remove the extra header because the host is not able to read it.

The complexity of this solution is present in the VTEP router

VTEP:

- mantains servers to VXLAN mapping
- performs encapsulation/decapsulation
- has one (or more) interfaces toward the local LAN and one toward the IP network (with a unique IP address)

- communication between VTEPs (to inform each VTEP of the host that are connecting with themselves) uses **multicast** (IP technology to create groups) → a multicast group for each LAN
- upon receiving a frame directed to unknown MAC address (not in its VXLAN) it broadcast the frame to all other VTEPs in the group

OBS: it is possible to move the VTEP functionalities from the routers to the switches → the novel switches are able to perform the encapsulation function.

Just to make a recap of all this part:
- Data center → thousand of tenants → thousand of VLANs
- Each VLAN has its own VLAN ID → 12 bit → about 4K different LAN segments
- VXLAN ID → 24 bit → about 16M different LAN segments

NB: There are cases where VXLAN is used in a single data center because there are so many virtual machines that VLAN IDs are not sufficient → **flexibility** of VXLAN.
The other 2 properties of VXLAN are **scalability** (up to 16 M different tenants (each with a dedicated LAN segment) can be supported) and **path availability** (layer 3 computation strategies (instead of 2 which are more difficult to manage).

Another similar technology is **NVGRE** (Network Virtualization using Generic Routing Encapsulation) which consists of two headers:
- TNI (Tenant Network Identifier) 24 bit
- GRE that provides the encapsulation mechanism.

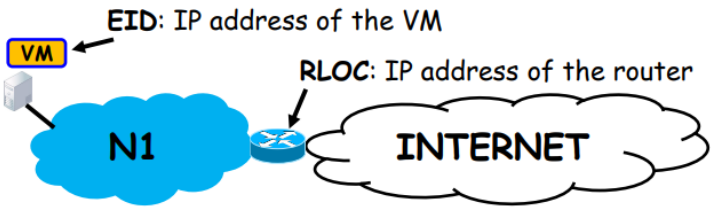
Different devices rely on these two different solutions: VXLAN (Cisco, VMware, etc.), NVGRE (Microsoft, Intel, HP, etc.).
LISP protocol

We're now in a multiple Data Centers scenario → let's now focus on the connection between a user and the DC which requires Layer 3.

Problem: Suppose an host is doing work on a VM, if for some reason (typically for optimization purposes) this VM is moved to another DC → **VM migration** → when performing a VM migration, the IP address of the VM changes; if the VM hosts a service for Internet users, the connection among users and the VM is lost.

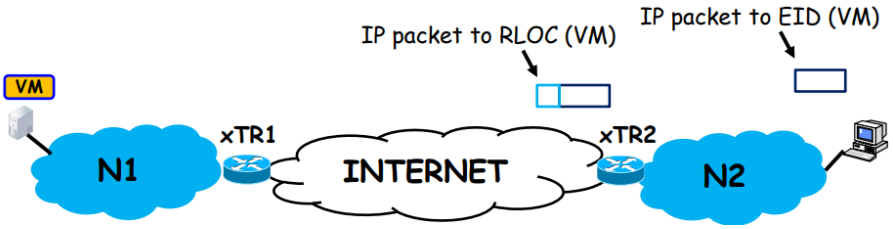
Solution: Locator/Identifier Separation Protocol (**LISP**) → allows the VM migration to happen without losing connection.

The main idea behind LISP is to define a two level IP addressing scheme:
- Endpoint Identifier (**EID**) identifying the VM (host)
- Routing LOCator (**RLOC**) identifying the location where the VM is hosted



However, when we send a pkt to a VM, we have to know only the RLOC → The association among an EID (VM identifier) and its RLOC (location identifier) is performed by routers.

Example:
Suppose you want to send a pkt from N2 to N1. In our packet there is only the IP add. of the destination VM.
When the pkt arrives at the router xTR2→ LISP encapsulation → xTR2 (xTR stands for **Ingress/Egress Tunnel router**) adds the RLOC of the router of network N1, where the destination VM is located.
xTR1 then routes the packet to EID of the destination VM.



Now, let us suppose that the Vm is going to move to a different network with a different locator → we simply update the RLOC in the routing table → basically the host is not affected by this, he is always sending a packet to the same VM.

How is it possible to have this routing table updated? We need a sort of distributed control system → the **Mapping System** (MP) is a control infrastructure that maintains the EID-RLOC mapping. An xTR query the MP through a map request for a specific mapping and receives a map reply stored in the map cache.
When a VM is relocated we only need to update its RLOC and the host is not losing the connection as he does not need to change the EID.

Another advantage of LISP is that we can substantially reduce the routing table of the internet as we only need to maintain the RLOCs rather than all the EIDs and this was the main motivation behind the use of this protocol.

An additional functionality of LISP is **traffic engineering** that consists in managing the incoming traffic when more than one paths are available to reach the xTR.

LECTURE 7 - Application Layer - HTTP

A network application is a distributed set of processes: I have processes running in different hosts, on each host I have a specific process that the application is running and the process running on different hosts communicate with each other.

Our goal will be to understand how these processes (= programs running within a host) communicate with each other through the application-layer protocol.

To use a specific application we need a **user agent** → an interface between the user and the network application.

The vast majority of applications use a **client-server approach** → the two hosts have different roles:

- the **client** contact with server requesting a service (for Web, client is implemented in browser; for e-mail, in mail reader);
- the **server** provides the requested server (typically always active waiting for requests) → e.g., Web server sends requested Web page, mail server delivers email.

The interface between application and transport layer is defined by an **Application Programming Interface** (API) → is the interface that is used by the client to request data and by the server to send it.

Question: How does a process "identify" the other process (a different process running on a different host of the same application) with which it wants to communicate?

Answer: A process running on the server is identified by:

- **IP address** of host running other process
- **Port number** → allows receiving host to determine to which local process the message should be delivered

World Wide Web

After recalling some basic concepts, let's focus on a specific application → World Wide Web → the application that provides, to users, web pages → through this application you can have access to a huge amount of web pages.

A **web page** is a set of objects (text, images, etc.) organized in the **HTML** (Hyper Text Markup Language) language and reachable through its **URL** (Uniform Resource Locator).

The URL is composed of:

- **hostname** → allows to identify the server where the web page is stored (red)
- **pathname** → allows to identify where the webpage is located inside the server (blue)

www.someSchool.edu/someDept/pic.gif

MEMO: the number of objects a page is composed of determines the number of messages between the client and the server.

The user agent for the Web is called **browser** (Chrome, Firefox ecc…).

What about the communication client-server in the WWW?

Communication takes place via the HTTP protocol (that relies on a TCP protocol connection):

- client initiates TCP connection (creates socket) to server (port 80) → handshaking
- server accepts TCP connection from client
- http messages (application layer protocol messages) exchanged between browser (http client) and Web server (http server)
- TCP connection closed

Remember that HTTP is **stateless** → means that server does not maintain any information about past client requests.

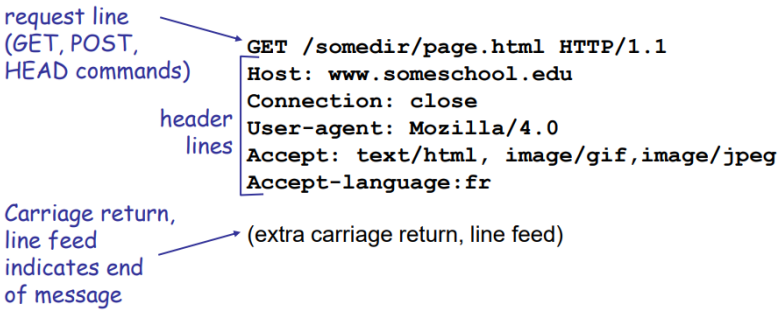
Let's see more in detail how this communication takes place → there are two different ways in which this communication can be:

- **Non-persistent** → you have to open a tcp connection each time you're asking for a new page → server parses request, responds, and closes TCP connection → 2 RTTs to fetch each object + each object transfer suffers from slow start.
  Not so efficient → can be solved by using **parallel** TCP connections

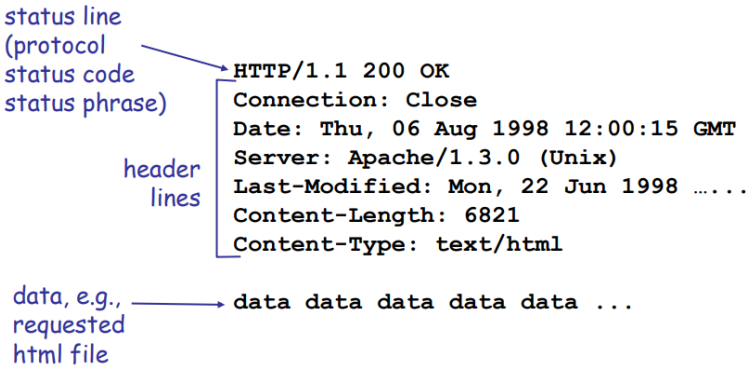
- **Persistent** → the server leaves the TCP connection open after sending responses and hence the subsequent requests and responses between the same client and server can be sent.
  The server closes the connection only when it is not used for a certain configurable amount of time.
  All requests are handled on the same TCP connection, the client sends requests for all objects as soon as it receives base HTML → fewer RTTs and less slow start.

HTTP has two types of messages, request and response which have the following structure:

- **Request:**
  - request line:
    - the command GET to specify what kind of request you are doing → get means "I want a web page".
    - the path of the page
    - the type of http → in this case 1.1
  - header lines:
    - host → the name of the server
    - connection → the kind of connection you want → "close" if you want a non persistent connection
    - user-agent → which browser you're using
    - the type of the object you're going to accept
    - the language you want



- **Response:**
  - status line, where we have 2 informations:
    - http type → 1.1
    - type of response → 200 is the code to indicate that the response is the correct one (if there are anomalies the reply is different)
  - header lines:
    - connection type → identical to the request
    - date
    - server → which server is replying
    - when the content has been modified last time
    - content length (in bytes)
    - content type



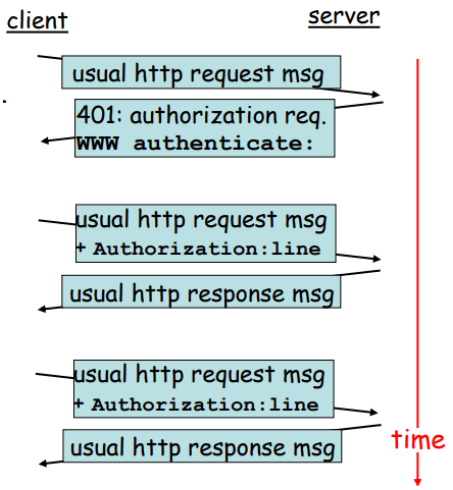
We saw that In the first line in the server→client response message there is a code. In the case we've seen there is 200 but we have different types of codes:

- 200 OK: request succeeded
- 301 Moved permanently: request object moved → new location specified later in this message (Location)
- 400 Bad Request: message not understood
- 404 Not Found: document not found
- 505 HTTP Version Not Supported

User-server interaction: <u>Authentication</u>

Some pages require **authorisation**, typically username and password.
Since the http protocol is stateless, clients must present authorization in each request.
A line is added to the header lines → **`authorization`**: header line in request.

If no authorization is presented, the server refuses access and redirects to the access page → add the **`www authenticate`**: header line in response



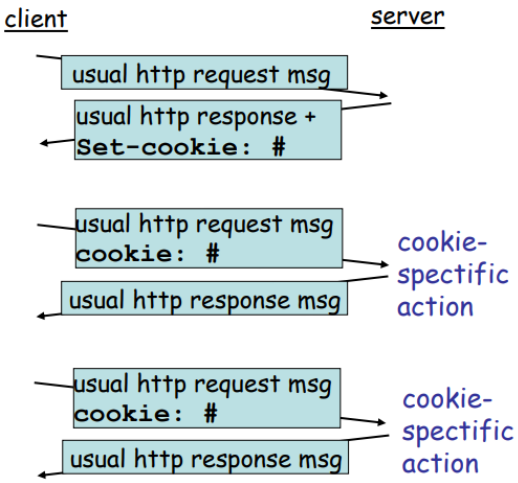
Normally, the browser caches name & password so that the user doesn't have to input it every time.

User-server interaction: cookies

The server can provide customized content to the user by requesting **cookies** to the client → basically it is an identification number.

The client makes a request; the server responds in the manner we have seen but adds **`Set-cookie`**: 1678453 → Server sends “cookie” to client in response msg, which is basically an identification number.

The server asks the client to maintain that number for all the successful requests, so the server is able to associate to you all the activities you’re doing.

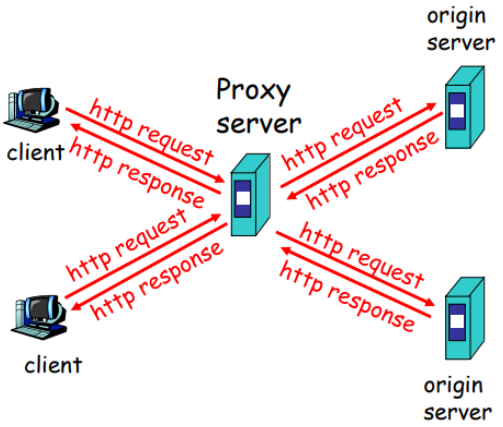


User-server interaction: conditional GET

Goal: don’t send objects if the client has up-to-date stored (cached) version.

Client: specify date of cached copy in http request → **`If-modified-since: <date>`**
Server: response contains no object if cached copy upto-date.

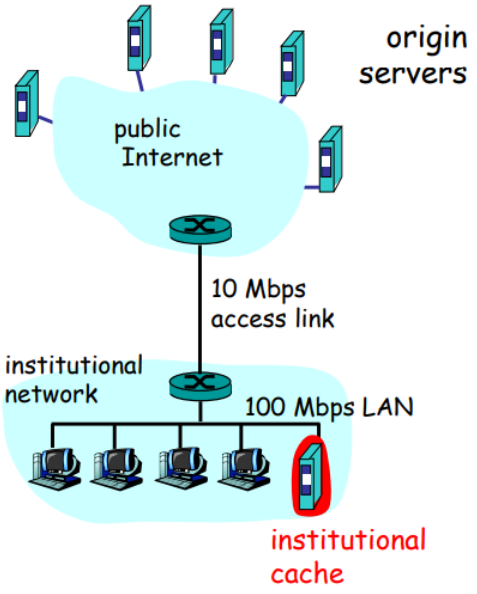
Another feature of the http → **Http proxy (**also known as **http cache)**→ is a device which is set in the middle of a set of clients and a set of servers.



How does the proxy work? Instead to establish a connection between a client and a server we put a proxy server → it aims to satisfy client request without involving the origin server, so that if the requested object is cached within the proxy it sends it to the client (**web caching**); otherwise it asks the object to the origin server.

So we have two different connections → one client - proxy and another proxy server - origin server.

Why web caching? What is the advantage to having that proxy server (institutional cache)?



Let us suppose that you ask for a page that someone else required before. We have two advantages:

- improving the performance → smaller response time (if the cache “closer” to the client) + decrease traffic to distant servers
- provide a sort of security level → you can trace all the activities done by all the clients → if you want to block a specific web page , you can implement a simple rule into the cache and avoid the connection with that page.

LECTURE 7 - Application Layer - DNS

The main aim of DNS (Domain Name System) is to identify the web servers → each time you ask for a specific web page you need to know the web site, and so the web server.

A DNS maintains a mapping between IP addresses and (host)names, it is composed of:

- distributed database, organized in a hierarchical way, of many DNS servers, also known as **name servers** (remember that a name server is a device of the DNS architecture → DNS distributed database is composed by many name servers)

- application-layer protocol to interact with the database which contains the addresses informations → host, routers and name server communicate to "resolve" names (address translation)

Why a "distributed database"? This system has been decentralized for two reasons:
- **robustness**: to avoid a single point of failure
- **scalability**: to reduce the amount of traffic and the size of the database it would be necessary to maintain

What does "organized in a hierarchical way" mean? We have 3 kind of name servers:

- **local name servers**: receives requests from hosts to "resolve" names (try to reply to this query) stored in its cache → each time we use our pc to search a web page, we're asking the name server the address of that specific web page.

- **authoritative name server**: responsible for storing the associations of IP address of the machine you're using and name of the web server → can perform name/address translation for the host → through the information in this server, the local server is able to reply to the client.

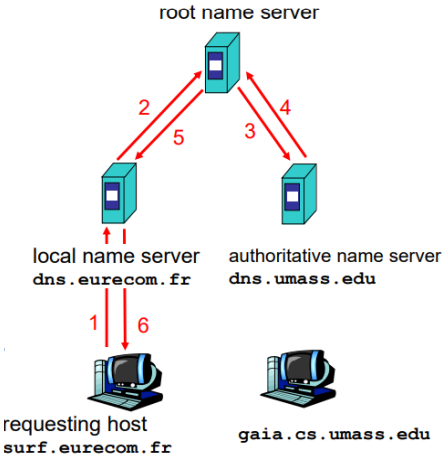
- **root name server**: are the servers with the highest hierarchy degree because are the devices responsible for the communication between local and authoritative name servers.
  There are 2 different kind of services they can provide:
  - they can directly provide the association between a name and an IP address → solving the mapping problem
  - they know what is the authoritative server where the information is stored → when the local name server is not able to resolve a name it forwards the message to the root name server

  Basically they know the information or where to find the information.
  There are 13 designated Root name servers worldwide.

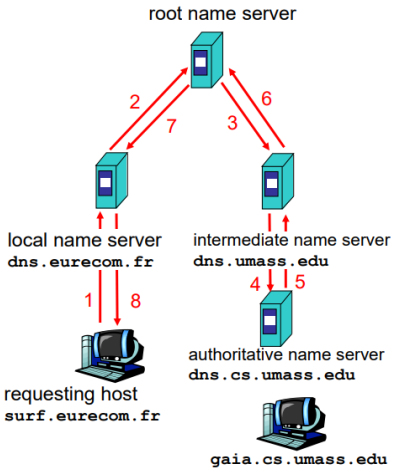
MEMO: A host can only communicate with the local name server.

Example:
- Host **`surf.eurecom.fr`** wants the IP address of **`gaia.cs.umass.edu.`**
- Host contacts its local DNS server, **`dns.eurecom.fr`** → how the host knows what is its local DNS server? Is an information which is obtained exploiting the DHCP protocol → the host must know this information.
- **`dns.eurecom.fr`** contacts the root name server (if necessary).

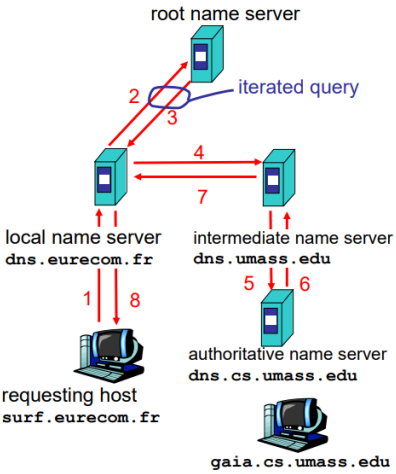
- Root name server contacts authoritative name server, **`dns.umass.edu`** (if necessary).



- There is a particular case in which the root name server doesn't know the authoritative name server → the r.n.s. contact the **intermediate name server** → who to contact to find an authoritative name server.

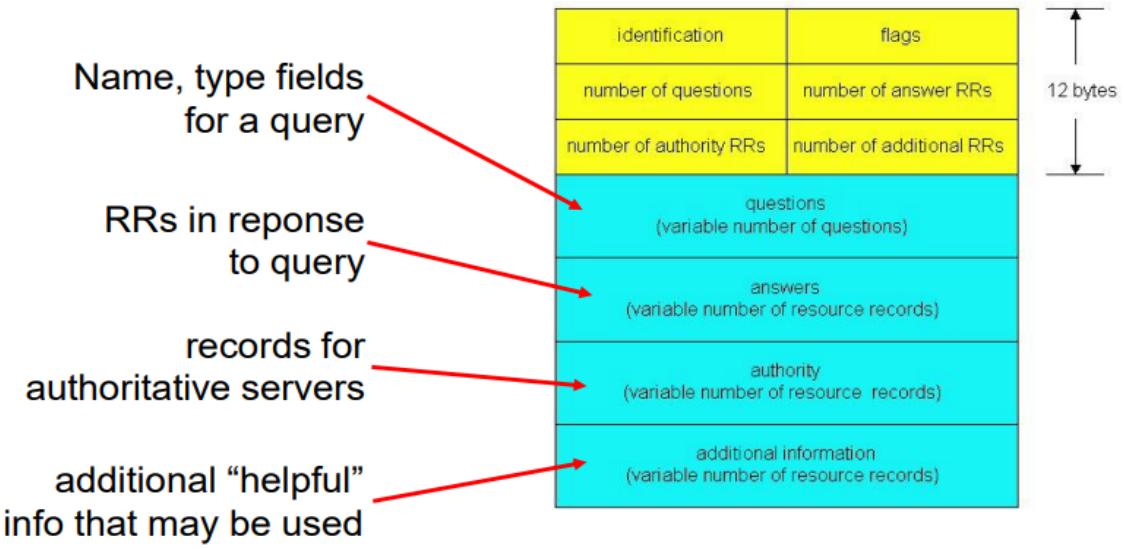


- The problem with this approach (**recursive query** → puts the responsibility of name resolution on the contacted server) is that now the root name server has to communicate with both the local and intermediate server → heavy load!
- The solution is to have the root name reply to the local by saying which intermediate server should contact to resolve the target name → **iterated query** → contacted server replies with name of server to contact: "I don't know this name, but ask this server".



DNS records → DNS distributed database stores resource records (RR) with the format → (name, value, type, TTL).
The type value determines the content of the name and value fields.

DNS protocol → query and reply messages have both the same message format:



LECTURE 8 - SDN

The IP protocol can be seen as the bottleneck in a network, though being quite old it is hard to modify. Moreover, network devices have different configuration procedures, one for each network function. The introduction of a new protocol would require updating the device OS or to replace old devices. This led about 10 years ago to the Clean State project which was run to answer the question "how would we build the internet if we started from scratch today?".

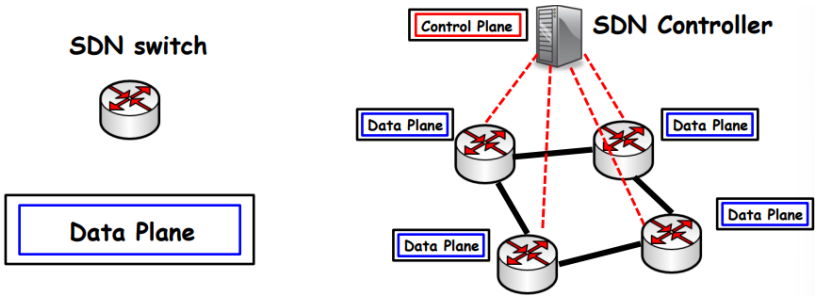
A usual legacy network device can performs:

- **Data plane** actions (forwarding of packets)
- **Control plane** action (protocols → network intelligence)

The idea behind **SDN** (Software Defined Networking) is to decouple these two parts:

- the control plane is logically centralized
- the data plane is programmable and abstracted from the control plane

The control plane software runs on general purpose hardware → the **SDN Controller**, which controls and programs the SDN switches, executing Network Applications decisions.



NB: Routers are now referred to as SDN **switches** to highlight that they're simpler devices, in fact they only contain a routing table and should be able to forward packets.

Let us look in detail at the two types of devices that use this protocol.

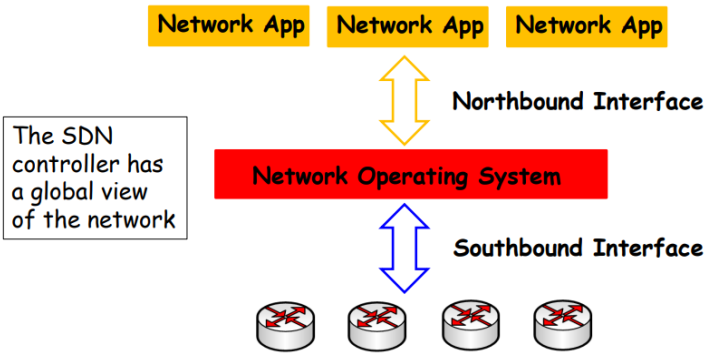
SDN Controller

The SDN controller runs on general purpose hardware and is responsible for communicating with all the switches and updating their routing tables.

The two main components of the network are:

- **network operating system**: the SDN controller is the intermediate layer between network data plane devices through two interfaces:
    - **southbound** interface → the NOS uses it to configure the switches (writing routing tables), and SDN switches use it for notification → **OpenFlow** (developed at Stanford, is the southbound interface of an SDN architecture → provides communication between the control plane and data plane).
    - **northbound** interface: necessary for the network applications to communicate with the SDN controller
- **network applications**: any application related to the network (routing, traffic, engineering, etc.)

With this structure if we wish to add a new functionality to the network we just need to write the network application and connect it to the controller (**fully open and programmable** network) → no need to touch SDN switches!

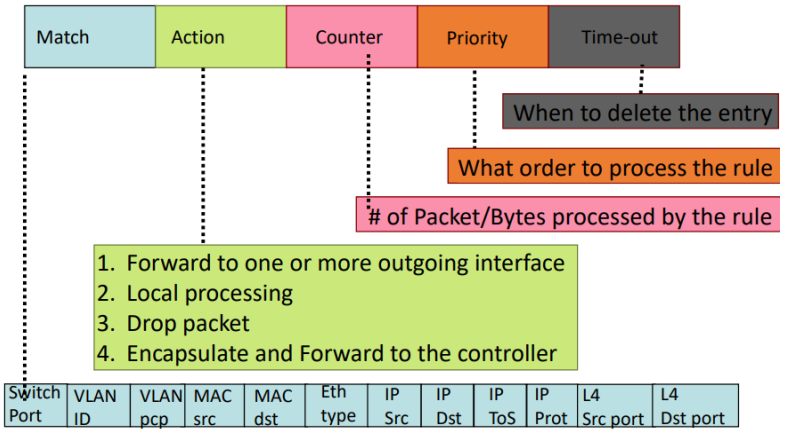


SDN Switches

The SDN switch is a forwarding device containing a **flow table**.
The main difference from a normal router is that, in addition to the destination address, it can also examine other header fields, including levels 2 and 4.
This allows greater precision in deciding how to forward packets.

OpenFlow: a Flow Table Entry

A Flow Table Entry specifies the data plane actions (Forwarding) to perform on an incoming packet belonging to a specific flow → in this protocol a switch can perform different actions as seen in the picture.



A flow table looks like this:

Port	Src MAC	Dst MAC	VLAN ID	Priority	EtherType	Src IP	Dst IP	IP Proto	IP ToS	Src L4 Port ICMP Type	Dst L4 Port ICMP Code	Action	Counter
*	*	0A:C8:*	*	*	*	*	*	*	*	*	*	Port 1	102
*	*	*	*	*	*	*	192.168.*.*	*	*	*	*	Port 2	202
*	*	*	*	*	*	*	*	*	*	21	21	Drop	420
*	*	*	*	*	*	*	*	0x806	*	*	*	Local	444
*	*	*	*	*	*	*	*	0x1*	*	*	*	Controller	1

Here is a list of the possible OpenFlow messages:

- **Controller → Switch:**
  - Flow_mode: to add/delete/modify an entry in the Flow Table
  - Read-state: to collect statistics about Flow Table, ports and flows (used collect data from the switches to implement advanced network applications)
- **Switch → Controller:**
  - Packet_in: when there is not matching entries in the table
  - Flow_removed: to notify that a flow entry expired (TimeOut field in the table entry)
  - Port-status: port configuration or state changed

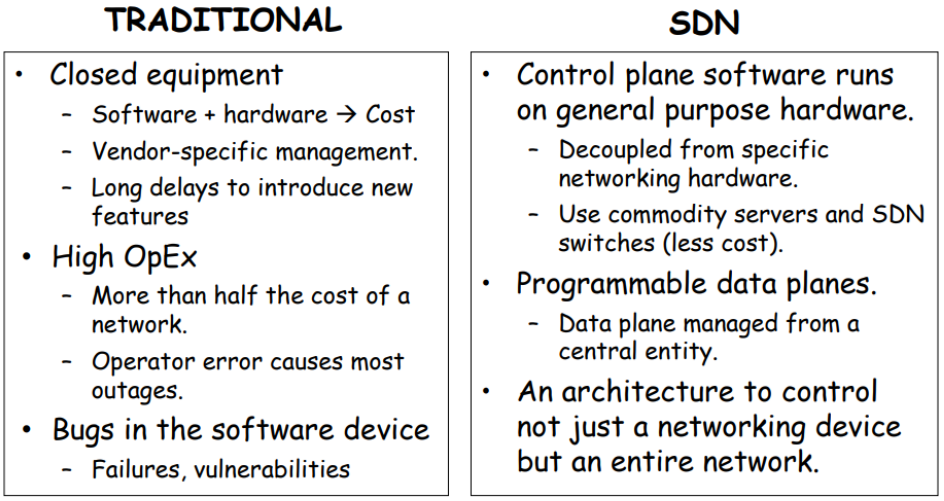
Entries installation strategy

There are two ways in which the SDN controller can manage the Flow Table of a switch:

- **Reactive** approach → inserts a new line only when it receives a packet from a new stream.
  It involves an overhead time for forwarding packets, as there is a setup time for the new flow.
  It is a good approach when the main goal is to reduce the size of tables → I have in my table only the rows I really need.

- **Proactive** approach → flow entries are pre-installed by the controller in the switches
  It is necessary to know all future traffic patterns since creating an entry for each possible flow would result in too many combinations.
  There is no setup time but the flow tables size could significantly increase.

To summarize, the differences between the traditional approach and the use of SDN are:



Other issues in SDN networks

There are basically 3 kind of issues with SDN:

- <u>The control plane bottleneck</u>
  The Controller is a bottleneck of the SDN infrastructure for two reasons:
  - Scalability

    - Security

  However, a single controller is not enough to manage a large scale network → use of Multi Controllers in large networks.

  Open questions:
  - Number of controllers → How many controllers are needed?
  - Assignment of switches to each controller → How to assign switch management to each controller?
  - synchronization of controllers → How to realize the communication among controllers?

- The switch's Flow Table size →the flow table is more complex than an IP routing table since:
  - matching is possible in many different fields.
  - acts on flows → there is an high number of flows in a large scale network
  - there are more tables in the new versions of OpenFlow

  Moreover, the Flow Table is realized using memory technology for high speed searching applications (**TCAM**), but this has a limit of 1k entries → not enough to cover all possible flows in a medium-size network.

  Open questions:
  - Do we have to wait for a TCAM (or new) technology evolution?
  - Do we have, instead, to try to optimize the assignment (in terms of flow table size minimization) of flow table entries to network switches?

- The communication channel switches/controller
  The original idea was to have a dedicated channel to enable communication between the controller and the switches, even if there are no specifications about how to implement them. This is not feasible in practice, so the same infrastructure as the data plane is used (→ a single physical network).

  Open Questions:
  - How to solve congestion problems? → need for congestion control mechanisms → we should introduce a sort of **priority** in the network to determine which packages to give priority to.

LECTURE 8 - NFV

NFV is an initiative to virtualize network services, so that instead of having a specific machine for a service, we have a virtual machine for the service that can run on generic hardware.

There are many benefits to this approach → the most important are:
- **Flexible provisioning**: no need to physically install the service onto the device, but can be run whenever the hardware is available
- **Mobility** (migration): can move the virtual machine across the network (for instance based on traffic)
- Automation, hardware independence, fault tolerance, etc.

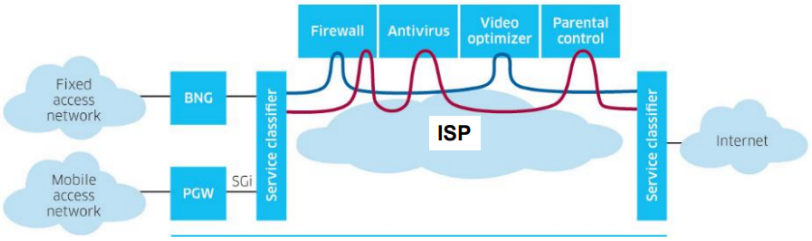
Everything is great but **how** to implement this idea? We need something called **Service Function Chaining** (SFV) → is simply a different way to look at the path in the network.

Until now we said that in the network some paths are defined and the traffic crosses the network following these paths → what is the different approach of SVC?

The idea is that the path of the traffic in the network can be translated as a chain of services (virtual machines/identifiers) through which it must pass.

Let us suppose that we have 2 different kinds of flows in my network (red and blue).
The blue one is generated by a user that is asking to have a firewall and a video optimized while the red traffic requires a firewall, an antivirus and a parental control.

As an internet service provider, I'm defining the path inside my network so that the red traffic is going to cross the virtual machine of the firewall and the VM of the video optimizer while the red one is going to cross the VMs of the firewall, antivirus and parental control.



Of course I want the fastest path → I need to know where the VMs I'm interested in are .
Instead of using the classical IP destination address, I'm inserting a sort of new header (**NSH** → Network Service Header) exploiting an encapsulation mechanism in which I'm inserting the identifier of the VMs I have to cross.

So, before forwarding the pkt, I have a classifier which is going to classify the traffic to understand which are the services required. Then it inserts VMs to cross → each NF has its own ID and the SFC is defined as a list of IDs to be crossed.

In conclusion, we can say that:

- NFV and SDN are independent and complementary.
- SDN is a network solution providing a **high flexibility** in routing compatible with NFV Service Chaining