Freie Universitat Berlin
Erasmus Program

10 ECTS

Telematics

Professor:
Prof. Dr. Ing. Jochen Schiller

Autor:
Filippo Ghirardini

Winter Semester 2024-2025

Contents

Telematics

Autor: Ghirardini Filippo

Winter Semester 2024-2025

---

1 Basics

1.1 Network composition

A network consists of three elements:

- **End systems**: can vary in size and usage
- **Intermediate systems**: these (e.g. routers) are the components that allow the network to work.
- **Links**: they connect the end systems and can be *optical*, *copper* or *wireless*. Even if wireless is becoming more and more important, cables are still fundamental (undersea cables, underground cables).

Question 1.1.1 (Why fiber optic?)**.** Because this medium has not reached it's maximum and still has a huge potential of **bandwidth**. Also, while copper cable start acting as an **antenna** (and a receiver), disturbing near copper cable, fiber optic doesn't have this problem. Furthermore, copper cables need amplifiers which increase **latency**.

Question 1.1.2 (Why copper?)**.** It's **cheaper** and **easier** to handle.

Question 1.1.3 (Why cables over wireless?)**.** Because of **stability** and **latency**. Usually the problem is tampered by buffers but obviously it doesn't work with interactive applications.

Question 1.1.4 (What are threats to cable?)**.**

1.2 Communication principles

There are two basics principles:

- **Synchronous**: joint action of sender and receiver. Requires **waiting** until all parties are ready (e.g. phone calls)
- **Asynchronous**: sender and receiver operate decoupled (e.g. SMS, email). Requires **buffering**.

Note 1.2.0.1. There is also **isochronous**, which means the messages are sent every predetermined amount of time.

1.2.1 Direction

Communication channels may allow traffic flow in different directions:

- **Simple duplex**: one direction
- **Half-duplex**: both directions in different moments
- **Full-duplex**: both directions at the same time

1.2.2 Distribution

The communication distribution can happen in different ways:

- **Unicast**: one to one
- **Broadcast**: one to all
- **Multicast**: one to a subset
- **Anycast**: one to the nearest, e.g. when requesting to a redundant database you don't care which one responds
- **Concast**: many to one, e.g. we collect sensor data and send it to one
- **Geocast**: one to a certain region

Note 1.2.2.1. Even if multicast would be easier and cheaper, companies usually go for unicast because they want to know who the clients are.

Note 1.2.2.2. Broadcast guarantees anonymity while multicast does not.

1.2.3 Topologies

The main topologies are:

- **Full mesh**: too expensive
- **Chain**: in cars and trains
- **Star**: ideal for switches
- **Partial mesh**: the best compromise
- **Tree**: not ideal for big networks since if you cut a side, you lose contact

1.3 Sharing

1.3.1 Cons

Sharing may create a lot of problems, like **bottlenecks**: links and intermediate nodes are shared between end systems. One solution may be to *reroute* or to start *dropping packets* (e.g. when streaming the resolution lowers down).

1.3.2 Pros

At the same time, sharing means more efficient (less expensive) mechanism to **exchange data** between different components of distributed systems and **minimize blocking** due to multiplexing.

1.3.3 How?

There are two possible ways of sharing:

- **Reservation**: you reserve in advance the resource so that it is guaranteed, e.g. remote surgery. When the peak demand and the flow duration varies, there are two options:

  1. *First Come First Served*
  2. Everyone gets $10Mbps$

  It is implemented with **circuit-switching**: establishes dynamically a dedicated communication channel. It has predictable performance and a simple and fast switching but it's inefficient for bursty traffic, complex to setup and not easily adaptive to failures.

- **On-demand**: when there is a resource available you take it (variable *delay*, **jitter**), e.g. email. It is implemented with **packet-switching**: splitting the resource in packets and multiplex them. Much more flexible but requires buffers, packets overhead and has unpredictable performances.

Observation 1.3.1. It all depends on the application. Each flow has a **peak rate** and an **average rate**.To decide if *reservation* works well for a specific case, we must look at the ratio $\frac{P}{A}$. If it's small then it works well, otherwise it's wasting resources.

1.4 Internet

Internet is a network of networks. It enables processes on different hosts to exchange data: it's a bit delivery system.
ISPs enable you to access and use Internet services: well defined and commonly required functions. There are two roles: **client** and **server** that can be on different machine (or not, like with P2P).

Definition 1.4.1 (Internet)**.** *The set of all reachable parties (IP addresses).*

1.5 Protocols, layer and standards

Definition 1.5.1 (Digital Data Communication)**.** *Processing and transport of digital data between interconnected computers.*

Definition 1.5.2 (Data)**.** ***Representation*** *of facts, concepts and statement in a formal way which is suitable for communication, interpretation and processing by human beings or technical means.*

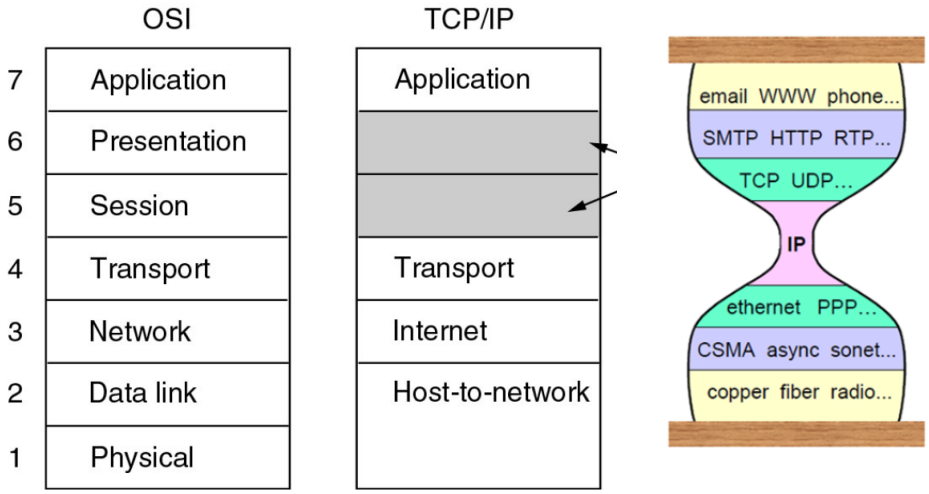
Note 1.5.0.1. **Information** is different from the data.

Definition 1.5.3 (Signal)**.** *A signal is the physical representation of data by spatial or timely variation of physical characteristics.*

In this context we need rules of communication for the different devices to communicate: we have heterogeneous computer architectures, network infrastructure and distributed application. A **protocol** is a conversational convention, consisting of syntax and semantic.

Definition 1.5.4 (Protocol)**.** *Protocols define format, order of messages sent and received among network entities, and actions taken on message transmission and receipt.*
A protocol is a set of ***unambiguous*** *specifications defining how processes communicate with one another through a connection (wire, radio etc.).*

To provide structure to the design of networks protocols, network designers organize protocols in **layers**. We use two models, either the ISO/OSI or the TCP/IP.



All the different layers need additional information, which is added via **headers** to the data payload via **encapsulation**. This could cause a lot of **overhead**.
All the protocols rely on the Internet Protocol. This maximizes **interoperability** and does not ensure anything, therefore no one has expectations.

1.6 Quality of service

To define the quality of communication we check:

- **Technical performance**: delay, jitter, throughput, data rate, etc.
- **Costs**
- **Reliability**: fault tolerance, system stability, immunity, availability
- **Security and Protection**: eavesdropping, authentication, denial of service, etc.

1.6.1 Latency

The main parameters we check are:

- **One-way delay**: measured in seconds

$$d_1 = t_1' - t_1 \tag{1}$$

- **Round-trip-time**: measured in seconds

$$r_1 = t_2 - t_1 \tag{2}$$

  It should also integrate the processing time of the other device.

1.6.2 Stability

The main parameter that measures stability is the **Jitter**. It's measured in seconds and calculated using the delay:

$$d_i = t_i' - t_i \qquad j_i = d_{i+1} - d_i \tag{3}$$

1.6.3 Capacity

From a capacity perspective, we measure the **throughput** in $\frac{bit}{s}$ as follows:

$$T = \frac{\sum \text{data}_i}{\Delta t} \tag{4}$$

The **goodput** instead, is the amount of **useful** throughput from a user perspective.

Note 1.6.3.1. **Bandwidth** is used for the description of the channel characteristics.

There is also the **Delay-Throughput-Produc** which measures how much data can be on the medium itself while traveling. E.g. with a connection of $1Mbps$ that has $200ms$ of delay we have

$$1Mbps \times 0.2s = 200kbit$$

2 DNS

2.1 Introduction

Names provide a level of abstraction from the IP address: for humans it's easier to remember. It also provides **load balancing** and easy **aliasing**.
The decision for DNS adding is handled by two organizations:

- **IETF**: how entries are entered and read from the phone book
- **ICANN**: how to decide *what* names should be entered in the phone book

To use naming you need two things:

- **Unique** names
- **Resolution** of names to locator (IP address) or other services

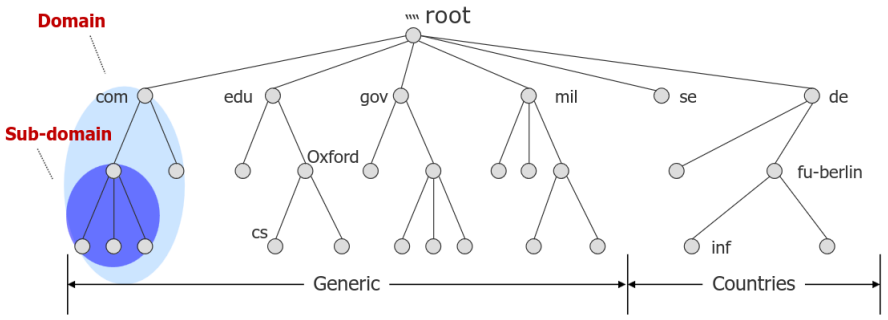
2.1.1 Scaling

To allow scaling, DNS uses **delegation** and **caching**. In particular for delegation, DNS adopts three intertwined **hierarchies**:

- **Name space**: hierarchy of names
- **Management**: hierarchy of authorities over names. Who owns which name part.
- **Infrastructure**: hierarchy of DNS server. Where is the mapping stored.

2.2 Namespace

DNS namespace is implemented as a tree structure: each node has a **label** which identifies it relatively to its parent node. Each node is **root** of a sub-tree (if it's not a leaf). In particular direct children of the root are called **Top Level Domains**. Each subtree represents a **domain** and each domain can be divided in **sub-domains**.



There is a limited number of TLDs: originally it was 7 plus one for each country. Now there are many more, even in non Latin alphabets.

2.2.1 Leafs

The name of a domain consists of a sequence of labels beginning with the root of the domain and going up to the root of the whole tree. Each label is separated by ".".
In the leaf nodes the IP addresses are associated with the names.
Furthermore, there could be **Domain Name Aliases**: pointers of one domain to another (Canonical Domain Name).

2.2.2 DNS Database

There are a few rules for the database:

- The **depth** of the tree is limited to 127
- Each label can have up to 63 characters
- The whole domain name has a maximum of 255 characters (even if the average is 10)
- A label of length 0 is reserved for the root

The full address to a host is the **Fully Qualified Domain Name**, which includes the leaf, each node and the root. The **Relative Domain Name** instead, is an incomplete domain name.

2.3 Management

The management of domain names also follows a hierarchy structure: **ICANN** manages the root domain and delegates someone (**DENIC** for Germany) to handle the *de* domain. They then delegate FUB to handle the *fu* domain. And so on.
This solution ensures that the names are unique.

2.4 Name servers and zones

2.4.1 Domains

Domains are administrative concepts managed by single organizations. The name of the domain corresponds to the name of the root node. They can delegate the responsibility for subdomains to other organizations but maintains the pointer to them to be able to forward requests.

2.4.2 Name servers and zones

On the other hand, name servers and zones are technical concepts. The name server is a process that maintains a database for a domain space. The part of the name space known to the server is called a **zone** and it's stored in a *zone file*. The name server may have multiple zones and has authority over them.

Primary Master It's a name server that must exist. Reads the data from a local file and has a database describing subdomains and computer in a selected zone.

Secondary Master It's optional and is a replication of the master for reliability reasons. It receives the data from another server which is authoritative.

2.5 Resolution

There are two types of Name Resolution:

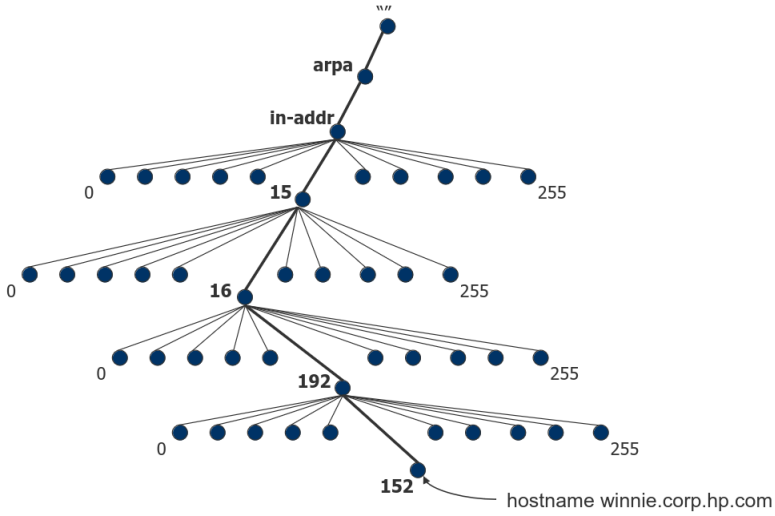
- **Recursive**: the name server replies either with the answer or with an error and it's responsible to contact the other nodes
- **Iterative**: a name server replies with the address of another one, it's the host duty to contact additional name servers for the answer

Question 2.5.1 (Why do root servers not support recursive solution?)**.** Using the recursive option implies that every intermediate needs to wait for all the others, depleting its resources.

Question 2.5.2 (How does this all contribute to scalability?)**.** We do not have **strong consistency** and

2.5.1 Reverse lookup

While mapping a name to a *global* IP address is simple, doing the other way round it's really difficult because we need to do a complete search of the tree.
Because of this, there is a special area in the database called **in-addr.arpa** that contains 256 sub domains, each one having 256 and so on.



Note 2.5.1.1. This is useful against **spoofing**: as an example if you get an email and you want to check if the sender is who claims to be, you can do a reverse lookup on the IP of the email server.

Question 2.5.3 (Why does reverse lookup not always work?)**.** Because the entries are not always present in the database.

2.6 Database entries

A **resource record** is the entry in the database to get the address or other information of a name. It's composed of a tuple:

```
(name, TTL, class, type, value)
```

TTL It's the Time To Live: after a certain amounts of seconds the record will be deleted from the cache and updated. With a shorter TTL you have a very updated cache while longer TTL means outdated caches but less requests for the server.

Type Indicates the type of data to be returned. **A** is the actual IPv4 address corresponding to the name (**AAAA** for IPv6).

Class Nowadays it's only **IN** but there were in the past other options for different networks with independent DNS zones.

Observation 2.6.1 (Load balancing)**.** DNS is very useful for load balancing: depending on the region when you ask for a DNS entry the answer will be the closest one. It can also be used for **evil purposes** (censorship, marketing).

2.6.1 Name Server

For each name server of a zone a **Name Server** record is created in the cache. E.g. when you want to visit *arnold.movie.edu* you may have in cache a NS entry for *movie.edu*.

2.6.2 CNAME

A **CN** record is an optional entry in the database that illustrates aliases on its canonical names.

2.6.3 Pointer

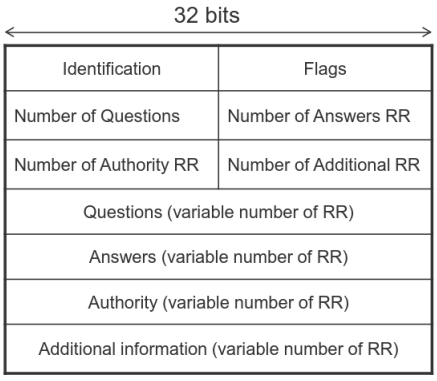
The **PTR** record provides information for the mapping of an address to names. If you do not have any entry for an IP address you then have to do a reverse lookup. Addresses should refer only to one name.

2.6.4 Mail Exchanger

The **MX** record serves for the controlling of the email routing. It specifies an email server responsible for a domain name with the option to indicate a preference if multiple servers are present (the smallest value is preferred).

2.7 Protocol

The resolver software triggers the resolution process and tries first for the cache. Then it sends the request to the local DNS server which is either static (resolv.conf) or dynamic (DHCP).
The protocol consists of a single packet used for inquiries and responses:



Identification 16*bits* for the mapping of an inquiry to a response.

Flags 16*bits* of various flags that indicate if the packet is a request or a response, if it's **authoritative** or not, if it's *iterative* or *recursive*.

Numbers These fields indicate the contained number of inquiries responses data records.

Questions Contains the names to be resolved.

Answers Resource records to the previous inquiry.

Authority Contains the ID of the passed responsible NS.

Additional information If the name searched is only an alias, the belonging resource record for the correct name is placed here.

The packet is sent through UDP on port 53 and the **reliability** is only implemented via repeating the requests. Also it is not protected.

2.8 Scalability

The scalability is achieved mainly with local caching of recent results. The cache can be in the network and also in the local client.
One of the main problem is how long should you keep the entries? You need to achieve both **consistency** and not doing too many requests. You also need to detect and flush the **stale entries**. You have to avoid **cache poisoning**: when a malicious person changes the value in the cache to redirect you to an evil software.

2.9 Extension

2.9.1 Dynamic DNS

The problem comes up when, as an example, you restart the router and your public IP address is changed (or maybe the ISP changes it every 24 hours). The DDNS allows you to tell the changed IP address.

2.9.2 Characters

The original DNS supports only ASCII, so there is an extension for UTF characters.

2.9.3 DNSSEC

The **security** is important because DNS is the most crucial indirection to access the data. Controlling DNS response implies controlling the discovery of the communication endpoints. It may be use in an evil way for political and economical reasons.

3 Email

3.1 Introduction

Email is an example of an application that works on the different layers. It's **asynchronous**, **decentralized** (to improve *scaling*), **client-server** and based on simple ASCII text.

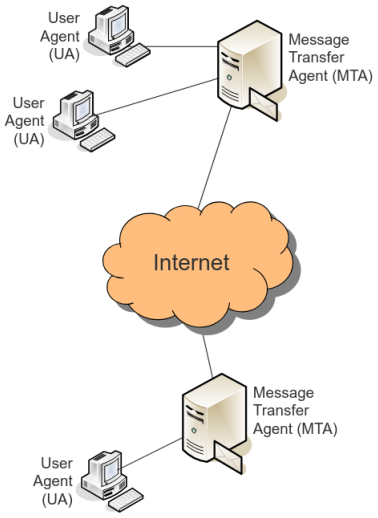
3.1.1 Motivation

Email was the first *killer application*. It started in the 1980's with a simple terminal interface, evolving in the 1990's with the *web-mail* and then *mobile email*. Today social network are trying to swallow the email concept.

3.2 Architecture

There are two actors involved:

- **User Agent**, with email clients. Runs on the computer of the user and is intermittently on. E.g. Thunderbird or Outlook
- **Message Transfer Agent**: the email servers. They run on a remote machine and stores and forwards on behalf of the User Agents. It's always on



3.3 Message

Message are viewed as having an **envelope**, the fundamental part for the delivery, and a **content**. The latter contains a **header** with a certain coding and a **body** consisting of simple characters.

3.3.1 Envelope

The envelope is created by the **MTA** or the **MSA** and includes all the information for transporting the message. Some information are redundant with the header (like the sending and receiving address) but there are some differences, like when you send a Blind Carbon Copy message.

Note 3.3.1.1. You can't know if the sender is correct. This can be used for evil purposes. The only way to avoid that is by encrypting or signing the email.

3.3.2 Content

Header It contains characters with the following syntax:

```
<key>:<value>
```

Body It's the content of the email. It's separated from the header by a blank line.

Since originally the content could only contain 7*bit* ASCII, **Multipurpose Internet Mail Extensions** was invented to extend the classical format. It adds additional headers, content types and sub-types:

- **MIME-version**
- **content-description**: string that describes the content of the message
- **content-id**: identifier for the content
- **content-transfer-encoding**: selected coding for the content (BASE64, ASCII)
- **content-type**: specifies the type of the body in the format *type/subtype*, e.g. text, image, audio, video, etc..

3.4 Protocols

3.4.1 SMTP

Simple Mail Transport Protocol delivers the mail to the final inbox. It can't ensure that the message arrives to the final user because it expects the receiver to be always online.
It uses **reliable data transfer** based on TCP on port 25 and it's *best effort*. It provides **little security**: no encryption, optional authentication on port 587 to reach MSA but nothing between MTAs.
The protocol follows these steps:

1. **Write** an email, the client formats it and sends it to it's own mail server
2. The mail server sets up a connection with the receiver's server and **sends** a copy of the email
3. The **receiver**'s server creates the header of the email and places the message in the inbox

Graylisting A first attempt to block spam. If a combination of IP address of the sender, their email and the receiver's one is seen for the first time, the message is discarded and an error is returned. From the second time on, the message goes through. This is based on the idea that scammers won't send the email twice.

3.4.2 POP3

This protocol pulls emails from the server over a connection on port 110. It's text based and allows basic functionalities such as *logging*, *copying locally* and *deleting* from the server.
It works in two phases:

1. **Authorization** phase: *user*name and *pass*word for authentication, either successful or not
2. **Transaction** phase: a *list* of the messages and their sizes is provided, then via *retr* its possible to retrieve a message using the number of the list and with *dele* to delete an email.

Note 3.4.2.1. POP3 is heavily limited due to problems with multiple users handling and always-on connections.

3.4.3 IMAP

This protocol works on port 143. In this case the emails remains on the server and may be cached by the client. All the actions are performed on the server. Ideal when you need to access it from different locations.

3.4.4 HTTP

The **webmail** allows the user to interact with emails via WEB. E.g. Gmail or Outlook.

4 HTTP

HTTP is a protocol that allows the user to request a **resource** (e.g. HTML page) that is on a server. They may contain references to other resources, therefore creating a *web*, called **World Wide Web**.

4.1 History

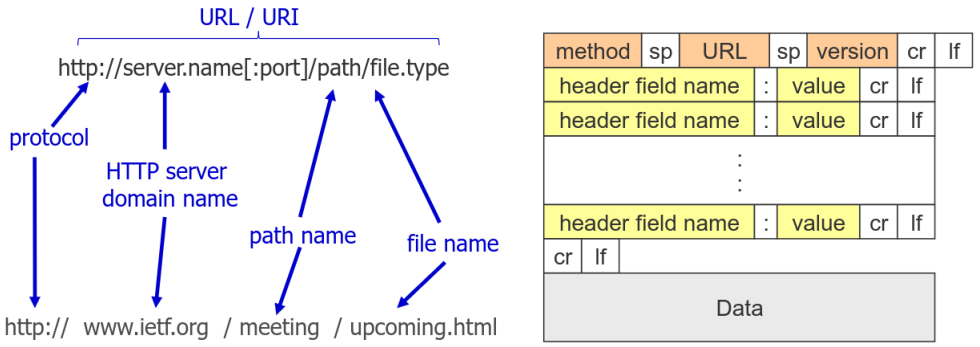
The first idea came in 1945 by Vannever Bush, with his **Memtex**, a desk containing different information categorized accordingly: **hypertex** context was born. Then in 1962 Doug Engelbart started to work on its actual implementation and by 1989 Tim Berners Lee connected that with TCP/IP and DNS protocols, effectively creating the WWW.
Today it gives access to **intelinked documents** distributed across several computers in the world, using the internet as exchange.

4.2 Communication

4.2.1 HTTP/1

The standard way of communicating is between a **client** (e.g. Firefox, Chrome) and a **web server** (e.g. Apache, Nginx).
The communication is handled by the HTTP protocol (usually on port 80). It's a text based **request/response** protocol.
It was **stateless** until version 2. This means that the server maintains no information about previous requests and thus the specification of the context is needed every time.

Request HTTP requests follow the **REST** API principle, allowing for performance, scalability, simplicity, modifiability, portability and reliability. The resources are retrieved via a **URL**

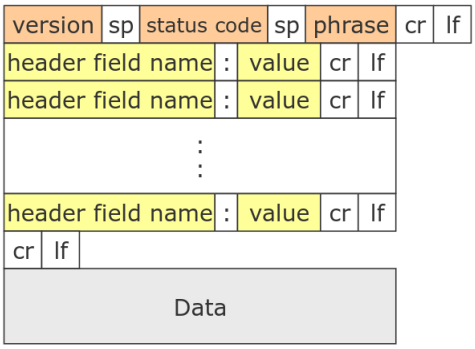


The specified commands that can be used with a URL are:

- **GET**: load a web page
- **HEAD**: load only the header of the web page, used for *debugging*
- **PUT**: store a page on the web server
- **POST**: append something to the request passed to the web server
- **DELETE**: delete a web page

Response The HTTP response contains the protocol used, the header lines and the **status code** , that can be:

- **1xx**: only for information
- **2xx**: successful inquiry
- **3xx**: further activities are necessary
- **4xx**: client error (syntax)
- **5xx**: server error



4.2.2 Web Sockets

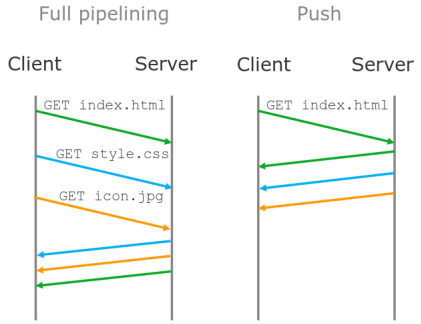
HTTP first problem is that the client needs to poll explicitly for content, causing a huge **overhead**. Thus Web Sockets were created: they allow a full duplex communication between the server and the client without the need of HTTP. It uses the same ports and it's set up using an HTTP request to "upgrade", which is then answered with a "switching protocol" response.

4.2.3 WebRTC

HTTP second problems is to enable communications between multiple browsers without creating a web server for each one of them. WebRTC implements a P2P communication that provides functions to establish media and data exchange, e.g. for videoconferencing.

4.2.4 HTTP/2

The second version of HTTP is **binary** instead of text based. It is fully **multiplexed**, associating requests and response and allowing a bi-directional stream. Therefore it can use only one connection while still granting **parallelism**. Furthermore it uses **header compression** to reduce overhead and allows server to push responses into client caches, reducing the number of requests to render web pages.



4.2.5 HTTP/3

This version uses **QUIC** protocol over UDP instead of TCP and TLS for security, avoiding *head of line* blocking.

4.3 Cookies

The main problem with HTTP is that it's **stateless**, this meaning that after every request/reply the web server forgets everything. While this is not a problem for simple browsing, it means that we cannot store user content to personalize the experience.
The solution is the **cookies**: tags stored in the web browser and set by the server so that they can be sent again to allow the latter to identify the client.

4.3.1 Structure

Cookies are stored as name-value pairs defined by the server. They can have optional parameters such as an **expiry date**.

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=497793521	15-10-18 17:00	Yes
joes-store.com	/	Cart=1-00501;1-07031	11-10-18 12:00	No
aportal.com	/	Prefs=Stk:SUNW+ORCL	31-12-18 17:30	No
sneaky.com	/	UserID=2344537333744	31-12-18 18:00	No

4.3.2 Pros and cons

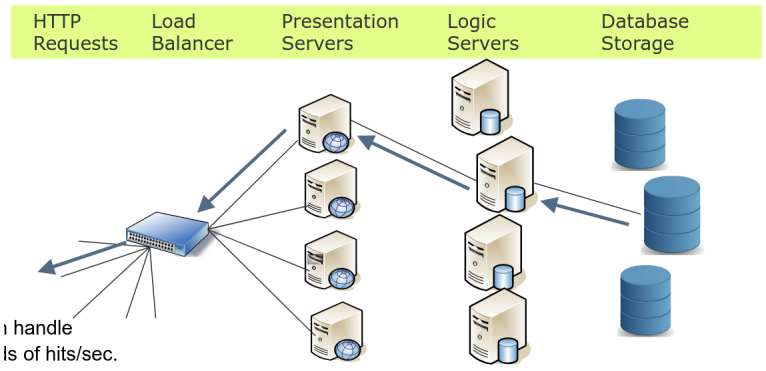
They enable **authorization**, shopping carts, recommendations and **user session state** (e.g. for web mail). The biggest problem is about **privacy**: cookies are identified by **Etags**, an opaque identifier for a specific version of a resource, and can be used to track users.

4.4 Proxy

A **proxy** is an intermediate cache between multiple clients and a server. The main goal is to have a more **efficient** page loading, improving **scalability**. It temporarily stores the pages loaded by the browsers: if the client requests it and it hasn't changed yet, it's loaded from the proxy, otherwise a new request to the server is made and the cache is updated.
It can also enable support for protocols such as FTP or Telnet without the need for a new browser implementation.
It can also work as a **firewall**.

4.5 Scaling

To handle huge loads (top 1000 websites) we use **3-tier** architecture, which separates the web server in **presentation servers**, **logic servers** and **database storage**.



If, instead, we want to deal with medium and small web servers, we usually virtualize a lot of them on a single machine and we do the multiplexing with the server URL field in HTTP.

4.6 DNS

It's possible to use DNS over HTTP by sending a request (either *GET* or *POST*) to the DNS server. It improves privacy and security but the user looses control.

5 SNMP

The Simple Network Management Protocol allows the management of devices and services using a simple datagram service with a **client-server** architecture:

- **Agent**: process continuously running on each managed node collecting information
- **Manager**: process running intermittently on a management workstation that requires information about the devices in the network

There may also be a **proxy agent** that integrates non-SNMP capable systems

5.1 Motivation

If we have problems in the network we need a **management tool** to be able to correctly identify them and their causes. In particular the tool needs to manage:

- **Performance**: measure and analyze network performance to provide good network service
- **Configuration**: monitor or modify configuration settings or HW or SW elements
- **Accounting**: measure network utilization parameters per user or group of users
- **Fault**: detect, log, notify users and automatically fix problems while running
- **Security**: control access to network resources according to local guidelines to avoid sabotage and unauthorized access to sensitive information

Observation 5.1.1. It should be achieved **remotely** over the existing network, meaning a protocol above IP.

5.1.1 Goals

The main goals of network management are:

- **Monitoring** HW equipment
- **Statistics** of network usage
- Remote **diagnostics**
- **Protected** and **safe** networking
- **Efficient** internetworking
- Simple model of **network status**
- Gather data for **network planning**

5.2 Overview

A network management framework is based on three building blocks:

- **SNMP**: defines **format of messages** exchanged by management systems and agents and **basic operations**
- **SMI**: Structure of Management Information specifies how the monitored information is structured, defining the objects that SNMP protocol accesses over the network
- **MIB**: Management Information Base describes the concrete managed objects and is an open concept for data storage. It may be public (RFC) or proprietary.

5.2.1 History

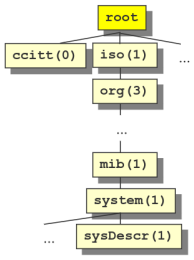
There have been three major versions of SNMP during history:

- **SNMPv1**, 1998, was designed originally as an interim solution but became the standard. It had very weak security model with complex bulk requests but was simpler than CMIS/CMIP
- **SNMPv2**, which was then split in
  - *SNMPv2u*: user-based security
  - *SNMPv2**: user-based security and additional features
  - *SNMPv2c*: without security but with *GetBulk* operation
- **SNMPv3**, the current one, that provides an advanced security model: now each message has security parameters, integrity and authentication.

5.3 Managed objects

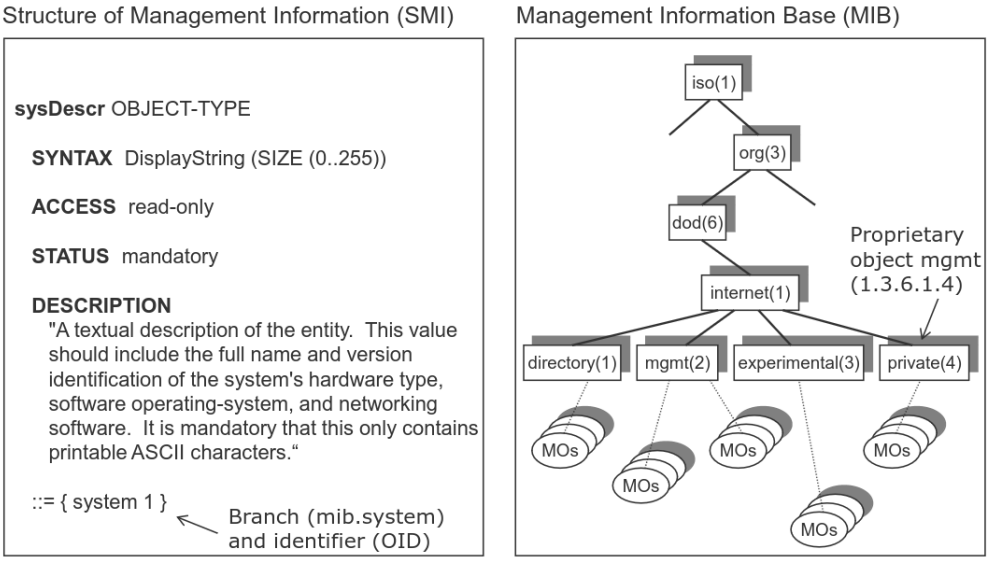
An *agent* monitors the network resources that are abstracted as **managed objects**. Each one has a **unique ID** and a **name** and models various property of the resource. Its standard components are:

- **Common prefix + Unique name**: e.g. *iso.org.dod.internet.mgmt.mib.system.sysDescr*
- **Syntax**: simple data types *integer*, *string* and *array*
- **Access rights**: *read-only* or *read-write*
- **Status**: *mandatory* or *optional*



MIB The Management Information Base is a distributed virtual database that hosts the collection of managed objects that belong to the same context. It defines the capabilities of the device that can be managed.
In particular **MIB-2** defines the generics for all manageable internet devices. Its prefix is *iso(1).org(3).dod(6).internet(1).mgmt(2).mib2(1)*. Some examples are *batteryAgingNotification* with OID *1.3.6.1.2.1.233.0.5* and *sysLocation* with *1.3.6.1.2.1.1.6*.
Another specific MIB is **RMON** for Remote Monitoring, it has more advanced functions such as gathering of statistics, alarms, events, on-board evaluations.

MIT Each managed object has a unique position in the Management Information Tree, thus providing a unique reference.



5.3.1 Encoding

Since different systems use different data representation, it's necessary to recode data while maintaining its meaning. To do that the message is first composed in ASN.1 syntax and then transferred using BER.

ASN.1 Abstract Syntax Notation One is an ISO standardized language for representation-independent specification of data types. It's used in SNMP to describe managed objects. It consists of:

- **Elementary** types, such as *boolean*, *integer*, *bitstring*, ...
- **Structured** types:
  - **Sequence**: ordered list of data types
  - **Set**: unordered set of data types
  - **Sequence Of**: like *array* in C
  - **Set Of**: unordered set of elements from the same data type
  - **Choice**: like *union* in C

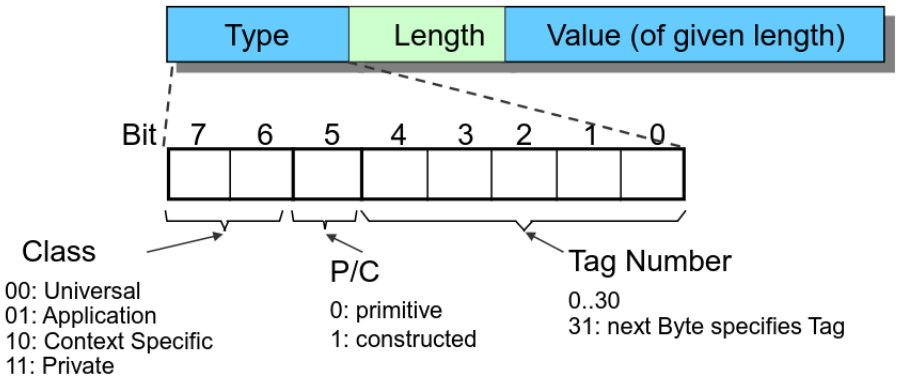
Example 5.3.1. Some types defined by ASN.1:

```
- INTEGER
   - signed 32-bit int
- OCTET STRING
- OBJECT IDENTIFIER (OID)
```

and some defined by SMI:

```
- IpAddress
   - OCTET STRING of size 4, in network byte order
- Counter
   - unsigned 32-bit int (rolls over)
- Gauge
   - unsigned 32-bit int (will top out and stay there)
- TimeTicks
   - unsigned 32-bit int (rolls over after 497 days)
- Opaque
   - used to create new data types not in SNMPv1
- DateAndTime, DisplayString, MacAddress, PhysAddress, TimeInterval, TimeStamp,
    TruthValue, VariablePointer
   - textual conventions used as types
```

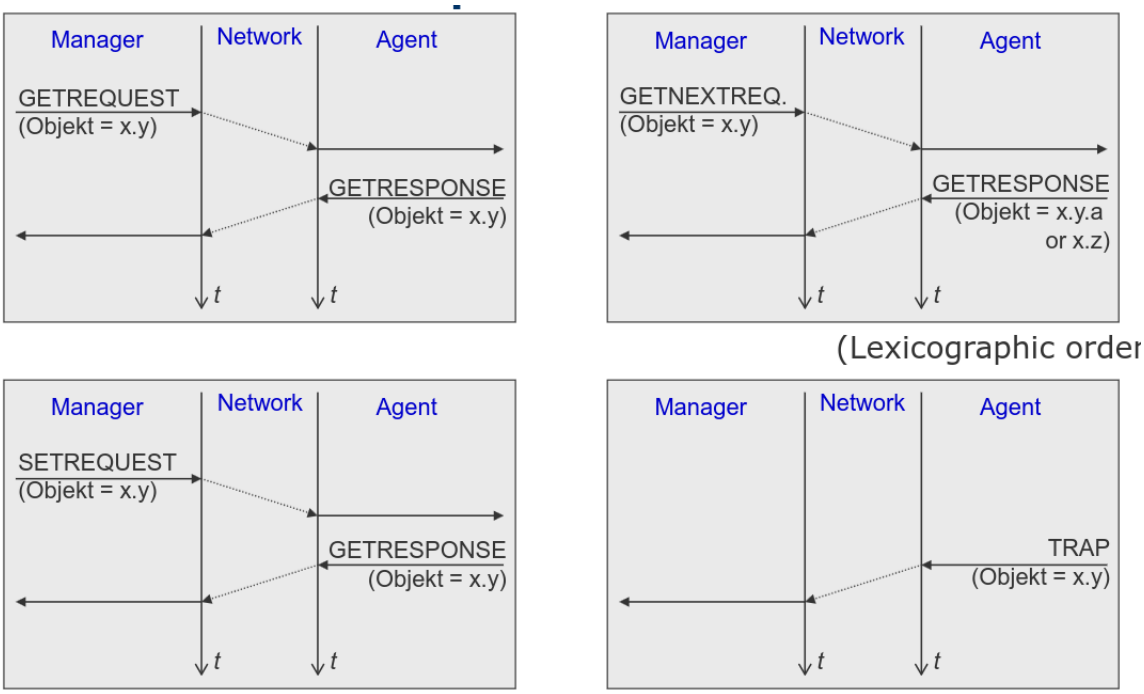
BER ASN.1 content is then converted into smaller binary data that follows the Basic Encoding Rules, like source code is converted to machine code.



5.4 Operations

SNMP has five main operations:

- **GetRequest**, **GetNextRequest**, **SetRequest** and **GetResponse** that are initiated by the SNMP manager
- **Trap** that allows an agent to push information to the manager
- **GetBulkRequest** is a series of *GetNextRequest*

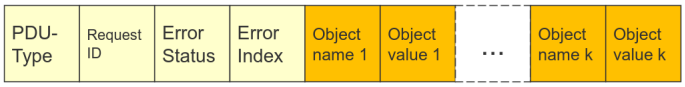


SNMP works on a well-known UDP port: 161 for everything except of *Trap* that is on 162.

5.4.1 Packet format

The packet is divided in:

- **Version**: version number of SNMP
- **Community**: string used for authentication, transmitted in plain text
- Command dependent**PDU**:



  - **PDU-Type**
  - **Request Id**: identification of pending request
  - **Error Status**, based on the result of the query
  - **Error Index**: reference to the variable binding pair that caused the failure
  - Variable length list of pairs of (Object Name, Object Value)

0	noError
1	tooBig
2	noSuchName
3	badValue
4	readOnly
5	genErr

5.5 Tools

There are different tools:

- **Command line** like *snmpget*, *snmpnext*, etc that allows the generation and decoding of SNMP data units and sometimes also support for MIB files
- **MIB Browser**
- **Management Platforms** like *CiscoWorks* and **Nagios** (open source)

All of the tools need to do the **discovery** and **polling** of the network topology through ICMP, SNMP and HTTP, among diffent devices with common interfaces but also vendor specific functionalities.

5.6 NETCONF

SNMP is used for monitoring only and therefore a tool for configuration is needed. NETCONF is based on XML and messages are exchanged through a secure transport protocol. The included operations are:

- **get**: retrieve running configuration and device state information
- **get-config**: retrieve all or of a specified configuration datastore
- **edit-config**: edit a configuration datastore by creating, deleting, merging or replacing content
- **copy-config**: copy an entire configuration datastore to another configuration datastore
- **delete-config**: delete a configuration datastore
- **lock**: lock an entire configuration datastore of a device
- **unlock**: release a configuration datastore lock previously obtained with the *lock* operation
- **close-session**: request graceful termination of a NETCONF session
- **kill-session**: force the termination of a NETCONF session

An extension of NETCONF is **RESTCONF**, which allows accessing data defined in **YANG**. YANG (Yet Another Next Generation) is a data modeling language for the definition of the data. It can be used for configuration data, status of devices, events or notification. It's protocol independent but can be converted into XML or JSON. It's **modular** and represents data structures as XML tree, with many data types.

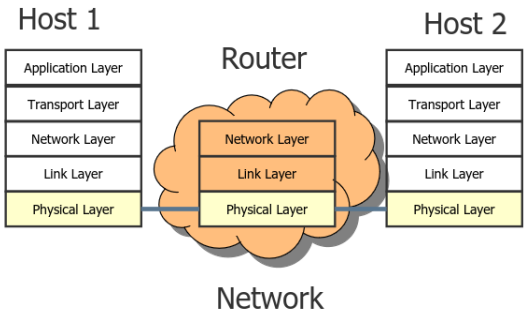
5.6.1 NETCONF vs SNMP

Let's analyze the main differences:

- **Security**: NETCONF offers more granular and flexible access control mechanism, it runs over SSH or TLS (RESTCONF on HTTPS), while SNMP lacks proper encryption and authentication
- **Structure**: NETCONF uses structured data models to define the configuration and operational state of network devices, a clear and standardized way that reduces misconfigurations. On the other hand, SNMP is less structured and less intuitive (tree structure), often requiring complex OIDs and more prone to errors
- **Functionality**: SNMP can only retrieve device data while NETCONF can also modify or replace configurations

6 Physical

In the network we have the **hosts**, end systems hosting user applications, and the **routers**, intermediate systems providing network connectivity.



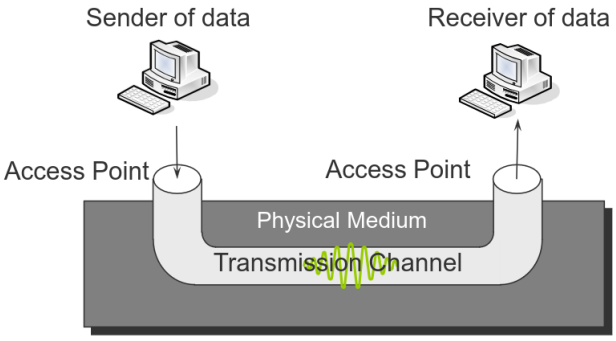
The hosts have the full stack implementation while the routers only up to the Network Layer.

6.1 Signals

Definition 6.1.1 (Signal)**.** *A signal is the physical representation of the data. It can be:*

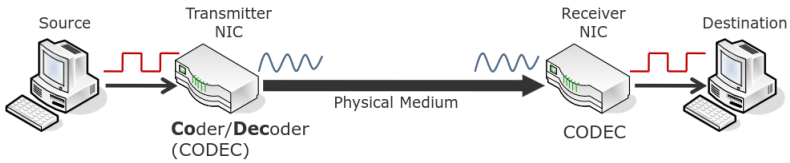
- ***Analogue****: sequence of continuous values*
- ***Digital****: sequence of discrete values*

Data is converted to signal which is then sent over the **transmission channel**, which is composed by **access points** and a **physical medium** (e.g. copper).



Since computers deal with digital signals, transmitting one bit via at a time a given medium, they need:

- **Quantization**: convert from digital signal to analog signal and vice versa
- **Sampling**: must rely on periodical measurements of the physical medium



Observation 6.1.1. We have different challenges during the transmission of data:

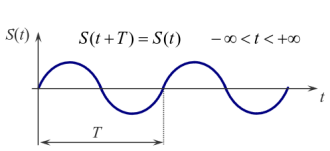
- **Internal**: **collision** and **synchronization**
- **External**: **noise**

6.1.1 Periodic signal

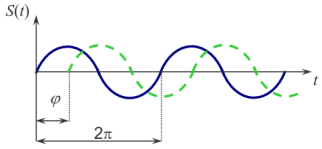
Periodic signals are the simplest signals. They take the following parameters:

- **Period** T
- **Frequency** $f = \frac{1}{T}$
- **Amplitude** $S(t)$
- **Phase** φ

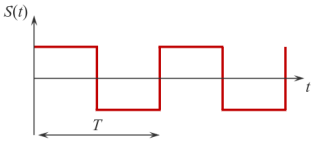
Example 6.1.1. Some examples of signals:



(a) Sine wave



(b) Phase



(c) Square wave

Fourier Analysis Like the composition of functions, it's possible to **compose** signals, generating new ones. In fact, per the **Fourier Analysis**, any period function can be constructed as the sum of a number of sines and cosines, resulting in a **Fourier Series**.

$$g(t) = \frac{1}{2}c + \sum_{n=1}^{\infty} a_n \sin(2\pi n f t) + \sum_{n=1}^{\infty} b_n \cos(2\pi n f t) \tag{5}$$

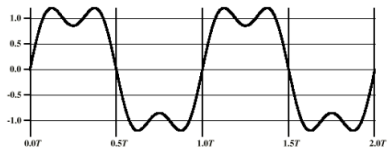
Fourier Transform is a mathematical transformation used to transform signals between time domain and frequency domain. Exists also in two dimensional space.

Distortion If all Fourier components were equally diminished the resulting signal would be reduced in amplitude but not distorted. Unfortunately all transmission facilities diminish different components by different amounts, introducing **distortion**.

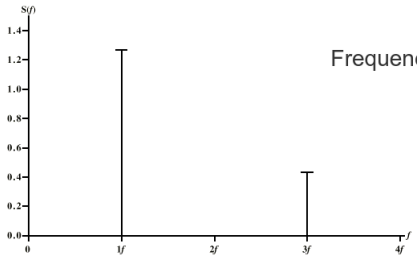
Frequency domain The frequency domain is described by:

- **Spectrum**: the range of frequencies a signals consists in
- **Bandwidth**: width of the *spectrum*. In theory many signals have infinite bandwidth. The **effective bandwidth** is the narrow band of frequencies where most of the energy is contained.

Example 6.1.2. In the following signal we have a **spectrum** from f to $3f$ and a **bandwidth** of $2f$.



(d) Time domain



(e) Frequency domain

6.1.2 Bandwidth

We have two possible digital signal:

- **Binary**: two possible values, 0 and 1
- **Multilevel**: more than two possible values (e.g. ternary, quaternary)

Definition 6.1.2 (Symbol rate)**.** *Number of physical signaling events per unit of time on the transmission medium. The unit of measure is a* ***baud****.*

Definition 6.1.3 (Data rate)**.** *Rate of bits decoded from symbol rate per unit of time. The unit of measure is* $\frac{bit}{s}$*. There are two cases:*

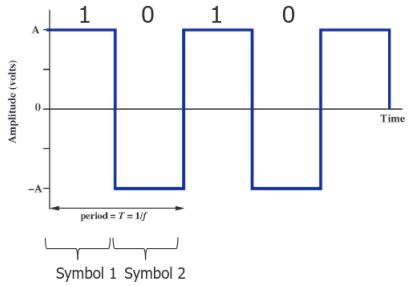
- ***Binary*** *signals with frequency* v*, each signaling event codes one bit*

$$Data\ rate = v$$

- ***Multilevel*** *signals with* n *possible values*

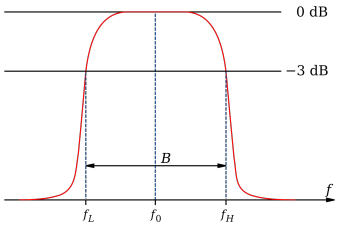
$$Data\ rate = v \cdot \log_2(n)$$

Example 6.1.3. In this image we have a square wave with a negative (0) and a positive (1) pulse



The duration of a symbol is $\frac{1}{2} \cdot T = \frac{1}{2f}$, hence the **symbol rate** (which in this case is equal to the **data rate**) is $2f$ bits per second.

Definition 6.1.4 (Bandwidth)**.** *The bandwidth of the medium is the highest* f_H *minus lowest* f_L *frequency which can be transmitted over this medium (in Hz).* f_0 *is the center frequency.*



A **square wave** is composed of an infinite number of **harmonics**, where the amplitude of the kth one is $\frac{1}{k}$.

$$s(t) = A \cdot \frac{4}{\pi} \cdot \sum_{k=1,k\ \text{odd}}^{\infty} \frac{1}{k} \sin(2\pi k f t) \tag{6}$$

While theoretically we would need an infinite bandwidth to get a square wave, the medium limits the number of harmonics.

6.2 Transmission

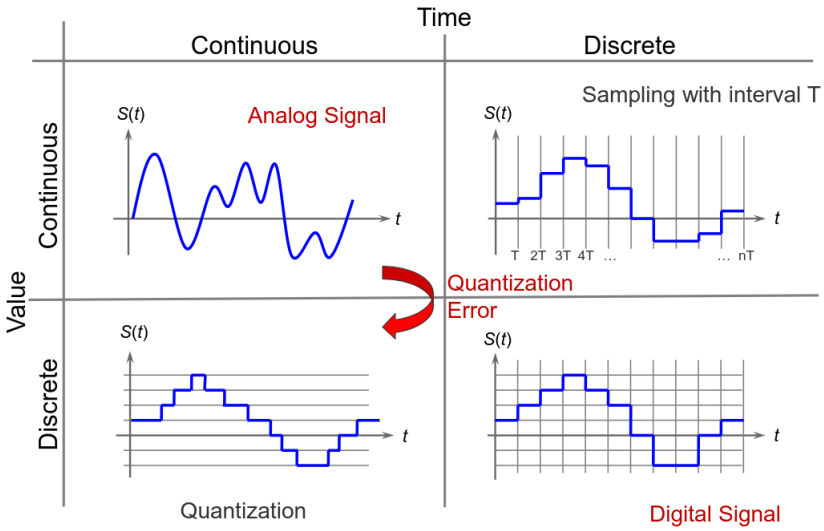
The fundamental problem of communication consists in reproducing on one side exactly or approximated a message selected on the other side. The ideal transmission would be a square wave while the actual one is an approximation.
To get from an analog to a digital signal, there are two main operations:

- **Quantization**: from a continuous value we get a discrete one with an error
- **Sampling**: periodical measures at a given rate with an interval T

  Theorem 6.2.1 (Nyquist Sampling Theorem)**.** To allow the reconstruction of the original analog signal it is sufficient that the sampling frequency f_s is such that

  $$f_s > 2W \tag{7}$$

  where W is the bandwidth in Hz.



To get an analog signal from a digital one we perform the **coding** operation, where the quantization intervals are assigned to a binary code and then transmitted.

6.2.1 Channel capacity

To define the **capacity** of an analog physical channel, we need to consider that there is always some **noise**. This means that the capacity will be **finite** and will depend on several parameters, usually:

- **Additive**: the noise value is added to the signal and that's what the receiver gets
- **White noise**: independent, random noise values with constant spectral density
- **Gaussian**: probability distribution of the amplitude of random noise values

While on the analog signals noise will cause degradation of the quality, on digital ones it causes **bit errors**. It is possible to reduce the effect of noise by boosting signal amplitude, but requires energy and causes more interferences.
Transmission impairments essentially come from:

- Signal **attenuation** and **attenuation distortion**
- **Delay distortion**
- **Noise**
  - *Thermal* noise
  - *Intermodulation* noise
  - *Crosstalk*
  - *Impulse* noise

Definition 6.2.1 (BER)**.** *Bit Error Rate is a metric for bit errors. It depends on the* ***environment****, on the* ***communication medium*** *and the* ***length*** *of the transmission line (higher frequencies attenuated stronger than lower, different frequencies have different speed).*

$$BER = \frac{Number\ of\ erroneous\ bits}{Number\ of\ transmitted\ bits} \tag{8}$$

Theorem 6.2.2 (Shannon Theorem)**.** A channel with W bandwidth, P average signal power and N average noise power has a maximum data rate of:

$$\text{maximum data rate} = W \cdot \log_2(1 + \underbrace{\frac{P}{N}}_{\text{SNR}^1}) \tag{9}$$

Even if we don't consider noise, throughput is still **finite** due to:

- **Quantization** at the transmitter and **discrete levels** of the signals
- **Sampling** at the receiver

While Shannon Theorem gives us an upper bound for the data rate, we can use Nyquist theorem to calculate the amount of discrete signals needed to achieve that.

Theorem 6.2.3 (Nyquist theorem)**.** Given a channel of W bandwidth and n discrete levels of the signal, the maximum data rate is

$$\text{maximum data rate} = 2W \cdot \log_2(n) \tag{10}$$

6.3 Data encoding

To transmit individual bits there are two options:

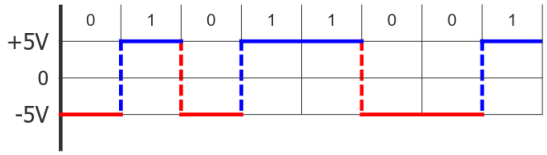
- **Baseband**: the original data is transmitted "as is" over the medium, requiring **data encoding**
- **Broadband**: the data is transmitted by **modulating** it onto a **carrier** analog signal

A data encoding technique needs:

- **Robustness**: tolerance to distortion
- **Efficiency**: high transmission rate, achieved using coded words (binary, ternary, quaternary)
- **Synchronization** with receiver: less opportunities for out-of-sync. Achieved by frequent changes of voltage level regarding to a fixed cycle. Needs to avoid direct current: positive and negative signals should alternatively arise. Bipolar/Unipolar encoding.

6.3.1 NRZ

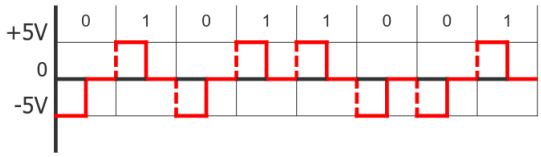
Non Return to Zero is a simple approach where 1 is coded with a positive voltage ($+5V$) and 0 as a negative one ($-5V$). It's very **simple** and the smaller the clock pulse, the higher the data rate. It's prone to **loss of synchronization** and has **direct current** during long sequences of the same bit.



[1] Signal to Noise Ratio

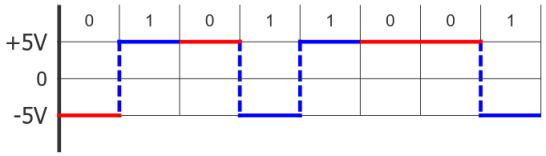
6.3.2 RZ

Return to zero works on the same principle of NRZ but after each bit the signal goes back to zero first. This way, the signal is **self-clocking** and there is no direct current. It needs twice the bandwidth.



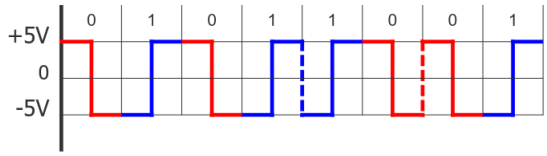
6.3.3 Differential NRZ

Similar principle of NRZ but 1 is encoded as a **voltage level change** and 0 as a missing change, thus having the disadvantages of NRZ only for a sequence of zeros.



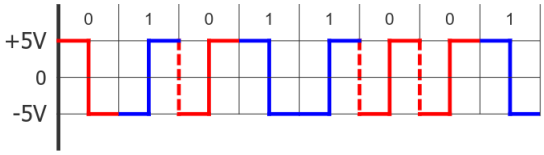
6.3.4 Manchester code

With each code element the clock pulse is transferred: voltage level change occurs in the middle of each bit. 0 is encoded as voltage level change from positive $(+5V)$ to negative $(-5V)$ while 1 as voltage level change from negative $(-5V)$ to positive $(+5V)$.
Clock synchronization happens for each bit and the end of the transmission is easily recognizable. The capacity used is only half.



6.3.5 Differential Manchester code

A variant of the Manchester code. 0 is encoded as a voltage level change while 1 as a missing one.



6.3.6 4B/5B

To improve the efficiency of the Manchester code, this techniques codes a hexadecimal character of four bits in five, avoiding long zero blocks. Uses the same principle of the differential NRZ. Allows some combinations for control information. The transmission provides clocking.
It's used in the USB and FastEthernet context and in GigabitEthernet with different variants (8B/10B, 64B/66B, ...).

Name	4b	5b	Description
0	0000	11110	hex data 0
1	0001	01001	hex data 1
2	0010	10100	hex data 2
3	0011	10101	hex data 3
4	0100	01010	hex data 4
5	0101	01011	hex data 5
6	0110	01110	hex data 6
7	0111	01111	hex data 7
8	1000	10010	hex data 8
9	1001	10011	hex data 9
A	1010	10110	hex data A
B	1011	10111	hex data B
C	1100	11010	hex data C
D	1101	11011	hex data D
E	1110	11100	hex data E
F	1111	11101	hex data F
I	-NONE-	11111	Idle
J	-NONE-	11000	Start #1
K	-NONE-	10001	Start #2
T	-NONE-	01101	End
R	-NONE-	00111	Reset
H	-NONE-	00100	Halt

6.4 Modulation

To transmit with **broadband** we need to modulate the signal, that being **shaping** a carrier frequency via the baseband signal.

$$s(t) = A \cdot \sin(2 \cdot \pi f t + \varphi) \tag{11}$$

6.4.1 ASK

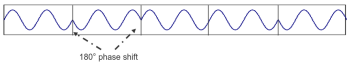
Modulation of the **amplitude** A. It's easy to realize and doesn't need much bandwidth but it's not robust against distortions. Often used in optical transmissions.

6.4.2 FSK

Modulation of the **frequency** f. It was the first used in data transmission using phone lines. Needs a lot of bandwidth and it's a waste of frequencies.

6.4.3 PSK

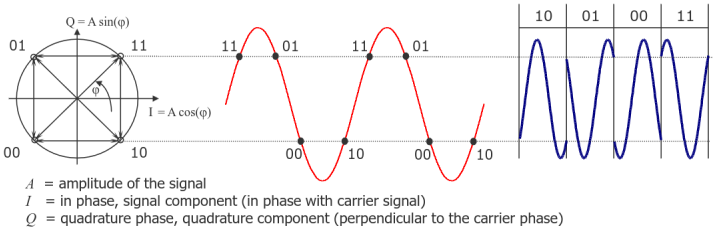
Modulation of the **phase** φ. It has a complex demodulation process but it's robust against disturbances.



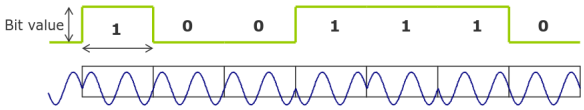
Also called **Binary** PSK.

6.4.4 PSK variants

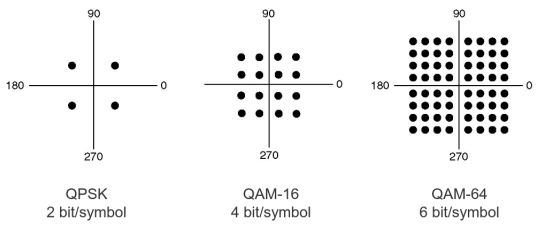
Quadrature Phase Shifting Key (also 2B1Q) allows the shifting between 4 phases, allowing for 4 states and thus 2 bits at a time (doubling the data rate).



Differential BPSK works with two different phases like PSK but it shifts only if 1 is the next bit.



Quadrature Amplitude Modulation is a combination of ASK and QPSK. $n > 2$ bit can be transferred at the same time. Bit error rate increases with n but is still less than similar techniques. Used frequently in wireless communication.

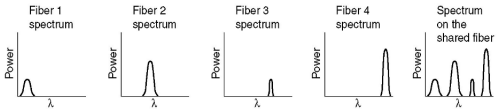


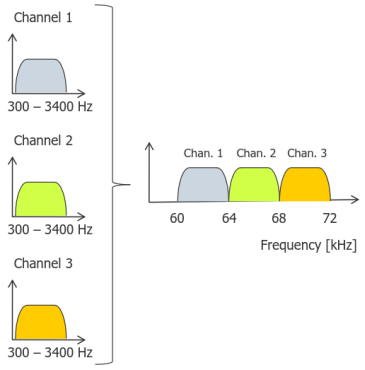
6.5 Multiplexing

Since lines are expensive it's important to share the resources. Multiplexing is a technique that provides simultaneous transmission over a single medium.

6.5.1 FDM

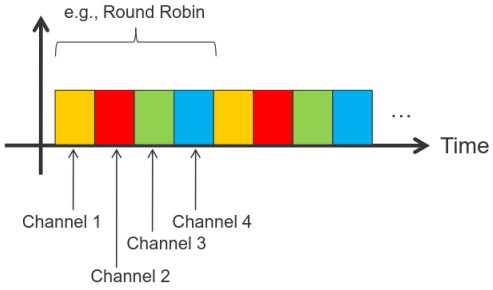
Frequency Division Multiplexing divides the frequency spectrum in frequency **bands**, which are used exclusively and simultaneously. When used for optical transmission is called **Wavelength Division Multiplexing**.



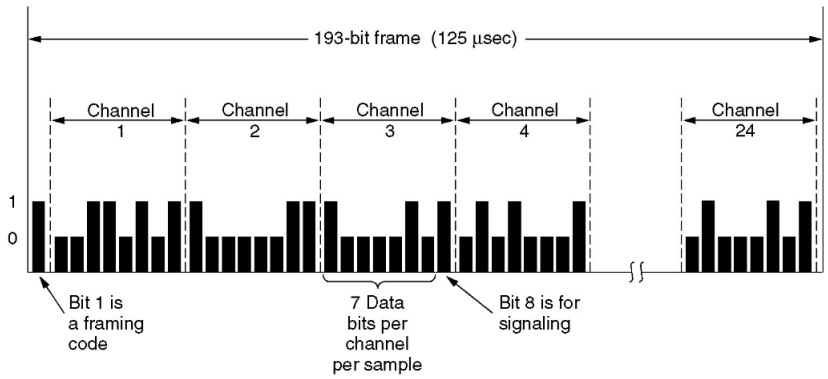


6.5.2 TDM

Time Division Multiplexing divides time into slots of fixed or variable length. Each timeslot represents one sub-channel.



A classical TDM is the T1, having 24 channels in parallel with 8bit per channel (one is control), a 193 bit frame that lasts for 125μsec for 1.554Mbit/s.

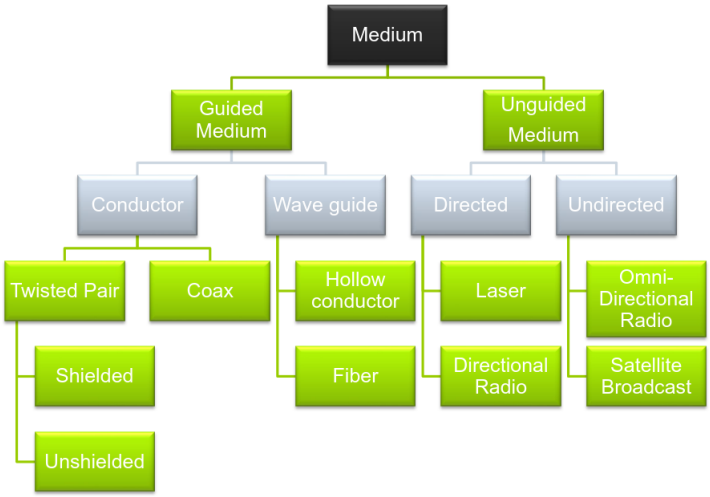


It's possible to multiplex T1 into higher carriers to get T2, T3, ...

Note 6.5.2.1. Multiplexing techniques plus algorithm that control how to do it result in **Multiple Access** technologies (e.g. FDMA, TDMA) on layer 2.

6.6 Physical media

There are many different **medias**.



And different type of **networks**, classified over the size of their scope.

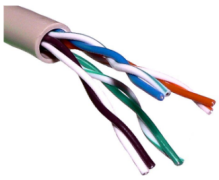
Name	Scope	Example
Body/Personal	1mt	Body
Local	10mt-100mt	Room, Building
Metropolitan	1Km-10Km	Campus, Town
Wide	100Km-1000Km	Country, Continent
Internet	10000Km	Planet

6.6.1 Electromagnetic waves

Electromagnetic waves are used to transmit the signal over cables and wireless. In a vacuum they travel at the speed of light c but in copper or fiber they slow down at about $\frac{2}{3}$ of c.
The **fundamental relationship** between wavelength λ, frequency f and c:

$$\lambda \cdot f = c \tag{12}$$

6.6.2 Guided medium

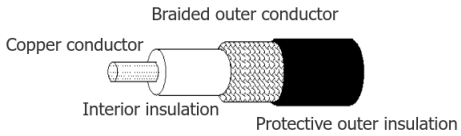


Twisted pair This medium transmits data through **electrical signals**. Electromagnetic signals from the environment can disturb it, thus it's necessary to have **insulation** and **twisting** (also done at different rates to reduce **crosstalk**). It's cheap and simple and it's used both for digital and analog signals. They have a bit-error rate of $\approx 10^{-5}$.
They are divided in:

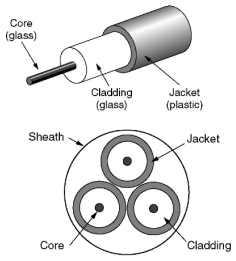
- **Unshielded** (U)
- **Foil shielded** (F)
- **Screen shielded** (S)

The **shield** can be individual, overall or both. They are also divided in **categories** based on their shielding level and maximum speed.

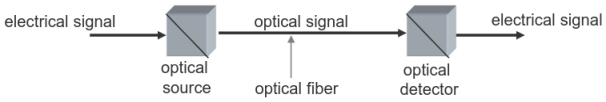
Coaxial It's a copper cable with braided outer conductor to reduce disturbances and interior insulation between them. Has a bit-rate error of $\approx 10^{-9}$, has higher data rates over longer distances compared to the twisted pair and has a better signal quality.



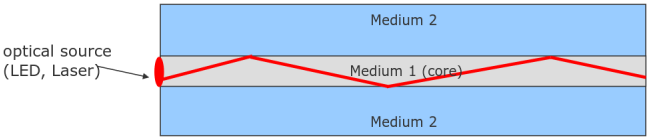
Optical Fiber Huge capacity with nearly unlimited data rate. It's insensitive to electromagnetic disturbances. Has a good signal-to-noise ratio. It's smaller and lighter and has a bit-error rate of $\approx 10^{-12}$.
The structure of an optical transmission system has:



- **Light source**: converts electrical into optical signals, 1 is a light pulse and 0 is no light pulse
- **Transmission medium**, the optical fiber
- **Detector**: converts optical into electrical signals

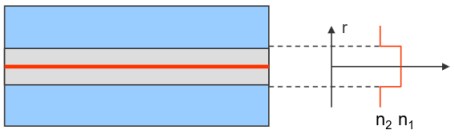


The cable has a **core** made of optical glass (super thin), an internal **glass cladding** and a protective **plastic covering**. The transmission takes place in the core, which has a high **refractive index** (refraction effect relatively to vacuum), where a ray of light is reflected between the two mediums.

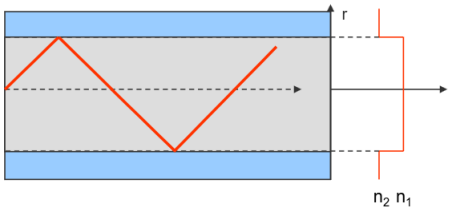


There are three types of fiber cables:

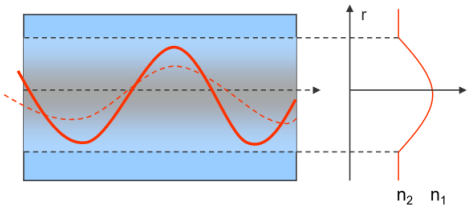
- **Single mode**: with a diameter core of $8 - 10\mu m$, all rays can go in only one direction and therefore having no dispersion (homogeneous signal delay). Expensive due to small core.



- **Simple multimode** (step index): core diameter of $50\mu m$, uses different wavelengths with different delay signals and therefore has a high dispersion



- **Multimode** with gradient index: same as the simple multimode but the refracting index changes continuously, hence having a low dispersion



Radiation sources Radiation sources can be of two types:

- **LED**: cheap and reliable, has a broad wavelength spectrum thus a **high dispersion** (small range) and a **low capacity**
- **Laser**: **expensive** but with **high capacity** and a small wavelength spectrum and thus **high range**. They are sensitive to **temperatures**.

On the other side of the signals there are **photodiodes**.

Attenuation The ray of light is increasingly weakened along the medium due to **absorption** and **impurities** in it.

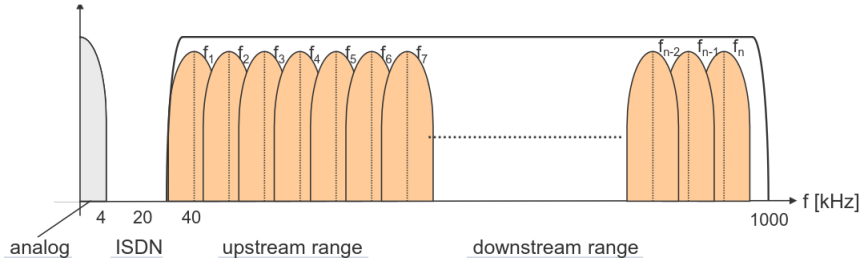
Dispersion Rays of light spread in the medium at different speeds, refractive index in the medium is not constant.

6.6.3 Home connection

There are multiple solutions to connect homes to the internet:

- Existing **phone lines** local loop (DSL). It was the first approach. It needed a MoDem (Modulator and Demodulator) to convert digital data into analogs and viceversa. Today uses the whole spectrum of the copper cable and modulates through **Discrete Multi-Tone** or **Carrierless Amplitude Phase**. Calls are now over IP. The main standards are VDSL and VDSL2
- Existing **cable TV** network connections, using coaxial and upstream multiplexing through **Cable Modem Termination System**. Uses the **DOCSIS** standard.
- Deployed **cellular networks**
- Existing **powerline connections**
- **Satellite** communication
- **Microwave** links
- **Fiber-to-the-home**

Discrete Multi-Tone Modulation Uses multiple carriers where each channel uses a suitable optimal modulation method (QAM). Channels in high frequencies have a lower quality over long distances.



7 Link

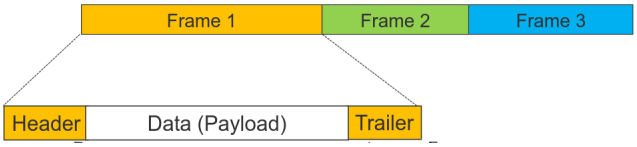
7.1 NIC

The **Network Interface Card** is a piece of hardware that provides link layer abstraction to the device network's stack. They are of two types:

- **Point-to-point**: high bandwidth bidirectional links, with a dedicated channel per direction thus avoiding collisions
- **Broadcast**: medium shared in a way that creates collisions if more devices transmit at the same time

7.2 Framing

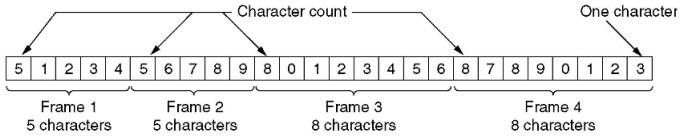
Messages are organized in **frames**, well defined structures, as follows:



- **Preamble**: before the header, flag byte sequence marking the beginning of a frame and sometimes the frame length
- **Header**: control information (e.g. addresses, frame numbers)
- **Error check**: inside the Trailer, contains **Frame Checking Sequence**
- **Postamble**: after the Trailer, flag byte sequence marking end of a frame

7.2.1 Character count

A good addition is using the first value to advertise the number of characters in the frame, thus eliminating the *Postamble*.



7.2.2 Character stuffing

Start and end of a frame are represented by a special byte sequence. Since the flag byte may occur in the payload, **character** stuffing is done: a special escape byte *ESC* is inserted by the sender and then removed by the receiver. The escape byte may also appear in the payload and thus it may also be stuffed itself.

The generic version is the **bit stuffing**: inserting a 0 after five consecutive 1.

7.2.3 Physical layer

Framing can also exploit special symbols from the physical layer:

- **4B5B**: use the special symbols as control characters
- **PHY coding violations**: in Manchester code no mid-bit edge can appear, therefore the start and end of a frame can be coded as a violation of that rule

7.3 Error detection and correction

Transmissions over the physical layer are not error free. Hence, error **detection** and **correction** is necessary. The solution is to add **error control data** to each frame: a frame of m bits receives r check bits, creating a **codeword** of $n = m + r$ bits.

7.3.1 Frame Check Sequence

A very naive solution is to repeat the payload and then do a bit-wise comparison. It's very effective but has a huge **overhead**.

Parity bit This technique reduces the overhead by counting the number of 1 in the payload and setting the **parity bit** to 1 if there is an **even** amount of them. This adds a 1bit overhead only but doesn't check for *even* errors and doesn't allow correction.

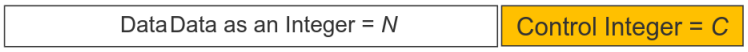
Double parity Group bits together to form a matrix and then compute parity bits for each row and column. Compared to single parity, can identify more errors (not all) and correct some of them.

Sender:			Receiver:		
1011	1		1011	1	
0010	1		0110	0	
1100	0		1100	0	
0110	0		0110	0	
0011			0111		

Modulo check Parity bits don't handle well error **bursts**[2]. Modulo check tries to solve this by computing a frame check sequence based on a modulo operation: interprets the payload as a number N and appends a control integer C such that

$$(N + C)\%11 = 0$$

Bit errors are detected by recomputing the modulo at the receiver end.



The generic version is the **Cyclic Code Checksum**, which allows you to choose the number of bits dedicated to the checksum and the base of the modulo, extremely lowering the probability of undetected errors (if tuned correctly).

Cyclic Redundancy Checksum A payload of m bit $a_{m-1}, \ldots, a_0$ is seen as a **polynomial** $a_{m-1}x^{m-1} + \ldots + a_0$. The sender and the receiver then agree on a **generator polynomial**

$$G(x) = g_r x^r + g_{r-1}x^{r-1} + \ldots + g_1 x^1 + g_0 x^0 \qquad g_r = g_0 = 1$$

The sender then interprets a data block of length m as a polynomial

$$M(x) = a_{m-1}x^{m-1} + \ldots + a_1 x^1 + a_0 x^0$$

and adds redundant bits so that the extended polynomial $M'(x)$ is divisible by $G(x)$.

```
r = degree(generator_polynomial)
extended_data_bits = data_bits + (r zeros) // Concatenate r zeros
remainder_bits = binary_division(extended_data_bits, generator_polynomial)
checksummed_frame = extended_data_bits XOR remainder_bits
```

On the other hand the receiver divides the received extended polynomial $M'(x)$ by $G(x)$ and if the remainder is 0 then no error occurred.
CRC will detect all error bursts of length $\leq r$. It **will not recognize** if instead of $T(x)$, $T(x) + E(x)$ with $E(x)$ containing $G(x)$ as a factor.

[2] Group of errors close together, very frequent in data communication.

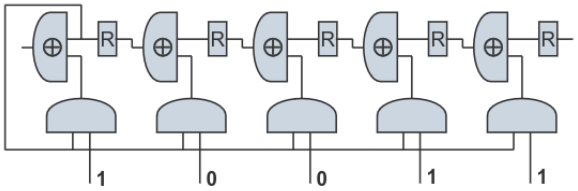
Note 7.3.1.1. Common 16-bit generators are:

- **CRC-16** $G(x) = x^{16} + x^{15} + x^2 + 1$
- **CRC CITT** $G(x) = x^{16} + x^{12} + x^5 + 1$
- **Ethernet** $G(x) = x^32 + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

The **hardware** implementation for CRC uses **shift registers** with XOR for subtraction and AND for applying it.

Example 7.3.1. Circuit for the generator polynomial

$$G(x) = x^4 + x + 1$$



7.3.2 Hamming Distance

The Hamming Distance is the number of places in which two binary sequences differ. Given an $m$bit long **data** with 2^m possible *data words*, r **check bits** and thus $n = m + r$ bit **codewords**, build a list of all the possible 2^n codewords and find the two with the minimum distance. This allows to:

- **Detect** d errors, having a distance of at least $d + 1$
- **Correct** d errors, having a distance of at least $2d + 1$

Example 7.3.2. Given a code with only two possible codewords:

$$w_1 = 000$$
$$w_2 = 111$$

We have a distance of 3, meaning we can **detect** 2 bits errors and correct 1bit ones:

- 001 received, we can correct it
- 110 received, cannot be recovered

While the Hamming Code can correctly identify and correct 1bit errors, it's expensive in terms of required check bits and can't correct 2bits errors and cannot identify 3bits ones.

7.3.3 Correction mechanism

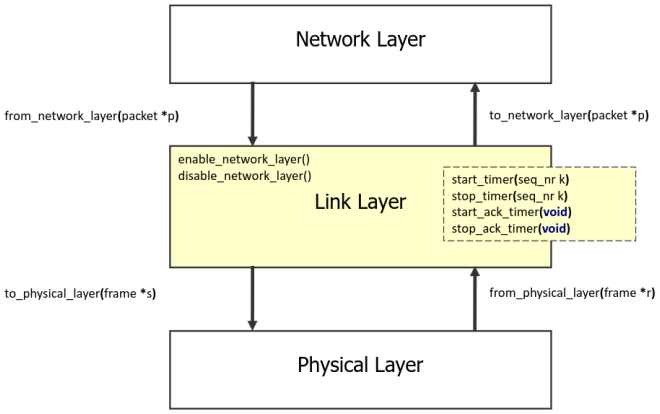
Forward Error Correction Uses error correcting codes (RS, BCH): they can be corrected in most cases, **discarded** otherwise. It's low latency since the feedback from the receiver to the sender is not needed, therefore suitable for **delay sensitive** transmissions.

Automatic Repeat reQuest Uses error-detecting codes (CRC): if data contains errors, it needs to be sent again from the sender. Suitable for **error sensitive** transmissions. Using ARQ means managing the **control flow** (numbering data blocks, receipt acknowledgment, retransmission).

7.4 Flow control

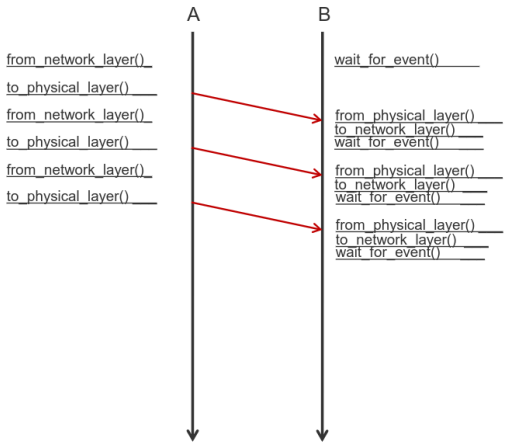
Let's define a sketch of the interface between the Link layer and the Network and Physical layers.

```c
#define MAX_PKT 1024 /* determines packet size in bytes */

typedef enum {false, true} boolean; /* boolean type */

typedef unsigned int seq_nr; /* sequence or ack numbers */

typedef struct {
   unsigned char data[MAX_PKT];
} packet; /* packet definition */

typedef enum {data, ack, nak} frame_kind; /* kinds of frames */

typedef struct { /* frames are transported in this layer */
   frame_type type; /* what kind of a frame is it? */
   seq_nr seq; /* sequence number */
   seq_nr ack; /* acknowledgement number */
   packet info; /* the network layer packet */
} frame;
```



7.4.1 Simplex

This is the simplest mechanism of flow control: transmission happens in **one direction** without *sequence number* or *acknowledgment* and ignoring *processing time*. The sender **fetches** and **sends** data in an infinite loop, while the receiver **gets** it and **forwards** it to the network layer. While being extremely **simple**, it assumes that the channel never damages or loses frames and that the **network layer** is **always ready**.

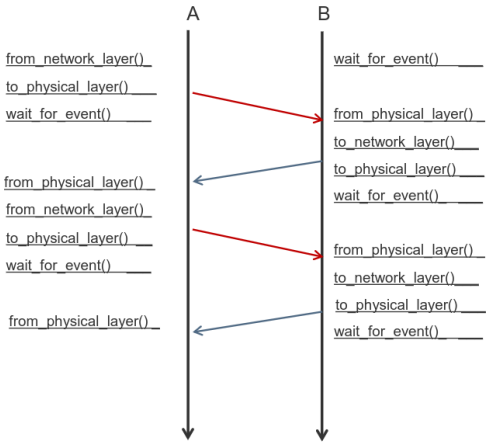


To solve all of these problems we have two approaches:

- **Feedback** based flow control: receiver sends information back to the sender allowing him to send more data
- **Rate** based flow control: protocols limits the data a sender may transmit without feedback from the receiver

7.4.2 Stop-and-wait

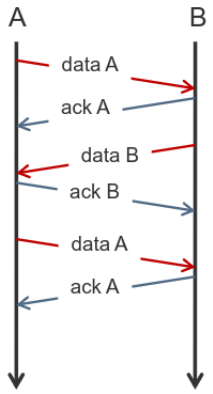
This approach uses a **bidirectional channel**: the sender sends a data blocks and waits until an **acknowledgment** from the receiver arrives or a **timeout** is reached. An unacknowledged data block is resent.



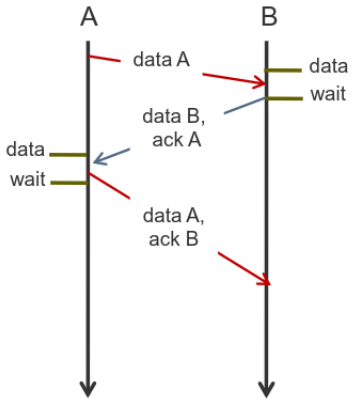
Since the channel is not error-free, we use the **Alternating Bit Protocol** to signal a duplicate data block by using one bit flag.

The main drawback is that there are large waiting periods between the transmissions, hence **capacity is wasted**.

Piggybacking To get a **full-duplex** communication we could use two simplex channels with **stop-and-wait**. That would be a waste of resource, hence the technique of **piggybacking**.
It consists of **stop-and-wait** on a single channel for both directions: data frames and ACK are intermixed and distinguished by the **Type** field in the header.
An even more efficient approach is to include the ACK message in header of the data frame. When a data packet arrives, the receiver waits a particular time interval to send the ACK back.



(f) Separate messages



(g) Single message

7.4.3 Sliding window

To avoid long waiting periods of the sender, both parts agree on a **transmission window** of size W. Any message sent in that window doesn't need a specific ACK, when ACK is sent all previous messages in the window are acknowledged.
For this technique, **sequence numbers** need to be introduced, to allow sequentially numbering of the messages. If coded on $n$bits, the sequence number seqnum $\in \{0, 1, \ldots, 2^n - 1\}$ and **wraps around** the end.

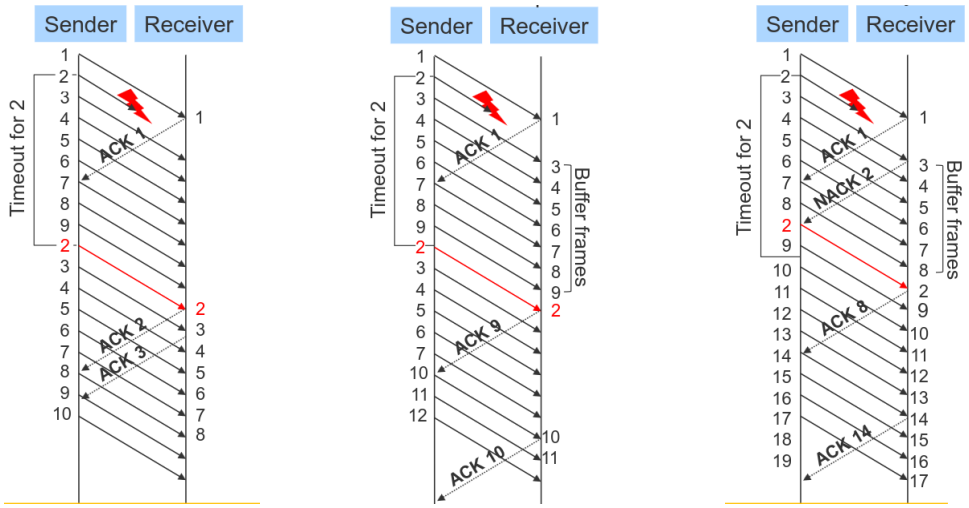
Note 7.4.3.1. All frames in the window must be **buffered**, hence for a window of size W a buffer for W frames is needed.

Observation 7.4.1. It's important to have a window of size $W < 2^n$ and not $W \leq 2^n$ to avoid ambiguities.

Pipelining Long round-trip time is an issue in terms of efficiency. If bandwidth $\cdot$ round-trip-delay is large, a big window is needed to fill the pipe capacity. Having a bigger window size means also having more bit-errors and packet loss.

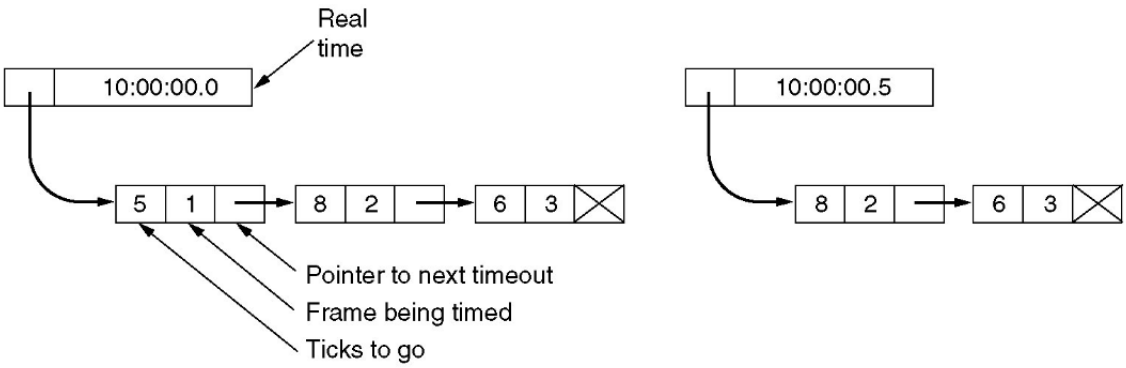
There are three ways of handling the **pipelining**:

- **Go-back-N**: each frame has an associated **timer** of the expected time to receive an ACK. When an ACK is received a new packet is sent and the buffer is updated (window slide). When the timer expires, the sender retransmits all buffered frames.
- **Selective repeat**: when a frame is correct, send ACK. When it's missing, **buffer** the following correct frames. When the missing message arrives, send ACK for that one and all the buffered frames. Capacity is used more efficiently but the receiver needs more buffer.
- **Selective reject**: works similar to selective repeat but when a frame is missing, a negative acknowledgment is sent and just that frame is repeated. A variation is to send a list of missing frames instead of a single NACK. This method enhances **efficiency** but it's more complex.



(h) Go-back-N (i) Selective repeat (j) Selective reject

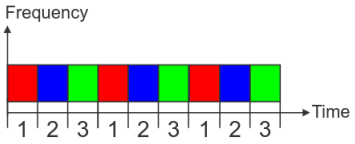
Note 7.4.3.2. Some protocols require many timers: implement them by using one hardware timer and store expiration times in a linked list that's updated during protocol runtime.



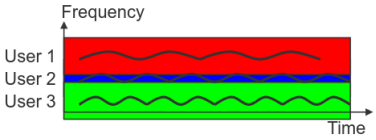
7.5 Medium Access Control

When dealing with a medium shared by $n > 2$ nodes, a NIC needs to manage:

- **Channel allocation**: medium access control, organizes the order of transmitting nodes on the channel. There are two different approaches:
  - **Distributed MAC**: there are n nodes that generate frames for transmission on shared single channel. If two frames are transmitted simultaneously, the frames are lost. Time management can happen in two ways:
    * **Continuous** time: no master clock, transmission of frames can begin at any time
    * **Slotted** time: time is divided into discrete intervals (slot), frame transmission begins always at the start of a slot
  - **Centralized MAC**: a **master device** manages channel allocation
    * **Round-Robin**: the master device polls each node periodically (fixed TDMA) and each node gets the entire transmission capacity for a fixed time interval



    * **FDMA**: the master allocates a different frequency to each node, which gets a portion of the transmission capacity for the whole time

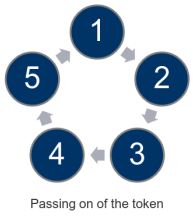


Since users are typically **bursty**, most subchannels will be idle most of the time.

- **Node identification**: addresses of the nodes on the medium, **MAC Addresses**, typically 6bytes long in hexadecimal notation

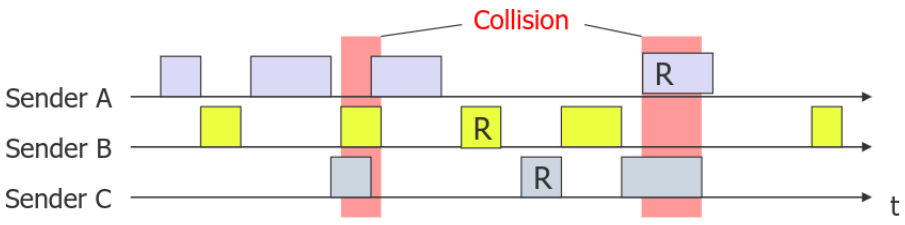
7.5.1 Token

Multiple access using a **token** (a bit sequence) means that only the owner of the token is allowed to send. The token is then passed between all the nodes. It's particularly suitable for a **ring** topology. It guarantees access without collisions, efficient and fair. It's very complex, e.g. when handling a lost token.



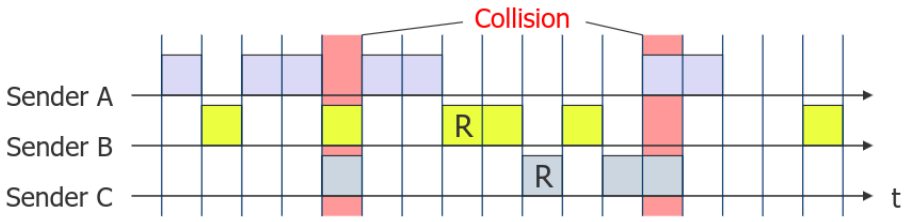
7.5.2 ALOHA

This multiple access technique was derived from the principles of *ALOHANET*[3]. Nodes are **uncoordinated** and they can send at any time on every frequency. When several nodes are sending at the same time, a **collision** occurs: the frame are lost and each sender wait a **random time** to transmit again.



[3]Developed by Norman Abramson on the Hawaiian Islands in 1970s, a network connecting computers on islands over radio. There were two channels: uplink shared by nodes (collision may occur) and downlink used by main computer. The packets were ACKed by the main computer. Good performance only in low traffic.

Slotted ALOHA Even small overlaps would cause a collision, hence having a **low throughput** and no guaranteed response time. The improvement was **slotted** ALOHA: the time axis was divided into **time slots** and each sender could send at any time but only at the beginning of a time slot. This brought fewer collisions but now **synchronization** of the computers was needed.



Performance Given an infinite number of users generating data, the transmissions (and retransmissions) are generated according to a Poisson distribution with intensity λ. Hence, the probability of k transmission attempts in an interval $[0, t)$ is

$$P(X = k) = \frac{(\lambda t)^k}{k!} \cdot e^{-\lambda t}$$

The **throughput** S is given by the load G and the probability of a **successful** transmission P_0

$$S = G \cdot P_0$$

A frame is **successfully** transmitted if no other frames are within the **vulnerability period** t

$$P_0 = P(X = 0) = \frac{(Gt)^0}{0!} \cdot e^{-Gt} = e^{-Gt}$$

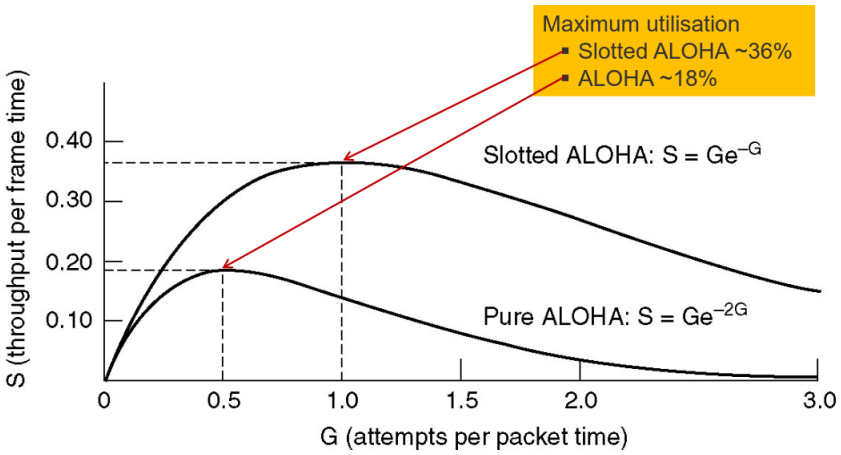
Then, assuming that time unit is the time to send a single frame, we have:

- **ALOHA** with vulnerability period of 2

$$S = G \cdot P_0 = G \cdot e^{-2G}$$

- **Slotted ALOHA** with vulnerability period of 1

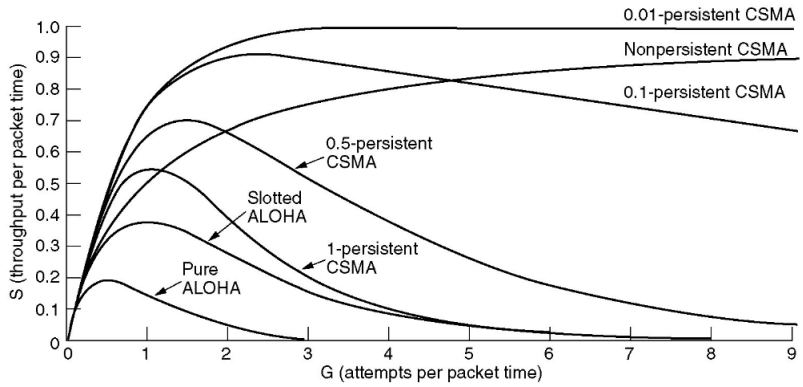
$$S = G \cdot P_0 = G \cdot e^{-G}$$



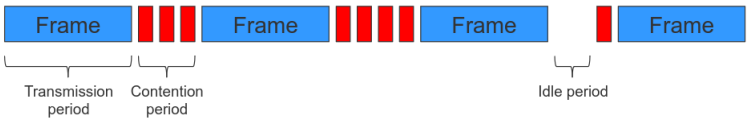
7.5.3 CSMA

Since ALOHA doesn't perform well under high loads, a solution is to reduce collisions: a sender first examines if another node is already transmitting and if nobody is currently sending, they are free to start. This is called **Carrier Sense Multiple Access** and it's a **simple** way with good utilization of the network. The main drawback is that there is **no guarantee** for the access to the medium. It only works with **short transmission delay**.
There are three variants:

- **Non persistent**: when a node has data to send, it first listens to the channel. If it's busy, the computer listens and waits for availability. As soon as the channel is idle, the frame is transmitted. If a collision occurs, the node waits a random amount of time and starts again.
- **Persistent**: when a node has data to send, it first listens to the channel. If it's busy, the node waits a random amount of time and starts again. Else, it starts transmitting.
- p**-persistent**: if the channel is idle, a node that has data to send transmits with **probability** p in current slot and defers until next slot with probability $(1-p)$ going then back to the beginning. If the channel is busy, the node waits from the next slot and goes back to the beginning. If a collision occurs, the node waits a random amount of time and goes back to the beginning. It's applicable in slotted time environments (slotted ALOHA).



CSMA/CD CSMA with **Collision Detection** aims to reduce the waste produced by collisions between long frames: the node listens through their own transmission, if what they hear is different from what they sent, stop the transmission and wait a random amount of time.



7.5.4 Reservation mechanisms

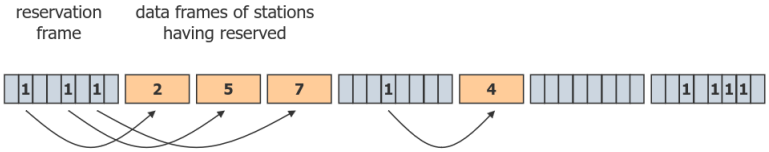
Reserve the channel to avoid collisions. Works in two **alternating phases**:

1. **Reservation**: the sender makes a reservation by indicating the wish to send data (in some cases also the length)
2. **Transmission**: if reservation is ok, the data communication takes place

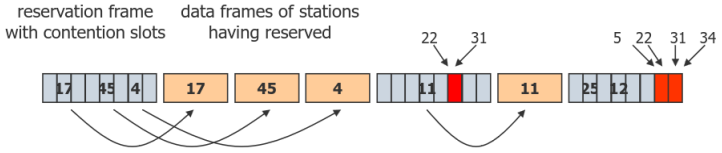
It's very efficient in terms of capacity but there is more delay due to the reservation procedure.
The reservation can be:

- **Centralized**: a master periodically queries all the nodes and checks if they have to send data, assigning sending rights
- **Distributed**

- *Explicit*: **Bit-Map** approach, uses a small **reservation frame** in the first phase and a **data frame** of constant size in the second phase. There are two variants:
  - **Without contention**: each user i is assigned to the ith slot in the reservation frame. If it wants to send the data, it sets the bit to 1. After the reservation phase, all nodes have set their bit and can send the data in the order of the bits in the frame



  - **With contention**: the reservation frame consists of a limited number of contention slots $< i$. Users try to get a random contention slot writing their computer ID into a slot. If there is no collision in the reservation phase, a node may send.



- *Implicit*
  - **Binary Countdown**: there are priorities based on the ID of the computer. When nodes need to send data, they enter a **contention** phase where they broadcast their ID bit by bit. The smaller addresses give up.
  - **Adaptive Tree Walk**: the nodes are the leaves of a binary tree. In the first contention slot following a successful frame (slot 0), all nodes are allowed to try to acquire the channel. If there is a collision, only the nodes from the left subtree are allowed to try. If successful, the slot is reserved for the right subtree.

7.6 Protocols

7.6.1 PPP

Point to Point Protocol establishes a direct connection between two nodes. It supports synchronous and asynchronous connections. It uses:

- Specified **frame format** with **error detection**
- **Sliding window** approach for flow control
- Specified connection establishment and tear down

It doesn't use medium access control since there is no need to on point-to-point links.

Connection The connection happens in three steps:

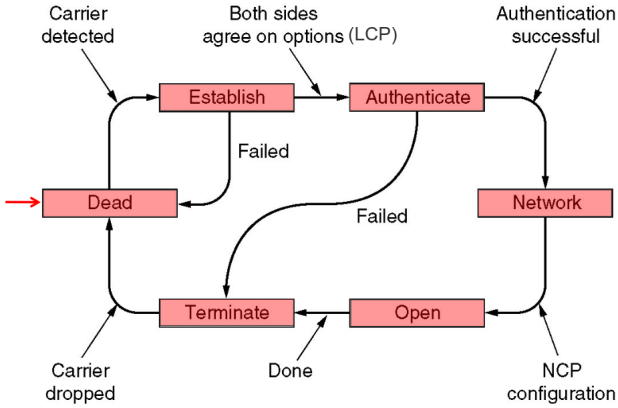
1. PPP establishes a **basic physical layer connection**, testing that the layer is ready
2. **Link Control Protocol** configures the NIC on each end of the link:
   - *Maximum Frame Size* (MTU)
   - *Escape characters*
   - *Magic numbers*: identifying an end or to detect looped links
   - *Authentication method*

   It also manages the **termination** of the link layer connection.

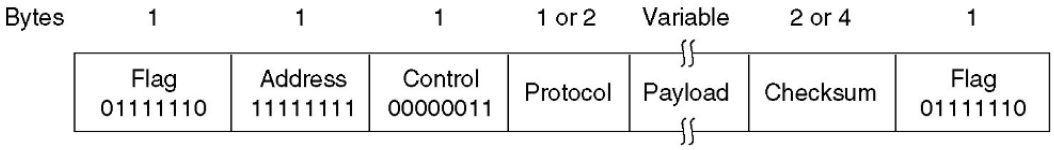
The **frame types** are defined between the *initiator* and the *responder* as follows:

Name	Direction	Description
configure-request	$I \rightarrow R$	List of proposed options and values
configure-ack	$I \leftarrow R$	All options are accepted
configure-nack	$I \leftarrow R$	Some options are not accepted
configure-reject	$I \leftarrow R$	Some options are non negotiable
terminate-request	$I \rightarrow R$	Request to shut the line down
terminate-ack	$I \leftarrow R$	OK, line shutdown
code-reject	$I \leftarrow R$	Unknown request received
protocol-reject	$I \leftarrow R$	Unknown protocol requested
echo-request	$I \rightarrow R$	Send this frame back
echo-reply	$I \leftarrow R$	Here is the frame back
discard-request	$I \rightarrow R$	Just discard this frame (testing)

3. **Network Control Protocol** negotiates the network layer conditions (e.g. Internet Protocol Control Protocol configures IPv4 networks address or compression options) and manages the tear down of the network layer connection (frees IP)



PPP is **character oriented** (byte oriented) and uses **byte stuffing**.



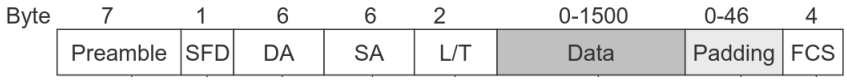
- **Flag**: start of frame with special flag 01111110 while end of frame is omitted for successive frames
- **Address**: useless, inherited from HDLC, it's set to 11111111
- **Control**: set to **unnumbered mode** 00000011, meaning without sequence numbering and acknowledgments
- **Protocol**: specifies to which network layer the payload should be delivered. For IP it's 00000110
- **Checksum**: just **detection**, default is 16 bit CRC with generator polynomial $x^{16} + x^{12} + x^5 + 1$ computed over *Address*, *Control*, *Protocol*, *Payload* and *Padding*

Usage PPP used to be the default link layer protocol for **dial-up** internet access. Now it's still used on **point-to-point optical links** (PPPoE) and for the last mile with DSL (PPPoA).

7.6.2 Ethernet

History After ALOHA, Xerox began experimentation on network over **coaxial** cables. In 1976 Robert Metcalf created Ethernet, an improvement over ALOHA with CSMA/CD. Between 1978 and 1983 the new standard IEEE 802.3 was created.
Initially it was a network over a **bus** topology: it was hard to maintain and debug if the bus got damaged. The solution was to implement a **star** topology where the cable goes through a central point: initially a **hub** (just a repeater, still CSMA/CD) and later a **switch** (one NIC per interface, no need of CSMA/CD).

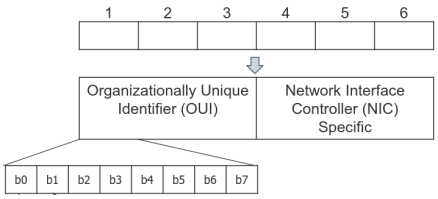
Frame



- **Preamble**: 7 bytes, each one 10101010, flag for synchronization
- **Start Frame Delimiter**: one byte 10101011 marking the beginning of the frame
- **Destination address**: 6 bytes containing the MAC address of the receiver
- **Source Address**: 6 bytes containing the MAC address of the sender
- Depending on the version
  - **Data length** in 802.3, ranging from 0 to 1500
  - Identification of the **upper layer protocol** in Ethernet II
- **Data**
- From 0 to 46 bytes of **padding** to fill up the frame to at least 64 bytes, otherwise small fragments are discarded
- **Frame Check Sequence**: 4 bytes that uses a 32bit CRC computed over *DA*, *SA*, *length/type*,*data* and *padding*

MAC Address The MAC address is a 6 bytes long number assigned uniquely by IEEE and locally administered. It can be:

- **Unicast**, starting with 0
- **Multicast**, starting with 1
- **Broadcast** with all 1



Multiple Access Control Ethernet collision resolution mechanism is based on CSMA/CD: listen before sending and stop as soon as simultaneous transmissions are detected. A simple solution but with a possible large delay before sending.

The waiting period after a collision needs to be kept small to avoid long delays but at the same time that makes the risk of subsequent conflict higher. The solution is **Binary Exponential Back-off**: the random waiting period is drawn from an **increasing interval**. After i collisions the node waits drawing a random number between $[0, 2^i - 1]$, after 10 collisions the interval becomes $0, 2^{10} - 1$ and after 16 the node gives up.

When the channel is free again the node waits x time slots before starting again, where a **time slot** corresponds to the minimum Ethernet frame length.

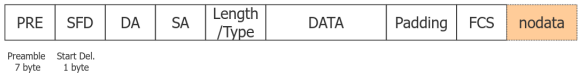
The major drawback is that it tends to prefer last contention winners and new contending nodes over the others, being **unfair**.

Observation 7.6.1 (Collision detection)**.** Since the propagation speed is **finite**, collision detection is not instantaneous. The solution is to adjust **minimum packet length** and **maximum traveled distance**. A node needs to be sure that the beginning of its message reached the receiver before assuming there was no collision and stops listening.

Physical layer The physical layer used with Ethernet is **baseband** encoding with **Manchester Encoding**. The voltages used are $\pm 0.85V$.

Standards Some Ethernet standards are:

- **10Base-2** (cheapnet): cheap coaxial cable. Terminals are attached with BNC connectors, maximum 5 segments, 30 nodes per segment, at least $0.5m$ between each node, $185m$ maximum segment length and a maximum expansion of $925m$
- **10Base-T** (twisted pair): star topology using twisted pair where several nodes are connected through a hub. Devices are attached with a RJ45 plug where 2 out of 4 cables are used. Maximum cable length to the hub is $100m$ and maximum extension $200m$
- **10Base-F**: Ethernet with fiber optics, expensive but excellent against noise. Used to connect distant buildings
- **Fast Ethernet**: concept based on *10Base-T* with a central hub or switch. 100Mbps transmission rate. The speed was a problem for the collision detection, hence the maximum distance was reduced to 200m. It also brought **auto-configuration** of NICs (speed and communication mode)
  - **100Base-T4**: UTP cable CAT3, using all the 4 cables pairs, and 8B6T encoding
  - **100Base-TX**: UTP cable CAT5, using only 2 cable pairs, and 4B5B encoding
  - **100Base-FX**: optical fiber, one per direction, with a maximum cable length to the hub of 400m. There is a variant where only switches are allowed and the maximum length is 2Km
- **Gigabit Ethernet**: to avoid reducing the cable length while maintaining a high speed, a new minimum frame length of 512 bytes was specified. The **Carrier Extension** was made through a *nodata* field after the FCS. This field is added by the hardware while the software is kept unaware.



Now the sending of several successive frames (**Frame Bursts**) is possible without the need of repeated CSMA/CD: the sending MAC fills the gaps between the frames with **Interframe-Bits** (IFG), signaling to other nodes that the medium is still occupied.

  - **1000Base-T**: UTP cable CAT5/6/7, uses 4 pairs, maximum segment length of 100m
  - **1000Base-CX**: STP cable, uses 2 pairs, maximum segment length of 25m
  - **1000Base-SX**: multimode fiber with a maximum length of 550m and on the 850nm band
  - **1000Base-LX**: single or multimode fiber over 5000m and on 1300nm band
- **10-Gigabit Ethernet**: star topology using switch and optical fibers. CSMA/CD is no longer used since collisions cannot occur. There are two specifications on the physical layer:
  - **LAN** with 10Gbps
  - **WAN** with 9.6215Gbps for compatibility with SDH/SONET

The **copper** variants are:

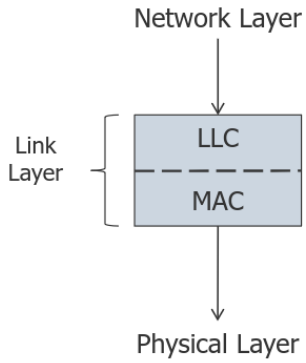
- **10GBase-CX4**: four pairs of coaxial cables for each direction, maximum length of 15m
- **10GBase-T**: either CAT6 (50m) or CAT7 ($100m$) cables, uses all the pairs in both directions in parallel, there is a filter for each cable to separate the sending and the receiving. On the physical layer a variant of **Pulse Amplitude Modulation** is used, with 16 discrete levels between $\pm 1V$

The **fiber** variants are:

Name	Type	Wavelength (nm)	PHY	Coding	Fiber	Range(m)
10Base-SR	Serial[4]	850	LAN	64B66B	Multimode	$26 - 400$
10Base-LR	Serial	310	LAN	64B66B	Singlemode	10.000
10Base-ER	Serial	1550	LAN	64B66B	Singlemode	40.000
10Base-LX4	WDM[5]	$1275, 1300, 1325, 1350$	LAN	8B10B	Both	10.000
10Base-SW	Serial	850	WAN	64B66B	Multimode	$26 - 65$
10Base-LW	Serial	1310	WAN	64B66B	Singlemode	10.000
10Base-EW	Serial	1550	WAN	64B66B	Singlemode	40.000

Logical Link Control LLC interfaces with the network layer to provide:

- **Unreliable** datagram service
- **Acknowledged** datagram service
- **Reliable connection-oriented** service



The header contains:

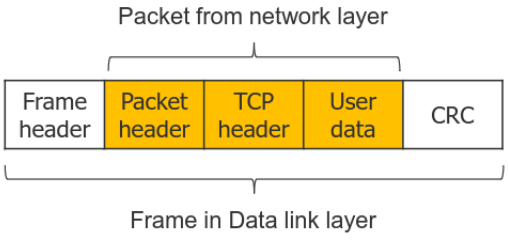
- **Destination** access point: which process to deliver
- **Source** access point
- **Control** field for sequence and ack numbers

Today it's **disbanded**.

7.7 Infrastructure

Different devices are used in different layers of a network.

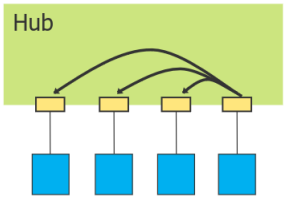
Application layer	Application gateway
Transport layer	Transport gateway
Network layer	Router
Data link layer	Bridge, Switch
Physical layer	Repeater, Hub



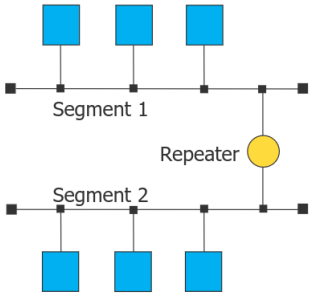
[5]Only one transmission at a time.
[5]Wavelength Division Multiplexing: several transmissions in parallel.

7.7.1 Physical elements

Hub and Repeaters A **Hub** sends the received packets on one side to all of the devices on the other side. A **repeater** links two different networks. Both of them do not understand packets, frames or headers: they just refresh the signal and **increase** the network **range**. They offer one channel shared between all nodes, a low cost but also low security solution.

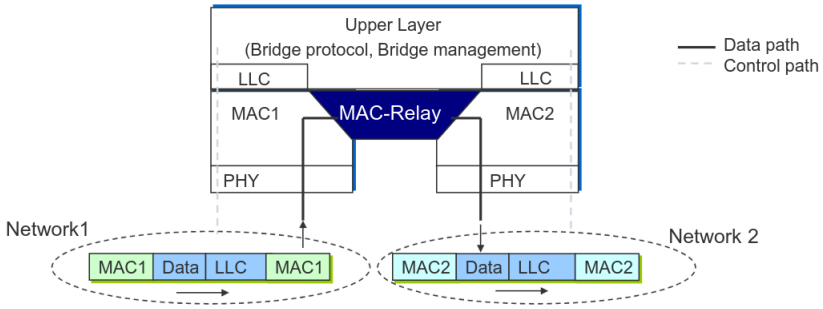


(k) Hub



(l) Repeater

Bridging Since many LAN exists due to load management reasons, security reasons and maximum length reasons, it's fundamental to connect them. A bridge should be **transparent** (different formats, data rates, lengths) and **flexible** (moving a node between networks should not require new HW or SW).
A bridge operates at the link layer, it processes frame addresses and supports different network types.



Each bridge has a **forwarding database** with at least two entries. Each entry has:

- **MAC Address**: host ID
- **Port**: port number of the bridge used to send data to the host
- **Age**: aging time of entry (optional)

The source field of a frame that arrives on a port tells which hosts are reachable from this port. For each frame received the bridge stores the source field in the database together with the port where the frame was received. All entries are deleted after a certain amount of time.

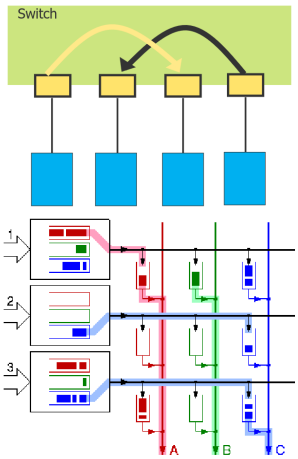
Spanning Tree Protocol Adding redundancy in a network will introduce **endless loops**. The idea behind a Spanning Tree is to relay frames only along the edges of a tree structure. It works as follows:

- At the beginning each bridge assumes to be root and floods a packet containing its ID and current cost initialized at 0 over all of its ports

root ID	cost	bridge ID	port ID

- A bridge that receives such packet checks the root ID and compares it with its own. Both the ID and the costs are updated (its own cost for the sender bridge is summed with current cost value) for receiving packets with smaller ID and forwarded
- When the updated packets of all bridges have passed all other, there is an agreement on the **root bridge**. The received packets containing the smallest costs determine the **designated bridge** for a LAN and designated ports for the bridges to send out data

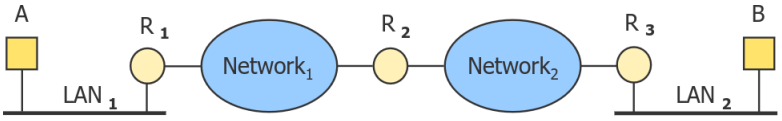
Switches They are similar to bridges, except they have **point-to-point** NICs on each port. They have a **buffer** for each node/port and connected nodes can send and receive at the same time. There are also layer 3 and 4 switches with corresponding layer functionalities.



The most used HW implementation is **buffered crossbar**: for each input port provide buffers for the output ports. At any time only one input port can be connected to an output line. Collisions are almost impossible.

Routers Since bridges uses non-scalable addresses, they are limit with the number of supported nodes. Moreover, they pass broadcast frames on every port, causing **broadcast storms** and do not communicate with hosts to hand over information on errors.

Routers, on the other hand, work on the Network layer: packets are forwarded towards destination based on a **global address**, hence overcoming the restriction on nodes number. Moreover they do not let broadcast through and improve performances by communicating with hosts.



Gateways There are two types:

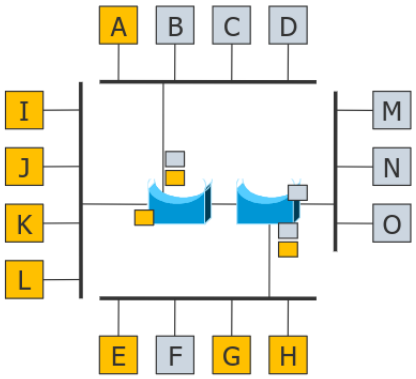
- **Transport** layer: connect computers using different transport protocols, e.g. TCP/IP to ATM
- **Application** layer: understand the format and contents of the data and translate the message, e.g. email to SMS

Structured cabling The idea is to **partition** a network through its cable infrastructure, which is connected to a **backbone** or a central switch. Each user outlet is cabled to a communication closet with an individual cable. In there, the user outlets terminate on **path panels**, usually mounted on 19" racks.
The **advantages** are:

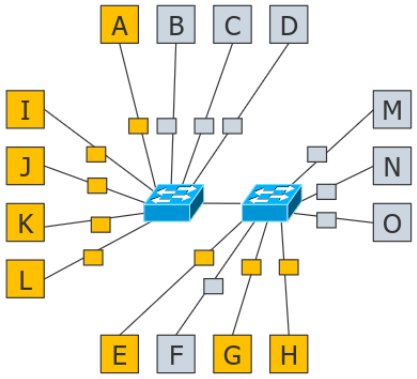
- **Consistency**: usage of the same cabling systems for data, voice and video
- Support **multi-vendor** equipment
- **Simple modifications**: support the changes in within the system (e.g. adding, changing, moving)
- **Simple troubleshooting**: problems are less likely to down the entire network and simplify the isolation and fixing of problems
- **Fault isolation**: by dividing the entire infrastructure into simple management blocks, it's easy to test and isolate specific points of fault and correct them

7.7.2 VLAN

While initially computers of an enterprise network were on a single LAN, today there are several because of new cabling technologies and security/load management necessities.
Virtual LANs allow the configuration on LANs **logically** instead of physically, provided that there is a **decoupling** between the two topologies.



(m) Physical topology

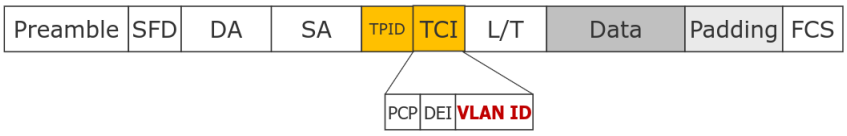


(n) Logical topology

VLAN-aware devices are needed, such as switches that have a table which tells which VLAN is accessible via which port. There are three possibilities:

- Assign each port of the device to a VLAN-ID, allowing only the devices with that ID to attach to it
- Each MAC address is assigned to a VLAN, having a table that keep tracks of that
- Each Layer 3 protocol (IP address) is assigned to a VLAN but violating the layer independence

IEEE 802.1Q specifies a field in the frame header telling the VLAN assignment. The first VLAN-aware device adds the TAG based on the MAC address while the last one removes it. Newer NICs support this.



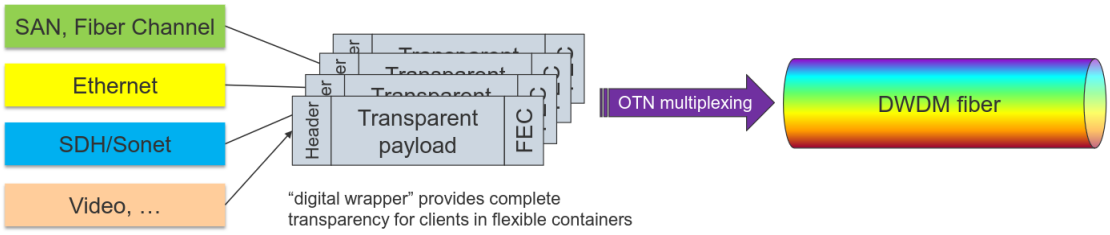
The new fields are:

- **Tag Protocol Identifier** ($0x8100$): serves as flag to differentiate with beginning of L/T field in a non-VLAN frame
- **Tag Control Information**
  - **Priority Code Point**: 3bit priority field, refers to 802.1p
  - **Drop Eligible Indicator**: indicates frames that can be dropped in case of congestion
  - **VLAN Identifier**: 12bit VLAN-ID, between 0 and 4095

Note 7.7.2.1. **802.1ad**, then improved by 802.1ah, allows nested VLAN tags for bridging VLANs over providers.

7.7.3 Optical Transport Networks

OTN creates an optical virtual private network by encapsulating data frames meaning multiple data sources can use the same channel. It replaced SDH/Sonet systems with the standards **G.709**, **G.798** and later. Supports the encapsulation of different protocols (e.g. IP, Ethernet frames) and both variable and constant bit rates.



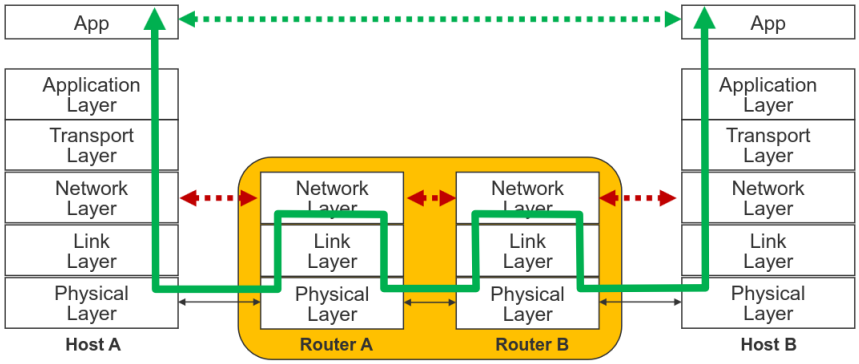
8 Network

The idea behind the network layer is to **interconnect** all the LANs, MANs and WANs. To do that it's necessary to introduce uniform internetwork addresses in an end-to-end packet format, abstracting the lower layers and creating a common format that's **independent** of intermediate link layer technologies. This brings also the necessity to do:

- **Routing**: automatically acquiring and updating next hop information for each destination
- **Forwarding**: moving incoming packets from the input interface to the appropriate output

The main **goal** of the network layer is to manage end-to-end connectivity across multiple networks. To do that it defines:

- Internetwork identifiers
- Uniform end-to-end packet
- Routing and forwarding mechanisms



There are two philosophies for **end-to-end connectivity**:

- Connection-**less**: data is chunked and transferred as packets of variable length, with source and destination addresses in each one. Sending is made spontaneous without reservation. It's easy to implement but brings more challenges (wrong order of packets, delays, unreliability).
- Connection-**oriented**, with three phases:
  - *Connection establishment*
  - *Data transmission*: information exchange between the partners
  - *Connection termination*: release of the terminals and the channels

  It's more complex to implement but brings reservation of capacity, flow control and no problem with sequence numbering.

Intermediate nodes need to **acquire** and **maintain** the **state** to establish paths, scope broadcasts and guarantee service quality. At the same time they need to handle **constraints**:

- **Limited memory**: they cannot store an arbitrary amount of data
- **Limited throughput**: they cannot rely on an arbitrary amounts of control traffic to acquire the state
- **Limited computing capacity** and **speed**: they cannot compute arbitrary complex operations

8.1 Addressing

Addresses are used to **identify** end hosts in multi-access networks and, in case of multi-address hosts, they help to select the right interface. They require:

- **Compactness** of representation
- **Independence** from lower layers
- Built-in support for efficient and decentralized **path finding**
- **Uniqueness** within a network scope

Aggregation Aggregation leads to much smaller intermediate states in intermediate nodes. Some example are:

- **Geographical**: aggregate based on geographical area with a strict assignment policy
- **Hierarchical**: aggregate by organizations and sub-organizations, has a higher flexibility in assignment but may lead to less compression:
  - *Allocation*: initially a range of continuous addresses (determined by **prefix**) is given to an organization, which then can recursively delegate authority over parts of it. At the top the **Internet Assigned Numbers Authority** delegates to the **Regional Internet Registry** (AfriNIC, APNIC, ARIN, LACNIC and RIPE). They then delegate to **Local Internet Registry**, which are mostly ISP, enterprises or academic institution. They in the end assign IP to customers
  - *Assignment*: give an address to a host/interface
  - *Match*: longest common prefix match (match on prefix length), basically forward data to the output interface that shares more digits with the destination

Challenges The main challenges are:

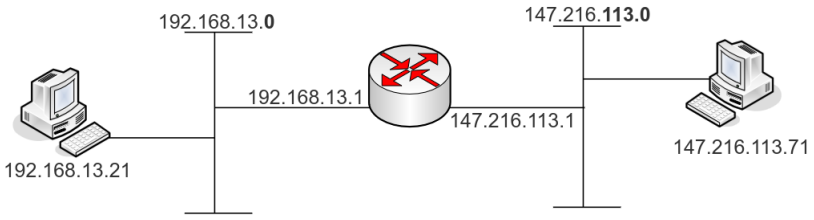
- **Security**: e.g. address spoofing when an attacker claims to own a network address of someone else since there is no global database
- **Efficiency**: effective aggregation requires careful address planning (keep large ranges, give unused addresses back)
- **Automation**: e.g. auto-configuration of address segment

8.1.1 IPv4

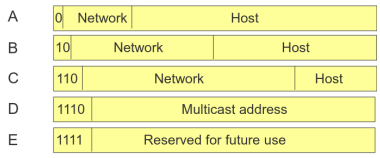
An IPv4 address is made of 4 bytes divided in **network** part and **host** part. They can be:

- **Public**: needs to be **unique**, typically one for each node
- **Private**: not unique and thus not globally routable.

Router or *gateways* usually have an IP address for each network they are linked to.



Classes IP addresses are divided in five classes based on the first bits values.



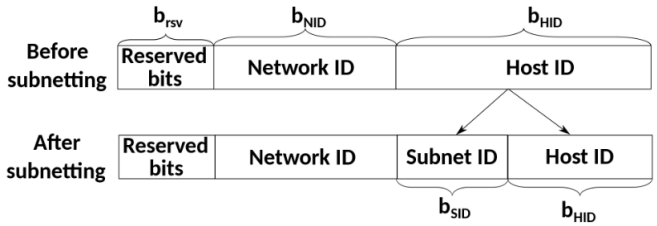
Since nobody expected such an explosive growth of the internet, too many class A addresses were given away. Furthermore, class C networks are very small while class B are often too large.

Variable-Length Subnet Masking To solve the problems of IP addresses, VLSM was introduced: static classes were replaced by network prefixes of any length.
Now an address is in the form $a.b.c.d/n$ where the first n bits are the **network** identification and the remaining $32-n$ are the **host** identification. Another notation is the address and the explicit netmask in the form $u.x.y.z$.

Note 8.1.1.1. VLSM is at the base of **Classless Inter-Domain Routing**.

Definition 8.1.1 (Subnet)**.** *A subnet is a subset of addresses in a class A,B or C network. The principle is that some bits of the host addresses part are used to complement the network ID.*

All hosts on the same physical network use the same subnet mask. Combining IP address and subnet mask, a router determines to which subnet a packet must be sent.

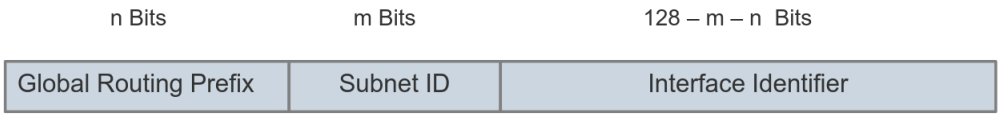


8.1.2 IPv6

In 2011 IANA gives the last available blocks of IPv4 addresses. An extension was needed, hence the creation of IPv6: 128 bits divided in 8 couples of octets (8 bits).

Note 8.1.2.1. IPv6 syntax allows to remove **leading zeros** in each one of the eight blocks and **replace** consecutive blocks of zeros with "::".

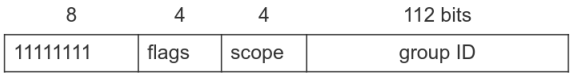
An IPv6 address is divided in three sections:



- **Global Routing Prefix**: depends on RIR policy, for RIPE LIRs get /32 and ISPs get /48
- **Subnet Id**: assigned by network administrator
- **Interface identifier**: hash value identifying the NIC

There are different types of addresses:

- **Unicast**
  - **Unique Local** $FC00 :: /7$
  - **Link Local** $FE80 :: /10$
  - *IPv4-mapped* $:: FF : 0.0.0.0/96$
  - *Loopback* $:: 1/128$
  - *Unspecified* $:: /128$
  - **Global**: every other one
- **Multicast** $FF00 :: /8$, it's an identifier for a group of interfaces. It has the following format:



  - **Flags**: 4 bits (0RPT) divided in:
    - T: if 1, non permanently assigned (dynamically, on-demand), 0 otherwise
    - P: multicast address assignment based on network prefix
    - R: embedding of the rendezvous point
  - **Scope**: defines the scope of the multicast

Value	Scope
0	Reserved
1	Interface-Local
2	Link-Local
3	Reserved
4	Admin-Local
5	Site-Local
6, 7	Unassigned
8	Organization-Local
9-D	Unassigned
E	Global scope
F	Reserved

- **Anycast**: chosen from unicast range and can be used to identify the router in a subnet, an organization or in a particular routing domain. It has the following format:

- **Broadcast**: dropped, use multicast

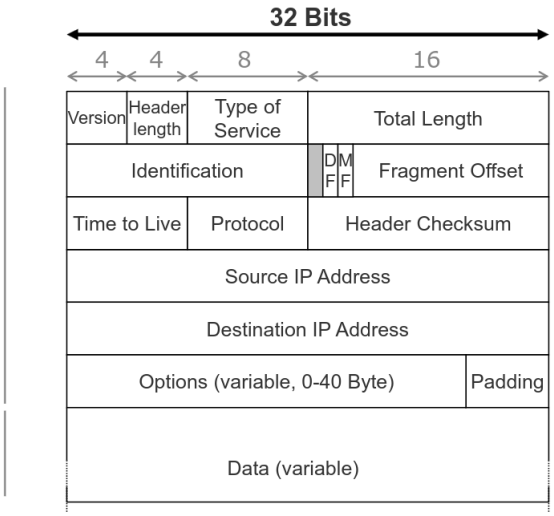
8.2 Internet Protocol

Designing a protocol that serves the need for more than 40 years of technological development is really non-trivial since there is a continuous change of the upper- and lower-layer technologies.
Internet Protocol version 4 is the fundamental network layer protocol of the current Internet. The basic **properties** are:

- **Transparent end-to-end communication** between hosts
- **Connectionless** and **packet-oriented**
- **Unreliable**, it provides a best-effort service
- **Hierarchical addressing**
- Support of **packet fragmentation**

8.2.1 IPv4

The IPv4 packet has the following format:



- **version**: either 4 for IPv4 or 6
- **length**: usually 20 bytes but depends on the IP options
- **type of service**: allows different packets to be treated differently (e.g. low delay for voice, high bandwidth for video). Today it's used for **Differentiated Service Code Point** and **Explicit Congestion Notification**

  DSCP This protocol is the introduction of **QoS** in IP network instead of best-effort. It uses a 6 bit flag and supports different types of services (e.g. video, voice). It defines various **Per-Hop Behaviors** that set the packet forwarding properties for a class of traffic:

  - **Default**: best-effort traffic
  - **Expedited**: loss-low, low-latency traffic
  - **Assured**: assurance of delivery under prescribed conditions
  - **Class Selector**: maintain backward compatibility with the old IP field

  It introduces the principle of **traffic classification**, where each data packet is classified into one of a limited number of classes. Each router is then configured to differentiate traffic based on its class.

- **total length**: number of bytes of the entire packet, the maximum being $2^{16} - 1$ bytes

Fragmentation Every link on the internet has a **Maximum Transmission Unit**, the number of bytes a link can carry as one unit. If the outgoing MTU is smaller than the total packet size, a router can **fragment** a packet, which are then recomposed at destination.

- **identification**: all fragments belonging to the same packet have this same value
- **More Fragments**: a single bit which indicates if this is the last fragment of a datagram 0 or if further will follow 1
- **Don't Fragment**: all router must forward packets up to a size of 576 bytes, everything beyond that is optional. Larger packets with this bit set to 1 might be discarded
- **fragment offset**: it's the position of a packet a fragment belongs to, counted in multiples of 8 bytes
- **Time To Live**: it's used to limit packet travel time and identify and mitigate loops. Its value is decremented by 1 at each router it passes and it's discarded when it reaches 0
- **protocol**: identifies the higher-level protocol, such as TCP or UDP
- **header checksum**: it's the sum of all 16 bits words in the header
- **source IP address**
- **destination IP address**
- **options**: were initially put to provide additional flexibility but now for security reasons are often deactivated since they could reveal information

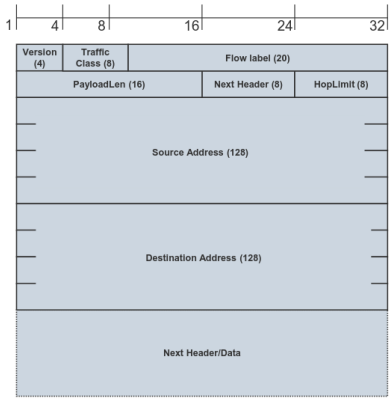
8.2.2 IPv6

IPv6 was implemented first of all to have more addresses. But that was not the only reason:

- Simpler **addressing** and **routing**
- Simpler **protocol architecture**: slim fixed header, optional extensions, format framework for header classes, no header checksum (done in other layers) and no fragmentation
- Simpler **administration**: auto-configuration of interfaces without DHCPv6 and renumbering via prefix change
- Additional **security**, improved multicast, anycast, **QoS**, support of **Jumbograms**

History IP next generation work began in the early 90s: in 1992 there were several proposal for the development of IP. After several merges in 1993 **Simple Internet Protocol Plus** and **Common Architecture for the Internet** were created. In 1994, SIPP was chosen.
In 1995 the IPv6 specification rolled out and in 1999 user addresses were available.
In May 2007 ARIN advises the beginning of IPv6 migration and in 2012 there is the world launch.

Packet The packet is formatted as follows:



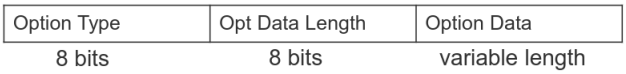
- **version**: IP version number
- **traffic class**: classifying packets
- **flow label**: virtual connection with certain characteristics or requirements
- **payloadLen**: packet length after the 40 bytes header
- **nextHeader**: indicates the type of the following extension header or the transport header
- **hopLimit**: decremented by one at each hop, if zero discarded
- **source address**
- **destination address**, not necessary if there is an optional routing header
- **next header/data**, if an extension header is specified, it follows after the main header, otherwise the data are following

Comparison The IPv6 header is longer than the IPv4 but it's only because of the longer addresses. Otherwise, it's better sorted and thus **faster** to process by routers.
Fragmentation was removed, among with checksum and header length, to leave the problem to the end host and simplify the handling.

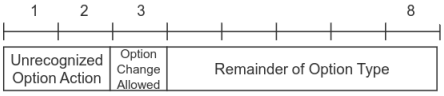
Extension headers They extend basic IPv6 functionalities. Each header may either reference another extension or the data. The main extensions are:

- **Routing**: definition of a full or partly specified route (deprecated since 2007)
- **Fragmentation**: same as IPv4 but now only the source can fragment and if a router finds a packet that's too big, an error is sent back
- **Authentication**: security information, authentication of the sender
- **Encapsulation**: tunneling e.g. for encrypted data
- **Hop-by-Hop options**: dedicated options to be processed by every router and host, at the moment only the Jumbogram implementation is supported
- **Destination options**: additional information for the destination

Options For maximum flexibility, options are possible in headers. They have no length limit and are encoded as **Type-Length-Value**:



The **Option Type** defines also:



- **Unrecognized Option Action**: the action that must be taken if a node does not recognize the Option Type
  - 00: skip this option and continue with the rest
  - 01: discard the packet
  - 10: discard the packet and ICMP the source with *unicast* and *multicast*
  - 11: discard the packet and ICMP the source with *unicast*
- **Option Change Allowed**: indicates whether or not the **Option Data** can be modified while the datagram is en route

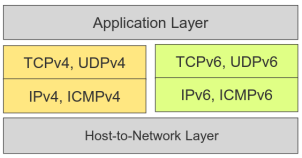
Observation 8.2.1 (Length)**.** IPv6 requires a minimum **MTU** of 1280 bytes. Links which cannot convey such big packets have to apply link-specific fragmentation and assembly at a layer below IPv6.

Path MTU It's the minimum link MTU of all the links in a path between a source node and a destination node. The originating node assumes that the PMTU is the MTU of the first hop. A **trial packet** of that size is sent out. If any link is unable to handle it, an ICMPv6 Packet **Too Big** is sent back and the originating node tries again with a smaller MTU.

8.2.3 Migration

IPv6 cannot be introduced overnight: for some time both variants will coexist. An **incremental deployment** is needed, which includes:

- **Porting**: making applications IPv6 ready, which means using a new API
- **Migration**: done step by step through different techniques
  - **Dual stack**: allowing the coexistence of both versions on the same device and subnet. The application chooses which one to use. It can continue indefinitely but means an increased use of IPv4 NAT.



    One of the main protocols is **Dual Stack Lite**: there is no IPv4 in the public internet, just in the private networks. IPv4 runs then over IPv6 and many customers can share a single globally unique IPv4 address.
  - **Tunneling**: connecting IPv6 regions over IPv4 regions. Routers encapsulates incoming IPv6 packets into a new IPv4 one with destination address of the next router also supporting IPv6. There is more **overhead** but no data loss.The main techniques are:
    * **6-over-4**: IPv6 islands in an IPv4 world with an IPv6 backbone. The end-user site network stuff must choose an IPv6 Internet Service to tunnel to or peer with other islands
    * **6-to-4**: isolated IPv6 islands in an IPv4 world. It allows the islands to automatically interconnect, defining **point-to-multipoint tunnels** and assigning an IPv6 prefix to each IPv4 address (DNS will return both). Then the nearest router identifies 6to4 packet based on a special IPv6 prefix and encapsulates it in IPv4
  - **Protocol Translator** (NAT): let IPv6 devices speak to IPv4 ones

8.3 Assignment and mapping

8.3.1 ARP

The **Address Resolution Protocol** discovers the MAC addresses associated with an IP address, which is needed for the lower layers that do not understand IP.
Each host stores known IP and MAC addresses in a table: the **ARP cache**. The entries become invalid after a certain amount of time, to avoid mistakes. The entries are inserted through ARP **requests** (broadcasts) and **responses**.
To optimize the process, each computer occasionally sends an ARP request to it's own IP address. On some systems a host periodically sends out requests for each entry.
The packet format is designed to be used with various network and link layers by specifying the length:

<table>
<tr><td colspan="2">Hardware type (2 bytes)</td><td>Protocol type (2 bytes)</td></tr>
<tr><td>Hardware address length (1 byte)</td><td>Protocol address length (1 byte)</td><td>Operation code (2 bytes)</td></tr>
<tr><td colspan="3">Source hardware address</td></tr>
<tr><td colspan="3">Source protocol address</td></tr>
<tr><td colspan="3">Target hardware address</td></tr>
<tr><td colspan="3">Target protocol address</td></tr>
</table>

Risks Since an ARP request-response is **stateless** and **not authenticated**, the ARP cache is updated every time it receives a reply, even if it didn't send out one. This creates risks for:

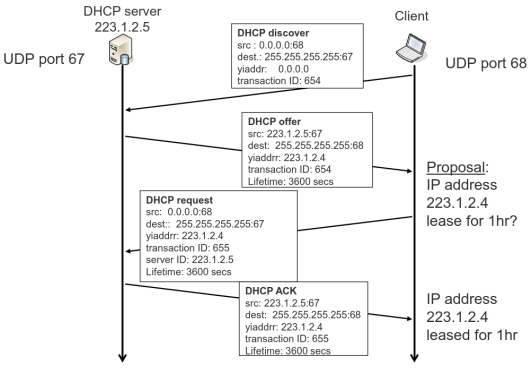
- **Spoofing**: a rogue machine can spoof other machines by replying with wrong data
- **Cache poisoning**: a rogue machine can poison an ARP cache by sending forged ARP replies

8.3.2 RARP

The **Reverse** Address Resolution Protocol a computer can ask it's own IP address by sending out his MAC address. A RARP server then replies with the IP. Deprecated.

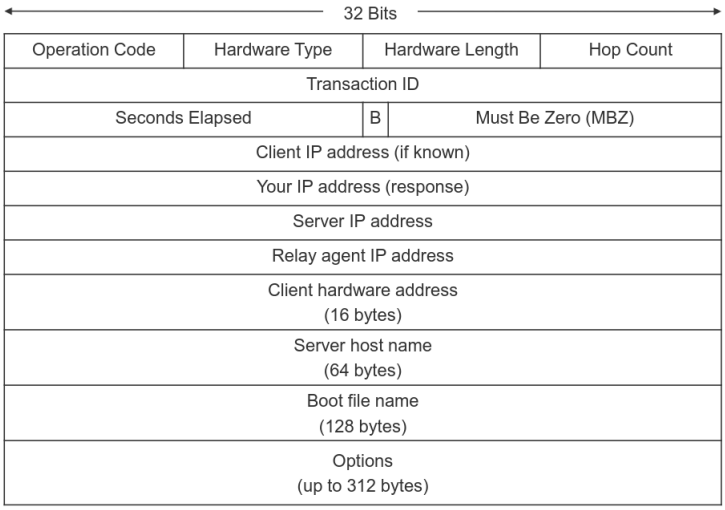
8.3.3 DHCP

Since RARP requests are not passed on by routers, the computer sends out a DHCP packet. In each subnet there is also a **DHCP Relay Agent** which passes those messages to the DHCP server.
Furthermore, to communicate globally you need more than the IP address (subnet mask, defaul router, DNS server) and those are configured by the DHCP.
DHCP works in three phases:



- **Service discovery**: the host does a **DHCP Discover** (broadcast), where he states his MAC address and asks for an IP one. The server then answers with a **DHCP Offer** (unicast) where he offers the IP address and other configurations (DHCPINFORM).
- **IP Address Assignment**: the host sends out a **DHCP Request** (broadcast) stating his MAC address and the IP address that he wants to use. The server sends out a **DHCP Ack** (unicast) to accept that (or NACK to deny).
- **Address release**: IP addresses may be permanently assigned or leased for a limited period of time. After the lease expiration you need to explicitly renew the lease or relinquish the address (**DHCPRELEASE**)

The DHCP packet format was mostly inherited from a previous protocol (BOOTP) and its the following:



Note 8.3.3.1. DHCP uses UDP and therefore is **unreliable**.

8.3.4 NDP

In IPv6 instead of ARP, **Network Discovery Protocol** is used. This protocol is also used for:

- **Neighbor Unreachability Detection**
- IPv6 **address autoconfiguration**
- **Router discovery**
- **On-link prefix discovery**
- **Next-hop determination**
- **Link parameter discovery**, e.g. MTU
- **Redirect**: indicate better next-hop
- **Duplicate Address Detection**

The packet format is a classic ICMPv6 one:

The possible operations are, compared to IPv4:

IPv6	**IPv4**
Neighbor Solicitation	ARP Request
Neighbor Advertisement	ARP Reply
Neighbor Cache	ARP Cache
Router Solicitation	Router Solicitation
Router Advertisement	Router Advertisement
Redirect Message	Redirect Message

Note 8.3.4.1. One advantage of using NDP is that it's a pure network protocol contrary to ARP, meaning that one can also apply network layer security mechanisms solving problems like spoofing.

8.3.5 SLAAC

State-Less Address Auto-Configuration is the IPv6 stateless equivalent for DHCP. It's divided in three phases:

1. **Acquire Link Local** IPv6 address: generate a tentative link-local based on the MAC address, used **Duplicate Address Detection** (send a *Neighbor Solicitation* to that address): if the address exists, fail and go to manual, otherwise if there is no response, convert the **Extended Unique Identifier** 48 bits MAC into a EUI 64 bit dynamically assigned host identifier:
   (a) Leftmost 24 bit of the MAC form the leftmost 24 bits of the EUI-64
   (b) Rightmost 24 bit of the MAC form the rightmost 24 bits of the EUI-64
   (c) Insert the constant $FFFE$ in the middle
   (d) Tweak left-most bit 7 from zero to one
   (e) Prepend $FE80$ ::
2. **Contact router**: once the link-local address is assigned to the interface, the node sends out a **router solicitation** to the well-known address $FF02 :: 1$ (All Routers) and the router responds with a **Router Advertisement**
3. **Configure Global IPv6 Addresses**: process the *router advertisement* message that has been received, which includes the network ID prefix and two flags:
   - **Managed Address Configuration**: if 1, stop and do stateful configuration, otherwise assign the global IP Global IP:Host ID/64 and check the other flag
   - **Other Stateful Configuration**: if 0, stop, otherwise use stateful configuration for other information such as name server

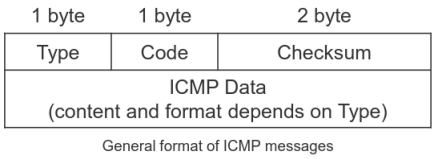
8.4 Diagnostic

8.4.1 ICMP

Internet Control Message Protocol is a **control** protocol of layer 3. It has two use cases:

- If a router cannot forward a packet, the source can be informed about it via an ICMP message
- **Ping** uses ICMP request and reply to check if host is online

The packet format is the following:



General format of ICMP messages

- **Type**: purpose of the message, 40+ types are defined and 41-255 are reserved for future use. The most important are:
  - 0: *destination unreachable*
  - 3: *echo* request and reply
  - 4: *Source Quench* (choke packet)
  - 11: TTL reached 0, packet discarded
  - 12: parameter problem on datagram
  - 15/16: information request and reply
  - 30: *traceroute*
- **Code**: additional information about the condition

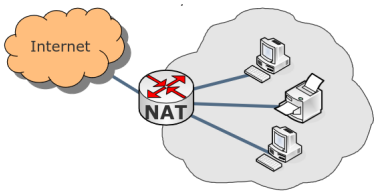
- **Checksum**
- **Data**: there are mainly two message categories
  - **Error**: to report an event. These messages don't have a response and include the IP header that generated the error
  - **Queries**: to get some information from a node, they have a matching response

IPv6 The packet format is the same for IPv6 but it's used for way more purposes, such as:

- Automatic address configuration (NDP/SLAAC)
- Automatic MTU Path discovery
- PacketTooBig is sent to source (IPv6 has no fragmentation)
- IGMP
- Detect malfunctioning router and inactive hosts
- Routing (RPL)

8.5 Network Address Translation

Each device needs an IP address, but they are scarce. Hence a well-known block of IP addresses were reserved for site-local use:



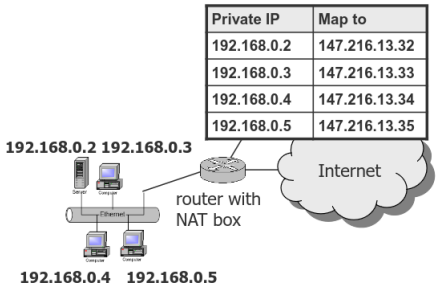
- Class A: 10.0.0.0 to 10.255.255.255
- Class B: 172.16.0.0 to 172.16.255.255
- Class B: 192.168.0.0 to 192.168.255.255

This allows for the reuse of the same private address within a different site **scalability** but introduces the need of sharing a public IP address through NAT.

Observation 8.5.1 (Security)**.** While some may argue that NAT hides the internal structure of a network and avoids exposing hosts, it doesn't actually improves security (e.g. doesn't prevent DDOS), it's only an excuse to prevent deployment of clean networks.

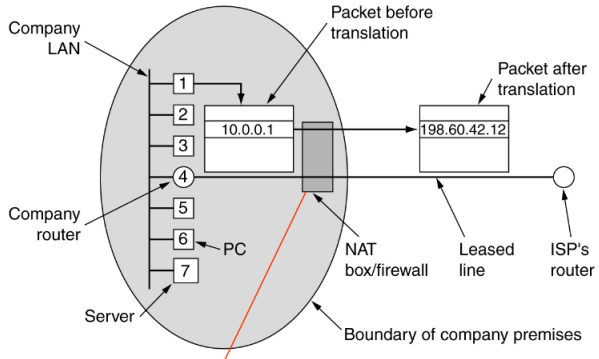
8.5.1 Static

Static NAT hides the structure of the network using one public IP address per private IP address, mapping them permanently. This means that we need as many public addresses as the private ones. **Temporary mapping** can be done, managing a pool of public addresses and assigning them dynamically when a request is sent out, but during peak traffic the number of public IPs used is almost the same as the private ones.



8.5.2 One-to-Many

Also known as **masquerading**, translates several local addresses into the same public address. Demultiplexing is now necessary and it's done through **Network Address Port Translation**: when a packet is sent out, the NAT box saves the local IP and port and maps them to a global port and IP, together with the destination IP and port. When the message comes back, it's remapped to the original address.



Protocol	Port (local)	IP (local)	Port (global)	IP (global)	IP (destination)	Port (destination)
TCP	1066	10.0.0.1	1066	198.60.42.12	147.216.12.221	21
TCP	1500	10.0.0.7	1500	198.60.42.12	207.17.4.21	80

The main **problems** with this NAT technique arise when the message arrives from the outside. Some static information need to be stored, such as "forward each request with port 80 to host x". Still, problems remains (e.g. instant messaging).

8.5.3 Carrier Grade

Also known as **Large Scale** NAT, it uses a NAT box for a large number of customers (e.g. a street) to further mitigate address exhaustion. It breaks end-to-hand principle and it's difficult to use well-known ports.

8.5.4 NAT64

It's a mechanism to connect IPv6 hosts to IPv4 servers: the client embeds an IPv4 address and then the NAT changes the IP headers, mapping IPv6 to IPv4.

8.5.5 Issues

The main issues with NAT are:

- **Header checksum**: changing IP addresses or port numbers on the fly requires also the recalculation of the checksum
- **Fragmentation**: since transport ports are part of the first fragment and packets may be delivered out of order, you need per fragment states in the NAT box
- **Encryption**: you cannot change fields values that are encrypted
- **Protocol-specific** issues above network layer, e.g. with DNS you will need static NAT mappings to match DNS entries

Overall, while NAT saved the internet, it killed the principle of **end-to-end** communication being initiated by one side.

8.6 Routing

A **router** (any host can be) accepts packets from third parties and forwards them. Router behavior coordinates within the network.
Inside a router there are:

- **Routing table**: contains the paths selected by the router for the forwarding table
  - *Destination network*
  - *Network mask*
  - *Meta information*, e.g. path length
  - *Next hop*
- **Forwarding table**: contains the destination network to which the router sends the packet

To find the entry that overlaps most with a destination IP address, **Longest Common Prefix Match** is used: the bit of the destination address are compared to the network mask of each entry and the one with the longest match is chosen. If no entry fits, the **default route** is chosen.

Path selection also depends on the **application scenario**:

- **Intra**-domain: when routers belong to the same administrative domain they all have the same understanding what a best path is
- **Inter**-domain: when routers belong to different domains, path selection with individual policies is needed

Note 8.6.0.1. Let's keep in mind two **prerequisites**:

- Each node must have at least one **unique** IP address (may have multiple)
- The **scope of validity** of the IP address should be appropriate (not link-local or private in case of global communication)

8.6.1 Router vs Forwarding

Definition 8.6.1 (Forwarding)**.** *A **local** process, which runs on a single machine, and moves incoming packets from input interface to appropriate output interface, towards their destination.*

Definition 8.6.2 (Routing)**.** *A **distributed** process, which requires the cooperation of several devices (routers) in the network. Monitoring and exchanging enough network topology information between them to allow correct forwarding decisions on each one.*

Definition 8.6.3 (Routing table)**.** *The **raw topology** information acquired by routing. It's shape and format depends on the used technique, sometimes it may not even exist.*

Definition 8.6.4 (Forwarding table)**.** *The **final product** of the routing process: it's derived from the raw topology information acquired by the routing scheme.*

Forwarding scheme Moves packets from one input interface to appropriate output interface, according to what is currently in the forwarding table. The requirements are:

- **Scalability** in therms of **IO transfer speed**
- **Fast table lookup**:
  - **One-dimensional**, depends only on the destination, paths from different sources coincide when they overlap
  - **Two-dimensional**, depends on the source and the destination, the paths may differ

  Furthermore, the match can be done:
  - **Longest Prefix Match**
  - **Exact Label Match**
- **Efficient** and sophisticated **data structures**

Forwarding schemes can be:

- **Deterministic**: decision based only on key lookup
- **Probabilistic**: decision based on key lookup and/or **random**

So, in the end, routing is the **control-plane** that computes, populates and maintains the forwarding table, while forwarding describes how the data flows, the **data-plane**.

	forwarding	**routing**
goal	direct packet to outgoing link	computes paths packet will follow
scope	local	network-wide
timescale	nanoseconds	milliseconds

8.6.2 Routing schemes

A routing scheme **monitors** and **exchanges network topology information** between routers to allow correct forwarding decisions on each one of them. The requirements are:

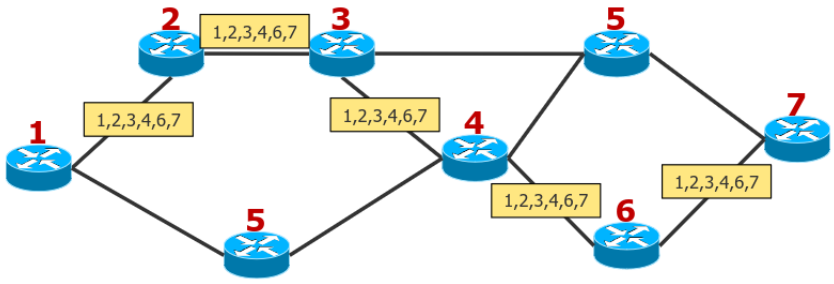
- **Scalability** in terms of **network-wide convergence speed**
- **Small control overhead** in terms of memory, throughput and CPU
- **Fault tolerance** to guarantee maximum availability
- **Loop-freedom**
- **Flexibility** to accommodate different path requirements for different applications

Static vs Dynamic

- **Static**: static forwarding tables are used, hence no reaction to changes in the network. It's **simple** and **stable** but in case of link or router breakdown, catastrophic
- **Dynamic**: routing tracks the network continuously and the forwarding tables are updated upon topology changes. It's **fault-tolerant** and more **reliable** but more **complex** and with more **overhead**

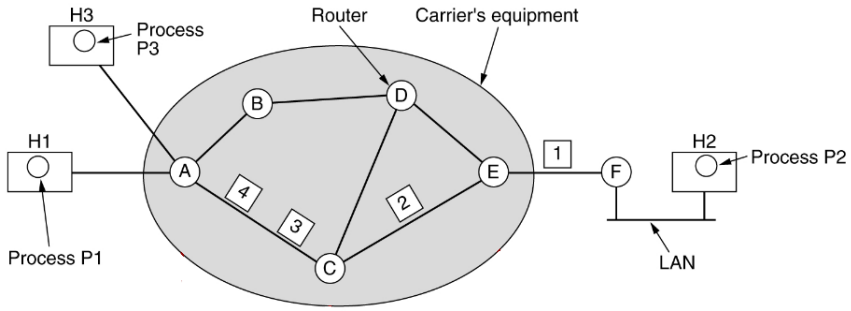
Source vs Hop-by-Hop

- **Source**: the path is determined by the source node and the whole path is included in the packet. In this scheme the routers do not need state but there is no fault tolerance if a downstream router or a link breaks

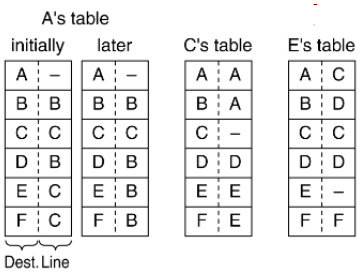


- **Hop-by-Hop**: the path is determined by each intermediate router, allowing for fault-tolerance but introducing state

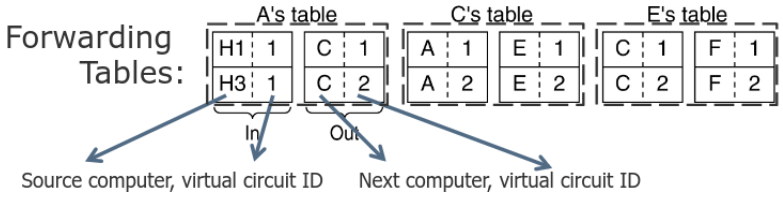
Packet vs Virtual Circuits



- **Packet**: **connectionless**, forwarding with the longest *prefix match* at each hop



- **Circuit switching**: **connection-oriented**, forwarding with exact match at each hop



	Packet	**Circuit Switching**
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC label
Forwarding	Longest prefix match	Exact label match
Routing	Dynamic routing	Static routing
Router failure impact	None, except for packet loss during the crash	All VCs that passed through the failed router are terminated
QoS	Difficult	Easy if enough resources can be allocated in advance
Congestion control	Difficult	Easy if enough resources can be allocated in advance

Reactive vs Proactive

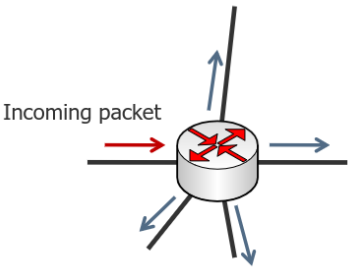
- **Reactive**: a path is discovered and maintained only when applications ask for it, less control overhead but path exploration delays, hence needs user data buffering
- **Proactive**: al possible path are discovered and maintained a priori, avoiding delays and buffering but introducing control overhead

Hierarchical vs Flat

- **Hierarchical**: definition of separate virtual regions in the network, hierarchically organized in a simil tree structure, with a central region as root. Less state required, as routers only need to know about their region, but may lead to worse paths since there is partial topology knowledge.
- **Flat**: all routers are part of the same region, having more state and overhead but being able to reach the best paths due to full topology knowledge

8.6.3 Basic concepts

Flooding Flooding means that each router **repeats** a received packet over all its interfaces, except for the source one. Each packet will reach all routers, meaning very **robust** in case of failure. At the same time there are several problems: **privacy**, **goodput decrease** and there must be a **loop detection**.



Note 8.6.3.1. In a **wireless** context, flooding means repeating packets on the interface they were received from.

In conclusion, flooding does not scale well but it's used by some routing algorithms (OSPF) and can be used as a benchmark.

Metrics Metrics compares the available paths discovered by routing to find the most favorable one. This depends on different criteria:

- **Response time**
- **Throughput capacity**
- Least number of **intermediate devices**
- **Avoidance** of local **overload**
- **Security** requirements

Routing algorithms generally take as input a **cost** per *link*. Unless specified otherwise, all links have the same cost and it may be updated by external mechanisms. The algorithms then find the available paths with the smallest **end-to-end cost**, which depends on three main metrics:

- **Additive**: add cost of each link on the path, e.g. delay
- **Multiplicative**: multiply cost of each link on the path, e.g. packet delivery ratio
- **Concave**: minimum cost of all links on the path, e.g. throughput

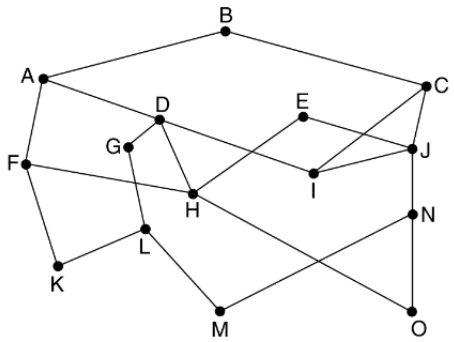
Note 8.6.3.2. The default metric, in case of equal cost, is the **hop-count**.

Determining the optimal path may be difficult because topology and information are never instantly available. Moreover, some metrics leads to **NP-complete** computations and **conflicts** between fairness and optimum can arise.

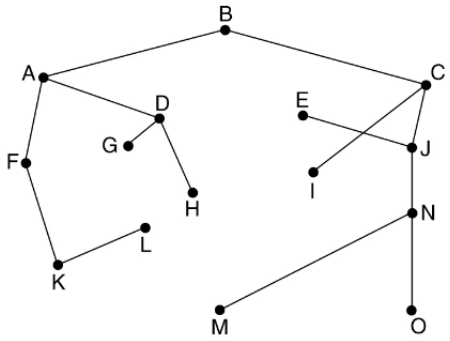
Graphs A computer network can be abstracted as a weighted graph where:

- **Vertex** is a router
- **Edge** is a link
- **Weight** is the cost of the link

Routing algorithms must discover and maintain a **partial**, but **sufficient** view of the topology to make paths available, from anywhere to everywhere.



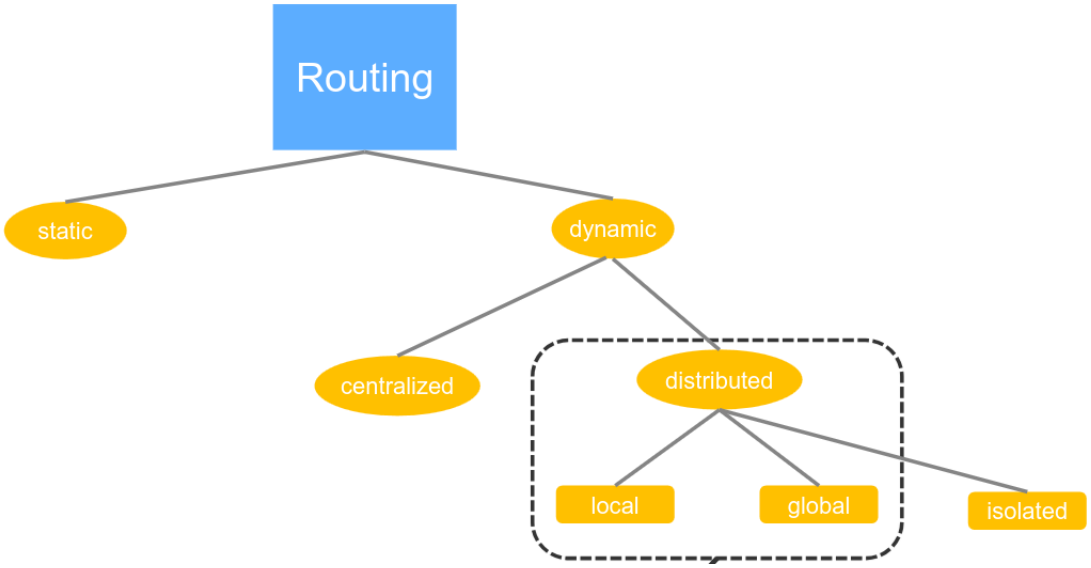
(o) Full topology



(p) Partial topology

8.6.4 Algorithms

Routing algorithms are divided as follows:



Distance Vector Routing E.g. **RIP**. The main principle is that **global information** is exchanged **locally**:

1. Each router periodically sends out a list of destinations it can reach including the costs to each one
2. If a router received that list and updates its own forwarding table with new paths or better ones
3. The algorithm converges towards a state where the forwarding table of each router indicates the least cost path to each possible destination

This corresponds to the **Bellman-Ford** algorithm:

```
Initialization:
   for all destinations y in N:
      D_x(y) = c(x,y) // If y not a neighbor then c(x,y) = ∞
      for each neighbor w
         D_w(y) = ∞ for all destinations y in N
      for each neighbor w
         send distance vector DV_x = [D_x(y) : y in N] to w

Loop:
   wait (until I see a link cost change to some neighbor w or
              until I receive a distance vector from some neighbor w)

   for each y in N:
      D_x(y) = min_v{c(x,v) + D_v(y)}

   if D_x(y) changed for any destination y
      send distance vector DV_x = [D_x(y) : y in N] to all neighbors
```

If N is the maximum length of a path, maximum N steps of the algorithm are required for the "**good news**" (a better path) to travel. On the other hand, if a router is **down**, there may be **loops** due to the link weight becoming infinite.

To avoid them we apply the **split horizon**: a router does not propagate information about a rout back over the same interface from which the route was received. Furthermore, we can use **Poisoned Reverse**: actively advertise routes as unreachable over the interface over which they were learned.

Another problem may arise if someone announces **wrong information**, such as a forwarding table with cost 0 to all destinations. In that case, nearly all traffic would be directed to that router, causing the collapse of the network.

Link State Routing E.g. **OSPF**. The main principle is that **local information** is exchanged **globally**:

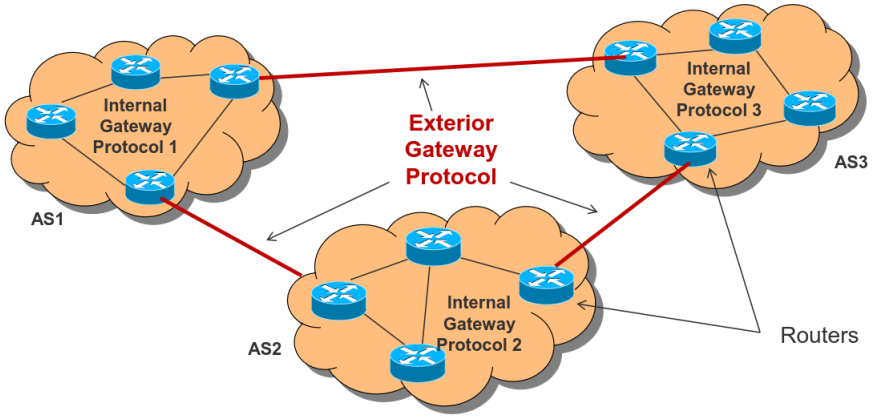
1. Each router periodically determines its neighbors via **HELLO** packets
2. They **flood** this information via **Link State Advertisement**, which contains
   - Identification of the sender
   - List of neighbors with associated link costs
   - Sequence number
   - Age
3. Each router receives LSAs and maintains its own **Link State DataBase** aggregating all received data
4. Each router can derive its **forwarding table** from its database. If it gets updated, the router recomputes the forwarding table. The algorithm used is **Dijkstra**:
   (a) Mark the source node as **permanent** (distance and line don't change), determining the **work node**
   (b) Consider **neighbors** and compute the distance based on current knowledge
   (c) Choose the node with the smallest distance to the source, mark it as permanent and set it as work node
   (d) If there are still non permanent nodes, go back to step (b)

```
#define MAX_NODES 1024 /* maximum number of nodes */
#define INFINITY 1000000000 /* a number larger than every maximum path */

int n;
int dist[MAX_NODES][MAX_NODES]; /* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{
   // s = source node, t = terminal node
   struct state { /* the path being worked on */
      int predecessor; /* previous node */
      int length; /* length from source to this node */
      enum {permanent, tentative} label; /* label state */
   } state[MAX_NODES];

   int i, k, min;
   struct state *p;

   for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
      p->predecessor = -1;
      p->length = INFINITY;
      p->label = tentative;
   }

   state[t].length = 0;
   state[t].label = permanent;
   k = t; /* k is the initial working node */

   /* The algorithm starts with the terminal node */
   do { /* Is there a better path from k? */
      for (i = 0; i < n; i++) /* this graph has n nodes */
      if (dist[k][i] != 0 && state[i].label == tentative) {
         if (state[k].length + dist[k][i] < state[i].length) {
            state[i].predecessor = k;
            state[i].length = state[k].length + dist[k][i];
         }
      }

      /* Find the tentatively labeled node with the smallest label. */
      k = 0;
      min = INFINITY;

      for (i = 0; i < n; i++)
      if (state[i].label == tentative && state[i].length < min) {
         min = state[i].length;
         k = i;
      }
      state[k].label = permanent;
   } while (k != s);

   /* Copy the path into the output array. */
   i = 0;
   k = s;
   do {
      path[i++] = k;
      k = state[k].predecessor;
   } while (k >= 0);
}
```

The main advantages of Link State Routing is that it avoids the **count-to-infinity** problems and in general wrong information from a single router do not crash the network.

8.6.5 Protocols

The Internet consists of a large number of **Autonomous Systems**, each one operated independently using an **Interior Gateway Protocol**. All gateways use the same **Exterior Gateway Protocol**.



RIP The **Routing Information Protocol** was the first IGP used. It's based on **distance vector** and it's **packet**, **flat** and **proactive**.
RIP messages are sent every 30 seconds as UDP datagrams and they contain up to 25 entries of the routing table. The metric used for evaluation is the **hop-number**, with a maximum of 15 ($16 = \infty$).

RIPv2 introduced the support for **subnets**, **CIDR**, **authentication**, **multicast** and more, but still limiting the max number of hops to 15.

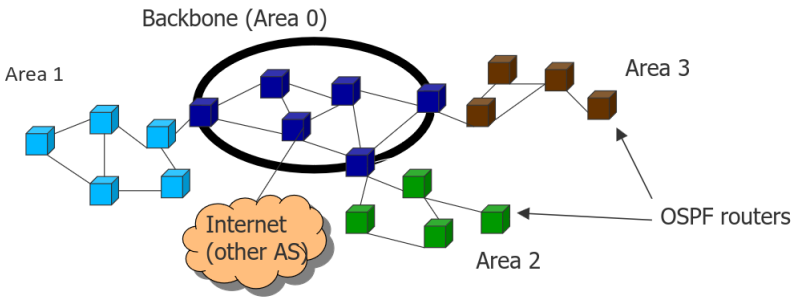
RIPng is an extension of RIPv2 to support IPv6.

OSPF **Open Shortest Path First** is an IGP based on **link-state** and it's **hierarchical**, **packet** and **proactive**. It supports many different metrics, load balancing, hierarchical systems and security mechanisms to protect routers from wrong information.
It has modular support for three categories of link layers:

- **Point-to-point** links between routers
- **Broadcast networks**, mostly LANs
- **Multi-access networks** without broadcasting, e.g. packet switching WANs

Since it's hierarchical, there are four types of routers:

- **Internal**: belong only to one area
- **Area-border**: connect two or more areas, thus part of the backbone
- **Backbone**
- **AS border**: connect to other autonomous systems



Within an area every router has the same LSDB, while an *area-border* one has LSDBs from each area it is connected to.

IS-IS **Intermediate System to Intermediate System** it's a very similar IGP to OSPF, with the difference of being **neutral to layer 3 protocol** and thus getting faster support for IPv6. In this protocol, routers also build a **map of the network** to calculate the shortest path.
Has a lower overhead compared to OSPF and thus it's used in networks with a large number of routers.

BGP In order to have an EGP, we need:

- **Scalability**
- **Privacy**: networks want to hide their internal topologies
- **Policy enforcement**: networks need to control independently where to send and receive traffic

Thus, Distance Vector and Link State routing do not work, mainly because they agree on a common distance while Internet routing is much more diverse.

Border Gateway Protocol is a **path vector** protocol. BGP routers exchange **path vectors** with neighbors (**peers**), which then either accept or discard based on **policies**. Neighbors are typically connected directly or via switch but they can also be not topologically adjacent (**Multihop Peering**).

1. BGP peers establish a TCP connection (**OPEN**, port 179) to exchange (**UPDATE**):
   - **Network Layer Information**: prefixes
   - **Paths**: list of AS numbers to reach a prefix

   or to withdraw them. Two more messages exist: **KEEPALIVE** to check if adjacent peers are still available and **NOTIFICATION** to handle errors.
2. BGP router collects all paths and stores them in the **Routing Information Base**
3. Based on local policies and attributes a **preference** is assigned to all RIB entries
4. For each IP prefix select one best route, choosing *preference* over *shortest path*
5. Before announcing entries to the neighbors, filter based on **local policies**, which can be
   - **Selection**: controls outbound traffic, specifying which path to use (e.g. customers over peers)
   - **Export**: controls inbound traffic, specifying which path to advertise

Note 8.6.5.1. Each AS appends itself when propagating announcement.

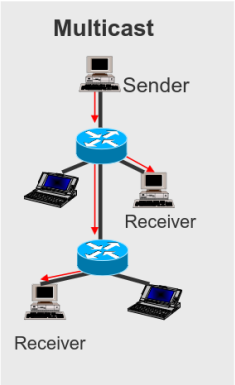
Note 8.6.5.2. To debug BGP, **Looking Glasses** and **Routing Information Services** are used, giving you access to routing tables of BGP routers.

Note 8.6.5.3. The duty of AS operators is to ensure **reachability** of their IP addresses or the IP addresses of their customers. There are two types of relations:

- **Customer-Provider**: the customer pays provider to get Internet connectivity. Usually records every 5 minutes the number of bytes sent and at the end of the month they bill in respect to the 95th percentile of the records
- **P2P Peering**: peers don't pay each other

BGP has a lot of **problems**: **stability**, **routing table growth**, **load-balancing**, **hijacking**, but most of all **security**. While BGP peering sessions can be encrypted, they cannot be verified since the protocol is based on mutual trust.

8.7 Multicast



Group communication is used in multiple contexts, such as voice and video conferencing, content broadcasting (IPTV), games, etc..

Among the different types of communication, **multicast** is the most efficient to transmit to $n > 1$ stations while sending the data only once. Multicast addresses a group of hosts based on a **single group address**, enabling the sender to deliver the data without knowing the receiving hosts.

The main idea is to use a **publish/surprise** paradigm, where the sender publishes independently of the receivers, which subscribes to the data independently of the first. The major protocols are **Internet Group Management Protocol** for IPv4 and **Multicast Listener Discovery** for IPv6.

There are two types of multicast:

- **Any Source Multicast**: receive what anybody sends to the group without knowing the sources
- **Source Specific Multicast**: receive only what a specified source sends, explicitly selecting it and filtering

8.7.1 Subscription

We need a group subscription mechanism to reduce broadcasts, meaning that we want **switches** that know which ports belong to a certain group and **routers** that know about receivers to establish dynamic routing states.

MLD and IGMP messages are encapsulated into IP packets and are used between receivers and routers. When a receiver subscribes to a group, it informs the router about the interest. Then, periodically, the routers ask which groups are still present. If at all, they switch to only listen IGMP and MLD messages.

8.7.2 IGMP

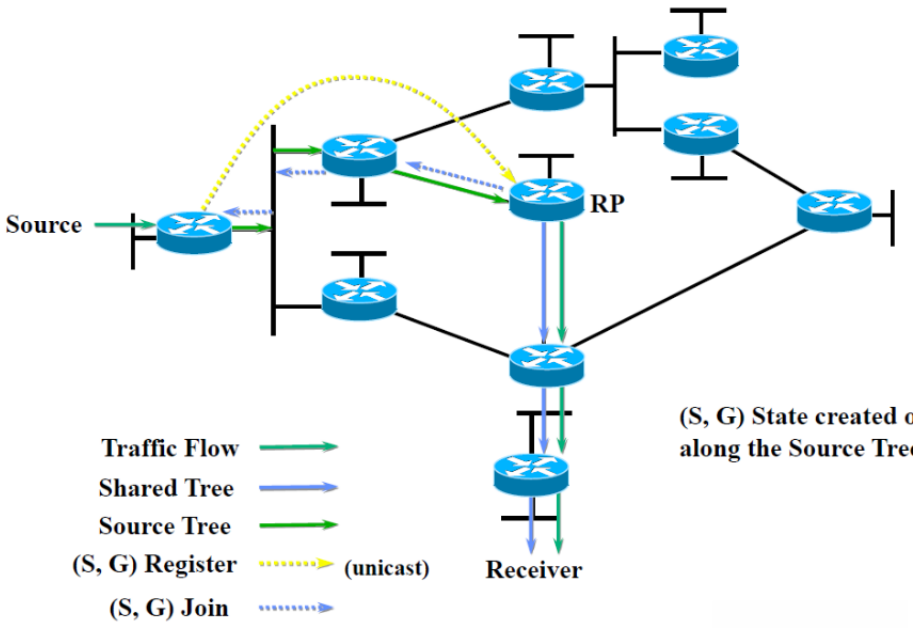
IGMP is a protocol for delivery of multicast messages to all group members that are located in different physical networks. For this, routers need information about **group associations**.
If groups are only **temporary**, routers have to acquire information about associated hosts by themselves. With IGMP messages, a multicast receiver subscribes to a group and thus informs routers about group interest, which periodically asks (**polling**), which groups of multicast are still present.
Routers exchange information to build **multicast routing trees** (routing protocol).

The creation of a **distribution tree** can be done in different ways:

- **Flooding and Pruning** (Distance Vector Multicast Routing Protocol): it first floods multicast packets to all neighbors and the uses **Reverse Path Forwarding** to prevent loops
- **Protocol Independent Multicast**: it does not require a specific unicast protocol and can work in two modes
  - **Dense**: like DVMRP
  - **Sparse**: can support Internet-wide groups and explicitly constructs a tree using **rendezvous points**

PIM Assuming that a low percentage of nodes will subscribe to a multicast group, we have:

- **Receiver** will join a group through an IGMP Join in the local subnet and then PIM Join to a rendezvous point
- **Sender** does a PIM Join with the rendezvous point and sends them a register message. Then it distributes data along the multicast tree



8.7.3 API

The multicast API uses Berkley sockets and has:

- **IP_ADD_MEMBERSHIP** to join a group on a specific interface
- **IP_DROP_MEMBERSHIP** to leave a group (IGMPv2)
- **IP_MULTICAST_IF** to set or get default interfaces for use with multicast sends
- **IP_MULTICAST_LOOP** to disable loopback of outgoing multicast diagrams
- **IP_MULTICAST_TTL** to set the IP TTL of outgoing multicast datagrams

8.7.4 Real applications

Multicast is mainly used in **intra-domain** contexts (e.g. ISPs) due to IPTV, gaming, etc...
There is very limited development in the **inter-domain** context due to:

- Difficult to achieve globally scalable routing
- No economic incentives for ISPs

Note 8.7.4.1. In the **mobile** context, routing gets more complex due to the publish/subscribe approach of multicast.

9 Transport

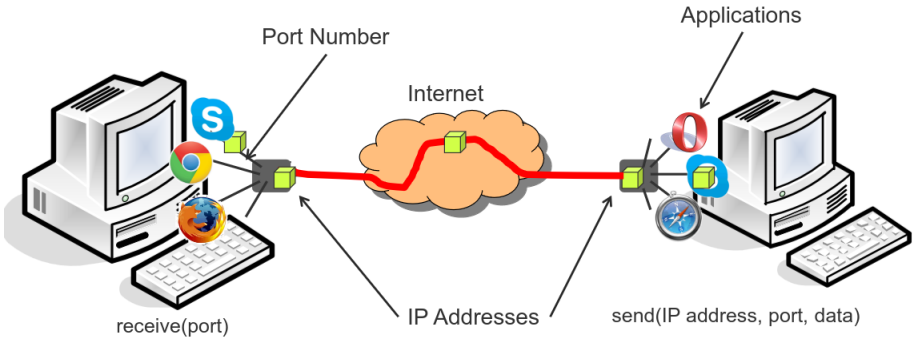
The transport layer operates in an **end-to-end** fashion, encapsulating the network layer. Its main goals are:

- **End-to-End reliability**:
  - *Deliver*: guarantee that the packets sent are indeed delivered
  - *Order*: guarantee that the packets arrive in the order they were sent
- **End-to-End flow control**: techniques to match the source sending rate with the destination receiving rate
- **Congestion control**: techniques to regulate sending rates of sources to **prevent network overload**
- **Multiplexing** and **demultiplexing** to allow sharing of an interface between multiple applications

9.1 Basic elements

9.1.1 De/Multiplexing

Since multiple applications share the same network interface, there is a need to multiplex and demultiplex between the applications and the network layer. This is done with **ports**: a networking abstraction that allows to identify applications. The port combined with the IP address is a **socket**: operating system abstraction that identifies processes to exchange network messages.



Port numbers are coded on 16 bits and are divided between:

- **well-known**: from 0 to 1023, used by known services and assigned by IANA and IETF
- **registered**: from 1024 to 49151, assigned by IANA and used also by OS
- **dynamic/private**: from 49152 to 65535

9.1.2 Primitives

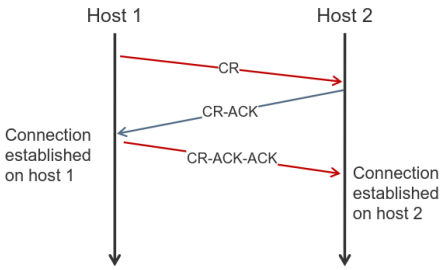
Transport layer provides:

- **Unreliable datagram service** (connectionless)
- **Reliable service** (connection-oriented), divided in three phases:
  1. *Establishment*
  2. *Communication*
  3. *Termination*

The **primitives** for a *connection-oriented* service are:

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONN REQ	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until DATA packet arrives
DISCONNECT	DISC REQ	This side wants to release connection

Establishment End-to-End communication setup is done with a **3-way handshake** so that the communication can be verified in both directions.



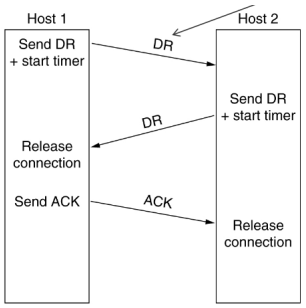
The main problem is **delayed duplicates**: we need to introduce a **sequence number** so that identically numbered transport packets are never out at the same time for different connections between hosts. Each connection will start numbering with a different initial sequence number.

If a delayed duplicate Connection Request arrives at host it gets rejected since it doesn't recognize the sequence number.

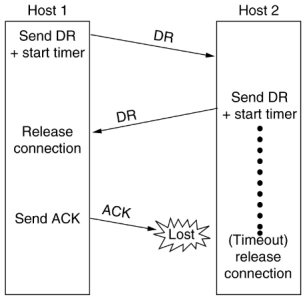
Termination There are two types of releases:

- **Asymmetric**: either one peer terminates the connection
- **Symmetric**: both peers have to terminate the connection explicitly

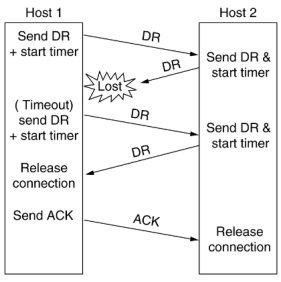
Both approaches could cause **data loss**. Thus a **timer** is introduced: it starts when a Disconnect Request is sent, if it timeouts the request is sent again and the connection is terminated abruptly only after N failed tries.



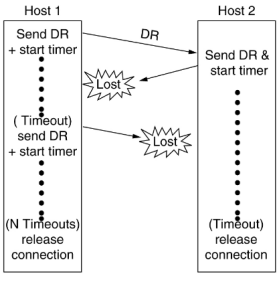
(q) Normal case



(r) Final ACK lost



(s) Response lost



(t) Response and DRs lost

Crash recovery If both hosts and routers are subject to crashes, recovery becomes an issue, e.g. a client sends a large file to a server, each chunk is ACKed but after a crash the server doesn't know the status.
In particular, a client could be in two **states**:

1. **No** outstanding ACK
2. **Outstanding** ACK

It has four possible strategies:

- Always retransmit last packet
- Never retransmit last packet
- Retransmit last packet in state (1)
- Retransmit last packet in state (2)

On the other hand the server has two possible strategies in case of a crash (C):

- First send ACK (A), then write (W) to application
- First write to application, then send ACK

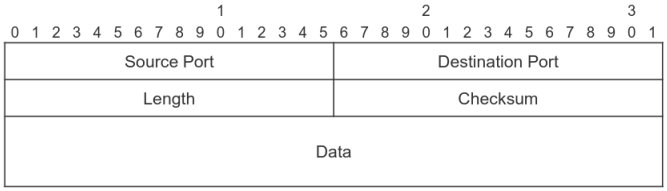
The different combination of the possible client/server strategies lead to different scenarios:

	First ACK, then write			First write, then ACK		
Strategy used by sending host	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

9.2 Protocols

9.2.1 UDP

The **User Datagram Protocol** follows the **KISS** (keep It Simple Stupid) principle. It uses a small 8 byte header and it's **connectionless** and **unreliable**, but **fast**.
There is no acknowledgment between peers: incorrect packets are discarded, there is only a basic checksum and duplicates/loss/packet permutations are possible.



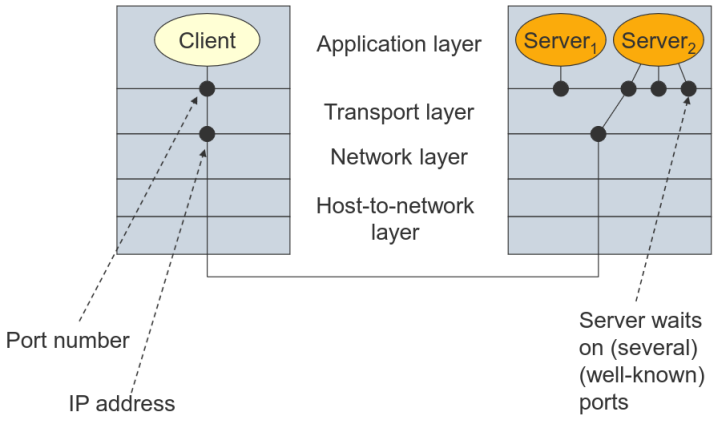
- **Source** and **destination port**: addressing of the applications by port number
- **Length**: total length of the datagram in 32 bits word
- **Checksum**, optional
- **Data**: the payload, filled up if necessary to an even byte number since the length is in 32bit words

The main **reasons** to choose UDP are:

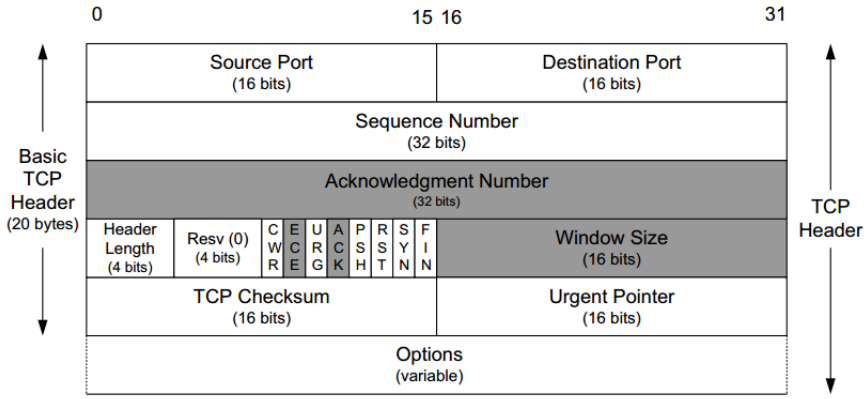
- **Multiplexing**: the tuple (port, network address) uniquely identifies end-points
- **Control**: application has finer-grain control, useful for real-time communication
- **Speed**: no delay due to connection establishment
- **Statelessness**: small packet header
- **Multicasting**

9.2.2 TCP

Transmission Control Protocol is: **point-to-point** (one sender, one receiver), **full duplex**, **connection-oriented**, **reliable**, **flow** and **congestion controlled** and **pipelined**.
It sends and receives segments to establish and terminate a connection, agree on a window size and transmit data.

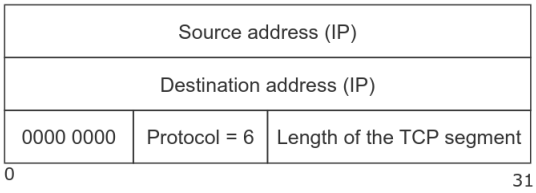


Header TCP has a 20 bytes header that supports plus options and up to 65495 data bytes.



- **Source** and **destination port**: port number of sender and receiver
- **Sequence** and acknowledgment number: they're used for the window mechanism in flow control (**Sliding Window**). The first indicates the sequence number of the first data byte in this segment and they are random initialized. The second indicates the next expected byte and acks all prior ones.
- **Header Length**: length of the header in terms of 32 bit words with a minimum of 5 and a maximum of 15

- **Window Size**: size of the receiver's buffer for the connection. The window indicates how many bytes at the same time can be sent.
- *Flags*
  - **Congestion Window Reduced**: set by a sender to inform a receiver that the congestion window has been reduced
  - **ECE**: indicates an ECN-Echo, it's set by a receiver to inform the sender that it received a packet with ECN set in the IP header, meaning that it experienced congestion. If **SYN** is set, indicates that a TCP peer is ECN capable
  - **URG**: signaling of important data, e.g. CTRL+C
  - **ACK**: set if it's an acknowledgment
  - **PSH**: immediate transmission of data without waiting for further data
  - **RST**: reset a connection e.g. during a crash
  - **SYN**: set to 1 for connection *establishment*
  - **FIN**: set to 1 for connection *termination*
- **Urgent Pointer**: if *URG* is set, indicates at which position in the data field the urgent data ends through a byte offset of the current sequence number
- **Options**, some of them are:
  - Negotiation of a **window scale**, allowing window size field to be shifted up to 14 bits
  - Use of **Selective Repeat** instead of Go-Back-N in case of an error
  - Indication of the **Maximum Segment Size** to determine the size of the data field
- **Checksum**: classical use and also verifies that the packet was delivered to the right host. It's computed using a **pseudo header**



  which is put in front of the main header. The checksum is then computed as the 1-complement of the sum of all 16-bit words of the segment, including the pseudo header.

Connection management A connection is divided in three phases:

1. **Establishment**: the server waits for connection requests with **LISTEN**. The client uses **CONNECT** and specifies IP address, port and MSS and sends them setting the SYN flag. If the server accepts through **ACCEPT**, it returns a segment with SYN and ACK flags. The client then sends back a segment with the ACK flag.
2. **Data Transmission**: it's a **full-duplex** connection where a byte stream is segmented. All hosts must accept at least segments of 556 bytes. Segments up to ACK-1 are confirmed and if there is a timeout before an ACK, the sender repeats, either with *Go-Back-N* o*Selective Repeat*.
3. **Termination**: a segment with the FIN flag is sent, if it gets confirmed that direction is switched off but the opposite one remains open and data can be still be sent

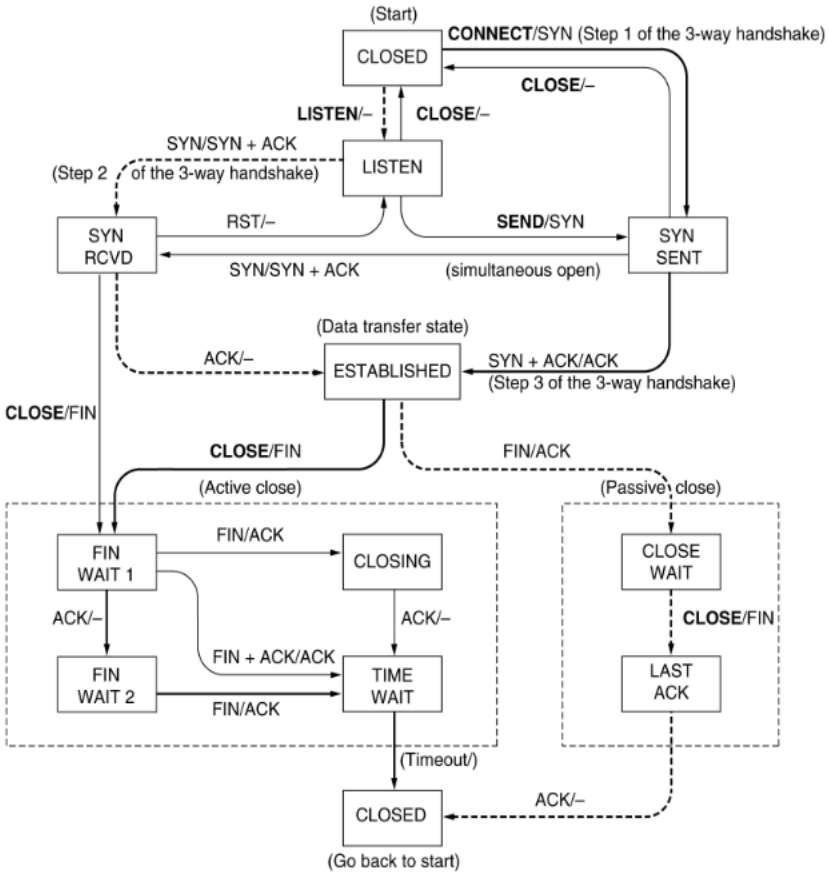


Figure 1: Finite state machine of a TCP connection

Timer management TCP uses several timers to schedule **retransmissions**. They are computed by estimating the probability density of the RTT and setting $T > RTT$. If the timer is too small, there are too many retransmissions but if it's too large there is a big delay to recover from packet loss. **SampleRTT** is the average of several recent measurements of time from segment transmission until ACK receipt. There are two algorithms to handle the retransmission timer:

- **Jacobson**: TCP manages a variable RTT that holds the best current estimation of the round trip time. When sending a segment a timer is started to measure the time needed for the ACK. If it arrives before the timeout , RTT is updated as follows

$$\text{RTT} = \alpha \cdot \text{RTT} + (1 - \alpha) \cdot \text{sampleRTT} \qquad \alpha = 0.875 \tag{13}$$

  The **timeout** was originally $\beta \cdot RTT$, with $\beta = 2$, but that was not reactive enough and was later redefined proportionally to the standard deviation of the arrival time of ACK

$$\text{devRTT} = \gamma \cdot \text{devRTT} + (1 - \gamma) \cdot |\text{RTT} - \text{sampleRTT}| \qquad \gamma = 0.75$$

  and thus

$$\text{Timeout} = \text{RTT} + 4 \cdot \text{devRTT} \tag{14}$$

- **Karn**: do not update RTT on any segments that have been retransmitted. Timeout is doubled on each failure until the segments get through.

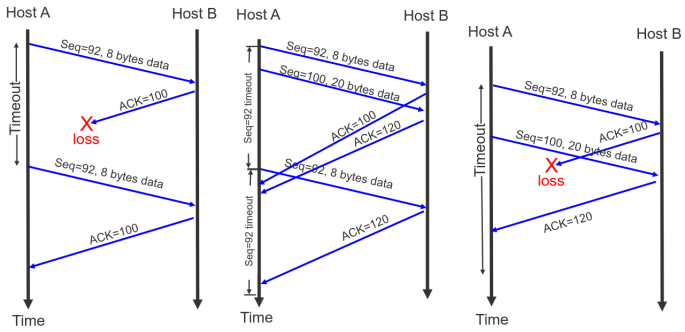
Other timers exist:

- **Persistence**: prevents a deadlock with a loss of the buffer release message of a receiver. If the timer expires, the sender transfers a test segment. The response contains the current buffer size of the receiver. If it's still 0 the timer starts again

- **Keep-Alive**: if a connection is alive for too long, at the end of the timer the other side is checked whether it's still alive. If not, the connection is terminated.
- **Time Wait**: during the termination of a connection, he timer runs for the double packet lifetime to be sure that no more late packets arrive

Reliable Transfer Management TCP creates reliable data transfer service on top of IP's unreliable service. It uses **retransmissions**, which are triggered either by:

- **Timeout**: for every segment, start a timer and transmit it. If timeout occurs, retransmit the segment and restart the timer. There are three scenarios:
  - **Lost ACK**: a segment arrives, ACK gets sent but gets lost, the segment is then retransmitted
  - **Premature timeout**: segments arrive and ACKs are sent back correctly but too late, retransmission begins and with the first response all previous correctly received messages are ACKed
  - **Cumulative ACK**: multiple segments arrive but the first ACK message gets lost, send an ACK message confirming all previous correctly received segments



(a) Lost ACK (b) Premature timeout (c) Cumulative ACK

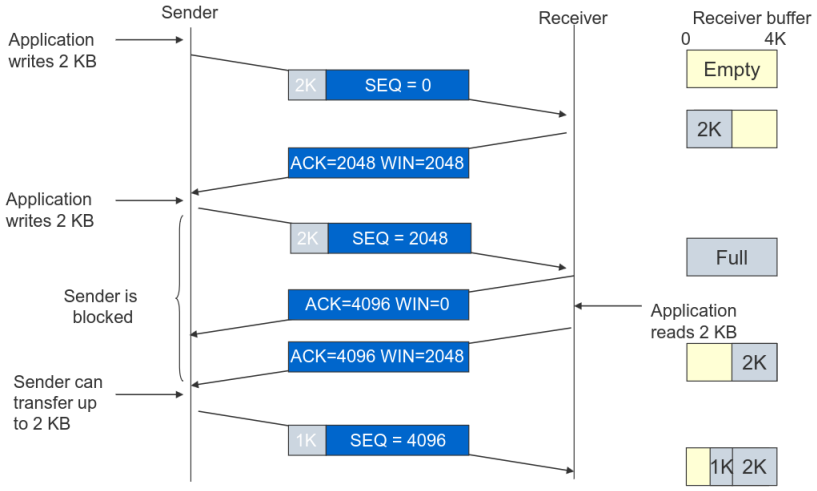
```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum
loop (forever) {
   switch(event)
   event: data received from application above
   create TCP segment with sequence number NextSeqNum
   if (timer currently not running)
   start timer
   pass segment to IP
   NextSeqNum = NextSeqNum + length(data)
   event: timer timeout
   retransmit not-yet-acknowledged segment with smallest sequence number
   start timer
   event: ACK received, with ACK field value of y
   if (y > SendBase) {
      SendBase = y
      if (there are currently not-yet-acknowledged segments)
      start timer
   }
}
```

- **Duplicate ACKs**: when the same ACK is received multiple times, it's very likely that an intermediate segment got lost and needs to be resent

ACK generation is done by the following rules:

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected SEQNUM. All data up to that already ACKed	**Delayed ACK**, wait at least 500ms for the next segment. If none arrives, send ACK.
Arrival of in-order segment with expected SEQNUM. One other has ACK pending	Immediately send single cumulative ACK, ACKing all in-order segments.
Arrival of out-of-order segment with higher than expected SEQNUM. Gap detected.	Immediately send **duplicate ACK** indicating SEQNUM of next expected byte
Arrival of segment that partially or completely fills the gap	Immediately send ACK, provided that segment starts at lower end of gap

Flow control To provide reliable data transfer, a **sliding window** mechanism is used: sender sends bytes according to the window size and the window gets shifted by n bytes as soon as an ACK for those arrives. The window size can be changed dynamically during the transmission.



Note 9.2.2.1. Urgent data is sent immediately disregarding the sliding window mechanism.

Congestion control When too many sources send too much data too fast for the network to handle it, **congestion** happens. It manifests trough **long delays** and **lost packets**.
Endpoints do not know about congestion, they just experience it. If a timeout happens, the host retransmits the packets leading to more traffic, more delay and so on.

Congestion control aims at solving three problems:

- **Bandwidth estimation**: how to adjust the bandwidth of a single flow to the bottleneck bandwidth
- **Bandwidth adaptation**: how to adjust the bandwidth of a single flow to variation of the bottleneck bandwidth
- **Fairness**: how to share bandwidth fairly among flows, without overloading the network

Detect There are two main approaches to detect congestion:

- **Network-Assisted**, through **Explicit Congestion Notification**: when supported by both endpoints (ECT flag), IP packets that traverse congested ares can be marked by routers. When the packet gets to the receiver, it echoes it back to signal that the sending rate should be reduced. The sender confirms that by setting the CWR flag.
- **End-to-End**: detecting losses can be done in two ways
  - **Duplicated ACKs**: mild congestion signal, packets are still making it through
  - **Timeout**: severe congestion signal, multiple consequent losses

  This approach cannot distinguish between congestion or loss on link layer, which can be problematic in wireless communication.

React Sender throttles transmission with the minimum between the congestion and the receiver window

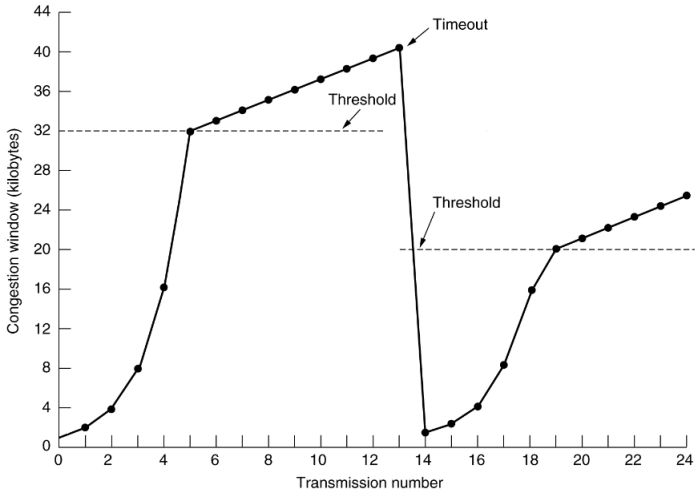
$$\text{LastByteSent} - \text{LastByteAcked} \leq \min(\text{CongWin}, \text{ReceiverWin})$$

Usually CongWin < ReceiverWin. The first one is dynamic and it's based on detection of loss events (timeout or 3 duplicate ACKs).

$$\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \frac{\text{bytes}}{\text{sec}}$$

The regulation of CongWin can be done with different mechanisms:

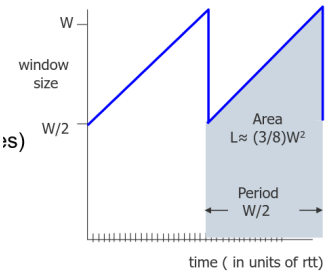
- **Additive Increase Multiplicative Decrease**: increase CongWin by 1 **Maximum Segment Size** every RTT until loss detected, in that case cut it in half
- **Slow Start**:
  1. Initialize CongWin to 1 MSS and ReceiverWin to the value specified by the other end
  2. A segment with MSS bytes of data is sent
  3. Apply **Slow Start Algorithm**: when an ACK arrives before the time out, double CongWin, otherwise reset it to 1 MSS. The growth stops when it reaches the flow control window size
- **Congestion avoidance** with **Inferring Loss**: introduce a threshold **ssthresh**, initialized at 64 kbyte. Above ssthresh, CongWin increases linearly by 1 MSS. In case of timeout, ssthresh is set to half of the maximum window size reached before it and CongWin is set to 1 MSS.
  After 3 duplicate ACKs, CongWin is halved and the windows then grows linearly



- **Fast Retransmit and Recovery**: when a single packet is lost, Slow Start is not well suited. With Fast **Retransmit** the receiver sends duplicate ACKs immediately when out-of-order segments arrive. If the sender receives triple duplicate ACKs, it retransmits the missing segment, avoiding a new slow start phase.
  With Fast **Recovery**:
  1. When the third ACK is received ssthresh $= \max(\frac{\text{ssthresh}}{2}, 2 \cdot \text{MSS})$
  2. Retransmit the missing segment and set CongWin $=$ ssthresh $+ 3 \cdot$ MSS
  3. For each subsequent duplicate ACK, increase CongWin by 1 MSS
  4. If timeout happens, go back to slow start

Throughput Given a *window size*, the *MSS*, the *RTT* and the *packet loss rate* P. Assuming no Slow Start, infinitely long TCP flow and periodic losses. The throughput of TCP is:

$$B = \frac{\text{Average number of bytes sent per cycle}}{\text{Average duration of a cycle}}$$



Since by definition each period delivers $\frac{1}{p}$ segments of MSS bytes and the total number of segments ACKed is the are under the saw tooth

$$\left(\frac{W}{2}\right)^2 + \frac{1}{2}\left(\frac{W}{2}\right)^2 = \left(\frac{3}{8}\right)W^2 \Longrightarrow W = \sqrt{\frac{8}{3W}}$$

Hence

$$B = \frac{\text{MSS} \cdot \frac{3W^2}{8}}{\text{RTT} \cdot \frac{W}{2}} = \frac{\frac{\text{MSS}}{p}}{\text{RTT} \cdot \sqrt{\frac{2}{3p}}} = \frac{\text{MSS} \cdot C}{\text{RTT} \cdot \sqrt{p}} \qquad C = \sqrt{\frac{3}{2}} \tag{15}$$

Fairness If k flows share the same bottleneck link of bandwidth R, each one of them should have an average rate of $\frac{R}{k}$. There are two approaches:

- **Max-Min**: aims to give each session equal access to each link's bandwidth
- **Proportional**: if packet loss is *synchronized* in different flows, there are times when the capacity is under-utilized, which is fair to no one. **Random Early Detection** in routers, tracks how the buffers fill up. If a buffer threatens to fill up soon, the router begins randomly dropping packets

Fairness can be difficult to achieve also because of **cheating**, misbehaving TCP flows, and **cohabitation** with non TCP traffic.

Wireless Transport layer protocols should be independent of lower layers, but TCP is optimized for wired networks since in wireless ones packet losses occur to other causes (hand off, bigger delays). For these reasons, TCP **performance** is very **poor** in wireless network. A few approaches to this problem are:

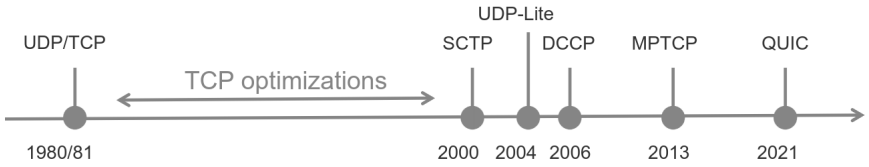
- **Split-connection** TCP: the end-to-end connection is broken in two parts
- **Modifications** of TCP, which may lead to compatibility issues
- **New protocols**

Security One of the attacks that can be performed on TCP is **SYN Flood**, which is a type of **Denial of Service**.

During connection establishment the client does not finish the three-way-handshake procedure and leaves a half open connection with the server still reserving resources. If this is repeated many times the server can be clogged.

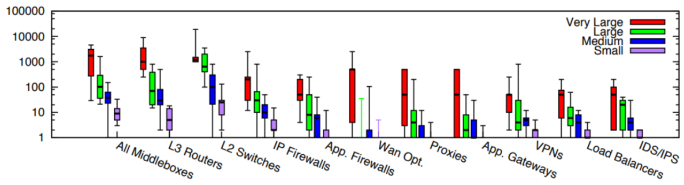
A possible solution is using **SYN Cookies**: the server does not create a half-open connection. It computes an initial sequence number y (the **cookie**) based on a hash function. When the client returns with ACK, the server checks again with the hash function and only then creates the connection.

9.3 Newer protocols



Choosing a transport protocol depends on the type of application: many of the new ones are driven by increased **readability**, reduced **latency** and **privacy**.
As of today, 85% of IPv4 and 90% of IPv6 traffic goes through TCP: there is an **ossification** of the transport layer, increasing the challenge to develop new protocols.

While originally protocols were focused on the end-to-end communication because we had dumb network and smart end devices, today it's not the same anymore: now we have as many **middleboxes** of different types as routers.



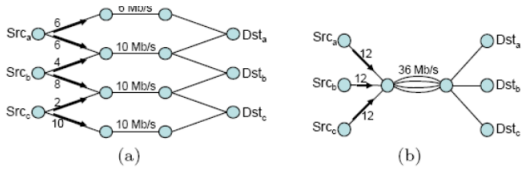
Middleboxes are the new core problem:

- They **assume** the format of the protocol header, making changes almost impossible
- They **modify some fields** of the transport or network layer, meaning that they cannot arrive at destination
- **Restrict** the usage of protocol extension that they don't support

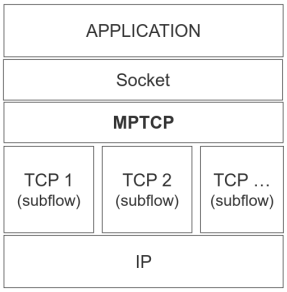
9.3.1 MPTCP

MultiPath TCP was developed to increase reliability, efficiency and flexibility by exploiting multiple paths based on different IP addresses of the host and quickly moving traffic from congested or failed links.

It's based on the idea of **Resource Pooling**: making a collection of resources behave like a single pooled one.



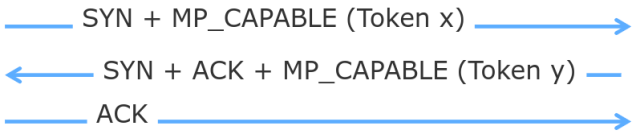
An MPTCP connection is composed of one or more regular TCP **subflows** that are combined. Each host maintains a state that glues them together. Each subflow is sent over a single path and appears like a regular TCP connection along it.
At least one of the following elements must differ between two subflows:



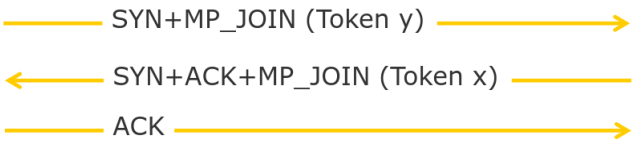
- Local IP address
- Remote IP address
- Local port
- Remote port

Note 9.3.1.1. Number of subflows can change during the lifetime of an MPTCP connection.

Connection establishment MPTCP connection establishment uses TCP mechanisms:



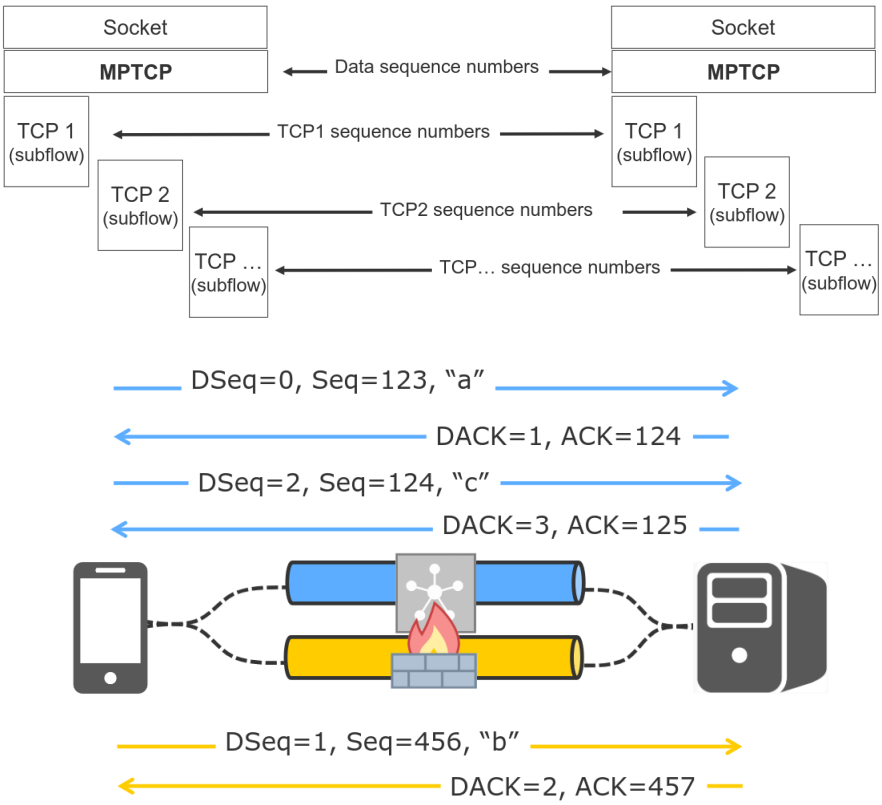
Subflow start



Data transfer Since in today's internet there are many middleboxes, to transfer data MPTCP uses two level of sequence numbers:

- **Data**: numbers the bytes in the *byte stream*
- **TCP**: numbers the bytes inside the *subflow*

Hence, regular TCP ACK confirms segments per subflows, while **DNS_ACK** implements cumulative ACK per data sequence and prevents deadlock in case of proactive ACKs from middleboxes.



Observation 9.3.1 (Losses)**.** When losses happen in a subflow, the subflow is completely lost.

9.3.2 QUIC

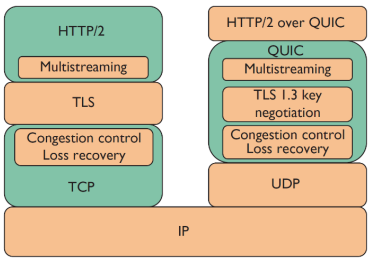
The main reasons for QUIC are:

- **Multiplexing**: for example, HTTP runs over TCP. HTTP/2 implemented full pipelining which is interfered by TCP: one lost packet in a stream requires all streams on the application layer to wait for successful retransmission. A simple solution is to run the protocol over UDP and improve connection management.
- **Surveillance**: implements minimal meta-data about the connection, reducing the ossification problem but making life more difficult for the providers

So overall the objectives are:

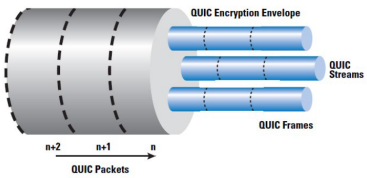
- Easy deployment
- Low latency connection setup
- Multi-streaming
- Better security
- More efficient loss recovery and congestion control
- Multipath for resilience and load balancing

QUIC can be viewed as a **bidirectional** UDP packet sequence with concealed payload, which is encrypted before getting to the UDP protocol. This means that now more mechanisms are in the user space and it's simpler and faster to update.



Connection A QUIC connection is identified by a pair of **ID**s, one for each endpoint: a persistent identity for a connection independent of changes at lower layers (merges the concept of TCP+TLS 1.3).

Packet QUIC packets are individually 62bit numbered and cannot be retransmitted using the same one (the next one is used). Multiple packets may form the payload of a UDP datagram with an assumed minimum of 1200byte (padded if necessary) and without fragmentation.
Encryption is based on individual packet, thus avoiding the wait for partially delivered ones.
The receiver **ACK**s the highest packet number received so far and adds a list of received contiguous packet numbers (maximum 256).



Streams Each connection contains one ore more streams, which is similar to a TCP one. They may have a priority defined by the application and are identified by an ID, with special bits to identify the initiator and if bi/unidirectional.
Stream creation is very **lightweight** compared to TCP since the QUIC connection is already open: open a stream, send data and close it in a single packed, reducing latency.
Streams are then segmented into **data frames** of 20 different types, identified with an offset (similar to TCP sequence number), and used for in-order delivery, loss detection and retransmission.

Recovery Loss detection is based on packets instead of frames. A packet is considered lost if while it's still unACKed a later sent one is ACKed **and** a lost threshold is met:

- Given an ACKed packet x, all unACKed packets with number $< x - t$ are considered lost, with t being the **reordering threshold**
- Given the time of the most recent ACK t, then all unACKed packets sent before time $t - w$ are considered lost, with w being the **waiting time**, derived from a weighted estimate RTT

Frames of lost packets are then placed in new ones for retransmission.

Flow/Congestion control Similar to the mechanism of TCP Window: there is a maximum amount of data a sender can send on an individual stream and on all others.

Issues The main issues with QUIC are:

- **Load-balancing**: load balancers often categorize packets based on protocol and socket pair, which are not stable for the entire QUIC session. Furthermore, high-capacity infrastructures assumes TCP is used for data-streams and not UDP, not allowing optimization.
- **Encryption cost**
- **Firewalls**: not all of them can handle QUIC and some block it
- **Performance**: it depends on the settings and will change over time due to newer implementations but at the moment it's not faster than TCP on most scenarios

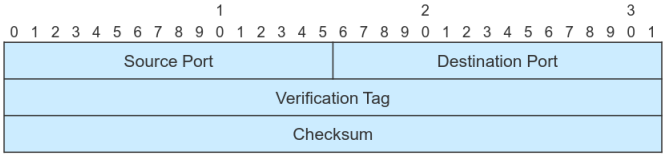
9.3.3 SCTP

Stream Control Transmission Protocol is a reliable, connection-oriented and message-oriented transport protocol. It uses **heartbeats** and **4-way-handshake** to support:

- **Multi-streaming**: allows the partitioning of data into multiple streams within a single **association**. Streams are **independent** of sequence delivery, meaning that the loss in one stream will only affect its delivery.
  Two sequence numbers are used:
  - **Transmission** SN: governs the transmission of messages and detects message losses
  - **Stream Identification/Stream** SN: determines the sequence of delivery of received data

  E.g. partial ordering of web page objects: each one is assigned to a stream without caring about delivery order. Even if an object doesn't get through, the others do, improving the user experience.
- **Multi-homing**: ability of a single endpoint to support multiple IP addresses. Used for redundancy: a fail on one network will not cause a failure of the association.
  One address is chosen as the **primary** and it's the destination for all data chunks for normal transmission, while retransmitted data use alternate addresses. A continued failure on the primary address results in transmitting all data to another one until heartbeats can reestablish the primary one.

SCTP packets are divided between a **common header**

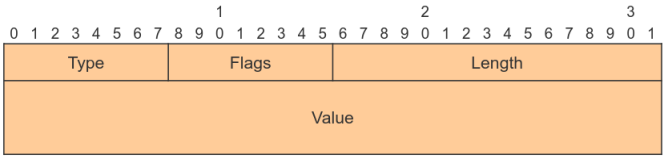


- **Ports**: 16 bits like TCP and UDP
- **Verification Tag**: validation of the sender, per association. It's set to the value of the Initiate Tag received during association initialization, which is a random number, or set to 0 in packets with the **INIT** chunk.
- **Checksum**: CRC32c algorithm using the polynomial

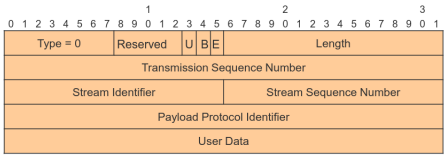
  $$x^{32}+x^{28}+x^{27}+x^{26}+x^{25}+x^{23}+x^{22}+x^{20}+x^{19}+x^{18}+x^{14}+x^{13}+x^{11}+x^{10}+x^{9}+x^{8}+x^{6}+1$$

  and calculated over the whole SCTP packet, including the common header (with checksum value at 0) and all the chunks.

and n **chunks** with the format



- **Type**, e.g. *Payload Data* 1, with its related flags



  - **U**: unordered data chunk, ignore SSN
  - **B** and **E**

B	E	Meaning
1	0	First fragment of a user message
0	0	Middle fragment of an user message
0	1	Last fragment of an user message
1	1	Unfragmented user message

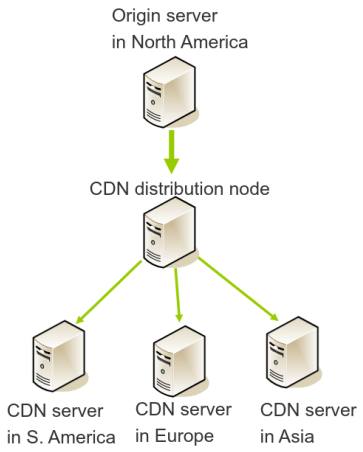
  - **Transmission Sequence number**
  - **Stream Identifier**: identification of data stream to which the following data belongs
  - **Stream Sequence Number**: sequence number of the following user data within the stream, when a user message is fragmented all fragments must carry the same SSN
  - **Payload Protocol Identifier**: identifies application layer protocol used by end or intermediate systems

- **Length**: overall length of chunk in bytes
- **Data**: actual content

10 Network types

10.1 CDN

Content Delivery Networks help to scale many distributed web requests (e.g. video streaming, drive). The principle is similar to DNS caches: the shared content is replicated everywhere and you access a close-by one instead of the central server.



10.1.1 How

Content replication can be done with two different strategies (often used at the same time):

- **PULL**: the first request will have to fetch the data from the central server and replicate it locally, when the cache is full erase recently least used content
- **PUSH**: content expected to be popular can be pre-provisioned in caches

Then, **DNS CNAME** feature is used to redirect hosts to the nearest cache server since the name resolution is done based on geographical position.

Requested from	Served from					
	Africa	Asia	Europe	N. America	Oceania	S. America
Africa	0.3	18.6	32.0	46.7	0.3	0.8
Asia	0.3	26.0	20.7	49.8	0.3	0.8
Europe	0.3	18.6	32.2	46.6	0.2	0.8
N. America	0.3	18.6	20.7	58.2	0.2	0.8
Oceania	0.3	20.8	20.5	49.2	5.9	0.8
S. America	0.2	18.7	20.6	49.3	0.2	10.1

10.1.2 Where

The service of a CDN company is to deliver the content fast. There are basically two design principles:

- **On-net**: deploy CDN servers directly into ISP points of presence, hence getting very close to the end-user but becoming complex to maintain the shuffling of content since they are scattered among different AS
- **Off-net**: connect few CDN data centers to ISPs, hence less maintenance but dependence from ISP POPs and a bit higher delays

10.1.3 Hypergiants

Hypergiants are large content providers, cloud providers and CDNs that are typically responsible for distributing the majority of traffic to the end user (e.g. Google, Netflix, Meta).
They reshape the internet by improving capacity, latency and congestion.

10.1.4 Advantages

The advantages of CDNs are:

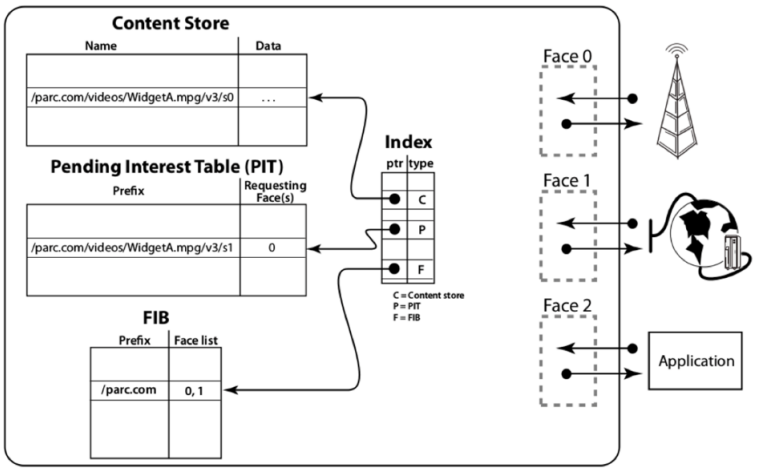
- **Robustness**: so many caches that you can't do a DoS attack
- **Scalability**: load is spread to thousands of servers
- **Lower latency**: content is accessed from a local cache

The main **disadvantage** is that it makes the internet and the services much more complex.

10.2 ICN

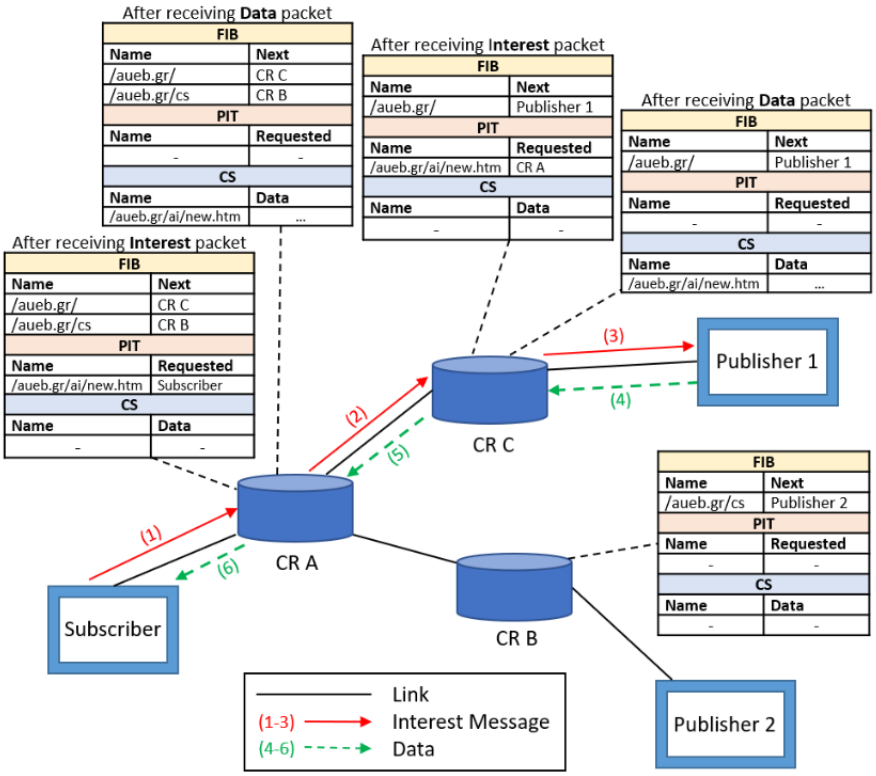
Information Centric Networking allows for improved **security** and more **efficient content distribution**. It's based on the principle to request access for the content and not the node, in a publish/subscribe paradigm: we address the content by a name, which is augmented with authentication. All of this with a lot of caching.

The content provider places the resource in a shared server. Then the customer specifies an **interest** to that object, which is store in the routers in a **Pending Interest Table**. Then the interest gets forwarded using the data in the **Forwarding Information Base** until it finds the requested resource, which gets back and gets cached along the way.



10.2.1 NDN

Named Data Networking is one way to implement an ICN.



SAIL is another implementation of an ICN that uses both regular routing techniques and Name-Based.

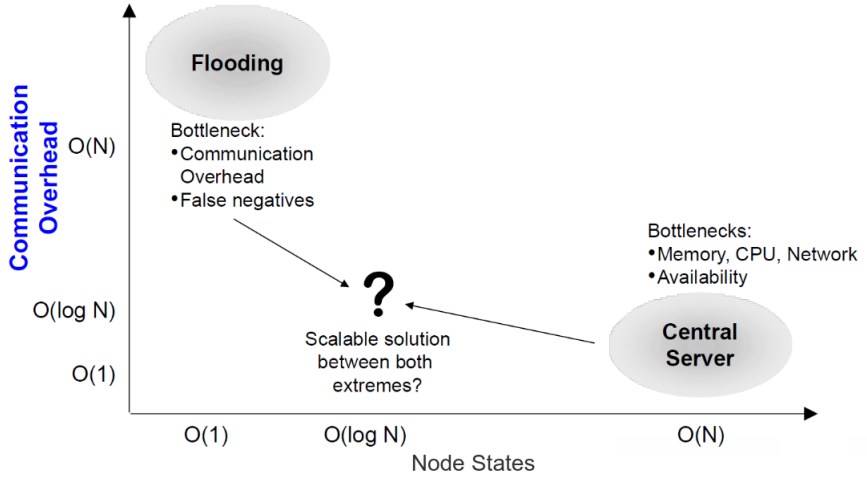
10.3 P2P

In a **Peer-To-Peer** architecture each member of the network is both client and server and it's called **peer**. They operate at the application layer. The main advantages are:

- No central entity, no single point of failure
- Reduces/eliminates the demand for server clusters
- Content replication
- Flexibility, self-organization
- Some levels of anonymity
- Efficiency

10.3.1 Unstructured

An unstructured P2P network has data placed **randomly**. Then decentralized **flooding** is used to spread it around the network, up to having it replicated on all peers.
The more data we have flooded, the less there is a need for central servers, but at the same time that increases **communication overhead**.



10.3.2 Structured

In a structured P2P data is placed **systematically** and there is a systematic overlay routing. The main challenge is where to store and how to find a certain item without any centralized control or coordination.

There are two **lookup** approaches:

- **Centralized**: using a server
- **Distributed**: using **Distributed Hash Tables**

DHT In a DHT nodes are structured according to some flat address space and are responsible for data in a certain part of it. Intermediate nodes maintain routing information to target nodes.

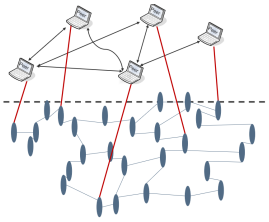
Each node manages a small number of references to other nodes and the data itself and needs only a small number ($O(\log N)$) of routing steps to reach the destination. Identifiers are distributed on the nodes nearly equally, hence balancing the load among them.

There are multiple algorithms to achieve this type of data structuring (Chord, Tapestry, Content Addressable Network). An example is **Pastry**: nodes are organized in a virtual ID ring from 0 to $2^{128} - 1$ and to each node and object a 128bit hash key is assigned.

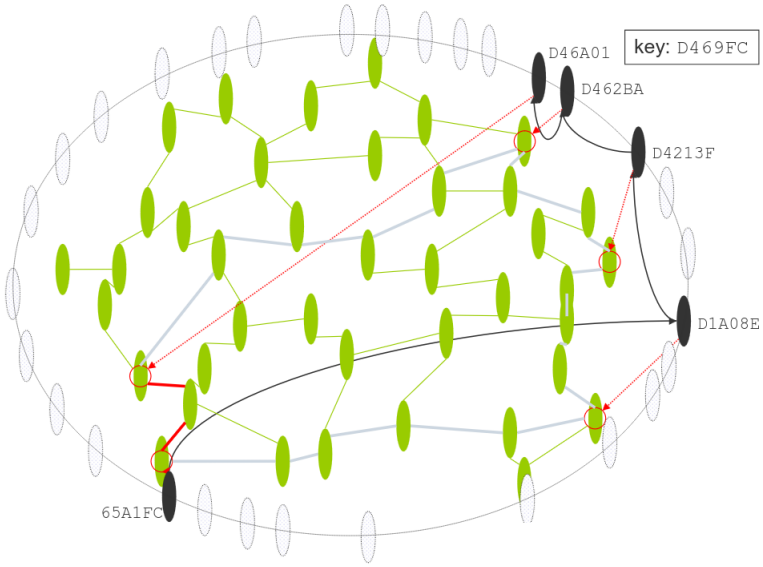
The routing is based on *longest prefix matching* considering the topological distance: each node maintains a forwarding table and a leaf set. Leaf sets provides shortcuts: contains the L closest nodes, $\frac{L}{2}$ smaller nodes by ID and $\frac{L}{2}$ larger nodes by ID, with $|L|$ usually being 2^b or $2 \cdot 2^b$.

1. Node n searches the node responsible for key k
2. n searches its forwarding table for a node with a better prefix matching (at least one or more digits matching k)
3. If such a node does not exist the leaf set is searched for a node which is lexicographically closer to k
4. Repeat in each node until found ($O(\log_2^b N)$)

DHTs can route very efficiently on the **overlay network**, however there is no relationship between that and the physical topology.



In fact, neighbors on one may not be neighbors in the other, hence causing an **overlay stretch**: the ratio between the accumulated physical routes during the overlay routing and the real path.



There are several method to reduce this phenomena, such as **Proximity Node Selection**: each node selects periodically a random entry from its routing table at row i. This node then replies with his i-th entry. After the response, the second node compares the distance between the own entries and the response and if physically closer it replaces it.

Another approach is the **landmarking**: a fixed set of nodes are set as landmarks; periodically each nodes compute their distance from them and nodes with the same one assign each other numerical close overlay IDs.

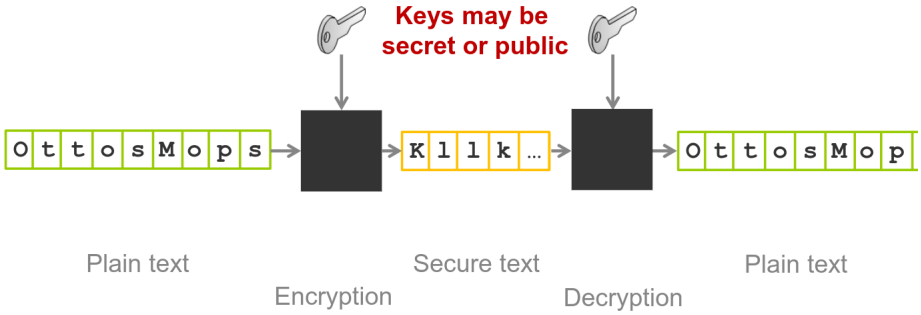
11 Infrastructure security

There are different security **objectives**:

- **Confidentiality**: to ensue nondisclosure of data
- **Integrity**: to ensure completeness and accuracy of data
- **Authenticity**: to allow for the verification of identity

11.1 Cryptography

Many security goals may be achieved through **cryptography**.



There are two types of cryptography:

- **Symmetric**: the same key, which remains always private, is used for encryption and decryption
- **Asymmetric**: different keys are used for encryption and decryption, **private** and **public**

	Public key	**Private key**
Confidentiality	Encryption	Decryption
Authenticity	Decryption	Encryption

11.1.1 Trust

The main challenge in the public/private key context is to obtain **trust** in the public key. A **Public Key Infrastructure** manages that a *public key* belongs to entities such as persons or organizations. If you trust the PKI, then you trust the key itself.
A **certificate authority** signs the public key of a third party by its own private key, attesting that the first one is correct. If you trust the CA, you also trust the third party key.

11.2 DNSSEC

DNS main problems are:

- DNS resolver can **lie** about DNS answer
- DNS client cannot verify the correctness of the answer

The objectives of DNSSEC are to provide **integrity** to prevent *spoofing* and *poisoning*. That is done through;

- **Authenticating messages** of name servers: check that no one on the way changed the DNS content

- **Authenticating resources records**: check that data comes originally from authoritative DNS servers
- **Proof of non-existence**: prevent DoS against names

DNSSEC is based on private key cryptography and introduces new resource records:

- **RSIG**: contains cryptographic signature
- **DNSKEY**: contains public sign key
- **DS**: contains hash of the *DNSKEY*
- **NSEC/NSEC/3**: for explicit denial-of-existence of DNS records
- **CDNSKEY/CDS**: for a child zone requesting updates to DNS records in the parent zones

The main objectives of DNSSEC are:

- **Secure a zone**: DNSSEC does not sign resource records individually but signs **Resource Record Sets**. It distinguishes between two types of pair:
  - **Zone**-Signing-Key: it starts with a **root zone key**, the highest possible in the DNS tree, and then follows cryptographic pointers to lower zones. Each pointer is validated with the previous validated **zone key**. This is called a **chain of trust** and works so that a resolver has only to carry the root key to validate the DNSSEC data on the Internet.
  - **Key**-Signing-Key: it operates the same as ZSK: signs the ZSK and stores it in DNSKEY and creates RRSIG for it
- **Secure zone delegation**

Observation 11.2.1. Deployment usually work two ways:

- **Full DNSSEC**: it's DNSSEC compliant and performs validation on its own
- **Stub resolver**: usually deployed on end hosts to perform DNS queries
  1. Client trusts completely the server
  2. Client decides autonomously how to handle unauthenticated data, either by enforcing validation or less

11.3 RPKI

Exploiting BGP Updates could lead to traffic **interception** to break privacy or service availability, since it's based on trust. Any BGP speaker can claim to own an IP prefix or modify an AS path. A receiver cannot verify the correctness of these data.

There are three threats models for the BGP protocols:

- **Route leaks**
- **AS Path manipulation**: change the AS path compared to the original traversal. The solution is to map IP prefixes to origin AS necessarily, including **cryptographic proof** so that prefix owner can authenticate it.
- **Prefix Origin Hijacking**: originate an IP prefix that you don't own. The solution is to sign paths so that BGP path information are cryptographically secured

Resource Public Key Infrastructure is a system that allows to attest the usage of IP addresses and internet resources, and thus includes cryptographically provable certificates that reflect IP/AS allocation in the Internet. Currently each RIR creates a self-signed root certificate.

11.4 ROA

A **Routing Origination Authority** legitimates an AS to originate IP addresses. It contains:

- Set of IP prefixes with minimal and maximal (optional) length
- An AS number allowed to announce the prefixes
- End-Entity-Certificate

They will be signed with the certificate of a RPKI