Redes de Computadores (RCOMP)

Lecture 02

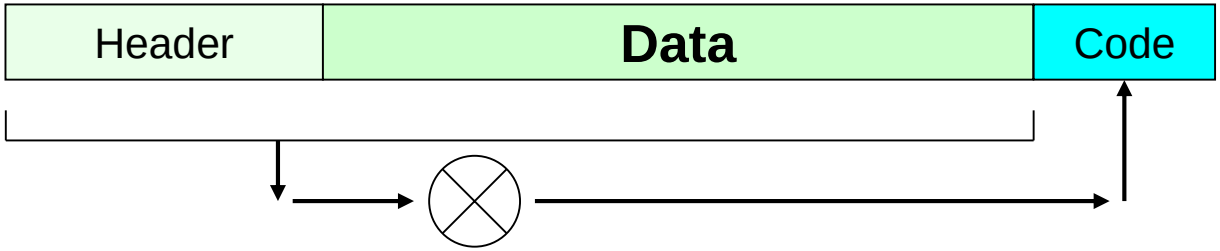
2017/2018

- Error detection.
- Network delays.
- Flow control.
- Error control.
- Network architectures.
- The OSI model.
- IEEE 802 Architecture (ISO 8802) .
- TCP/IP Architecture .
- The client-server model.

Error detection code

While a packet is being transmitted, it will not be unusual one symbol getting a wrong interpretation at the receiver. Errors may occur, part of the received data may be different from the originally sent.

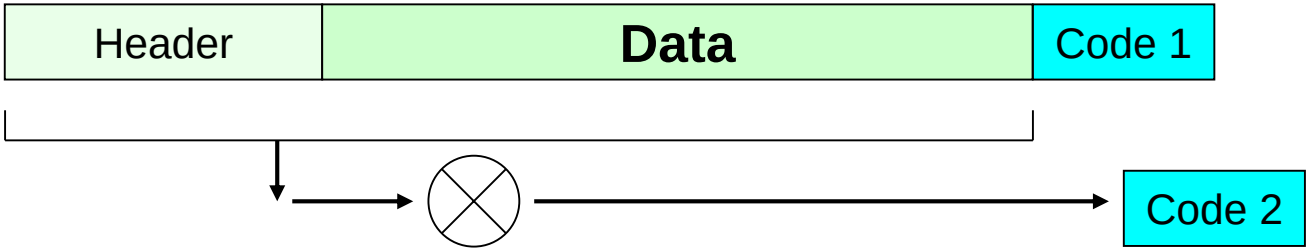
These errors may be detected if the sender adds a validation code to the packet. The code is generated by an appropriate function taking the content of the package as input and producing a small fixed size code representing the content. This function will produce a different code if there is any small change in input data.



The sender uses the function to calculate the code, and then, adds it to the packet. In some cases, the code is added to the packet header, but in can also be added to the tail as presented on the image above.

Error detection

The code produced by the sender is sent along with the packet. This gives the receiver the ability to repeat the process and confront the two codes it will then have:



At the image above, **Code 1** was calculated by the sender and **Code 2** by the receiver.

If the two codes differ, then the receiver knows **there has been an error**. If the two codes are equal, the receiver knows **probably there was no error**.

This method never absolutely guarantees there was no error. There is always a remote chance that a random change in the packet due to transmission errors may yet result in a packet and code that match each other. Though this is highly unlikely to happen.

Forward Error Correction (FEC)

Detecting errors is just the first step, once detected, something must be done about that. Error correction or error control handles the problem once an error is detected.

One approach for error control is **FEC**, it requires the use of a more extended validation information (not a simple code) and different functions to handle it. For errors with up to some extension, FEC allows immediate and autonomous correction by the receiver.

The other approach, **Backward Error Correction** (BEC) requires the receiver to ask for the packet to be retransmitted. This is also known as Automatic Repeat Request (ARQ).

Due to this dialogue, the network delay between the sender and the receiver have a significant impact on BEC efficiency.

FEC, however, is not a universal solution, it only solves errors if they are a limited to a short number of bits. When used, FEC must be combined with BEC for the cases where errors are too extensive.

Network delays - packet delivery time

Transferring a packet from the sender to the receiver through a network takes some time.

The total time for the transfer to be fully accomplished is called **packet delivery time**. It measures the elapsed time since the sender emits the packet first bit until the packet last bit arrives at the receiver.

The **packet delivery time** is the sum of two parcels:

- **Packet transmission time** – this is the amount of time it takes to emit or receive a given size packet at a given data rate.

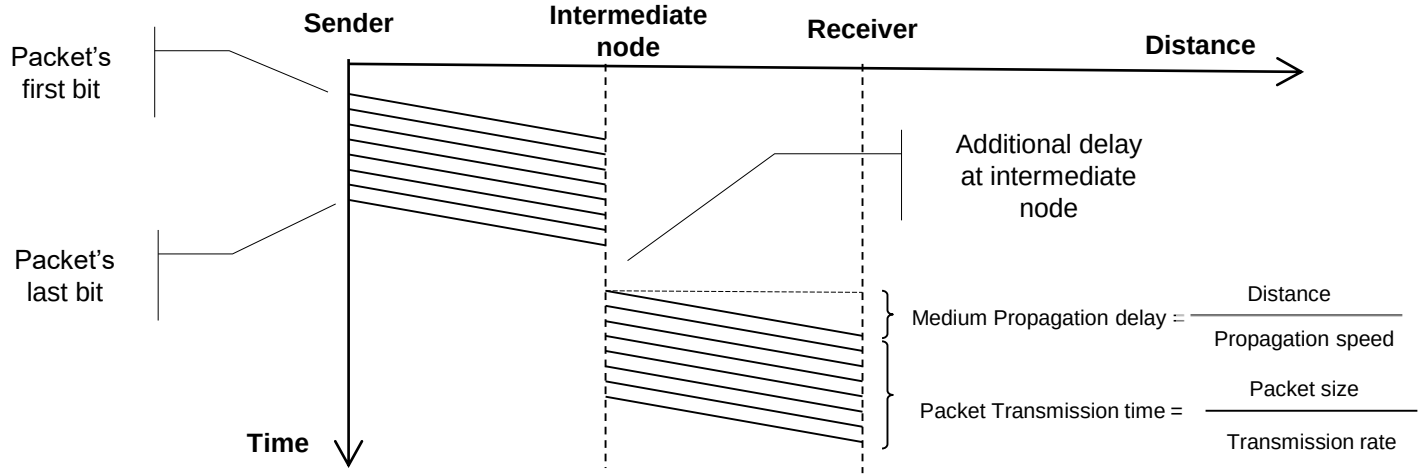
  Packet transmission time = (Packet size) / (Data rate)

- **Propagation delay** (network latency) – this is the amount of time it takes for one bit/symbol to travel through the network from the emitter to the receiver. If sender and receiver are connected to the same transmission medium (**no intermediate nodes**), then:

  Propagation delay = (Distance) / (Signal propagation speed)

Network delays – network latency

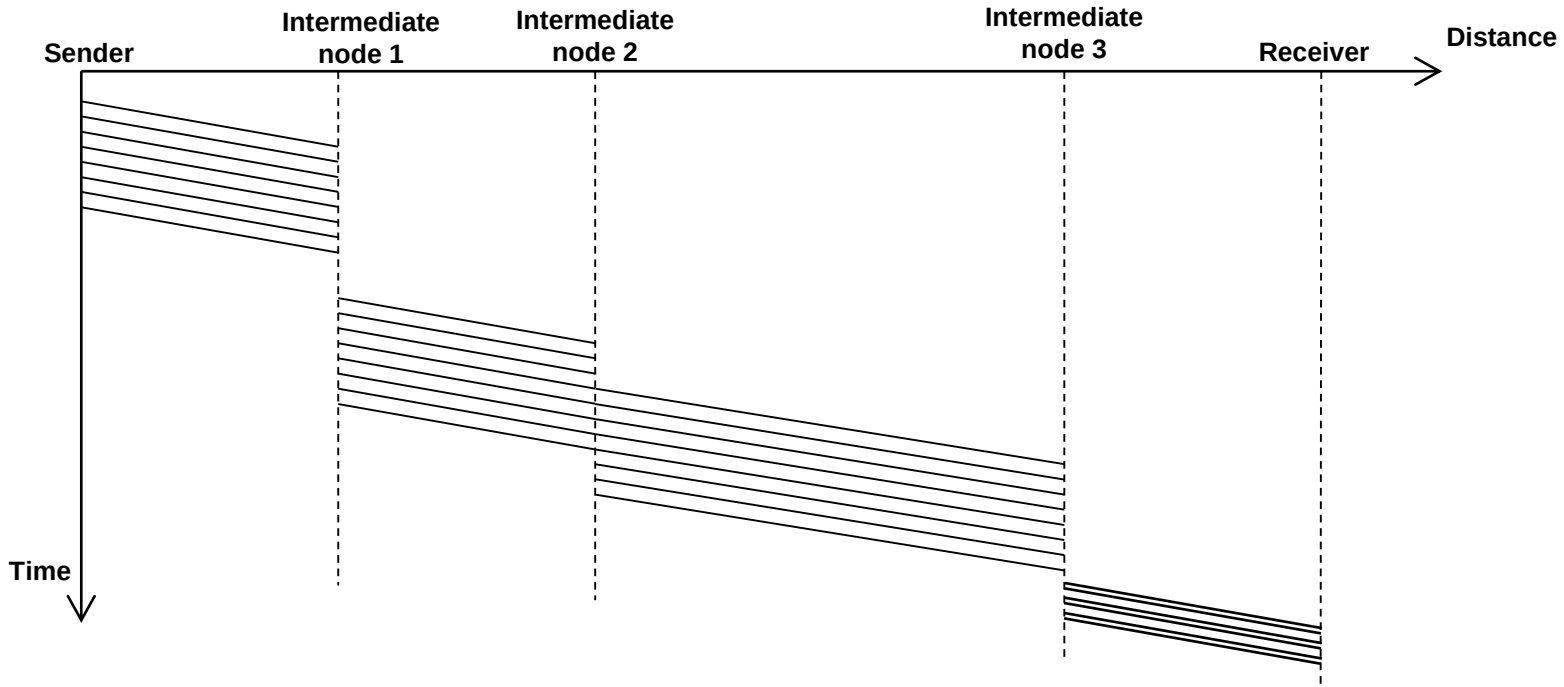
In a switching network, there are intermediate nodes between sender and receiver, here things get far more complex. Each intermediate node will add more time to network latency (propagation delay), it also receives and emits the packet, so the corresponding transmission time may have to be added.



As we can see on the image above, the latency added by an intermediate node equals the **packet transmission time**. However, intermediate nodes may not be able to send the packet as soon as it's received, this introduces an **additional delay**. Intermediate nodes use queues (FIFO) to store received packets before emitting them, these queues operate as buffers and are required because the intermediate node may not be able to emit packets at the same rate they are being received.

Network delays – network latency

Intermediate nodes may introduce other features affecting network latency calculation, the graphic represents a packet path through a switching network:



Intermediate nodes 1 and 3 fully receive packets before starting to emit them, this is called **store & forward**. Some types of networks allow, however, another operation mode called **cut-through**, that is the case of intermediate node 2. Another factor to take account for is intermediate nodes may receive and send at different data rates, that is the case of intermediate node 3.

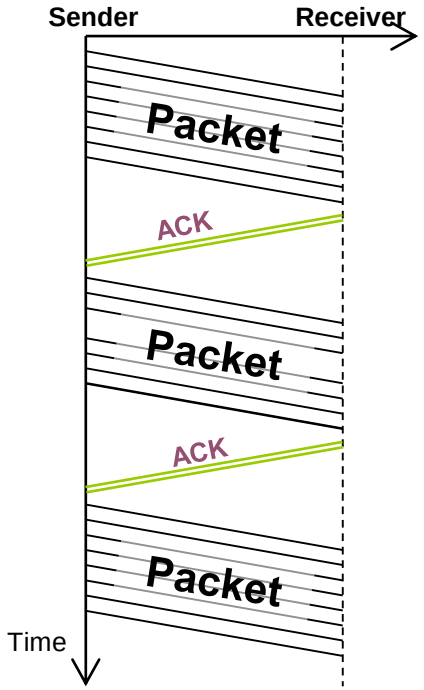
Flow control

Flow control objective is regulating the data flow between sender and receiver avoiding an overflow at the receiver (more packets than it can cope with).

The best way to achieve this is letting the receiver control the flow. In the technique called **stop and wait** the sender is forced to wait for the **ACK signal** from the receiver before sending the next packet:

The image on the right shows how inefficient this flow control technique may become due to propagation delays. The network may be idle, nevertheless, the sender cannot use it until the ACK is received.

To overcome this **stop and wait** flow control issue, a variant known as the **sliding window protocol** exists. Instead of the emitter having to wait for an ACK before sending the next packet, it can send a burst of W packets. Parameter W is the **window size** and is configurable.



Flow control – sliding window protocol

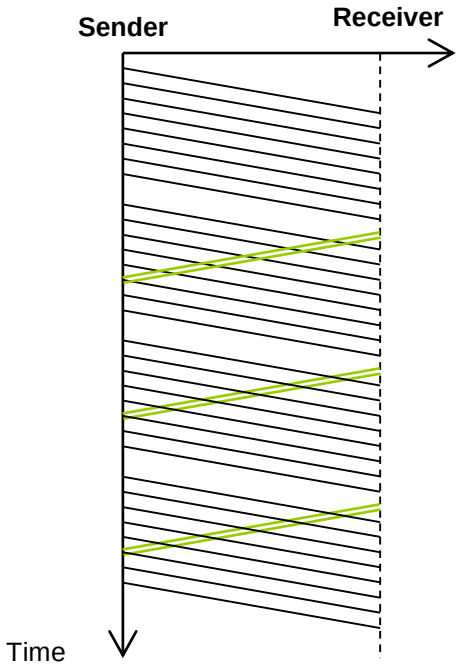
With the sliding window protocol the sender is allowed to send W packets (window size) without receiving any ACK from the receiver:

After sending W packets the sender then must wait because the window is exhausted, however, for every ACK that meanwhile arrives from the receiver, an additional packet can be sent.

Under ideal conditions (appropriate W value), there are no stops in packet transmission. This will happen if the ACK for the first packet arrives before W packets are sent.

It's important to point out that the absence of stops (100% efficiency) is possible only in a full-duplex link (simultaneous transmission in both directions).

Switching networks are inherently full-duplex because even if some links are half-duplex opposite direction packets can cross each other at intermediate nodes.



Error control

The purpose of error control is correcting errors found by error detection. Although self-correcting mechanisms are available (FEC - Forward Error Correction), in most cases the receiver will ask for the retransmission of another copy of the packet (BEC - Backward Error Correction).

Retransmission requesting is known as ARQ (Automatic Repeat Request), it may be implemented together with flow control. For this purpose, the receiver can reply to the sender either with ACK or NACK. The NACK signal means an error has been detected, and as such, the packet in reference must be retransmitted.

When used together with the sliding window protocol, this retransmission request mechanism, becomes known as **continuous ARQ**.

To use the sliding window protocol packets will have to be tagged (usually with a sequence number). This is required because in case of an error the receiver will have to tell the sender which is the packet to be retransmitted. For instance, the receiver will send NACK 5 to request the retransmission of packet number 5.

Network Architectures

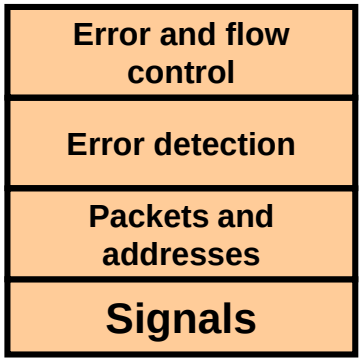
Communication between applications located in physically apart systems is a complicated process, involves many issues that must be addressed.

Due to this complexity, since the dawning of computer networks, in the 70s, a strategy of successive modules was adopted, usually designated layers. Each layer solves one part of the problems.

We call network model or architecture to how layers are organised and how they interact with each other.

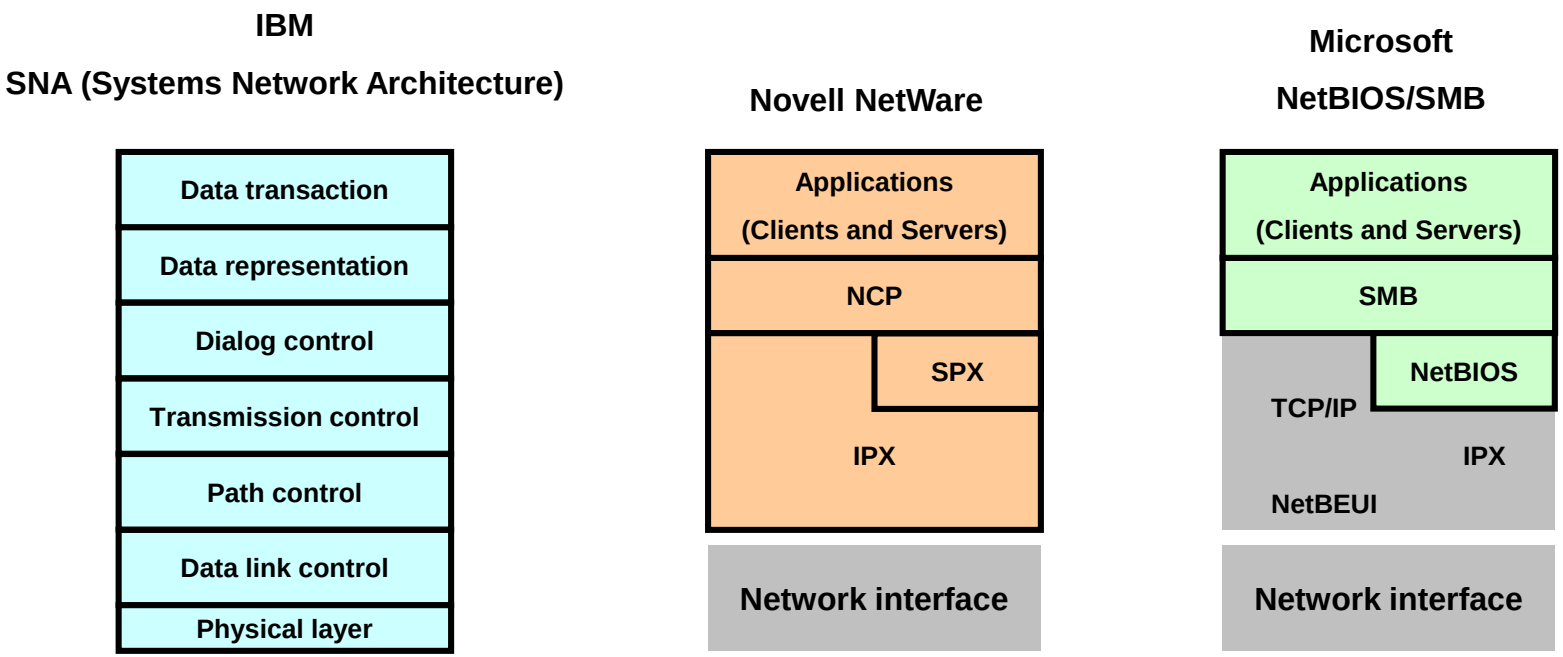
The main principle is, each layer uses the layer below and adds new features that will be made available to the layer above.

The image on the right summarizes the main issues addressed so far.



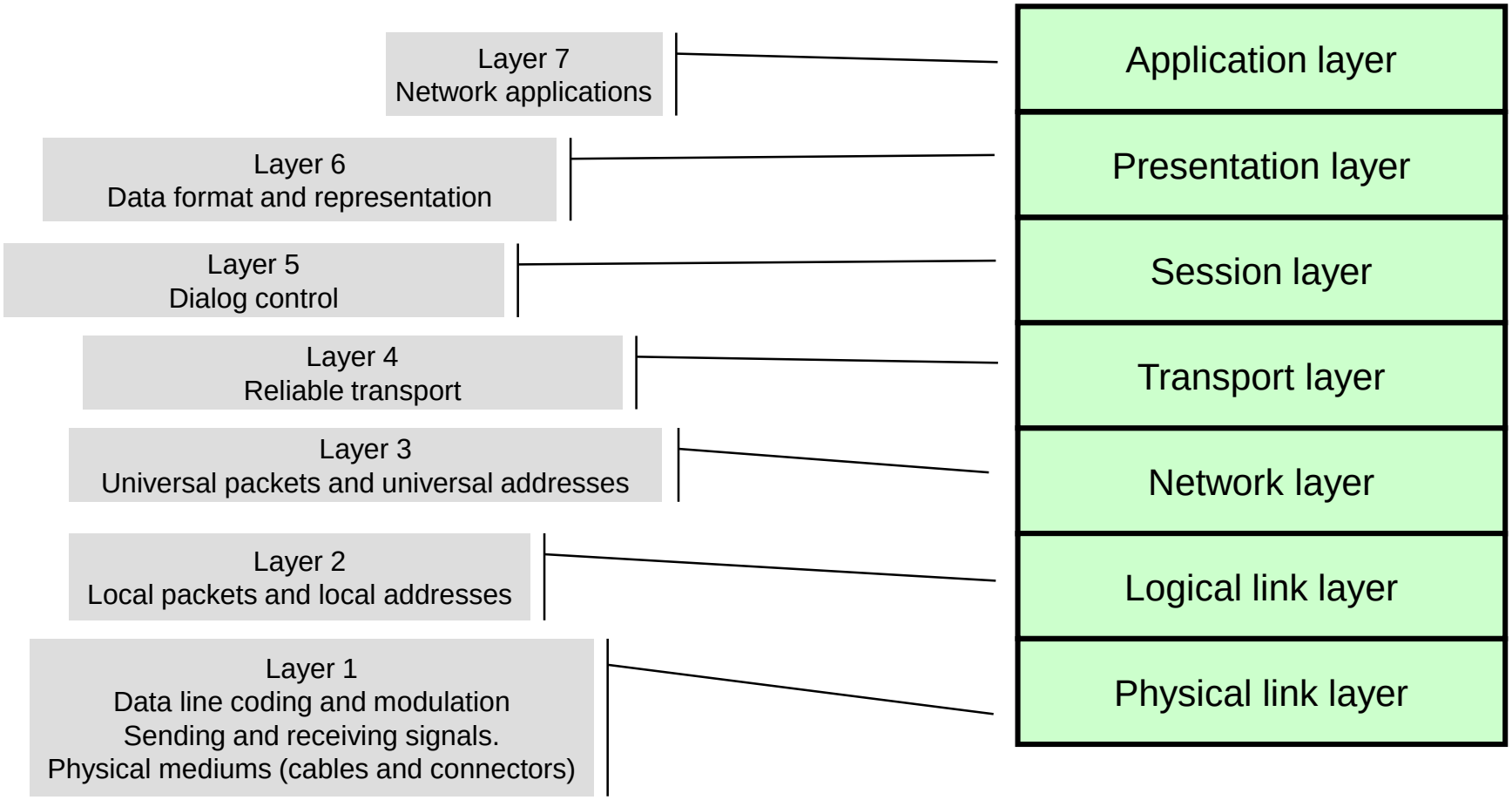
Proprietary Architectures

Computer networks have started to arise spontaneously in the early 70s. At that time each manufacturer developed its own closed system, following a patent culture to avoid being copied by others. These architectures are known as proprietary architectures, some of them are:



The OSI model

In an effort to normalize network architectures, ISO (International Organization for Standardization) established the OSI (Open Systems Interconnection) model. This model describes seven layers:



The OSI model - layers

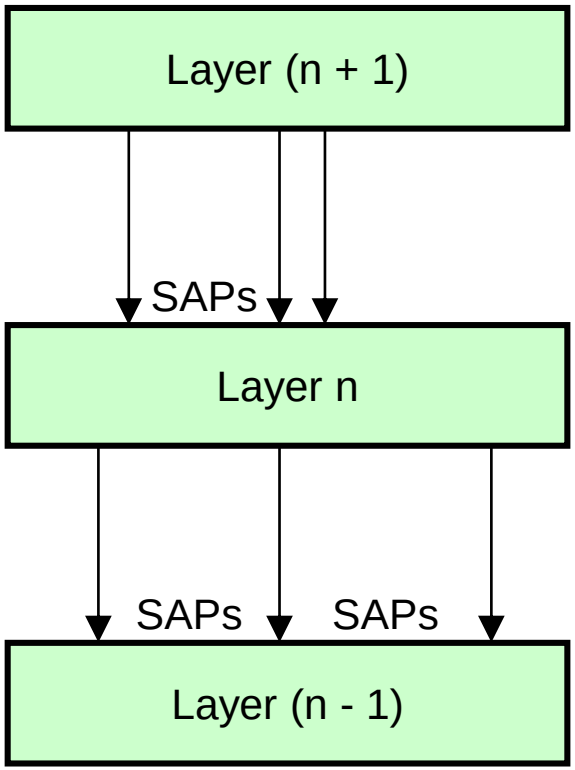
Successive layers of the stack interact with each other according to a downward service call model by service access points (SAP).

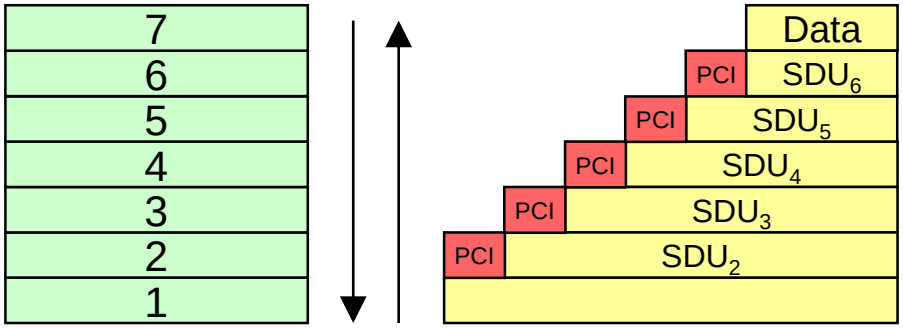
Each layer uses the services provided by layer below and adds them new features.

Typically the new features implemented by each layer require the addition of control information (PCI - Protocol Control Information).

PCI is added to the data (SDU - Service Data Unit) coming from the upper layer.

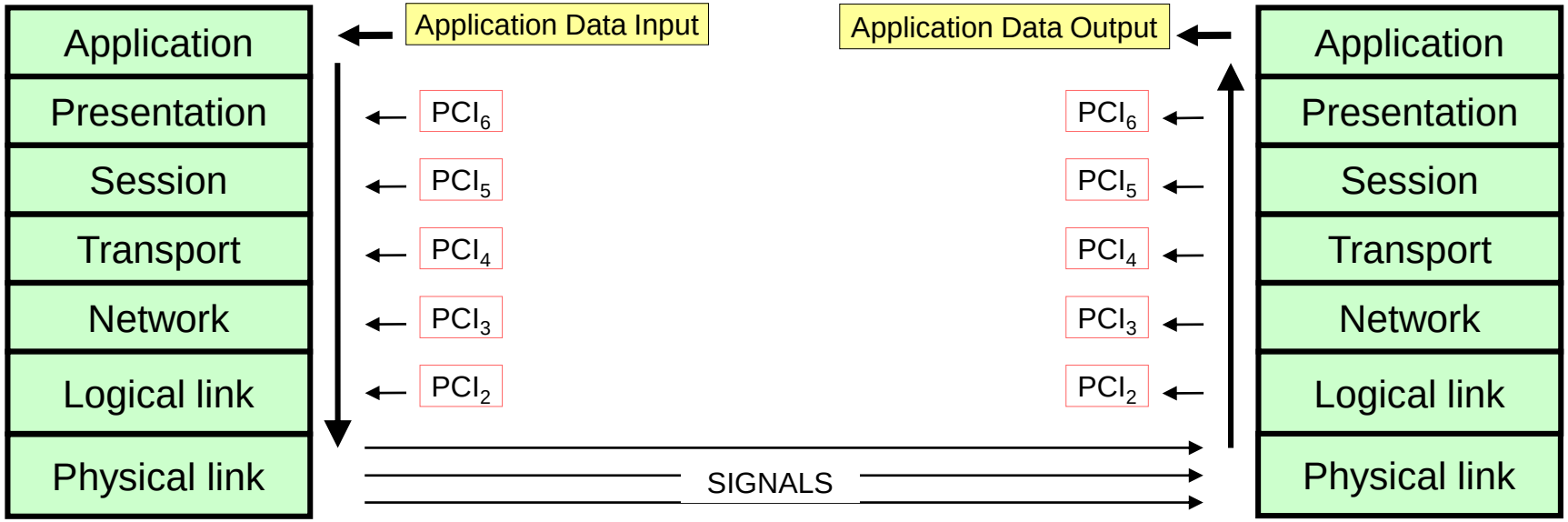
In each layer, the PCI + SDU assembly is referred to as PDU (Protocol Data Unit). The packet.





The OSI model - protocols

Direct interaction occurs only between successive layers and in the physical layer. However, the same level layers, hosted in different network nodes will communicate with each other using the PCI of that layer.



The exchange of information via layer PCI is related with that layer specific features and follows a set of rules known as protocol. Thus, **each layer has its own protocol**.

The OSI model as a reference

The main objectives of OSI were never totally achieved, in large part, this was due to the enormous complexity of developing an open model able to cover all the possibilities on one side, and also the reluctance of established proprietary architectures to give away done investments.

While under the open system interconnection point of view it has not been a success, the OSI model was a very important step because it includes a set of standards, nomenclature, techniques and ideas that have become a reference point for any discussion in computer networks area. Thus, it is frequently referred to as **OSI reference model**.

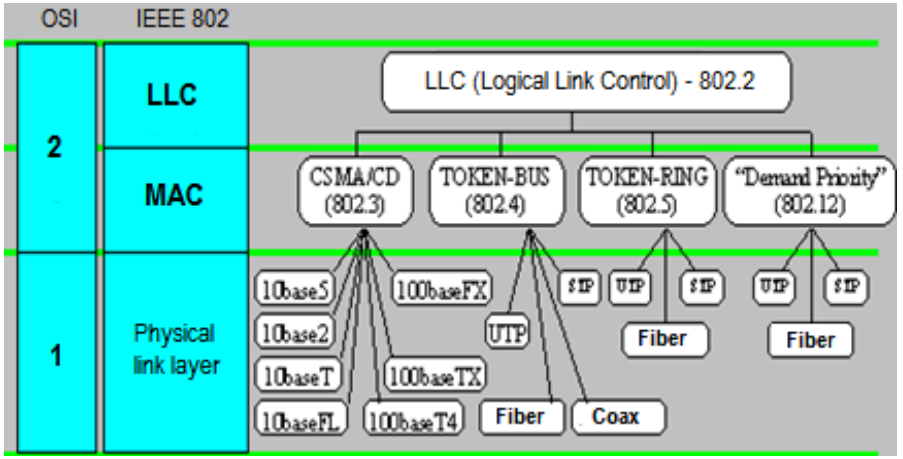
OSI main motivations: unite all networks architectures to allow all nodes to talk to each other, was ultimately achieved by another architecture: TCP/IP.

TCP/IP architecture achieved this by becoming global through the internet, by doing so, once all users start requiring an internet connection, manufacturers were compelled to adopt it.

The IEEE 802 Architecture (ISO 8802)

Most LAN technologies are standardized by IEEE and ISO, these technologies match layers 1 and 2 of the OSI model.

Each standard is identified by numbers and letters, for example Ethernet networks have the identifier IEEE 802.3 (ISO 8802-3). Whenever there are technical developments in these standards additions are made identified by lowercase letters.



Ethernet networks at 100 Mbps are defined in the 802.3u standard and the same Ethernet networks at 1 Gbps in the 802.3z standard.

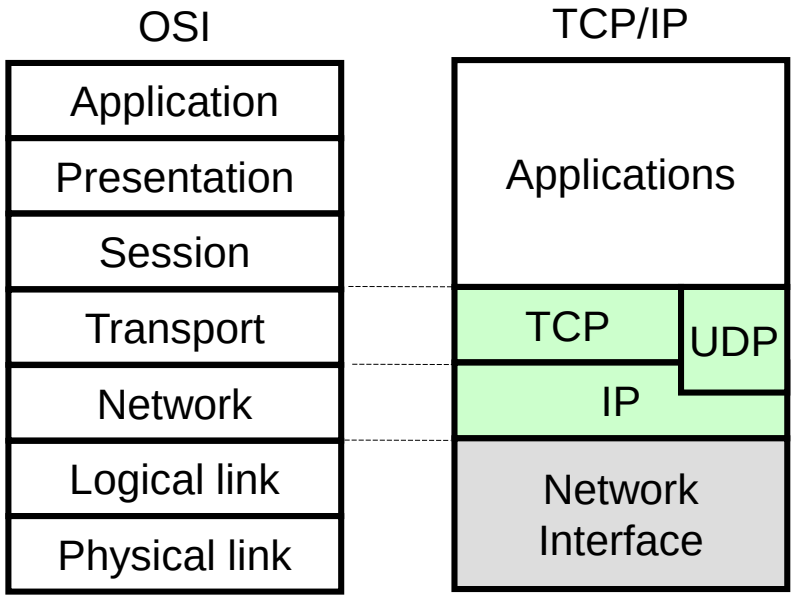
Most network implementations don't use the LLC layer and interact directly with the MAC layer. This is possible because, beyond the **medium access control** mechanism, the MAC layer also implements all basic functionality to transfer data packets locally, it defines node addressing and error detection.

The local networks evolve quickly to switching and access control mechanisms are now mostly not used. The exception is IEEE 802.11 wireless networks.

The TCP/IP architecture

The TCP/IP protocol stack resulted from a very different approach, in opposition to the OSI model it was developed without much theoretical planning, using a minimalist approach in which problems are solved as they arise in practice.

Nevertheless, TCP/IP was able to reach some of the initial OSI goals. Due to the internet generalization and the resulting requirement to use IP (Internet Protocol), there seems to be an irreversible migration trend of all systems to the TCP/IP stack, abandoning all other protocols. In this context, the interconnection of systems is necessarily resolved.



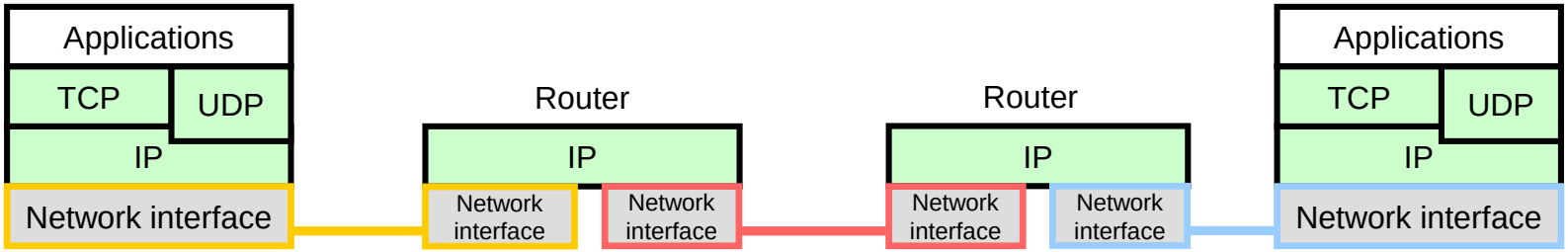
The TCP/IP stack consists of several protocols in addition to the IP, the most important are **UDP (User Datagram Protocol)** and **TCP (Transmission Control Protocol)**.
UDP – **connectionless** packet protocol without reliability, only simple error detection.
TCP – **connection-oriented** reliable protocol with sliding window flow and error control.
Of these two, the most widely used over the Internet is TCP.

IP routers

One advantage of TCP/IP is it does not define layers one neither layer two, the IP protocol is supposed to use any existing layer two packet transmission technology. IP defines a universal packet format and universal node addressing, thus allowing the interconnection of different network types. This allows the construction of a global network such as the internet.

However, when an IP node uses some specific layer two technology it will be able to deliver only to other IP nodes directly connected to that same technology. To overcome this, routers come in play.

Routers are the intermediate nodes of IP networks, they are usually connected to more than one layer two network (network interfaces) and each may use different technologies. Thus a router is capable of transferring IP packets between two networks, even if each uses a different layer two technology.



The diagram above represents three different layer two technologies being used: yellow, red, and blue.

IP network addresses

IP addressing introduces the **network address** concept, the aim is simplifying the routing task. Instead of node addresses based (like in layer two) routing is now network address based. By just looking at a node destination IP address its possible to determine to which network it belongs, each IP node address is made of two parts, one identifies the network it belongs to, the other identifies the node within that network.

For applications, all these routing issues across heterogeneous networks are completely transparent, applications just use IP addresses along with the UDP and TCP protocols. It´s up to routers analysing the destination IP address of each packet and find the appropriate path to reach there.

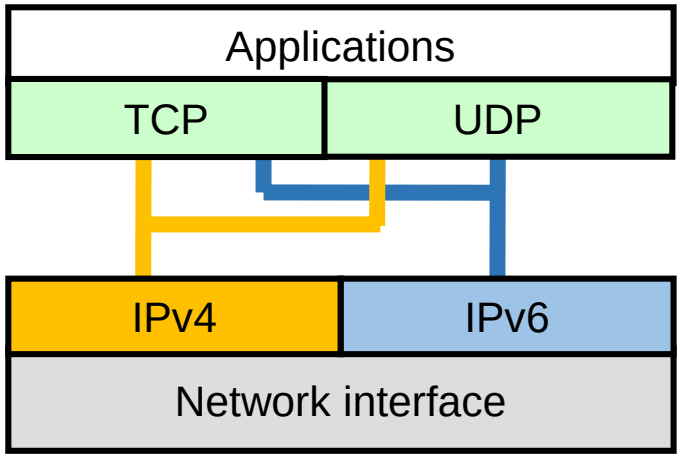
Although for IP the final destination for data is an IP node, for UDP and TCP protocols, final destinations for data are applications. Both use 16-bit numbers to tag data, knowing thereby to which specific application in the destination node it should be delivered. These labels are known as port or service numbers. Port numbers can be seen as addresses inside the node.

Internet protocol version 6

Since the internet has expanded, the most widely-used version of the Internet Protocol has been version 4, however, there is a new version 6, it's very slowly being introduced on the internet. One of the major differences between IPv4 and IPv6, is the increase on the size of node addresses (from 32 to 128 bits). Nevertheless, TCP and UDP protocols use IPv6 the same way they use IPv4.

To keep the internet running, the transition will necessarily be gradual, certainly, we are still going to have IPv4 on the internet for several years.

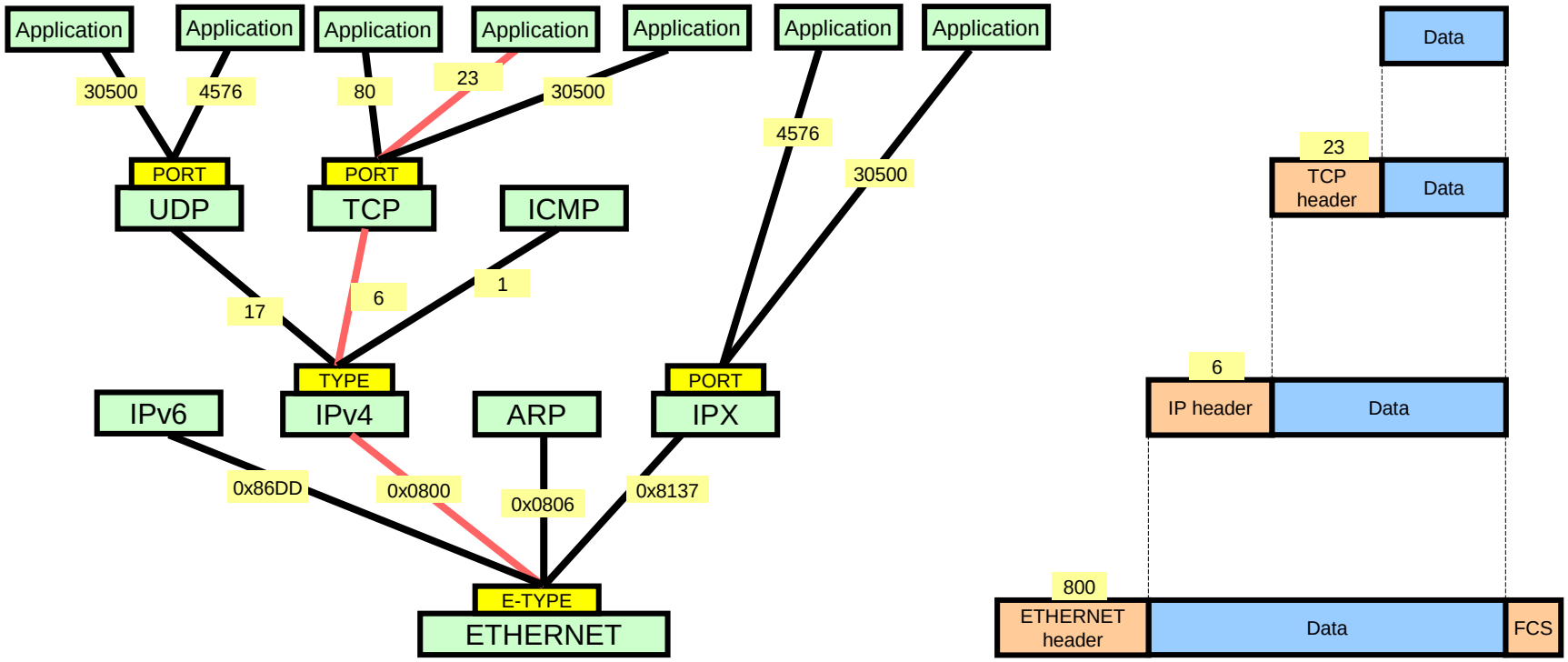
There are several co-existence strategies for IPv4 and IPv6, one of which is implementing both protocols in end nodes, this is known as **IPv4/IPv6 dual-stack** (right image).



As we have already noticed, in each layer, there are frequently several protocols running. This is possible thanks to tag based multiplexing. Tagging ensures data from a layer implementation X in one node, ends up being delivered to the same layer implementation X on the destination node.

Multi-protocol layers

The coexistence of several protocols at the same layer means that there are parallel data streams which diverge (upward) and converge (downward). In order for the converged flows to diverge elsewhere, each of them has to be tagged, this is known as multiplexing. This process is repeated successively along a protocol stack:



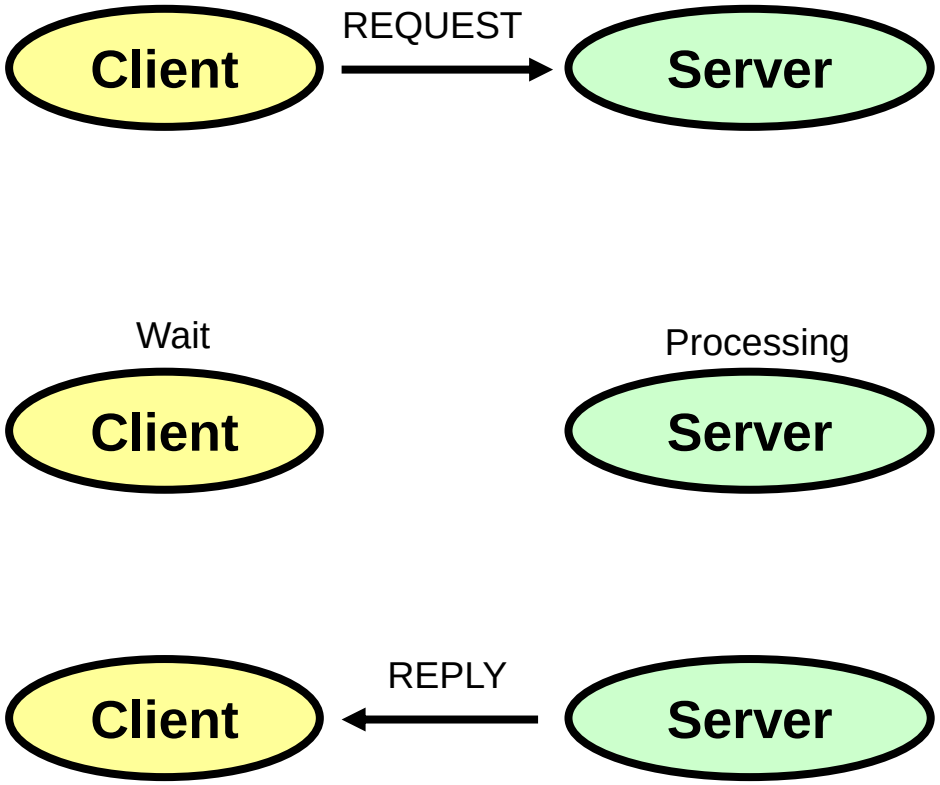
The client-server model

Almost all network communications follow a very simple dialog template known as **client - server model**.

FIRST: the client sends a request to the server, usually following a user action. The client must know how to reach the server, hence, the server network node address and port number. The network address is usually provided by the user, possibly in the form of a host name. The port number is standard for each type of service.

SECOND: after receiving the request, the server processes it. Meanwhile the client is waiting for a response.

THIRD: after processing the request the server responds to the client. To know the client's node address (and port number) the server simply checks the request source address.



These information exchanges follow the formats outlined in the application protocol. For certain services a reply may not be required, for others this dialogue may be repeated several times.