Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Telecommunication networks

Raffaele Di Nardo Di Maio

Anno Accademico 2018-2019

Indice

1 The story of internet **1**

2 Organization of internet **3**
2.1 Internet Service Provider(ISP) . . . 3
2.2 Network Access Point (NAP) . . . 4
2.3 Milan Internet eXchange (MIX) . . . 4
2.4 Internet Society (ISOC) . . . 4

3 Architecture of a network **7**
3.1 Classification based on the extension of the network . . . 7
3.2 Topology . . . 7
3.3 Classification based on the function of the network . . . 9
3.4 Open System Interconnection (OSI) . . . 9
3.4.1 Internet Protocol Suite (IPS) . . . 10
3.4.2 Encapsulation and decapsulation . . . 11
3.5 Structre of a telecommunication network . . . 11
3.6 Main application architectures . . . 13
3.7 Types of switching . . . 14
3.7.1 Circuit switching . . . 14
3.7.2 Packet switching . . . 14
3.7.3 Virtual circuit switching . . . 15

4 Performance metrics and QoS issues **17**
4.1 Traffic . . . 17
4.2 Delay . . . 18
4.3 Pipe capacity . . . 19
4.4 Reliability . . . 19

5 Performance analysis of switching methods **21**
5.1 Circuit Switching . . . 21
5.1.1 Message delivery time . . . 21
5.2 Packet Switching . . . 22
5.2.1 Pipeline . . . 22
5.2.2 Bottleneck . . . 22
5.2.3 Message delivery time . . . 23
5.3 Round Trip Time & Pipe Capacity . . . 23
5.3.1 BDP with specific types of delay . . . 24

6 Data Link Layer (DLL) **25**
6.1 Frame synchronization . . . 25
6.1.1 Character-oriented services . . . 25
6.1.2 Bit-oriented services . . . 26
6.2 Medium Access Control (DLL sublayer) . . . 28
6.3 Batch access problem . . . 30
6.3.1 Comparing MAC problems and Batch problems . . . 30

6.3.2 BRA resolution algorithms . . . 30
6.4 Error control . . . 31
6.4.1 Automatic Repeat reQuest (ARQ) . . . 32
6.4.1.1 Main protocols that uses ARQ . . . 33

7 Addressing **35**
7.1 Regional Internet Registr (RIR) . . . 35
7.2 Addressing at different layers . . . 35
7.3 Socket . . . 37
7.3.1 TCP socket . . . 37
7.3.2 UDP socket . . . 37
7.4 IP and MAC addresses . . . 37
7.5 Address Resolution Protocol (ARP) . . . 38
7.5.1 ARP packet format . . . 39
7.5.2 ARP uses . . . 41

8 Internet protocol **43**
8.1 IP: basic requirements and services . . . 43
8.2 IP address . . . 43
8.2.1 Multi-homed nodes . . . 44
8.3 Classful addressing . . . 45
8.3.1 Special addresses . . . 46
8.4 Classless addressing . . . 47
8.4.1 Netmask . . . 47
8.4.2 Block allocation . . . 48
8.4.3 Subnetting . . . 49
8.4.4 Subnet mask . . . 49
8.5 Supernetting . . . 51
8.6 Direct forwarding . . . 51
8.7 Indirect forwarding . . . 53
8.8 Router forwarding . . . 53
8.8.1 Routing table with classfull addresses . . . 53
8.8.2 Routing table with netmask . . . 54
8.9 Stub and Transit networks . . . 55
8.10 Dynamic addressing . . . 56

9 Private networks **57**
9.1 Private addressing . . . 57
9.2 Proxy . . . 58
9.3 Network Address Translation (NAT) . . . 59
9.3.1 Dynamic association . . . 59
9.3.2 Traditional NAT (Outbound NAT) . . . 60
9.3.2.1 Basic NAT . . . 60
9.3.2.2 NAPT (Network Address and Port Translator) . . . 60
9.3.3 Bi-Directional NAT . . . 61
9.3.4 Final remarks . . . 61

10 BOOTP & DHCP **63**
10.1 Bootstrap protocol (BOOTP) . . . 63
10.1.1 BOOTP packet format . . . 64
10.1.2 Limits of BOOTP . . . 65
10.2 Dynamic Host Configuration Protocol (DHCP) . . . 65
10.2.1 DHCP packet format . . . 65
10.2.2 Phases of DHCP . . . 66

11 IP datagram **69**
11.1 Fragmentation . . . 72
11.2 Different types of options . . . 72
11.2.1 No-Operation options (NO-OP) . . . 73
11.2.2 End of Option (END-OP) . . . 73
11.2.3 Record route option . . . 73
11.2.4 Strict source route . . . 73
11.2.5 Loose source route . . . 75
11.2.6 Time stamp . . . 75

12 ICMP **77**
12.1 Main rules of ICMP error messages . . . 78
12.2 Types of ICMP messages . . . 78
12.2.1 Echo . . . 78
12.2.2 Destination unreachable . . . 79
12.2.3 Time exceeded . . . 81
12.2.4 Parameter problem . . . 81
12.2.5 Redirect . . . 82
12.2.6 Timestamp request e reply . . . 82
12.2.7 Address mask request and reply . . . 83

13 IPv6 **85**
13.1 IPv6 format . . . 85
13.2 Global Unicast Address . . . 88
13.2.1 IEEE 64-bit Extended Unique Identifier (EUI-64) . . . 88
13.3 Link local addresses . . . 89
13.3.1 Special addresses . . . 89
13.3.2 Unique local IPV6 unicast (RFC 4193) . . . 89
13.3.3 Embedded IPv4 . . . 90
13.4 Multicast addresses . . . 90
13.4.1 Multicast scopes . . . 90
13.4.2 Solicited-Node Multicast Address . . . 91
13.5 Anycast addresses . . . 91
13.6 IPv6 new features . . . 91
13.7 IPv6 Header . . . 92
13.7.1 IPv6 Extension Headers . . . 92
13.8 Hierarchical routing . . . 93
13.9 ICMPv6 . . . 93
13.10 Neighbor Discovery Protocol . . . 94
13.10.1 Router Solicitation . . . 94
13.10.2 Router Advertisement . . . 94
13.10.3 Neighbor Solicitation . . . 95
13.10.4 Neighbor Advertisement . . . 95
13.10.5 ICMPv6 Address Resolution . . . 95
13.10.6 Router discovery . . . 96
13.11 Address autoconfiguration . . . 96
13.11.1 Duplicate Address Detection (DAD) . . . 97
13.12 IPv4-IPv6 Transition/Coexistence . . . 98
13.12.1 Dual stack . . . 98
13.12.2 Tunneling . . . 98
13.12.3 Network Address Translator – Protocol Translator (NAT-PT or NAT6to4) . . . 100
13.13 IPv6 popularity . . . 100

14 Network Layer: routing **103**
14.1 Intra-AS routing algorithm . . . 104
14.1.1 Static vs dynamic . . . 104
14.2 Intra-routing . . . 104
14.2.1 Link State . . . 105
14.2.1.1 Dijkstra's algorithm . . . 105
14.2.1.2 OSPF (Open Shortest Path First) . . . 108
14.2.1.3 LSA (Link State Advertisement) . . . 108
14.2.2 Distance vector . . . 110
14.2.2.1 Bellman-Ford's algorithm . . . 111
14.2.2.2 RIP (Routing Information Protocol) . . . 114
14.3 Inter-AS routing . . . 116
14.3.1 Border Gateway Protocol (BGP) . . . 116
14.3.2 Distance Vector with Path . . . 117

15 Transport layer **119**
15.1 User Datagram Protocol (UDP) . . . 119
15.1.1 UDP packet format . . . 119
15.2 Transport Control Protocol (TCP) . . . 121
15.2.1 TCP segment . . . 121
15.2.2 Connection set-up . . . 123
15.2.3 Bidirectional transmission . . . 123
15.2.4 Connecting closure . . . 123
15.2.5 Retransmission Time Out (RTO) . . . 124
15.2.5.1 Adaptive RTO . . . 124
15.2.5.2 RTO in pratice (*RFC 6298*) . . . 124
15.2.5.3 Karn's algorithm . . . 125
15.2.5.4 RTO management . . . 125
15.2.6 Delayed Acknowledgement . . . 126
15.2.7 Flow Control . . . 127
15.2.7.1 Deadlock . . . 128
15.2.7.2 Silly window syndrome . . . 128
15.2.7.3 Tiny datagram . . . 129
15.2.7.4 Pipeline . . . 129
15.2.8 Congestion control . . . 131
15.2.8.1 Slow Start/ Congestion Avoidance . . . 132
15.2.9 Reaction to segment loss . . . 134
15.2.9.1 TCP-old Tahoe) . . . 135
15.2.9.2 TCP-Reno . . . 136
15.2.9.3 TCP SACK (Selective acknowledgment) . . . 136
15.2.10 Standard TCP backdraws and limits . . . 137
15.2.11 New TCP proposals & upadates . . . 137
15.2.11.1 TCP Cubic . . . 137
15.2.12 TCP metrycs . . . 139
15.2.12.1 Delivery time . . . 140

16 Application layer **143**
16.1 Domain Name System (DNS) . . . 144
16.1.1 Distributed, Hierarchical Database . . . 145
16.1.1.1 Iterative resolution . . . 145
16.1.1.2 Recursive resolution . . . 145
16.1.2 DNS record . . . 146
16.1.2.1 Inserting records into DNS . . . 146
16.1.3 DNS protocol messages . . . 147
16.2 telnet . . . 149
16.3 HTTP . . . 149

16.3.1 Nonpersistent HTTP . . . 149
16.3.2 Persistent HTTP . . . 151
16.3.3 HTTP request message . . . 151
16.3.4 Cookies . . . 151
16.3.5 Web caching . . . 152
16.3.6 Dynamic Adaptive Streaming over HTTP (DASH) . . . 152
16.4 Mail Services . . . 153
16.4.1 Simple Mail Transfer Protocol (SMPT) . . . 153
16.4.1.1 How SMTP works . . . 153
16.4.2 Mail access protocols . . . 154
16.5 P2P file sharing . . . 154
16.5.1 P2P with centralized directory . . . 155
16.5.2 Query flooding: Gnutella . . . 155
16.5.3 Exploiting heterogeneity: KaZaA . . . 156
16.6 File Transfer Protocol (FTP) . . . 157

17 Quality of Service **159**
17.1 Classification . . . 160
17.1.1 Token Bucket Algorithm . . . 160
17.1.1.1 One Rate Two Colors (1R2C) . . . 160
17.1.1.2 Two-rate three color (2R3C) . . . 163
17.2 Traffic shaping . . . 163
17.2.1 Burstiness & Bandwidth Allocation . . . 164
17.2.2 Motivations of using traffic shaping . . . 164
17.2.3 Token Bucket for shaping bursty traffic . . . 164
17.2.3.1 Maximum burst length . . . 165
17.2.3.2 Maximum burst period . . . 165
17.2.3.3 Maximum packet delay . . . 166
17.2.4 Leaky Bucket . . . 166
17.2.4.1 Leaky Bucket vs Token Bucket . . . 166
17.2.5 Composite Shaper . . . 167

18 IEEE 802.x **169**
18.1 Ethernet . . . 169
18.1.1 Ethernet frame . . . 170
18.1.2 Classification of Ethernet cables . . . 171
18.2 Wireless LAN . . . 173
18.2.1 Frame types . . . 174
18.2.2 CSMA (Carrier Sense Multiple Access) . . . 175
18.2.3 Collision Avoidance (RTS/CTS) . . . 176
18.3 Bluetooth . . . 176
18.3.1 Voice and Data connections . . . 177

19 Netkit **179**
19.1 Simulation vs. emulation . . . 179
19.2 What is Netkit? . . . 179
19.3 User mode linux . . . 179
19.4 Netkit commands . . . 182
19.5 Preparing a lab . . . 182
19.5.1 A netkit lab as a single script . . . 182
19.5.2 Netkit labs using lcommands . . . 182
19.6 Zebra/Quagga . . . 184
19.6.1 Zebra configuration files . . . 184
19.6.1.1 deamons file . . . 186
19.6.1.2 zebra.conf . . . 186
19.6.1.3 ripd.conf . . . 186

19.6.2 Connection to the main zebra deamon . . . 187
19.6.3 Privileges on a router . . . 187

20 Firewall **189**
20.1 Packet filters . . . 189
20.2 iptables . . . 190
20.2.1 filter table . . . 192
20.2.2 nat table . . . 192
20.2.3 mangle table . . . 192
20.2.4 raw tablle . . . 193
20.3 iptables usage . . . 193

21 Real network devices **195**
21.1 Hub vs Switch . . . 195
21.2 Cables . . . 197
21.2.1 Patch cables . . . 197
21.2.2 Crossover cables . . . 197
21.3 CISCO router . . . 198
21.3.1 CISCO command modes . . . 198
21.3.2 Configuration of CISCO router . . . 199

22 Tools **201**
22.1 traceroute . . . 201
22.2 netstat . . . 202
22.3 tcpdump . . . 202
22.3.1 Packet sniffing with wireshark . . . 202
22.4 ping . . . 203
22.4.1 MTU . . . 203
22.5 ifconfig . . . 204
22.6 route . . . 205
22.7 arp . . . 206
22.8 telnet . . . 206
22.9 ssh . . . 206
22.10lynx . . . 207
22.11dig . . . 207
22.12HTTP request methods . . . 207
22.13SMTP commands . . . 208
22.14POP3 commands . . . 212
22.15IMAP commands . . . 213
22.16FTP commands . . . 213
22.17DNS files . . . 213

Capitolo 1

The story of internet

- **1961-64**
  Leonard Kleinrock, a brilliant MIT student, proves that packet switching is very efficient in presence of bursty traffic

- **1967**
  Lawrence Roberts submits a project to interconnect heterogeneous host computers through special nodes, called Interface Message Processor (IMP), a digital device that could route information trought the net. The project is funded by the Advanced Research Projects Agency (ARPA), of the Department Of Defense (DoD) of United States of America: it is the dawn of ARPAnet

- **1969**
  The first connection was extablished beetwen the UCLA and SRI hosts, as in Figure 1.1. SIGMA-7 supposed to send the message LOG and the SRI host had to reply with message IN. The first telnet crashed the system (PDP was down of service), after reading only **L** and **O**. ARPANET was up and running. In November, the network was composed by 4 nodes: UCLA, UCSB, Stanford Research Inst. (SRI), Utha Univ.

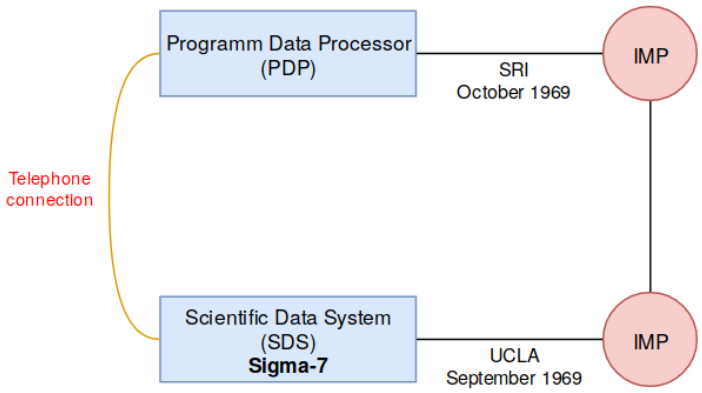


Figura 1.1: First connection beetwen different networks using IMPs. `https://www.youtube.com/watch?v=vuiBTJZfeo8`

- **1970**
  ALOHAnet A wireless network to interconnect campuses of Hawaii islands

- **1970-1980**
  proliferation of different (not interconnected) heterogeneous networks (Different packet sizes, interfaces, transmit rates, reliability...)

- **1972**

  - **Gateways**
    proposed by Cerf & Bob Kahn to interconnect heterogeneous networks (universal translator between heterogeneous networks

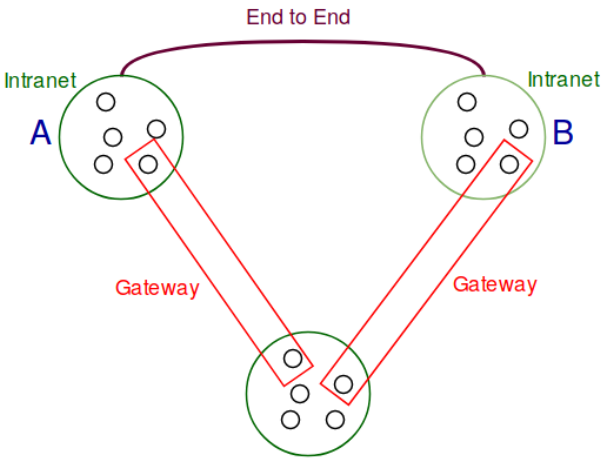


Figura 1.2: Example of gateways use inside a network.

  - **Transmission Control Protocol(TCP)**
    proposed to manage end-to- end communications over heterogeneous mix of networks (adding error control and recovery from IMP to end hosts

- **1977**
  - ARPANET, ALOHANET, Packet Radio network are as ARPA Internet
  - TCP gets split into TCP & IP
- **1981**
  UNIX includes the TCP/IP protocols
- **1983**
  - ARPANET (with more than 200 nodes) replaces NCP with TCP/IP proposed by Cerf & Kahn (CSNET-funded by the National Science Foundation to link computer science departments across the country – connects to ARPANET through TCP/IP)
  - Birth of the Domain Name Server (DNS) system
- **1990**
  ARPANET is decommissioned
- **1994**
  NFSNET is decommissioned
- **1995**
  the INTERNET becomes fully commercial
- **NOWADAYS**
  more than 40,000 networks are interconnected by means of TCP/IP protocols

Capitolo 2

Organization of internet

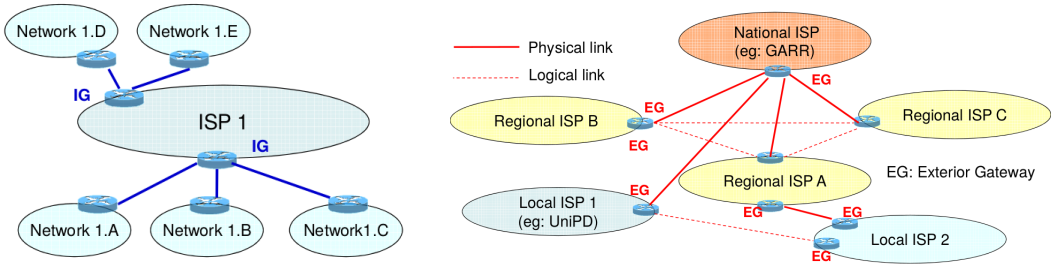
The Internet is a global system of interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to serve several billion users worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks, of local to global scope, that are linked by a broad array of electronic, wireless and optical networking technologies, Figure 1.2. The Internet carries an extensive range of information resources and services, such as the inter-linked hypertext documents of the World Wide Web (**WWW**), the infrastructure to support email, and peer-to-peer networks.

- **INTRANET**
  network based on the same protocol(same language) for the communication between the devices used

- **INTERNET**
  network that permit devices to talk to each other in the case that they are based with different protocols; The biggest group of interconnected networks based on the TCP/IP protocol stack

2.1 Internet Service Provider(ISP)

ISP is a telecommunication company that interconnect other (smaller) networks. It's connected to interior gatewayes (Figure 2.1a). The connection between different ISPs is guaranteed by exterior gateways and this structure is hierarchical (Figure 2.1a). GAAR is the national ISP that manages and provides connection to all the italian universities.



(a) ISP.

(b) ISP hierarchical structure.

Figura 2.1: How ISPs work.

2.2 Network Access Point (NAP)

NAP is a private company that provides ISPs interconnections (Figure 2.2). If NAP didn't exist, if there were problems on the maintenance of the common line, telephone opearators, that used that link, would have some argument trying to decide which of them must repair it (watching traffic statistichs and time of use from user of each of them). So this company manages the line and guarantees proper operation of the internet service, connecting different ISPs.

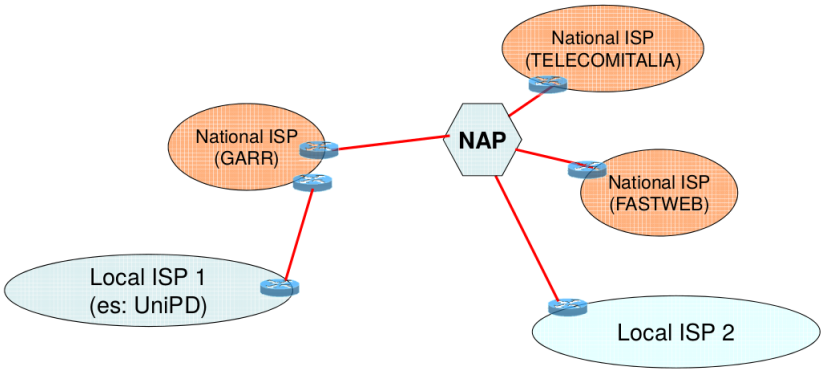


Figura 2.2: Example of NAP use.

2.3 Milan Internet eXchange (MIX)

It was born to promote the development of Internet in Italy, and to improve the interconnection between different ISPs operating in our country. Mix is a point of "multiple interconnection" in which the networks of each player (ISP, carrier, content provider, hoster etc...) interconnect themselves to exchange IP traffic (peering) efficiently and with advantageous costs compared to the transit. MIX is NOT an ISP, it does not provide Internet access to public accounts, not publish content, not sell web spaces, not offer web-hosting or web-housing services. The principle customers of MIX are:

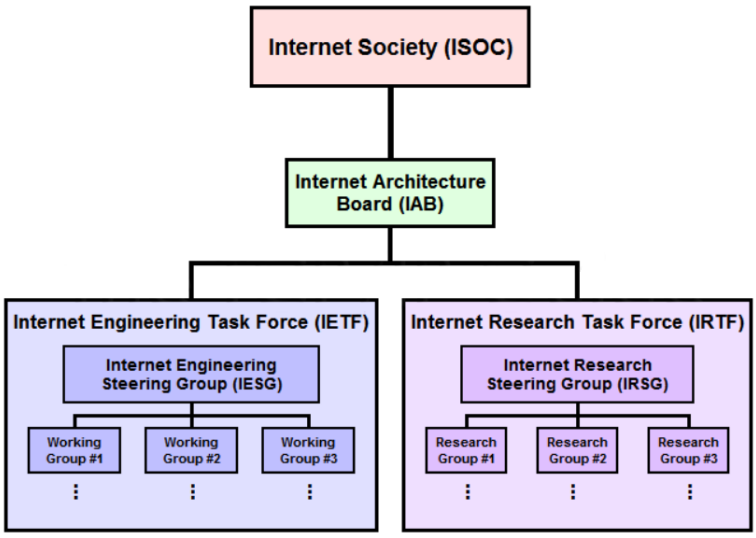
- **Internet operators (Peers)**
  realize peering agreements (public and/or private) with the other ISPs connected
- **Carriers**
  Telecommunication operators that use their equipment to give transit services from and to MIX
- **Operators of root-name servers and Top Level Domain**
  operators that provide super-partes services useful to the operation of Internet, supplied by means of direct peering

2.4 Internet Society (ISOC)

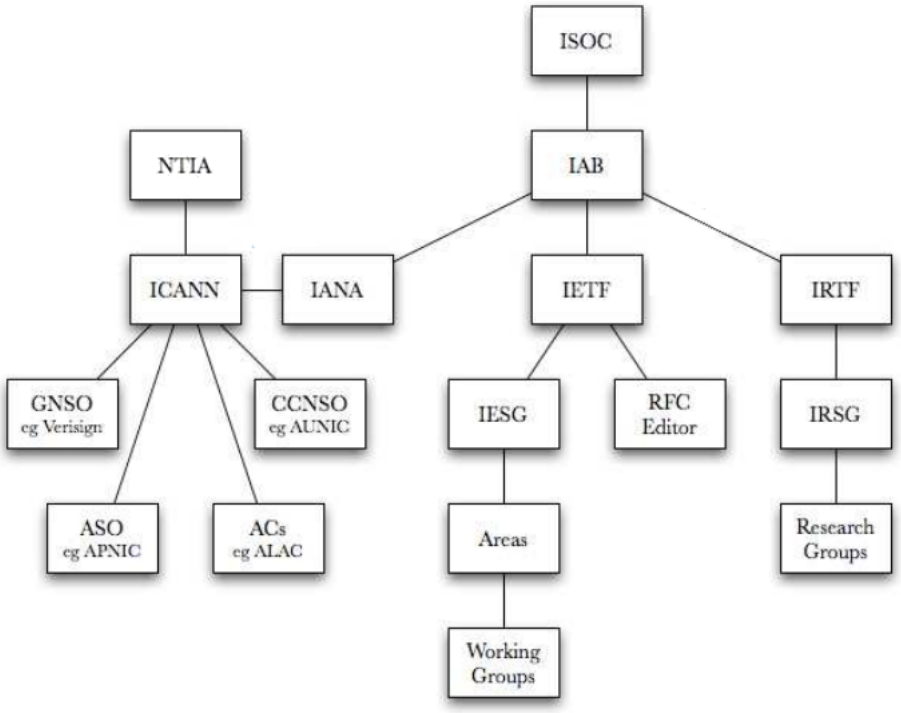
ISOC is a no-profit company that promotes the use of internet. Inside this agency, there is Internet Architecture Board (IAB) composed of 600 companies from USA and UE (Figure 2.3a). Inside IAB, there are three sections:

- **Internet Assigned Numbers Authority(IANA)** (Figure 2.3b)
  It defines DNS root, IPs and other protocols. The most famous sections inside this department are:
  - **Internet Corporation for Assigned Names and Numbers(ICANN)**
    Assignment of IPs and recognition of protocols
  - **National Telecommunications and Information Administration(NTIA)**
    Manages network and its improvement

- **Internet Engineering Task Force(IETF)**
  Development and Standard promotion (Request for comments *RFC*, document with all the innovations)
- **Internet research Task Force(IRTF)**
  Development of new protocols, research and experiments



(a) General organizzation of ISOC.



(b) Detailed organization of ISOC.

Figura 2.3: ISOC.

Capitolo 3

Architecture of a network

3.1 Classification based on the extension of the network

Looking to the physic extension of the network, there are 5 types of network:

- **Body area networks (BAN)**
  wearable devices that can define a network delimited by the body of a person
- **Personal area networks (PAN)**
  Like Bluetooth
- **Local area networks (LAN)**
  Like home network
- **Metropolitan area network (MAN)**
  Extension of a city
- **Wide area network (WAN)**
  Transatlantic network

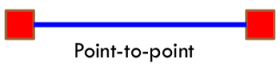
3.2 Topology

A graph can be used to describe a network. A node represents a device that composes the network and an arch is a link. There are two different topologies of a network:

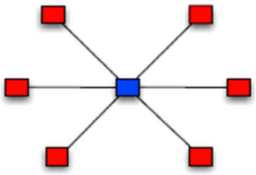
- **Physical topology**
  Graph formed by nodes and physical links
- **Logical topology**
  Graph formed by the information flows

A graph can be organized in different ways. The most important types of graphs are:

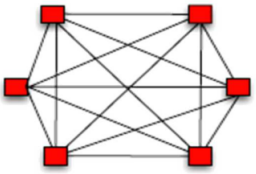
- **Point to Point**
  It can be used to represent a single link in physical topology or the connection between two end devices in logical topology



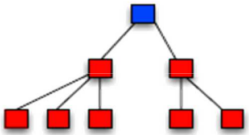
- **Star Topology**
  It can be the Wifi network (inside a house or a building) in a physical topology



- **Mesh Topology**
  It can represent the connection between every possible device in the world in logical topology

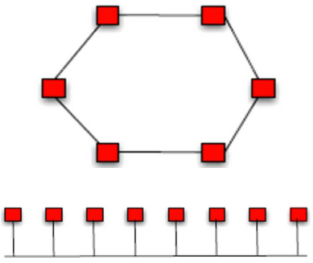


- **Tree Topology**
  Used for physical connection of sensors because of their short range. They are connected with a tree structure to central station (SINK)



- **Ring Topology**
  Used in WAN application because of the smaller amount of fibre that you need with respect to a Star topology

- **Bus Topology**
  Used in cases in which the line is shared between multiple devices. The stream of data is available on the line for each user



3.3 Classification based on the function of the network

There are two types of network (Figure 3.1):

- **Access Network (or Edge Network)**
  End devices can be connected to internet through this type of network. Its main features are:
  - Provide access to communication services to the end systems
  - May have different topologies

  The main links are composed by twisted pair, optical fiber (rarely), Wi-Fi, Internet, Bluetooth, LTE (smartphone internet), UMTS (3G) for phone call, Lora (Long Range), LPWAN (new family of connections like Lora), Zig Bee, Z-Wave, NB-IOT and WiMax (replaced than by HiperLAN).
- **Transport Network (or Core Network)**
  It interconnects different access networks. Its main features are:
  - Transfer Information Units over long distances
  - May have hierarchical structure

  It is usually composed by optical fibre or copper cable links.

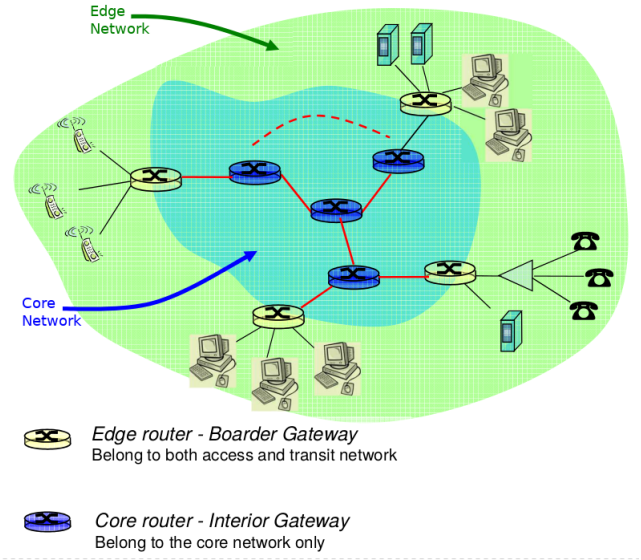


Figura 3.1: Edge and core networks.

3.4 Open System Interconnection (OSI)

All needed functionalities for a system interconnection are organized in a hierarchical structure with few "layers". This structure is a reference model for every device used in networks and it's called Open System Interconnection

(OSI).
Each layer is in charge for a class of functionalities (The lowest layer, i.e., PHYsical layer, is in charge to deliver a stream of bits from one device to another (one "hop")). The functionalities are realized by "entities" (Figure 3.2). Some devices, like switches and routers, have only functionalities of layers 1-3 because they only receive stream of data and send them again, routing them and deciding which is the next hop (Figura 3.3).

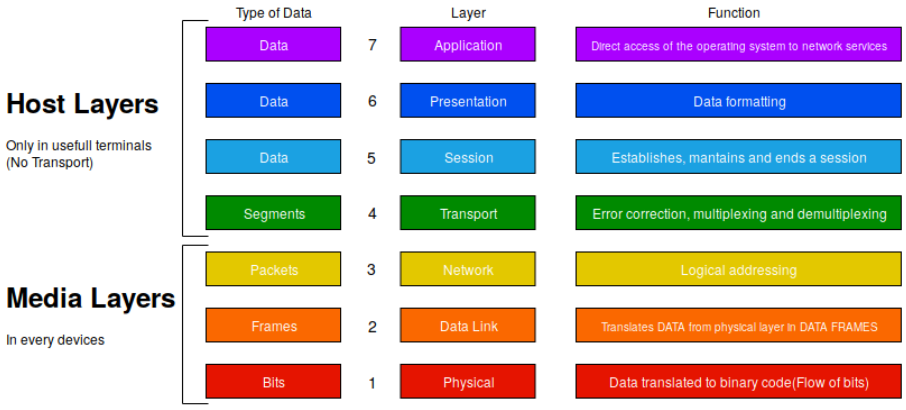


Figura 3.2: OSI layers and their functions.

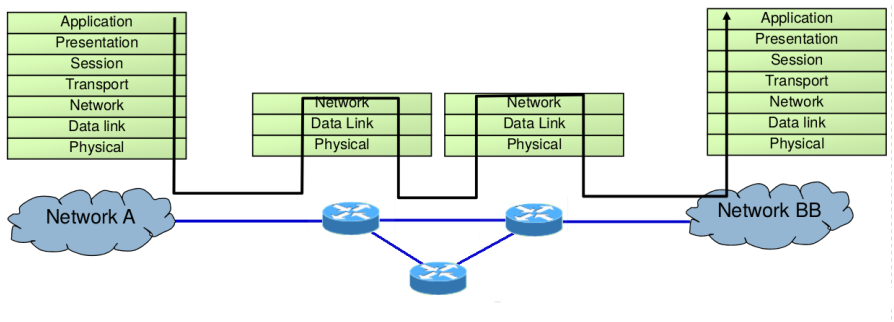


Figura 3.3: Example of OSI application.

3.4.1 Internet Protocol Suite (IPS)

In TCP/IP protocol there is another organization of the functionalities of a network device with Internet Protocol Suite (OSI). Some layers, of the reference model OSI, are joined together (Figure 3.4).

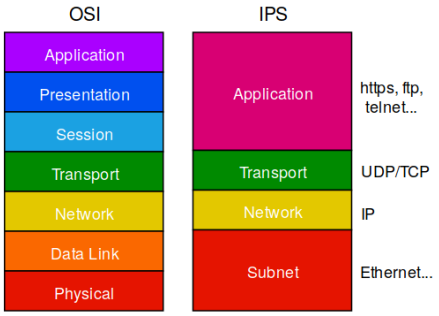


Figura 3.4: OSI layers with respect to ISO layers.

3.4.2 Encapsulation and decapsulation

Some control information is added to a message generated in an upper layer. This control sequence is called **header** and it's added to the message, called **payload**. A packet with his added layer define the Protocol Data Unit (PDU) of this layer. A PDU is the Information Unit (IU) exchanged between two entities of the same level.
The PDU of i^{th}-layer is the Service Data Unit (SDU) of the $(i-1)^{th}$-layer (Figure 3.5). When a packet has to be sent, it's usually done **Encapsulation** that consists on adding control information to original packet generated at Application Layer (or other lower layers).
When a packet is received by a device, the headers, added by lower layer to original packets, are used to understand how information is organized in packet, how it was coded and how error can be found or corrected by the device and the original packet is highlighted (Figura 3.6). This second phase is called **Decapsulation**.

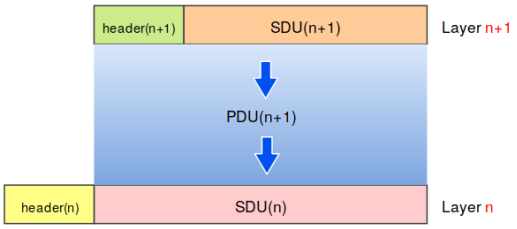


Figura 3.5: Example of PDU and SDU of two generic layers.

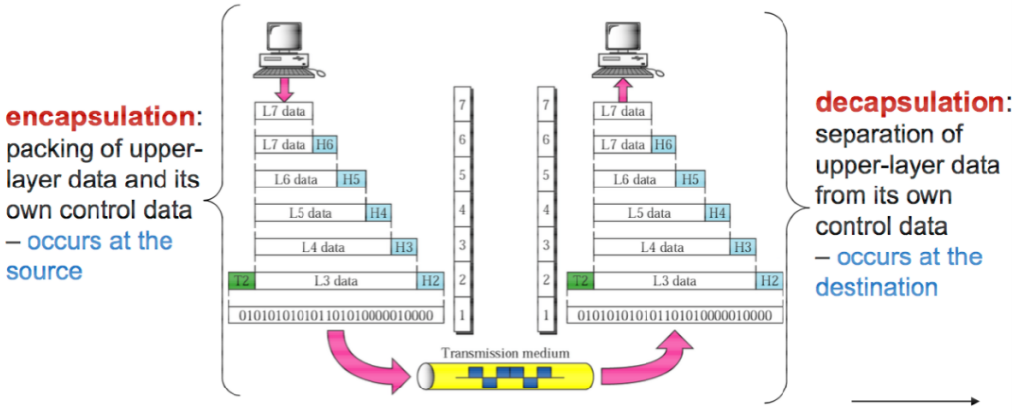


Figura 3.6: Example of encapsulation and decapsulation.

3.5 Structre of a telecommunication network

The main components of a telecommunication networks are:

- **Terminal Devices**
  PC, phones, screens, speakers, mics, storing unit...
- **Switching unit**
  Routers for data networks, switches for telephone networks. Each switching node is composed by:
  - **Ingress (or input) ports**
    devices that receive data from the connected links, generally provided with queueing buffer
  - **Egress (or output) ports**
    device that transmit data on the connected link, generally provided with queueing buffer

  - **Switching logic**
    mechanism that makes it possible to transfer an Information Unit from one input port to an output port

- **Links**
  Fibre optic, copper, twisted pairs, radio, ...

The "soul" of a TLC net are communication rules, based on:

- **Protocols**
  horizontal communication, language spoken by different hosts at the same level

  - Defines the format and the meaning of the language talked by peer entities (i.e., entity of the same layer) of different devices

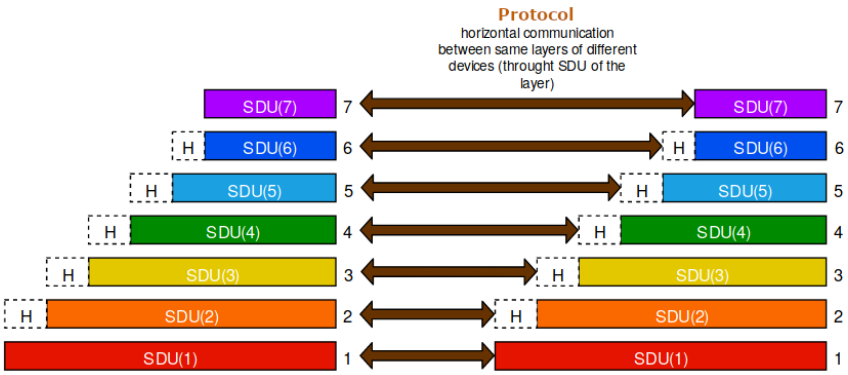


Figura 3.7: Example of protocols.

- **Interface**
  vertical communication, language spoken by the same host

  - defines the format and the meaning of the command primitives that can be exchanged between the entities of adjacent layers in the same device
  - Protocol Data Unit (PDU) of upper layer is the Service Data Unit (SDU) of lower layer

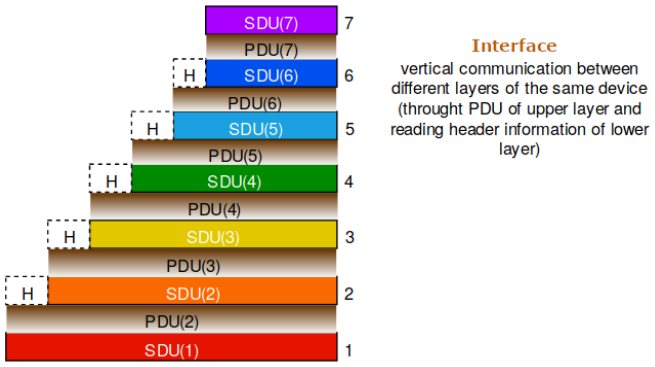


Figura 3.8: Example of interfaces.

3.6 Main application architectures

- **Client-server** (Figura 3.9)
  - **Server**
    1. always-on host
    2. permanent IP address
    3. server farms for scaling
  - **Client**
    1. communicate with server
    2. may be intermittently connected
    3. may have dynamic IP addresses
    4. do not communicate directly with each other

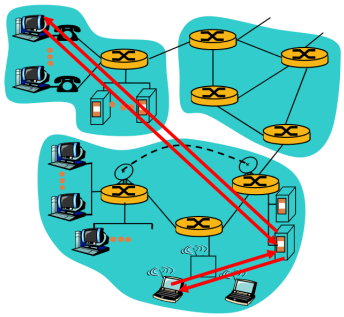


Figura 3.9: Client-server network.

- **Peer-to-peer (P2P)**
  1. not always-on server
  2. arbitrary end systems directly communicate
  3. peers are intermittently connected and change IP addresses
  4. Highly scalable but difficult to manage

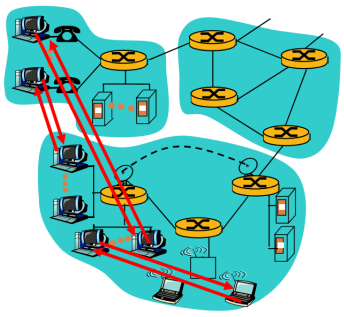


Figura 3.10: Peer-to-peer network.

- **Hybrid of client-server and P2P**

3.7 Types of switching

3.7.1 Circuit switching

Every connection is established through a physical connection(electric bridge). It was used in the past to create connection between two tephone terminals (the line was created because of switching made by a person). This type of connection is useless to transfer bursty data (multimedia) because this data is sent only in limited and short slot of time. The necessary time to establish the connection is bigger than the slot time, so it's useless for this type of data but it can be used for Skype or telephone calls, that are very long. Once the connection is established, the data is transfered only throught that line, guaranting performances but risking of blocking, wasting resources (the line is occupied since the end of the transmission hence no one else can use it).

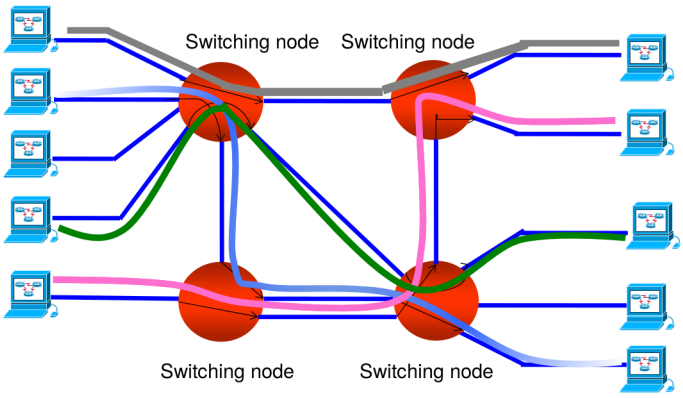


Figura 3.11: Circuit switching.

3.7.2 Packet switching

Multimedia are composed by a set of sequences (**Packets**). Sometimes very large messages are divided into small packets and then sent to the network. Packet switching consists in two basic operations:

- **Routing**
  Selection of the next hop (next node to reach) and insertion of the IU in the buffer of the corresponding output port. It consists in running an algorithm that decides the best output (the best line that it must choose). It is based to find the best:
  - **Mechanism**
    Organization inside the router
  - **Algorithm**
    Routing inter-routers strategy (between multiple routers)
- **Forwarding**
  Physical transmission of the IU to the next hop node (**Hardware side**). It is in relationship with hardware improvements.
- update the Network Interface Card (NIC)
- no need to change the routing software

On the line there is mechanism of **Multiplexing**, that consists on sending packets from different devices on the same line.
When the packets arrive to a node, there is the mechanism of **Demultiplexing**, that analizes the packets and understand what are the origins of them and so packets are sent to the true end devices, routing in a true way (Figure 3.12).
Data can be queued in a node if input rate is greater than the output rate. The drawbacks of this type of switching are packet loss and delayed transfer of messages.

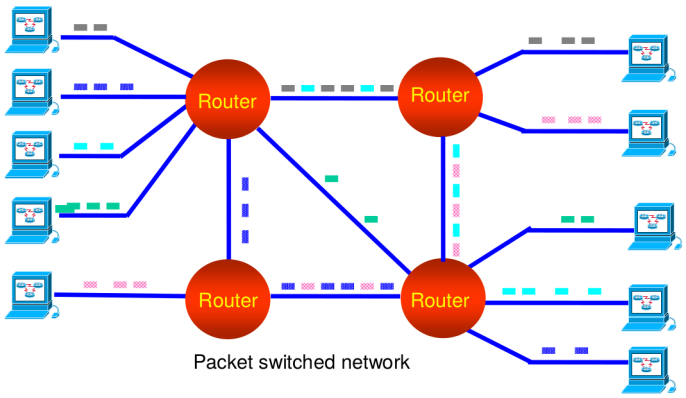


Figura 3.12: Packet switching.

3.7.3 Virtual circuit switching

Virtual circuit switching is a packet switching methodology whereby a path is established between the source and the final destination through which all the packets will be routed during a call. This path is called a virtual circuit because to the user, the connection appears to be a dedicated physical circuit.
However, other communications may also be sharing the parts of the same path. Before the data transfer begins, the source and destination identify a suitable path for the virtual circuit. All intermediate nodes between the two points put an entry of the routing in their routing table for the call.
Additional parameters, such as the maximum packet size, are also exchanged between the source and the destination during call setup. The virtual circuit is cleared after the data transfer is completed.

Advantages of virtual circuit switching are:

- Packets are delivered in order, since they all take the same route
- The overhead in the packets is smaller, since there is no need for each packet to contain the full address
- The connection is more reliable, network resources are allocated at call setup so that even during times of congestion, provided that a call has been setup, the subsequent packets should get through
- Billing is easier, since billing records need only be generated per call and not per packet

Disadvantages of a virtual circuit switched network are:

- The switching equipment needs to be more powerful, since each switch needs to store details of all the calls that are passing through it and to allocate capacity for any traffic that each call could generate
- Resilience to the loss of a trunk is more difficult, since if there is a failure all the calls must be dynamically reestablished over a different route

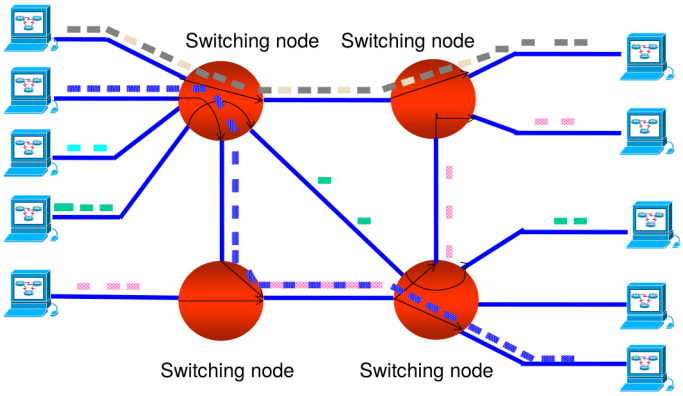


Figura 3.13: Virtual circuit switching.

Capitolo 4

Performance metrics and QoS issues

- **Bandwidth (W [Hz])**
  Range of frequencies contained in a modulated signal, or range of frequencies a channel can pass with acceptable attenuation

4.1 Traffic

- **Bitrate (R_0 [bit/s] or [bps])**
  Maximum transmission speed at bit level [bit/s] of a link $R_0 = W \log_2(1 + \Gamma)$
  - best transport rate it can offer to upper layer
  - typically max bit transfer rate at PHY or MAC layer
- **Throughput (S [bit/s], [pck/s], [pck/slot])**
  - Average data transfer rate from "service user" perspective
  - rate at which user information units are actually delivered to peer entity

  It is always $S \leq R$ because:
  - user may not "saturate" the connection
  - some transport capacity is taken by control signals (overhead)
  - connection can be congested or shared by multiple flows
  - Medium access control protocols may introduce idle times

  Denoting by $b(t)$ the total amount of bits delivered up to time t we can define the following throughput metrics:

  1. **Instantaneous throughput**
     $s(t) = \dot{b}(t)$ (derivative of b(t) wrt t)
  2. **Average throughput in a time slot T**
     $s(t;T) = \frac{b(t)-b(t-T)}{T}$
  3. **Long-term or steady-state or aymptotic throughput**

     $$S = \lim_{T \to \infty} s(t;T)$$

- **Goodput (S_g [bit/s], [pck/s], [pck/slot])**
  - **transfer rate seen by a certain entity**
    average speed at which the bits of the Service Data Unit (SDU) at a certain level are delivered to the destination peer entity

  - **throughput scaled by the overhead and other protocol inefficiencies**
    it's always $S_g \leq S$ because part of the transmission capacity (throughput) is taken by control signals (overhead) and sometimes might be wasted in idle periods or failed transmissions

For any given layer n, it holds $S_g(n) = S(n) * \frac{length(SDU(n))}{length(PDU(n))}$

4.2 Delay

- **Processing delay (d_{proc} [s])**
  Time required by intermediate and end nodes to "process" the message (check, Routing). Nowadays it is typically constant and often negligible but it was an important parameter in the past when the connection had usually been created manually.
  IT CAN BE REDUCED BY ADOPTING EFFICIENT ROUTING MECHANISMS

- **Queuing delay (d_{queue} [s])**
  Time spent by a message in a buffer, waiting to be processed by nodes. It depends not only on the structural characteristics of the links but also on the traffic intensity.
  (PARTIALLY) CONTROLLABLE using:

  - **Traffic priority**
    Reduce priority for a certain type of traffic of data
  - **Scheduling**
    Decide which information must be sent first
  - **Call admission**
    Prevent the use of some switches that are in congestion
  - **Control mechanisms**
  - **Increasing of number of output lines**

- **Transmission delay (d_{trans} [s])**
  Time between the reception of the first and the last bit of the message

$$d_{\text{trans}} = \frac{length(PDU)}{Throughput}$$

  IT CAN BE REDUCED BY INCREASING THE TRANSMIT RATE OF THE LINK AND USING ROUTING ALGORITHMS (that permit to understand which of the link is best performing)

- **Propagation delay (d_{prop} [s])**
  Time required for the physical propagation of the electromagnetic signal that carries the digital information from source to destination

$$d_{\text{prop}} = \frac{distance}{propagation\_speed}$$

  It is typically negligible, except in long-range communication (satellite, cross ocean, intercontinental) and very high speed transmission links. It plays a role even in local area network (vulnerability time).
  (E.g. EVD transmission of data, created in the past and based on the use of System Orbital Satellite, moving around the Earth, that permits to cover also Ocean with Internet. This type of connection was very expensive and no one bought it.)
  NOTHING TO DO ABOUT IT

- **End-to-End Delay (or Latency or Delay [s])**
  Overall time taken to deliver a complete message to the intended receiver, from the time in which the first bit is sent out from the source (end-to-end delay)

$$d_{\text{e2e}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

- **Jitter**
  It's the delay variation. Large jitter calls for long playout delays, large playout buffers and long queues. It's particularly critical for "streaming" services (e.g. in youtube video, constant delay is not a big issue, but when playout starts, image freezing is very annoying).

  In **M/G/1** queueing systems:

$$m_q = \frac{{\sigma_v}^2}{2(1-m_v)} - \frac{m_v}{2}$$

  with:
  m_q=mean queue length
  ${\sigma_v}^2$=jitter
  m_v=mean number of arrivals in a mean service time

4.3 Pipe capacity

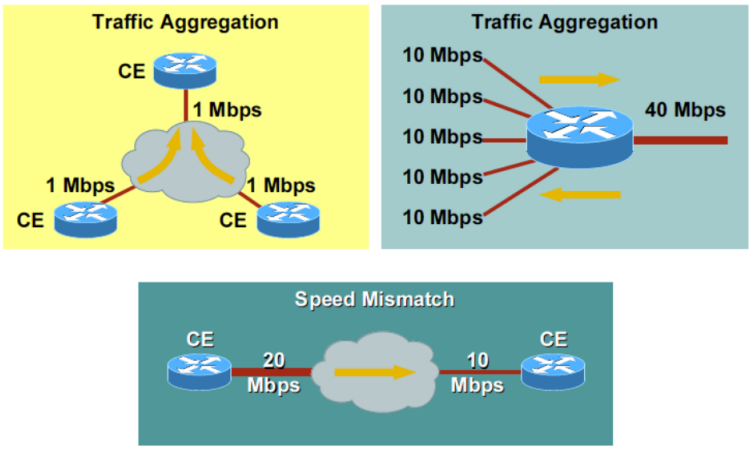
- **Bandwidth-delay product** (*BDP* **[bits] [packets]**)
  - Number of bits (or packets) that can be simultaneously "in flight" along a connection
  - Number of bits that can "fill" the link
  - Total number of bits (packets) that can be delivered to the receiver (in steady state) in a time equal to a packet end-to-end delay (d_{e2e})

  The BDP and the number of bits/packets that the sender needs to transmit while waiting for an acknowledgment from the receiver, are larger in order to fully exploit the link capacity. If systems knows what is BDP on the links, it can send a true amount of bits otherwise data are queued.

4.4 Reliability

Too many sources sending too much data too fast, for network to handle, can cause **congestion**. Possible manifestations are:

- lost packets (buffer overflow at routers)
- long delays (queueing in router buffers)



- **Packet Error Rate** (*PER*)
  - fraction of packets with errors over the total number of transmitted packets

  - probability that a packet transmission is affected by errors

- **Packet drop rate (P_{drop})**
  Fraction of total sent packets that are dropped.
  Packets, to be forwarded on a busy link, are queued in the output buffer of the switching node. A buffer has finite capacity hence, when it's full, it cannot host any more packets. So packets are dropped in two possible ways:

  - **Drop tail policy**
    new packets arriving to full buffer are dropped
  - **Random Early dropping**
    when buffer occupancy is above a critical threshold (e.g., 2/3 of storage capacity), queued packets are dropped at random.

  A congestion control is applied at the Edge Network. Watching the decreasing of transmission rate, for the reliability of links, the source understands that there could be congestion and sends a lower number of packets.

- **Packet Loss Rate (P_{LOSS})**
  Overall fraction of transmitted packets, that are not delivered to the destination, because of any reason (errors or buffer overflow)

- **Packet delivery ratio (PDR)**

  - Overall fraction of packets that are successfully delivered to the destination
  - Ratio between the number of successfully delivered packets over the total number of transmitted packets

$$PDR = 1 - P_{LOSS}$$

Capitolo 5

Performance analysis of switching methods

Hypothesis of the analysis:

- **N links** with transmission rate $\{R_{\mathrm{i}} : i = 1, .., N\}$ and propagation time

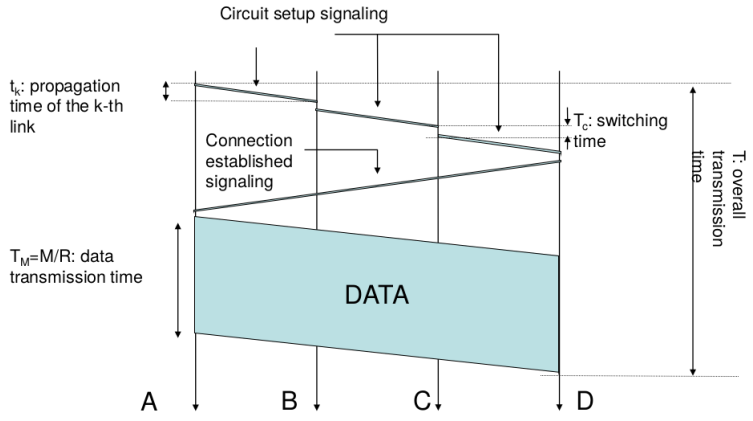
$$t_{\mathrm{i}} = \frac{length\ of\ i^{th}\ hop}{signal\ propagation\ speed} \quad \forall\, i \in [1, N]$$

- **N+1 nodes** (2 end nodes, $N - 1$ switches)
- **M**=message size [bits]

5.1 Circuit Switching

switching time=time of the switcher needed to establish the connection between input and output links

data transmission time=time needed to transfer the message M using a bitrate R



5.1.1 Message delivery time

- **Circuit set up time**
  $T_1 = \sum_{k=1}^{N} (t_{\mathrm{k}} + T_{\mathrm{c,k}})$
- **Connection established signaling**
  $T_2 = \sum_{k=1}^{N} t_{\mathrm{k}} = t_{\mathrm{p}}$

- **Dara transfer**
  $T_3 = \frac{M}{R} + \sum_{k=1}^{N} t_k$

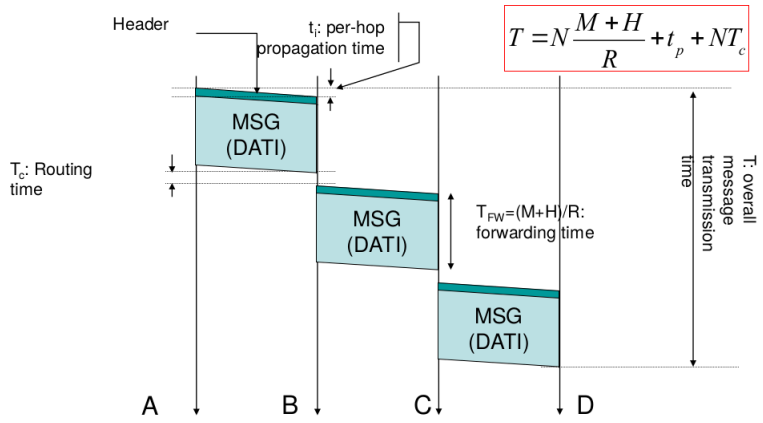
Hence the message delivery time is

$$T = T_1 + T_2 + T_3 \rightarrow \text{assuming } T_{c,k} = T_c \rightarrow T = \frac{M}{R} + 3t_p + Nt_c$$

5.2 Packet Switching

It's based on **"Store & Forward"**:

At each hop, the message is first completely received by a switch node, then stored in the output buffer, and finally forwarded along the output line.
It needs to add **routing information** to the message: a **header (H)** is attached to the message (OVERHEAD). The header contains information about destination address and source address (to be used for replies).
Sometimes it is used the mechanism of **CUT THROUGH**, in which a node receives and stores only header and, after this process, the corresponding data is transferred. This type of method is often applied in LANs networks.
t_c = **Routing time** = time necessary to take a packet from the input port and to copy that to the output



port, processing routing info

5.2.1 Pipeline

Different links can be used simultaneously to transmit different messages. A node can simultaneously receive from all its input ports and transmit over all its output ports. So the messages are transmitted continuously as a stream of bits and then they are seen by nodes as packets, watching the header and understanding what are the source and the destination.
For huge messages, the buffer would be full after a little bit of time if there are too many users. The message is divided in many smaller packets, and for each of them we need to copy and add the header.

- Exploit pipeline effect
- Avoid buffering overflow at intermediate nodes
- Ease multiplexing of packets from different flows

5.2.2 Bottleneck

The bottleneck of a connection is the slowest link. Since upstream links are faster, packets may arrive at a faster pace than the forwarding rate of the switch. In the long term, packets will be queued up at the output buffer port. If the upstream rate is permanently larger then the downstream, sooner or later the buffer will fill up and some packets will be dropped. The long-term throughput of a connection is upper limited by the maximum downstream rate of the bottleneck.

5.2.3 Message delivery time

- **if** $R_{\mathrm{i}} = R \ \forall \ i$
  $K=$ number of packets in which the message is divided

  - **1st packet propagation & routing times**
    $T_1 = \sum_{j=1}^{N} (t_{\mathrm{j}} + T_{\mathrm{c}})$
  - **Forwarding time of first packet**
    $T_2 = \sum_{j=1}^{N} \frac{H+\frac{M}{K}}{R}$
  - **Total Forwarding time of remaining packets**
    $T_3 = (K-1)\left(\frac{H+\frac{M}{K}}{R}\right)$

  We can also define $T_{\mathrm{fp}} = T_1 + T_2 =$**first packet delivery time**.
  Hence the message delivery time is

$$T = T_{\mathrm{fp}} + T_3 = \sum_{j=1}^{N} t_{\mathrm{j}} + NT_{\mathrm{c}} + (N+K-1)\left(\frac{H+\frac{M}{K}}{R}\right) = t_{\mathrm{p}} + NT_{\mathrm{c}} + (N-1)\frac{H+\frac{M}{K}}{R} + \frac{M}{R} + KH$$

- **different** $R_{\mathrm{i}} \ \forall \ i$ **(case of Bottleneck)**

$$T = T_{\mathrm{fp}} + (K-1)T' = t_{\mathrm{p}} + NT_{\mathrm{c}} + \left(H + \frac{M}{K}\right)\left(\sum_{h=1, h\neq 2}^{N} \frac{1}{R_{\mathrm{h}}}\right) + \frac{KH+M}{R_2}$$

  Let h* be the index of the bottleneck link. Pretending K is continuous, and setting to zero the derivative of T w.r.t. K, then solving for K we get

$$\frac{dT}{dK} = -\frac{M}{K^2}\left(\sum_{h=1, h\neq h^*}^{N} \frac{1}{R_{\mathrm{h}}}\right) + \frac{H}{R_{\mathrm{h}^*}} = 0 \Longrightarrow K = \sqrt{\frac{M}{H}\sum_{h=1, h\neq h^*}^{N} \frac{R_{\mathrm{h}^*}}{R_{\mathrm{h}}}}$$

  Hence the optimal number of packets (in case R_{h} is the same for all $h \neq h^* \Rightarrow R_{\mathrm{h}^*} = (N-1)R_{\mathrm{h}}$) is

$$K = \sqrt{\frac{M}{H}(N-1)}$$

5.3 Round Trip Time & Pipe Capacity

- **Round Trip Time** (RTT **[s]**)
  for an end-to-end (e2e) connection A→D, time taken by a Information Unit sent by A to reach D and for the corresponding acknowledgement packet (ACK) to return to A, assuming no buffering delay

- **Pipe Capacity** (C **[bits] [packets]**)
  maximum long-term average number of bits that can be transferred to D in a RTT interval, assuming no buffering and full exploitation of the pipeline effect.
  Hence the maximum long-term transfer rate, that can be achieved through the connection, is hence equal to

$$Max\ throughput = \frac{Pipe\ capacity}{RTT}$$

5.3.1 BDP with specific types of delay

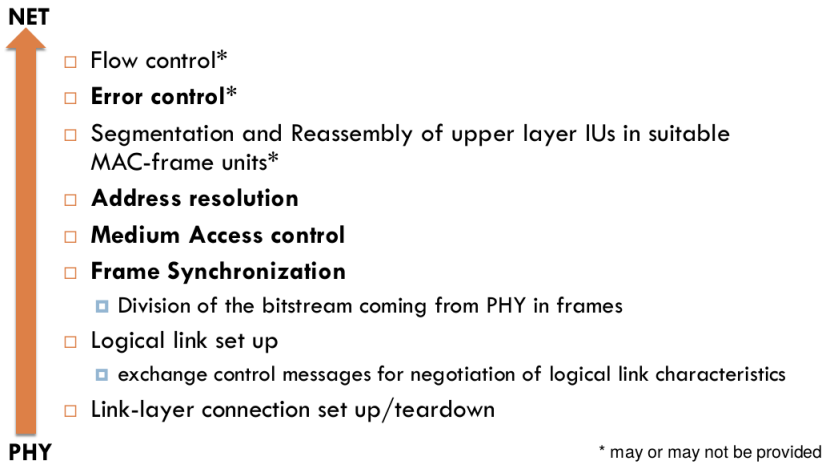
Maximum average throughput multiplied by delay

- **If delay = e2e delay and links have equal rate**
  BDP is the amount of bits that can be simultaneously sent along a (full) pipe without queueing
- **If delay = RTT**
  BDP is pipe capacity, i.e., average number of bits that can be delivered to the destination in a time equal to RTT, if the pipe is always full

Capitolo 6

Data Link Layer (DLL)

The main functionalities of this layer (classified according to the proximity to Physical and Network layers) are:



6.1 Frame synchronization

The DLL needs to find the byte and frame boundaries in the bit stream (generated in PHY layer). There are two possible situations:

- **Character-oriented services**
  The messages are composed of characters, hence DLL needs to split the bitstream in characters and, then, to find the boundaries of the MAC frame in the stream of characters

- **Bit-oriented services** The messages are composed of sequence of bits, hence DLL needs to find the boundaries of MAC frame in the stream of bits

6.1.1 Character-oriented services

1. **Step: Character synchronization**
   There isn't data to be transmitted and the sender keeps sending SYN characters to maintain bits (and characters) synchronization at the receiver.
   Receiver runs HUNT MODE: it slides a window of 8 bits until it matches with SYN. If SYN is observed for a sufficiently high number of characters, then characters synchronization is achieved and HUNT MODE is exited.
   SYN cannot be equal to any circular shift of itself (less than 8 bits), otherwise the receiver could be synchronized to wrong bit sequence. For this reason, SYN must be a prefix code.

2. **Step: Frame synchronization**
   Text beginning and ending are marked by special control characters:

   - **Data Link Escape (DLE)**⇒ 0001 0000
   - **Start of Text (STX)**⇒ 0000 0001
   - **End of Text (ETX)**⇒ 0000 0011

   Some special sequences of characters are used to understand where is the beginning or the end of the text:

   - **Beginning of the text**⇒DEL STX
   - **End of the text**⇒DEL ETX

   It's used a series of two special sequences to reduce the probability that receiver could understand in a wrong way the message, if some packets were lost, hence this is used to increase the robustness of the connection. If the text contains a DEL characters, transmitter will code it as DEL DEL.
   Any time a DLE symbol in text body is followed by a DLE symbol, receiver understand that the first is part of message and the second is stuffed and it's deleted (**Character stuffing**) (Figure 6.1).
   This type of approach can be used at other layer. At application layer, it can be used by a program to define that a frame is ended or started for the other program that uses it.

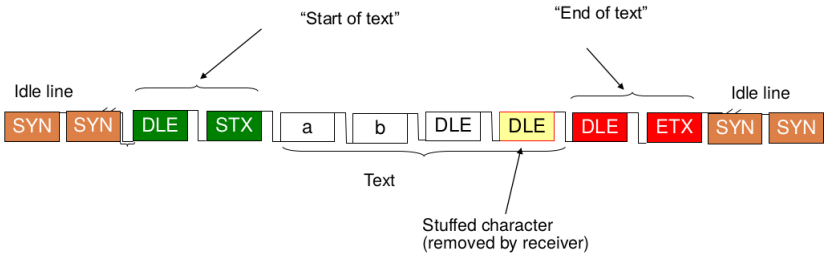


Figura 6.1: Example of char-stuffing.

6.1.2 Bit-oriented services

1. **Flags**
   Sender transmits an **idle** pattern, that contains logical bit variation to help PHY layer bit synchronization (E.g. ⇒ 01111111 01111111 01111111).
   Specific bit sequence (Flag) is used to mark start and end of message (e.g. flag for protocol HDLC ⇒ 0111 1110). The pre-processing phase is made by the sender. Every time there are 5 ones, the next bit is stuffed to 0 by the sender (**Bit-stuffing**) (Figure 6.2).
   Receiver checks the bit after a sequence of 5 bits equal to 1, if the next is:

   - **zero**
     this is silently dropped
   - **one** the receiver check the next bit, if it's:
     - **zero**
       this is the FLAG sequence and the transmission ends
     - **one**
       this is not the FLAG sequence and an error occurs (generated by receiver, decoding message) because there cannot be 7 bits not followed by a 0

Figura 6.2: Example of bit-stuffing.

2. **Start of Frame Delimiter & frame length field**
   It is used in LANs with broadcast channels, in which it is not possible to continuously transmit idle sequence. Hence to allow bit synchronization, each transmission is preceded by a synchronization **preamble** (E.g. $\Rightarrow$ 10101010101010...).
   The end of this preamble is marked by a specific sequence called **Start-of-frame delimiter** (E.g. $\Rightarrow$ 10101011).
   The start-of-frame delimiter is followed by a fixed-length header field that usually contains several information (e.g., source and destination MAC addresses, frame checksum sequence, type of frame, ...) among which, a field that encodes the frame length (usually, but not necessarily) expressed in bytes.
   This method has also drawbacks:

   - **Overhead**
     if the packet is very big and so it's divided in small packets, you need to copy this **preamble** + **fixed_header** for each packet
   - **If there is an error on header field**
     dyssynchrony of sender and receiver and wrong number of bits in the message

3. **Bit-encoding violations**
   This method is typically used with baseband modulation (E.g. Manchester encoding in Ethernet links).
   Each bit is encoded with a **low-to-high** or **high-to-low** transmission on the line, considering that there is always a signal translation in the mid of bit period.
   A controlled violation of this encoding scheme can be used to deliver signaling information. If there isn't a variation at the mid of a bit period (there is the variation only between two time slots). Otherwise the value, read in a bit period, is the value in which the signal arrives after the variation during a bit period.

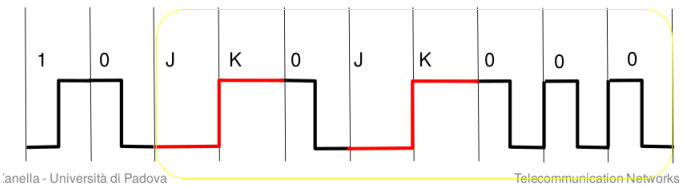


Figura 6.3: Example of bit-encoding.

E.g. **Start of Frame delimiter** $\Rightarrow JK0JK000$ (Figure 6.3), with:

- **J** = no transition in the whole bit period
- **K** = transition at the beginning of the bit period but not in the middle

The main drawback of this method is:
in a bit period, having half period with positive value and half period with negative value, it's useful

because it guarantees a signal with 0 energy. The main problem is that the bandwidth is twice in this period and hence the maximum bitrate is half of bitrate that you can use (Information is inside the bit period with variation in the middle).

6.2 Medium Access Control (DLL sublayer)

Mechanism to manage access to network, because network is usually shared by many users. In shared link, each user can interfere others during the transmission.

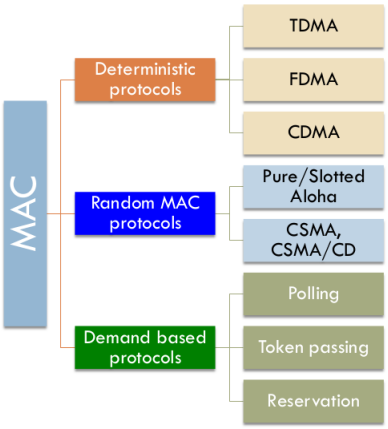


Figura 6.4: Medium Access Control (DLL sublayer).

- **Deterministic protocols**
  - **Time Division Multiple Access (TDMA)**
    Time period is divided in slots and there is an alternate use of channel in time (shared capacity splitted in time).
    Channel bandwidth can be used fully by a single user in its slot (user can use maximum bitrate but you can).
  - **Frequency Division Multiple Access (FDMA)**
    The bandwidth is divided in subchannels and each of them is assigned to a different user (user can use a fraction of maximum bitrate, but you can use it continuously).
  - **Code Division Multiple Access (CDMA)**
    All nodes can use maximum bandwidth in continuous time. To reduce interference, during overlapping of signals, all the signals of each user are coded in orthogonal manner.
    For each user, sending 1, CDMA codes it with a predetermined signature, different for each user ($signature_1$).
    Otherwise, sending 0, CDMA codes it with complementary of its predetermined signature ($signature_0$=$signature_1$*(-1)).
    The signature of each user is orthogonal to other users' signatures.
    For Example:

$Signature_1$	$Signature_0$	Name of user
11-1-1	-1-111	user1
1-11-1	-11-11	user2

If $signature_{0,1}(user1) * signature_{0,1}(user2) = 0$, hence this signatures are orthogonal.
For this reason, the message is multiplied by the $signature_1$ and $signature_0$ of each user and the transmitted messages by other users are deleted (because equal to 0 for orthogonality).

Message =11-1-1 11-1-1 1-11-1 -1-111 1-11-1 -11-11 -11-11 11-1-1
Message*$signature_1(user1)$=11000001
Message*$signature_0(user1)$=00010000
Message*$signature_1(user2)$=00101000
Message*$signature_0(user2)$=00000110

Hence the message, transmitted by user1, is 1101, by user2 is 1100. This is in an ideal situation where there is no noise, otherwise to use this type of code would be harder.

- **Random MAC protocols**
  - **Pure/Slotted Aloha**
    Whenever a station has packets to transmit, it transmits them. If there is a collision, it retransmits them after a random period of time.
    * **Pure Aloha**
      Transmission can start in any time (when a packet is ready to be transmitted).
      It doesn't need synchronization.
    * **Slotted Aloha**
      All stations must be synchronized and transmission can begin only at the beginning of a certain period (**Slot period**).
      When two stations collied, because of synchronization of stations, the two packets overlap perfectly (if they are colliding otherwise there is no collision) so the probability of collision is smaller than the probability of collision of pure aloha.
      The drawback of this protocol is the synchronization time between the stations.
  - **CSMA, CSMA/CD**
    * **CSMA (Carry Send Multiple Access)**
      Before starting the transmission, the stations listen to the channel. If they find channel busy (someone else is transmitting), they reframe some transmissions and try again in a second time.
    * **CSMA/CD (CSMA with collision detection)**
      It's the protocol used by internet. Before transmitting, this protocol works like the previous one. When you start transmitting over the channel, if you're unlucky and another station is transmitting at the same time, hence there is collision.
      The station continues to listen to the channel, if the power that you read from the channel is larger than the power that you are injecting in the channel, you understand that there is another station somewhere that is transmitting.
      Hence the station stops transmitting, because it knows that if it keeps transmitting, it will waste time, there will be a collision and other stations will also waste time waiting for a idle change or sending packets that are going to collide.
      So the station sends a `jam signal`, so that the other stations become aware of the collision too.
      After a random time(`backoff`), with a granularity (slot time) of 51.2 μs (=transmission time of the minimum frame length of 64 Bytes at 10Mbit/s), the frame is retransmitted.
      After **n** collisions in a row, the backoff time is chosen in the range $[0, 2^k - 1]$ slots, with $k = min(n, 10)$.
      After 16 failed (re)transmissions, the process is aborted.
- **Demand based protocols (Control access)**
  - **Polling**
    It asks all the station one by one if there is something to transmit. If a station had something to transmit it transmits otherwise the protocol asks to the next station.
  - **Token passing**
    There is a token that is given to a user in turn for only a while. You can transmit only if you have the token.

- **Reservation**
  A station needs to make a reservation before sending data and then, when it's its turn, it can transmit its packets. There are two periods:
  * **Reservation interval**
    it's divided into M slots and each station has one slot to make a reservation for its transmission (if there are M stations)
  * **Data transmission period**
    after reservation interval, each station knows which stations wish to transmit. Hence, during data transmission period, each station transfers its frame in order of reservation.

WHY NOT DETERMINISTIC IN EVERY CASE BUT RANDOM?
If the traffic is slow, probability collision is low.
In deterministic, we need a center controller. If you have a large population that usually generates small traffic (BURSTY TRAFFIC), in TDMA you lost time WAITING your slot time. Hence it's useful to determine the waiting time in a random-way.

6.3 Batch access problem

A batch access problem is caused by a large number of nodes (Batch), that interfere one to each other transmitting or receiving (Collision). It's difficult silent other nodes because the inquirer doesn't know address of nodes.
MAC protocol assume that data are generated stationary in time. Simultaneous transmissions by multiple nodes result into collision and all packets are lost. In many cases, network has to manage a set of sensors that send packets simultaneously to a center unit(E.g. building, RF tags illuminated by a reader, Wireless nodes that reply to neighbour-discovery request, mobile terminals that compete to reserve a channel slot).

6.3.1 Comparing MAC problems and Batch problems

MAC protocols generally look at the channel contention as a steady-state phenomenon

BRAs address scenarios where contention has a bursty nature.

6.3.2 BRA resolution algorithms

"Batch Resolution Algorithm" (BRA) is used to delete this collision problem. The BRA arbitrates the channel access in order to minimize the batch resolution interval (i.e. the mean time required to resolve all the nodes in the batch).
In each time slot, some nodes are activated and a node that sends its packet without collision is called **resolved**.
The algorithm organizes the nodes in subsets (with random number of elements because we don't know how many there are) and verify if there is a collision or not.
Feedback is returned after each slot:

- **Idle slot**: no nodes transmit
- **Successful slot**: a single node transmit
- **Collided slot**: two or more nodes transmit

The algorithm proceeds activating the subset L and if there is:

- **Collision**
  There is another subset into two subset.
- **No collision**
  The subset is resolved and the algorithm polls other nodes (using a first children order visit).

The algorithm doesn't know how many nodes are in the subset (Figure 6.5). If there is a collision and a child is empty (surely other child ⇒ Collision). So it sends IDLE, for the other child that will have collision and split immediately it with no waste of time.
First, if Algorithm knows (from history of the network) how many nodes usually are in BATCH, it divided

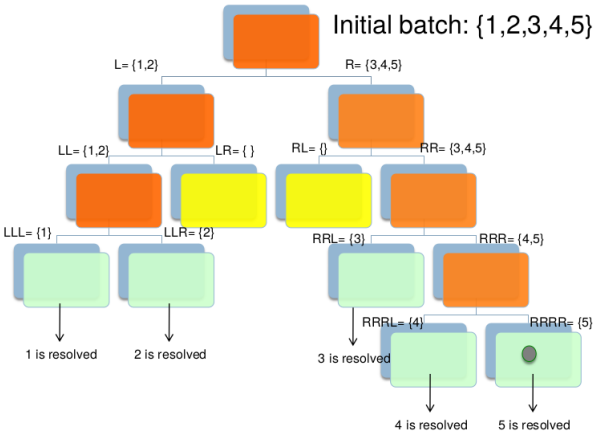


Figura 6.5: Example of application of BRA.

BATCH in subgroups (Figure 6.6).

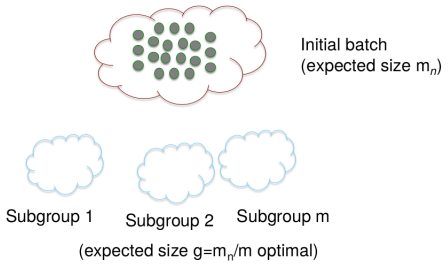


Figura 6.6: Example of organization of batch in subgroups.

6.4 Error control

The physical layer can introduce errors (dropped packets, packets can be lost). Given K redundancy bits, error control can be implemented using two approaches:

- **Forward Error Correction (FEC)**
  it tries to correct bits that can have errors (error correction). FEC can correct up to $\frac{K}{2}$ wrong bits (on ideal line).
  Bit error probability

$$P_{\mathrm{b}} \Rightarrow Prob[\ FEC\ fails] = Prob[\ packet\ contains\ more\ than \frac{K}{2}\ errors]$$

- **Automatic Repeat reQuest (ARQ) in combination with Cyclic Redundancy Code (CRC)**
  it does only error detection and it asks to senders to retransmit packets.

$$Prob[\ CRC fails] = Prob[\ packet\ contains\ more\ than\ K-1\ errors]$$

CRC is better because it hasn't to know the number of wrong packets. CRC is the most common but it isn't the only one.
CRC is usually applied on header because if payload has error, the header is correct (I can understand from header that there was an error on transmission).

Retransmission strategies:

- The PDU of layer N has an error detection code field (generally denoted by CRC – Cyclic Redundancy Code or FEC – Forward Error Correction)
- The DLL module at the receiver side uses the CRC to check the integrity of the received packet
- If the CRC check fails, the receiver discards the packet, while the sender retransmits the packet automatically, after a certain time

6.4.1 Automatic Repeat reQuest (ARQ)

ARQ protocols have to satisfy two requirements:

- **Accuracy**
  Packets have to be delivered to layer N+1 at the receiver side correctly (without any errors) and once and only once (no duplicate packets)
- **Efficiency**
  Capacity loss due to useless retransmissions and time waste, when waiting for packets or ACKs, has to be avoided

The sender waits for a positive feedback (ACK) from the receiver that acknowledges the correct reception of the frame:

- **Positive feedback packet = Acknowledgment (ACK)**
- **Negative feedback packet = Not-Acknowledgment (NACK or NAK)**
  it's not not necessarily sent (implicit NAK if ACK isn't received by sender)

To minimize occupation of line and to improve usage of bandwidth, ACK is sent in packet that must be resent back to sender (Piggybacking=ACK like the tail of a pig).
This guarantees to ACK that it cannot generate a new packet only for it, causing an occupation of the line for no reason.
Hence ACK is embedded in IU that has to be sent back, so its retransmission has to wait the time necessary to receiver B to generate IU and send it (Figure 6.7).

Figura 6.7: Embedding of ACK in IU.

In system with this type of control loop, if ACK is lost, the sender would be waiting for ACK feedback forever (DEADLOCK).
In reality it waits for **ACK time-out** and, if ACK is not received in this period of time, it sends again the packet of which it was waiting ACK.

Another problem can be introduced by the use of time-out:there can be a duplicate packet to the receiver, caused by the lost of an ACK.
The sender retransmitt a packet (already received by receiver) because ACK was lost and pass time-out seconds, hence the sender retransmits packet.

The solution can be assignment of a number (**Sequence Number (SN)**), so the receiver can discard consecutive frames with the same sequence number.

It can be organized in (Figure 6.8):

- **header**
  - **SN: Sequence Number**
    It contains the frame identifier number
  - **Type**
    It specifies whether the frame is a data frame or a control
  - **ACK**
    It contains information of acknowledgement (piggy backing)
- **Info**
  PDU of Layer 3 (Network), SDU of Layer 2 (DLL).
- **CRC (aka Frame Check Sequence, FCS)**
  It contains the Cyclic Redundancy Check computed either on the whole PDU of layer 2 or just on the INFO field (in this latter case the header must contain the Header Checksum Field)

All the yellow parts can be considered as the header of layer 2.

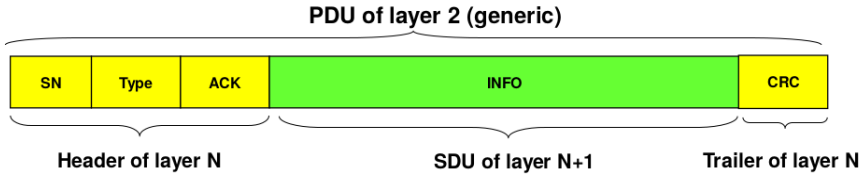


Figura 6.8: General frame structure.

6.4.1.1 Main protocols that uses ARQ

There are different protocols that use ARQ control, the main methods are:

- **Stop & Wait (S&W)**
  Before sending a new frame, the sender has to receive the ACK of the current one.
  In principle, an adequate buffer size at the sender and receiver sides is just one frame, hence a sequence number of just one bit suffices to distinguish new frames from duplicates.
  Backwards:
  If your pipe capacity is large, you lose a lot of resources waiting ACK for an entire Round Trip Time.
- **Go Back N (GBN)**
  It sends multiple packets in pipe (Window of packets) and then waits for all the pipe packets ACKs. Window slides of a position with the arrive of an ACK.
  If there is no ACK, the sender retransmits all the packets from packet with no ACK to the packets at the end of the window.
  The receiver has a buffer of packets in which packets are stored only once, so if ACK is lost and the sender retransmits the series of packet that receiver already has, the receiver drops them.
  For the same reason if a packet, sent by sender, is lost, the receiver doesn't see other packets next to lost packet but it drops them and it sends ACK for lost packet.

  The main problem is that if you transmit less than pipe capacity, you lose resources because receiver and sender are not synchronized.
  If ACK(0) is lost, but sender receives ACK(2), it understands that ACK(0) is lost and shifts the window to the new first packet that has to be sent, that is 2.

- **Selective Repeat (SR)**
  Packets already received are stored and not dropped like in GBN. For each frame correctly received (either new or duplicate), the receiver sends the ACK with SN of the first missing packet in the sequence of received packets (cumulative ACK).
  When a timeout occurs, the sender retransmits the corresponding frame. A different RTO can be kept for each in-flight packet.
  A single RTO can be kept for the whole in-flight window: the RTO is restarted after each retransmission.
  SNs of previous and current windows must not overlap:
  - **Window size** = **N** $\Rightarrow$ Sequence number goes from 0 to 2N-1
  - **Buffer size** = Window size x packet length

Capitolo 7

Addressing

7.1 Regional Internet Registr (RIR)

A Regional Internet Registry (RIR) is an organization that manages the allocation and registration of Internet number resources within a particular region of the world. Internet number resources include IP addresses and autonomous system (AS) numbers.
The main RIRs in the world are five, one for each continent, and they are (Figure 7.1):

- **APNIC**
  Asia Pacific Network Information Centre
- **AFRINIC**
  African Network Information Center
- **ARIN**
  American Registry for Internet Numbers
- **LACNIC**
  Latin America and Caribbean Network Information Centre
- **RIPE NCC**
  Réseaux IP Européens Network Coordination Centre

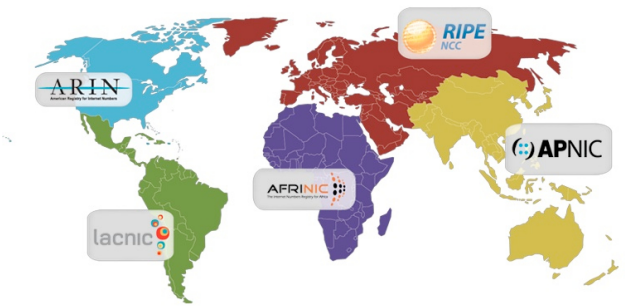


Figura 7.1: Types of RIRs and their area supports.

7.2 Addressing at different layers

- **PHY/MAC address**
  Uniquely identifies the Network Interface Card (NIC) in a local network. Each device is unique at local level but it is usually unique at global level in practice.
  NICs define devices connected to the network: each device MAC address is written on the same line in

which there is its IP address, used to connect device to the network.

Network boards MAC are usually identified by:

- **Exadecimal numbers separated by -**
  Bluetooth: 20-c9-d0-d5-aa-be
- **Exadecimal numbers separated by :**
  WiFi: 20:c9:d0:d5:aa:bd
  Ethernet: a0:4f:d4:4:91:cf

- **DLL (aka Logical Link Control) address**
  Uniquely identifies a logical point-to-point connection between two directly connected hosts. It must be unique for each point-to-point connection and it can be repeated in different connections.
  It's usually quite short (one or two bytes) hence it's more efficient than MAC address.

- **Network layer address (IP address)**
  Uniquely identifies a host in the entire Internet. It must be unique at global level (if we are considering public address).

- **Transport layer address (port numbers)**
  Unique within a host for a given transport protocol. It can be reused in different hosts or in the same host for different transport protocols (that is, TCP and UDP can reuse the same port numbers). It is usually composed of 2 bytes.

  There are many ports that are not yet used and that can be used by programmers to test created network apps.
  The main classification of port numbers is composed by:

  - **Unused:** 0
  - **Well-known ports:** from 1 to 1023
    port numbers reserved for well known applications.
    Examples of pre-assigned ports (TCP well-known port numbers]:
    21: *File Transfer Protocol (FTP)*
    22: *Secure Shell (SSH)*
    23: *Telnet remote login service*
    25: *Simple Mail Transfer Protocol (SMTP)*
    53: *Domain Name System (DNS) service*
    80: *Hypertext Transfer Protocol*

  - **Registered ports:** from 1024 to 49151
    numbers reserved to specific applications, though not officially allocated by ICANN that, nonetheless, lists such numbers to simplify their usage. The Operating System suggests to not use these numbers. UNIX has its specific services assigned to these port numbers in the path `/etc/services`.
  - **Dynamic/Private/Ephemeral ports:** from 49152 to 65535
    numbers that can be freely used by client applications or for nonstandard services

  There is an error if we try to connect to a reserved port number.

- **Application address**
  It is a human readable mnemonic address and it is unique at global level. It's composed of different parts called domains and separated by dots.

  Examples: *URL:* www.unipd.it
  Email: rossi@dei.unipd.it

7.3 Socket

An IP address identifies an host in the internet, a port number (& transport protocol) identifies a process in host. Socket is the union of previous addresses <**IP, port**> and identifies an endpoint of a connection in the Internet (Figure 7.2).

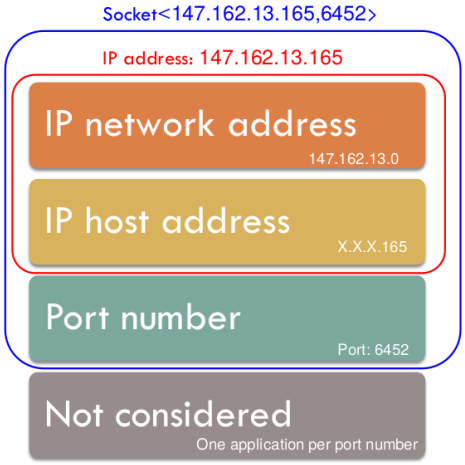


Figura 7.2: Use of socket for identification of nodes.

7.3.1 TCP socket

For connection-oriented transport service, a logical end-to-end connection is established between the sender and the receiver processes and this is full duplex.
Each endpoint is identified by <**IP, port**>. The TCP connection is identified by the sockets of its endpoints, hence this pair of sockets is referred to as

TCP socket: <**IP A, port A**> <**IP B, port B**>

This type of socket is used to identify a flow of information in the Internet.

7.3.2 UDP socket

For connectionless transport service, there is no logical end-to-end connection and sender sends the packet to a receiver. It suffices to specify the receiver socket <**IP, port**>.

7.4 IP and MAC addresses

Forwarding tables specify the MAC address of the next hop, because data needs it to understand where they must go.
In encapsulation, MAC addresses, specified inside the packet, change at each hop (because driver, in which data arrives, change). IP addresses don't change because they identify sender and receiver IP addresses for the protocol of used connection.

UDP socket: <**IP B, port B**>

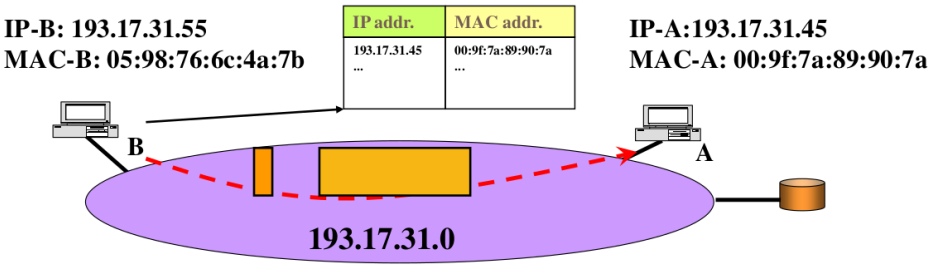


Figura 7.3: Example of forwarding table.

7.5 Address Resolution Protocol (ARP)

It translate IP addresses to their related MAC addresses. It's based on *RFC 826* and on the broadcasting capability of local network. Each host keeps an ARP-cache (to use it, local network must support broadcasting):

- **Store PHY address – NET address associations**
- **Entries are periodically deleted**
  The MAC address could change (from Ethernet to WiFi) but the IP address could remain the same

ARP is based on two main phases (Figure 7.6 and 7.5):

1. If a MAC address for a certain IP is unknown, the host broadcasts an **ARP-request** message that contains the IP address to be "resolved" into a MAC addres (Figure 7.4)
2. Each host compares its own IP address with the required one and, if matching, it replies with an **ARP-reply** message directly to the inquirer (Figure 7.5)

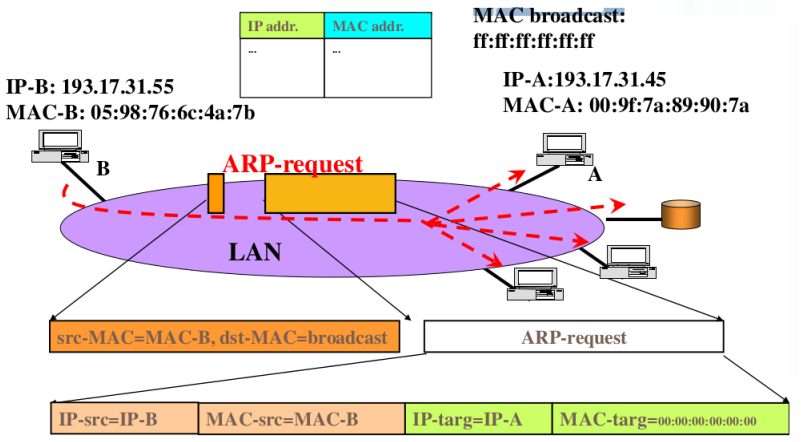


Figura 7.4: 1st phase: ARP-request

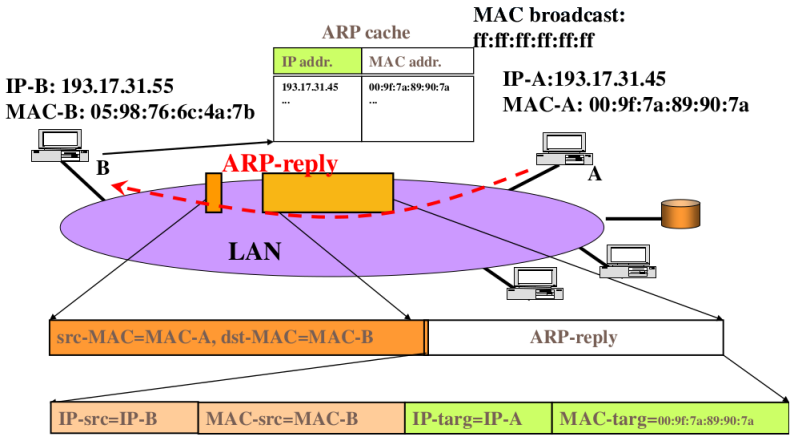


Figura 7.5: 2nd phase: ARP-reply

7.5.1 ARP packet format

A ARP packet is usually composed of two parts (Figure):

- **Frame header**
  its format depends on the network
- **ARP/RARP message**
  it's usually composed of:
  - **Hardware type**
    identifies PHY layer technology
    0x01 (10 Mb Ethernet)
  - **Protocol Type**
    Higher layer protocol using ARP
    IP=0x800
  - **HLEN**
    PHY layer address length
    Ethernet: 6 byte
  - **PLEN**
    NET layer address length
    IPv4: 4 byte (32 bit)
  - **ARP operation**
    type of ARP message
    ARP request = 1 RARP request=3
    ARP reply = 2 RARP reply=4

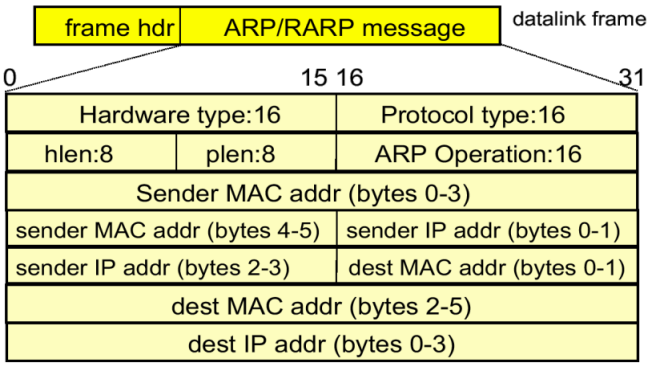


Figura 7.6: ARP packet format.

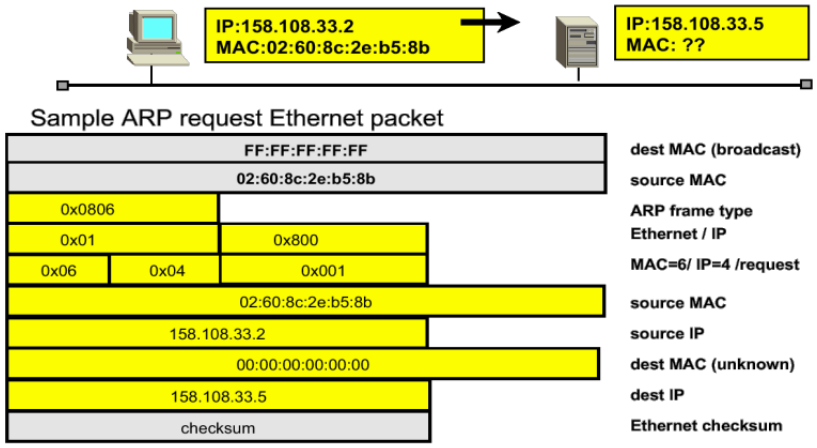


Figura 7.7: Example of ARP request packet.

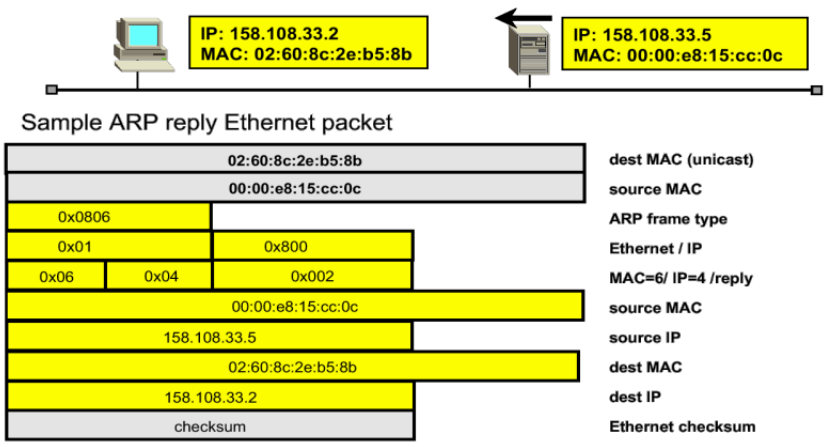


Figura 7.8: Example of ARP reply packet.

7.5.2 ARP uses

- **Gratuitous ARP**
  It's an ARP request for the same IP address of the sending node (ARP Request to itself) but no reply is expected because the node, that sends request, knows what is the MAC address.
  If a reply is received hence there is a IP address conflict (*RFC 5227*) because there is another node in the network with the same IP address of the sender of request.

  Its purpose is to update the ARP cache of other hosts:

  - To update ARP tables in L2 devices (switches) when a host gets connected to a different port (all the switches have a ARP tables that connects IP addresses, of devices connected to it, to their MAC addresses)
  - In case of new MAC/IP bindings (to understand if there is another device with the same IP address that you want to give to new device connected, verifying if there is a *RFC 5227*)
  - To check whether other hosts have the same IP address (verifying if there is a RFC 5227)

- **Unicast ARP request (RFC 1122)**
  Send ARP Request to specific MAC address. It's an optional way that can be implemented by some OS to refresh stale entries in the ARP table. Actively poll the remote host by periodically sending a point-to-point ARP Request to it, and delete the entry if no ARP Reply is received from N successive polls. It's usually used in LAN networks, in which router checks if a device is connected to the network or not (if there is no reply for a certain time, N polls, it supposes that the device is offline). The timeout should be about a minute, and typically N is 2.

- **Proxy ARP**
  A **Broadcast domain** is a set of all hosts that receive broadcast packets sent by any node in the same set. It's important to know that there are some devices that defines broadcast rules:

  - **Level-2 switches**
    transparent to broadcast.
    E.g. A LAN network can be considered as a single broadcast domain, because ARP requests are seen as simple packets and forward to the hosts that are waiting for it (all the devices).
  - **Level-3 routers**
    do not propagate broadcast packets. E.g. A LAN, partitioned in multiple subnetworks interconnected by routers, contains multiple separate broadcast domains

  With Level-3 routers, an ARP request cannot be sent to IP of an internal subnet. Hence **Proxy ARP** is implemented in each router of an internal network. A **Proxy ARP** "impersonates" all hosts in the other subnets (Figure 7.9):

  - Implemented in the router that interconnects the different subnetworks
  - Intercepts all ARP requests for IP addresses of other subnets and replies with its own MAC address
  - Forward all frames with its MAC address to the proper subnetwork

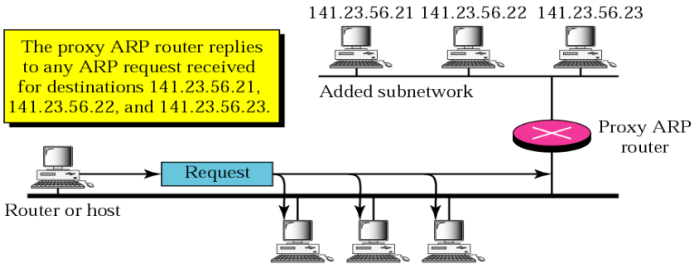


Figura 7.9: Forwarding

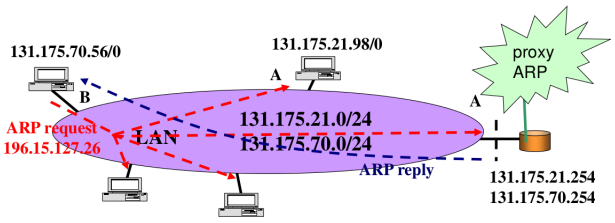


Figura 7.10: Example of ARP reply made by proxy.

An ARP-request is generated for any IP address. If the receiver is in the local broadcast domain, the node replies with ARP-reply (Figure 7.10). Otherwise, if the destination is outside the LAN, the proxy ARP replies with its own MAC and takes care of forwarding the packets to the Internet.
All packets that will be sent to the IP address, specified in ARP request, will be sent to proxy (indeed the device has only proxy MAC address).
Routing table is required only at the Proxy ARP, that will manage next packets sent to it and forward them to right IP address.

- **Reverse ARP (RARP)**
  maps MAC address to IP address. It was originally proposed to support initialization of hosts that need to get an IP address after booting.

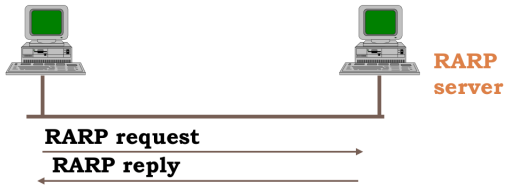


Figura 7.11: Example of network with RARP server.

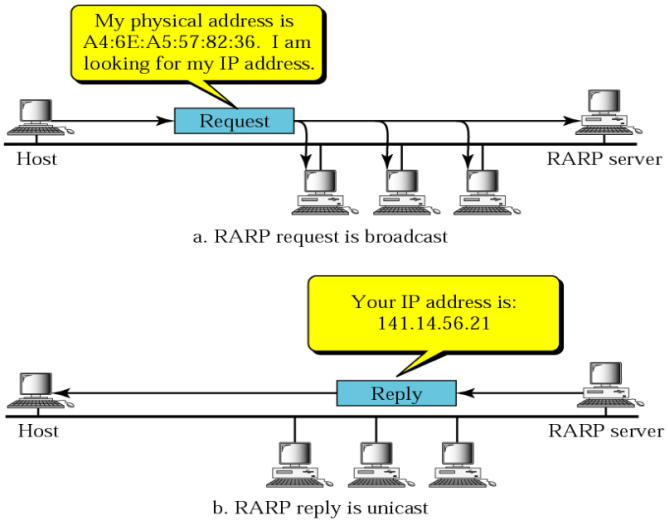


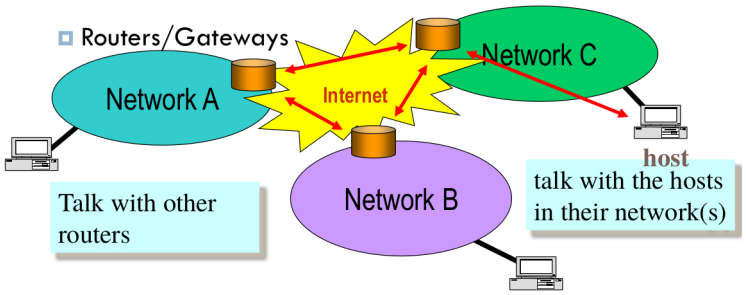
Figura 7.12: Example of application of RARP request and RARP reply.

Packet format used is the same used by ARP. It's an obsolete approach, replaced now by BOOTP and DHCP. (**Note:** *MAC addresses change at each hop, IP addresses don't change*)

Capitolo 8

Internet protocol

LAN are realized with specific technologies (E.g. Ethernet, WiFi, ZigBee). The interconnection of different networks requires dedicated devices (Routers or Gateways). The aim is to define a unique (virtual) network upon heterogeneous hardware platforms, using a global addressing architecture (IP addressing).

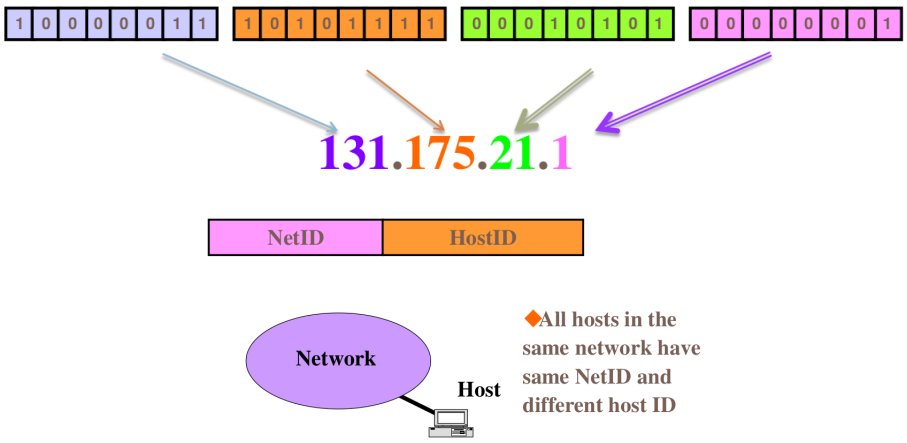


8.1 IP: basic requirements and services

- **Basic requirements**
  - IP relies upon basic network services provided by LANs
  - Local addressing capabilities (delivery using MAC address in the LAN)
  - Local data transfer (even not reliable)
  - Possibility of sending packets in broadcast over the LAN
- **Basic services**
  - Provide a universal addressing plan
  - "Datagram" transfer over the Internet (Best effort delivery service)
  - Provides packet fragmentation, if required (Datagram is larger than the Maximum Transfer Unit (MTU) allowed by the local link technology)
  - Reassemble fragments at the receiving side
  - Management of error and control messages by means of subsidiary protocol (ICMP)

8.2 IP address

IPv4 is composed of 32 bits, grouped in octets, represented as decimal number (from 0 to 255) separated by dots.
Every IP address can be divided into two address:

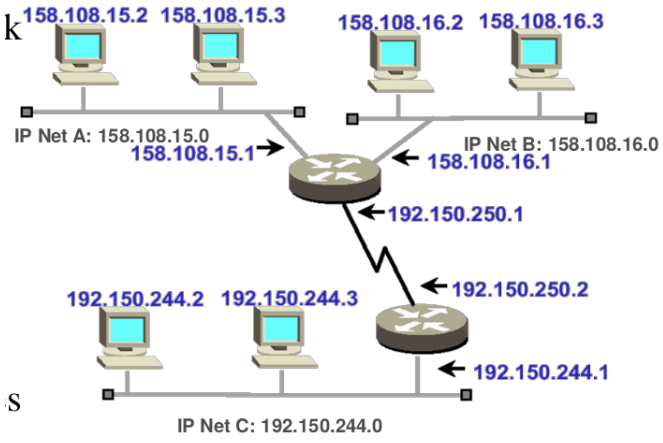


- **IP Network address (NetID)**
  most significant bits that uniquely identify a network in the Internet
- **IP host address (HostID)**
  least significant bits that uniquely identify a host in a network

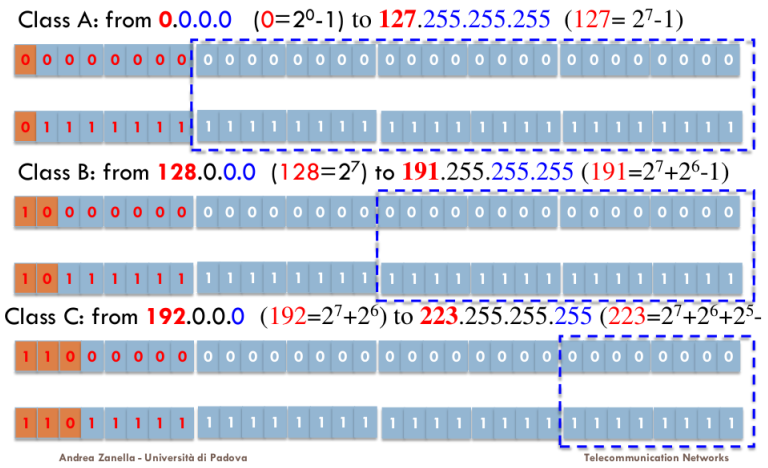
All hosts in the same network have same NetID and different host ID.
The particularity of this mask is that it's composed of ones and zeros. The ones are only together in the left part (more significant) of the number that identifies mask.
IPv6 addresses are not compatible with IPv4 addresses because they have a different design.

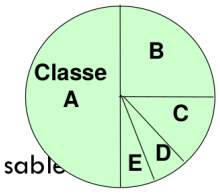
8.2.1 Multi-homed nodes

An IP address identifies a network interface. Multiple IP addresses are assigned to hosts with multiple network interfaces (these hosts are called **multi-homed nodes**).
Routers are always multi-homed but not all multi-homed devices are routers.



8.3 Classful addressing





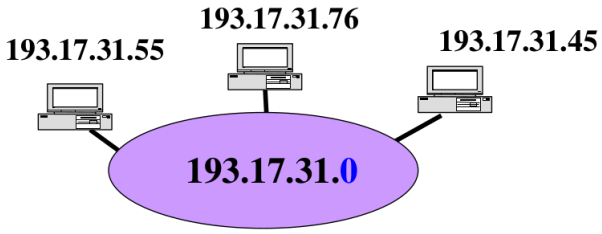
Class	Leading bits	Size of *network number* bit field	Size of *rest* bit field	Number of networks	Addresses per network	Total addresses in class	Start address	End address
Class A	0	8	24	128 (2^7)	16,777,216 (2^{24})	2,147,483,648 (2^{31})	0.0.0.0	127.255.255.255
Class B	10	16	16	16,384 (2^{14})	65,536 (2^{16})	1,073,741,824 (2^{30})	128.0.0.0	191.255.255.255
Class C	110	24	8	2,097,152 (2^{21})	256 (2^8)	536,870,912 (2^{29})	192.0.0.0	223.255.255.255
Class D (multicast)	1110	not defined	not defined	not defined	not defined	268,435,456 (2^{28})	224.0.0.0	239.255.255.255
Class E (reserved)	1111	not defined	not defined	not defined	not defined	268,435,456 (2^{28})	240.0.0.0	255.255.255.255

8.3.1 Special addresses

- **Network address**

  HostID all zeros

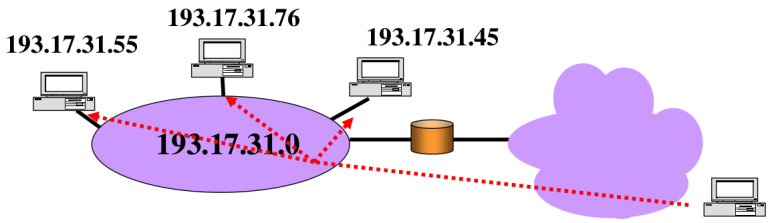
  It identifies the network with NetID (it's only used in the routing table)



- **Direct broadcast**

  HostID all ones

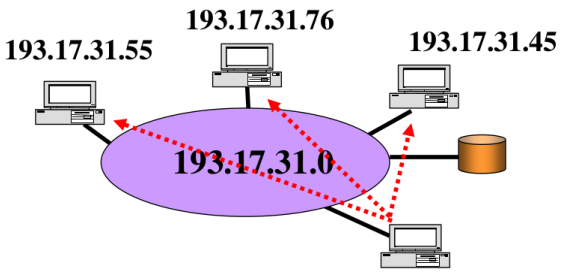
  It identifies all nodes in the network with NetID (broadcast to all nodes in the network $NetID|0...0$). A packet can be generated from outside, but it cannot go outside.



- **Limited broadcast**

  NetID and HostId all ones
  255.255.255.255

  It identifies all nodes in the network where packets was sent. Packet cannot cross network border (in either direction).



- **This Host in This Network**

  0.0.0.0

- **That Host in This Network**

  NetID all zeros

- **Private IP addresses**
  Reserved for use in private networks (not connected with other networks)

Class		From - to	Blocks
A	**[00001010]**.x.x.x	10.**0.0.0** - 10.**255.255.255**	1
B	**[10101100]**.[0001xxxx]	172.**16.0.0** - 172.**31.255.255**	16
C	**[11000000]**.[10101100].x.x	192.168.**0.0** - 192.168.**255.255**	256

These addresses can be seen only in Border Network and not in Core Network. Packets, transferred in Internet, don't have these addresses because gateways don't use them.

Special addresses	Netid	Hostid	Sorg/Dest	Class
Network address	Specific	All 0s	Dest	As for NetId
Direct Broadcast	Specific	All 1s	Dest	As for NetId
Limited Broadcast	All 1s	All 1s	Dest	E
This Host in this network	All 0s	All 0s	Source	A
Specific host in this network	All 0s	Specific	Dest	A
Loopback	127	Any	Dest	A

8.4 Classless addressing

Classfull address paradigm is not efficient because large portions of addresses are unusable and a company with a lot of host cannot have enough addresses to connect all the devices. Hence the **classless address architecture** has been proposed and implemented. The basic idea of this organization is based on two phases:

- **Subnetting**
  split large blocks in smaller ones
- **Supernetting**
  aggregate smaller blocks in larger ones

In other words, it *makes possible to freely place the boundary between NETid and HOSTid in the 32 bit IP address field*

8.4.1 Netmask

Netmask is a 32-bit vector with the form [1111...11000...0] in which 1s and 0s are never interleaved.
The number of 1s in the netmask identifies the number of bits in the IP address to be considered as NetworkAddress.
Hence,

$$NetworkAddress = IPaddress \ \wedge \ Netmask \ HostAddress = IPaddress \ \wedge \ \overline{Netmask}$$

The number of bit equal to 1, are usually specified in IP address with $/\#_{\text{bits}=1}$ (**Classless Inter Domain routing – CIDR,** *read "cider"*). For example:

IP address	#bit of Network	Netmask	IDs	
128.5.5.1/**16**	16	255.255.0.0	NetID	128.5.5.1
			HostID	0.0.5.1

8.4.2 Block allocation

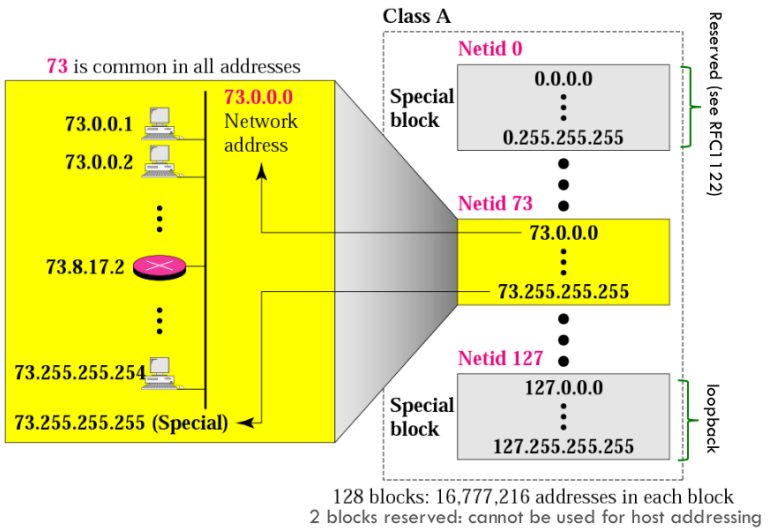
Global addresses (both IP and symbolic names) are assigned in a centralized manner by **Internet Corporation for Assigned Names and Numbers (ICANN)** that:

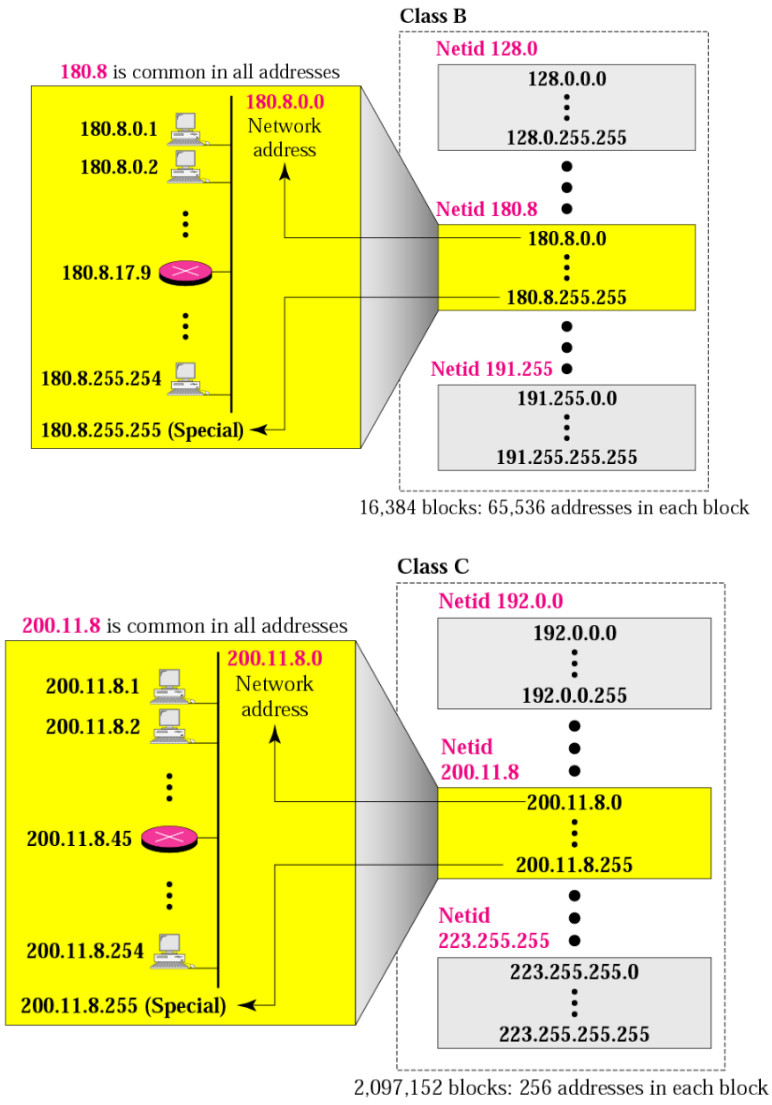
- allocates addresses
- manages DNS
- assigns domain names, resolves disputes

The number of addresses in a block, M, is always a power of 2

Net-ID length: $n = 32 - log2(M)$ e Host-ID length: $m = log2(M)$

The first address needs to be a multiple of the number M of addresses in the block and the number of addresses in the block is always a power of 2 (i.e. $M = 2^m$).
The NetAddress of the block, hence, will have a suffix of m zeros(it's a multiple of 2^m).





8.4.3 Subnetting

Subnetting divides big networks in smaller parts, each of them becomes an "independent network" in all respects. The main reasons of this organization of the network are:

- To Simplify management
- To Isolate malfunctioning
- Each subnet is a new broadcast domain ⇒ lower control traffic in each subnet
- It is possible to give access to a host only into a specific subnet ⇒ differentiated access permit

8.4.4 Subnet mask

HostID is obtained by bitwise multiplication (AND) of IP address and netmask.

To determine the subnets, you need to detach each interface from its router, creating islands of isolated networks. Each isolated network is called **subnet** (Figure 8.1 and 8.2).

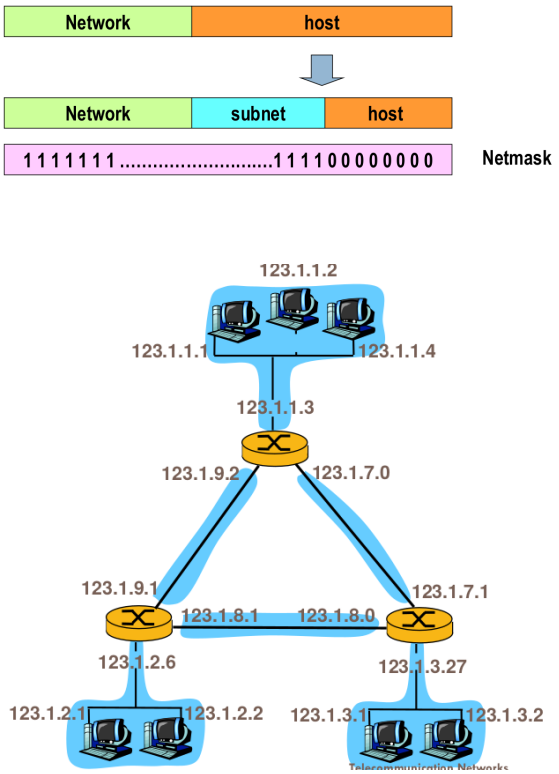


Figura 8.1: Example of subnets of a network.

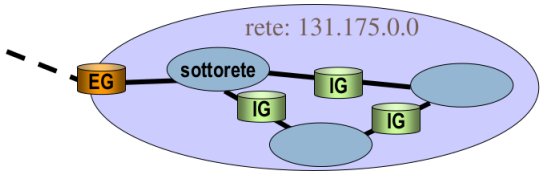
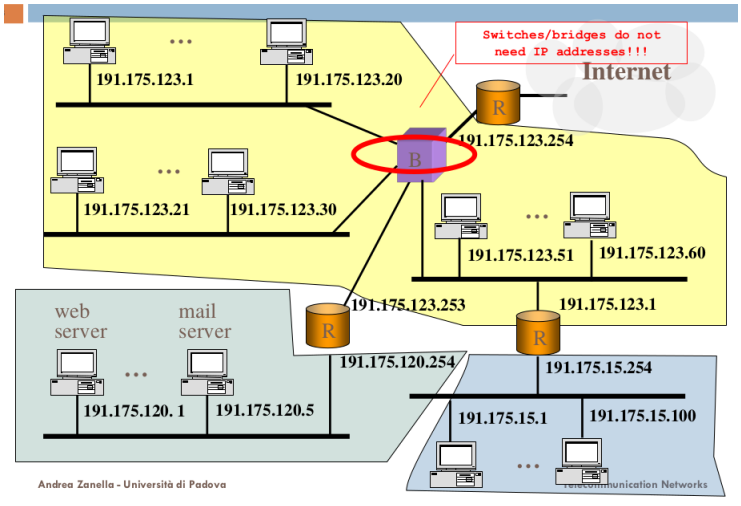


Figura 8.2: Example of subnets and how they interact each other.



Usually, subnetting is only known by routers within the network, while external routers will see only the (common) network addresses of all subnetworks (131.175.0.0). Internal routers need to manage subnets by using NetMasks.

Example
An ISP gives us a Class B address:

$$150.100.0.0/16 \Rightarrow NetMask\_original\_size = 16$$

All the possible addresses for my network hosts are from 200.100.0.0 to 200.100.255.255.
I want to connect 400 host addresses to the network. So I need to use:

$m = \lceil log_2 400 \rceil$ *bits* for the hostID
$NetID\_size = 23$ *bits* in my local network

It's important that NetIDsize≥NetMask_original_size because NetID from ISP is considered of NetIDsize bits.
$Netmask$: 255.255.254.0 → /23

The address of the host in my network are from 150.100.0.0 → 150.100.1.255.
Network Reserved address ⇒ 150.100.0.0/23
Broadcast reserved address ⇒ 150.100.1.255
$NetID\_size - NetMask_original_size = SubnetMask\_size$

8.5 Supernetting

It consists on merging together blocks of adiacent addresses (with the same prefix bits)

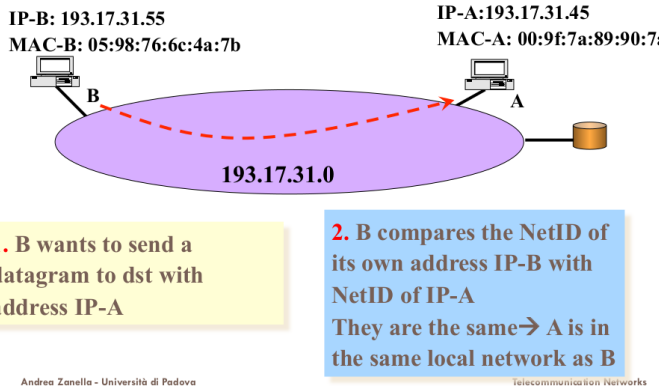
Example
Given 4 Class-C blocks can be organized into a single Supernet network choosing common NetID bits.

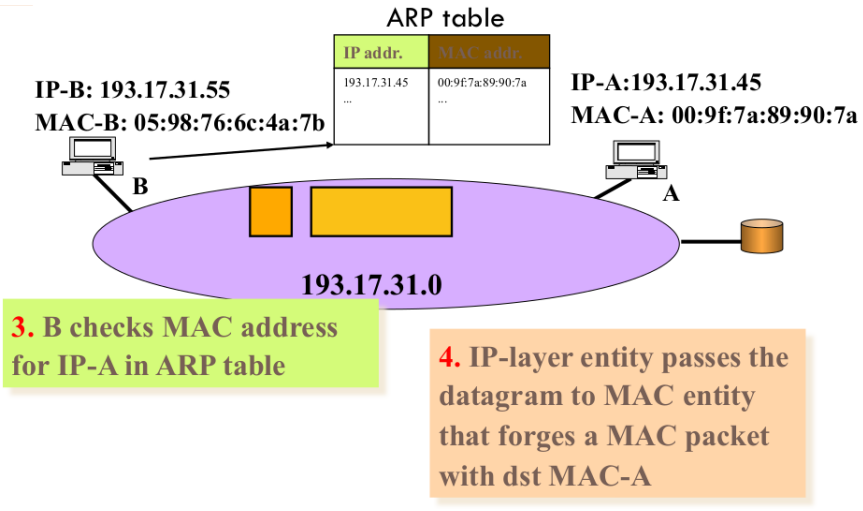
193.0.0.0/24
193.0.1.0/24
193.0.2.0/24
193.0.3.0/24
⇒ 193.0.0.0/22

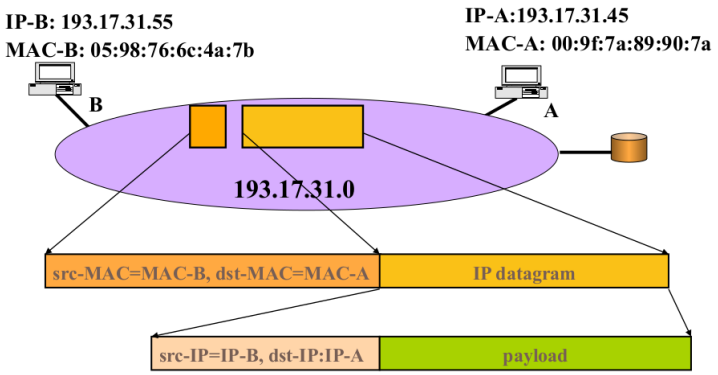
Network Reserved address ⇒ **193.0.0.0**
Broadcast reserved address ⇒ **193.0.3.255**

8.6 Direct forwarding

Destination of the packet is in the local network of the forwarder (**directly reachable**).

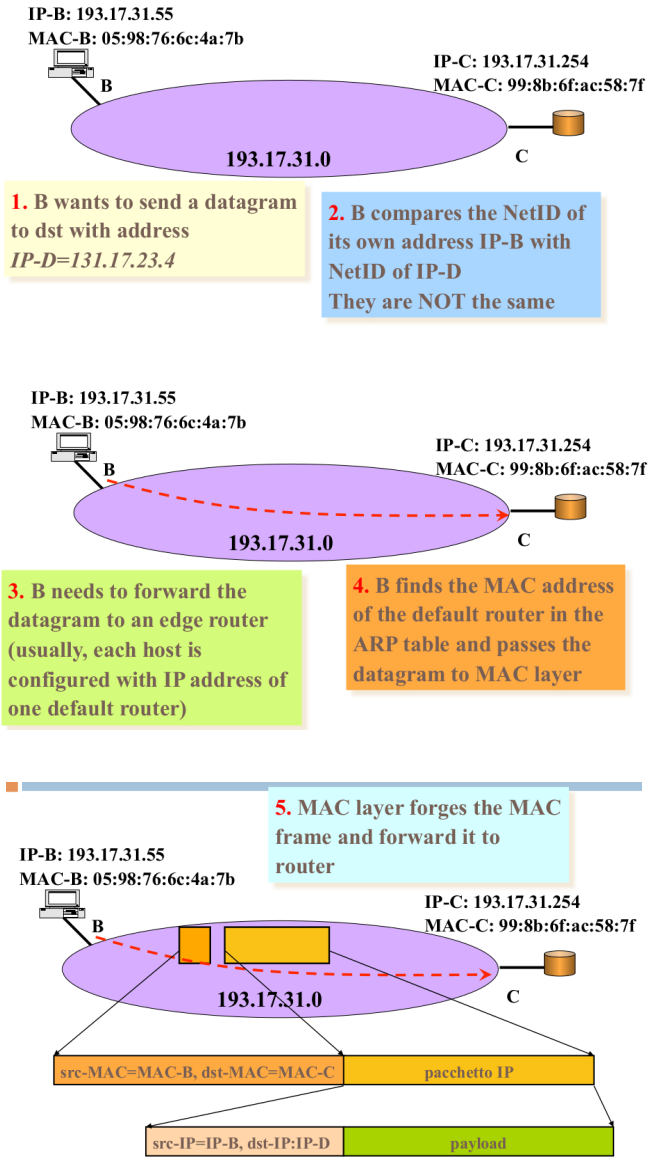

IP-B: 193.17.31.55
MAC-B: 05:98:76:6c:4a:7b
IP-A:193.17.31.45
MAC-A: 00:9f:7a:89:90:7a
B
A
193.17.31.0
1. B wants to send a datagram to dst with address IP-A
2. B compares the NetID of its own address IP-B with NetID of IP-A
They are the same→ A is in the same local network as B
Andrea Zanella - Università di Padova
Telecommunication Networks


ARP table
IP addr.
MAC addr.
193.17.31.45
00:9f:7a:89:90:7a
IP-B: 193.17.31.55
MAC-B: 05:98:76:6c:4a:7b
IP-A:193.17.31.45
MAC-A: 00:9f:7a:89:90:7a
B
A
193.17.31.0
3. B checks MAC address for IP-A in ARP table
4. IP-layer entity passes the datagram to MAC entity that forges a MAC packet with dst MAC-A


IP-B: 193.17.31.55
MAC-B: 05:98:76:6c:4a:7b
IP-A:193.17.31.45
MAC-A: 00:9f:7a:89:90:7a
B
A
193.17.31.0
src-MAC=MAC-B, dst-MAC=MAC-A
IP datagram
src-IP=IP-B, dst-IP:IP-A
payload

8.7 Indirect forwarding

Destination is on another network. Packet needs to go through at least another hop, specified in the **routing table**.



8.8 Router forwarding

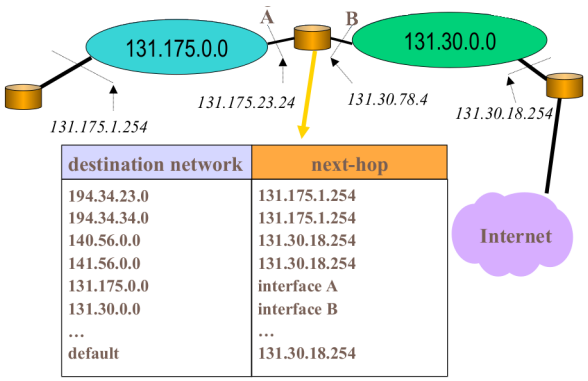
Routers as well may perform direct or indirect forwarding but they usually have more interfaces (they can perform direct forwarding on more local networks).
Every router has a routing table, that is a table, maintained by all the hosts in the Internet, that contains all the addresses reachable with one hop.
Routing table, in each router, contains for each possible destination network the address of the next-hop router.

8.8.1 Routing table with classfull addresses

Destination network has a fixed number of bits that identifies NetID. For each packet that arrives to the router, the packet is forwarded to the **next-hop** of the line, that matches its **destination network** with destination network address of the packet.

- **Direct forwading** ⇒ output port of the router that contains this routing table

- **Indirect forwarding** ⇒ IP address of next-hop router



destination network	next-hop
194.34.23.0	131.175.1.254
194.34.34.0	131.175.1.254
140.56.0.0	131.30.18.254
141.56.0.0	131.30.18.254
131.175.0.0	interface A
131.30.0.0	interface B
...	...
default	131.30.18.254

8.8.2 Routing table with netmask

The definition of which is next-hop is based on two phases:

1. **Configuration of Network interfaces**
   to configure a network interface we need both IP address and Netmask of the network to which the interface is connected.

Interface settings
(for direct forwarding)

Interface	eth0
IP address	131.17.123.1
Netmask	255.255.255.0

Interface	eth1
IP address	131.17.78.1
Netmask	255.255.255.0

Interface	eth2
IP address	131.17.15.12
Netmask	255.255.255.0

Figura 8.3: Interface settings (for direct forwarding).

2. **Configuration of routing tables**
   to establish which is the next hop, you need to know also what is the NetMask of a network to find it (because each network can have a different NetMask)

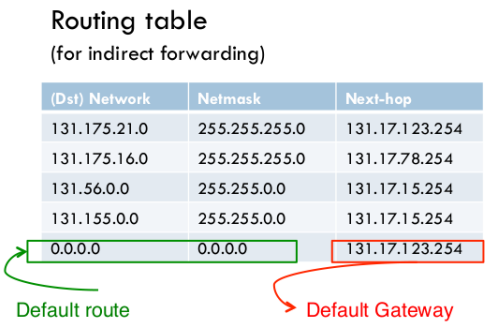
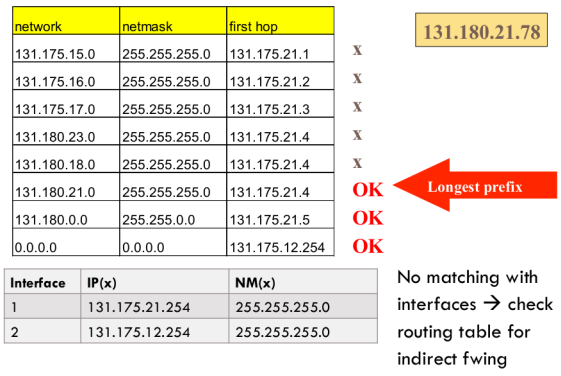


Figura 8.4: Routing table (for indirect forwarding).

When the router receives a packet, that has to be sent to a **Dest_IP_address**, the mask is applied to it:

1. **Control of direct forwarding**
   - **if** $Destination\_Network\_address = Dest\_IP\_address\ AND\ Netmask(X)$ **is equal to** $Interface\_Network\_IP\_address(X)\ AND\ Netmask(X)$
     then the packet is directly forwarded to the interface with matched settings
   - **else**
     the same operation is applied to the next interface, using its settings
2. **Control of indirect forwarding**
   if there aren't matches with the settings of the interfaces, specified in direct forwarding, we control routing table of the router:
   the packet is forwarded to the Next-hop of the line with settings such that the $Destination\_Network\_address = Dest\_IP\_address \wedge Netmask(X)$ is equal to $Interface\_Network\_address(X) = IP\_address(X) \wedge Netmask(X)$ and longest Netmask matching (more specific network address between them specified in the routing table).
   The network with *Network_Destination*=**0.0.0.0** and *Netmask*=**0.0.0.0** is called **Default route** because it matches with every possible network.
   This network is usually used to specify the **Default Gateway** as next-hop to have access to Internet in fact it represents the shortest Netmask that matches with a possible IP_address.

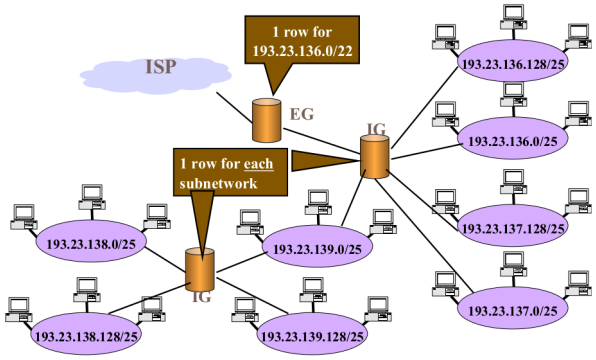


8.9 Stub and Transit networks

ISP gives you a Network address and a Netmask for you network, watching the number of hosts you want to connect. Internet is organized in subnetworks, connected between them by **gateways**. A network is divided into two sets of networks:

- **Stub networks**
  set of edge networks that are the leaves of the network

- **Transit networks**
  set of networks that guarantee the connection between multiple networks



8.10 Dynamic addressing

It's used when there are more hosts than available IP addresses and it's useful when hosts do not require permanent IP addresses (E.g. users in WLANs, exploit Statistical multiplexing of addresses,...).
Addresses are leased to a host for a limited time, after which the host must return it (or renew the lease). When all available addresses are assigned, new requests for an dynamic IP address are rejected.

Capitolo 9

Private networks

There are two types of private networks:

- **Intranet**
  private network that makes use of TCP/IP protocol stack (www, mail server, routers,...). It may or may not be connected to the Internet and it's never crossed by "external" traffic (traffic between end-points that are both external).
  Internal hosts may:
  - not require Internet access
  - require limited Internet access
  - require full Internet access
- **Extranet**
  private network that can provide some services that can be accessed by a set of enabled external users (e.g., on-line ordering of customers,...)

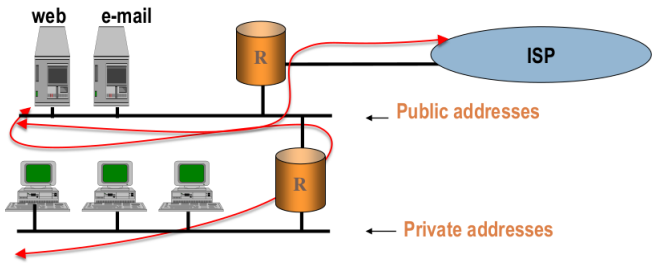
9.1 Private addressing

Hosts in Intranet are typically assigned to private IP addresses. Servers may or may not be assigned to public IP addresses.
Different Intranets can reuse same private IP addresses. Packets with private IP addresses are dropped by edge routers because they are never forwarded across the public Internet.
The main solutions to provide public services to a Intranet:

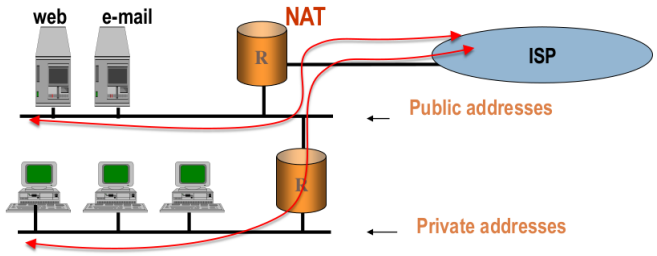
class A	10.xx.xx.xx	16 milions of addresses
class B	from 172.16.0.0 to 172.31.255.255	16 networks of 65536 addresses
class C	192.168.xx.xx	256 networks of 254 hosts

Tabella 9.1: RFC 1597, Address Allocation for Private Internets

- **First Solution**
  the solution consists on giving public addresses to server and private addresses to hosts in the Intranet. The hosts send packets to servers that are connected online and forward them.



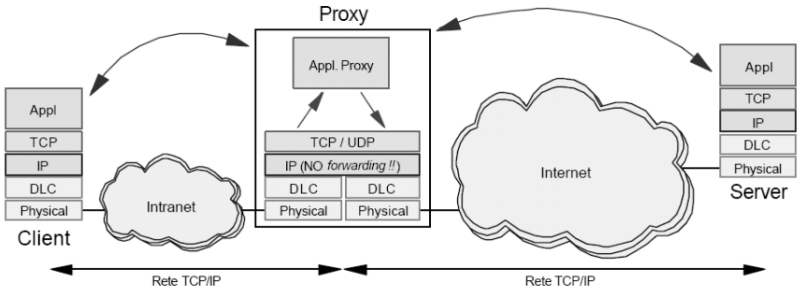
- **Proxy**
- **Traditional NAT**

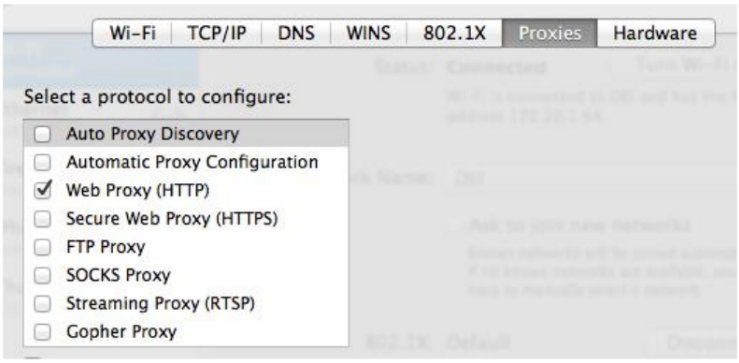


9.2 Proxy

Proxy splits the connection between private and public networks. All requests are addressed to the proxy that forwards it by using its own public IP address. The communication is actually obtained through two application layer connections.

Backward: this solution requires a proxy for each application.





For example, when you want to download from Google Scholar, you should often pay for it. If you download the proxy of UNIPD library (**Proxy Docile**) and you activate it, your request, for an article of Google Scholar, will be sent to server of UNIPD, that splits the connection and impersonates your device. If UNIPD has paid for the article, you can download it through UNIPD server as UNIPD client.

9.3 Network Address Translation (NAT)

A NAT performs as standard routers but in addition, they can map private to public addresses. To a NAT is assigned a pool of public addresses to be used to forward messages through the public Internet. If all the possible public addresses are already assigned to some private client IP, other requests of public IP addresses are blocked.
In order to permit two-ways communication, the mapping must be reversible and the association between public and private addresses is written in the NAT table.
There can be two types of correspondence:

- **Static correspondence**
  a MAC is always associated to the same IP address
- **Dynamic correspondence**
  a MAC is assigned to an available public IP address upon request

There are three types of NAT: Dynamic association, Traditional NAT (Outbound NAT), Bi-Directional NAT. Shared characteristics between them are:

- **Transparent Address Translation**
- **Transparent Routing**
  Routing shall be unaware of address translation
- **ICMP Packet Translation**
  ICMP messages carry IP addresses in their body and these addresses shall also be translated

9.3.1 Dynamic association

Dynamic binding is based on the concept of "session":

- Depends on which higher layer protocol generated the data
- **TCP & UDP** session is identified by the socket address
- **ICMP** 3-tuple (source IP, destination IP, Identifier)

When intercepting the first packet from a host, the NAT creates a session and binds the private address of the host with a public address. When the session is over, the binding is broken and the public address is released into the pool.
Once defined the session, NAT needs to find its start/end:

- **Session start**
  - *TCP:*SYN segment
  - *UDP, ICMP:*connectionless → different methods
- **Session end**
  - *TCP:*FIN segments on both sides

  Time-out is always used to avoid deadlock in case the protocol syntax is violated

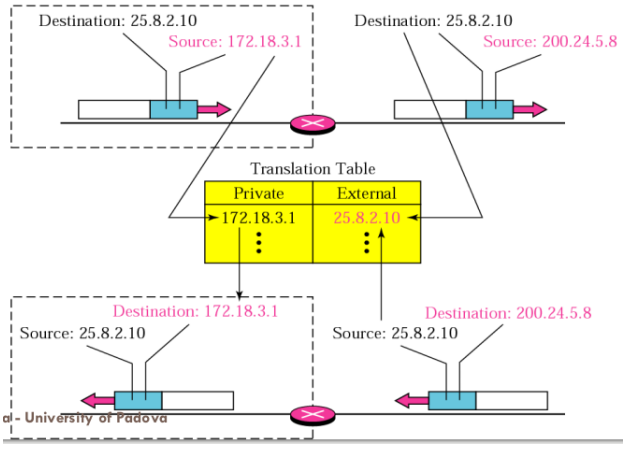
9.3.2 Traditional NAT (Outbound NAT)

It only supports connections starting from the Intranet toward the Internet (outbound).
Routing information, regarding the outer network, can be distributed within the Intranet, but the contrary is not possible.
Traditional NAT is composed of 2 subclasses:

- Basic NAT
- NAPT (Network Address and Port Translator)

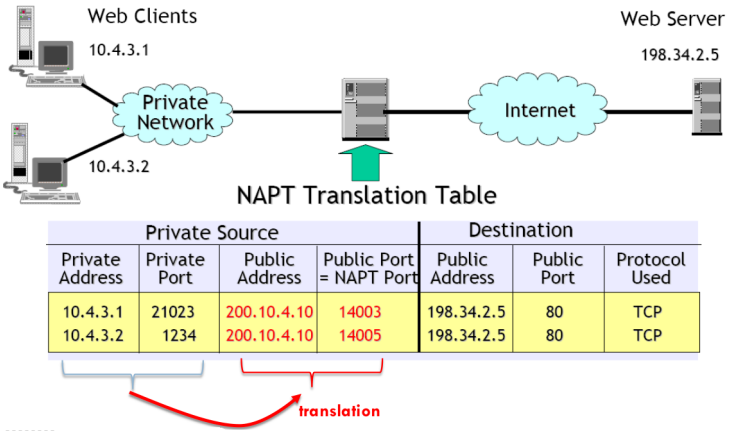
9.3.2.1 Basic NAT

It translates only the private IP address of the host into a public IP address.
There is a one-to-one correspondence between private and public addresses during a session (a public address can be used by a single host during one session).
If the number of public addresses is lower than that of hosts, some sessions may not be started. It cannot support multiple sessions with the same destination IP address, because when the NAT receives the reply for a request, it doesn't understand which device sends that request.



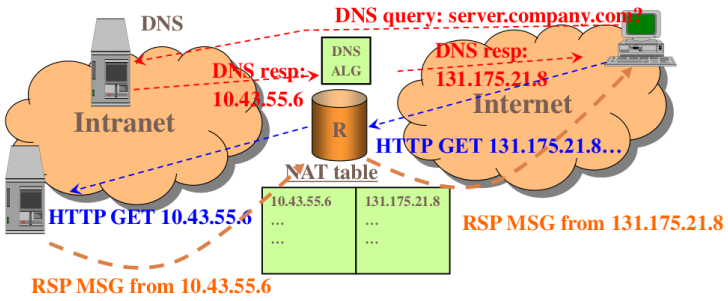
9.3.2.2 NAPT (Network Address and Port Translator)

The private socket <**Private IP, Source port**> must be translated to public socket <**Public IP, NAT source port**>. *NAT source port* is randomly chosen among the still available ephemeral ports of the NAT (probability to choose the same port is very small). The same public IP can be refused for multiple sessions, even with the same external host.
It needs to inspect the packet till layer 4 header because the port is defined at Transport Layer.



9.3.3 Bi-Directional NAT

It supports both internal and external session initiation. This type of NAT solves the problem in which ainiziazione host in the public Internet wants to start a connection with a host in the private Intranet, without knowing the public address to reach it.
The solution of this problem can be found with a local DNS inside the Intranet. The client sends a DNS request to DNS inside Intranet and this replies with a DNS reply that indicates the private IP address of the destination address in the Intranet.
Then a DNS algorithm translates it to its public IP address and updates routing table of the Intranet.
Hence the client sends its packets to the public IP address previously found, and a NAT translates it, watching NAT table, to its private IP address and then the host replies to that message in a complementary way.



9.3.4 Final remarks

Address translation isn't cost free:

- Header Checksum must be recomputed
- ICMP fields, with IP addresses, must be replaced and header checksum recomputed
- Checksum, in TCP or UDP, must be recomputed with new pseudo-header

The recomputing is given by the continuous modification of IP address in packets, during the transfer.

Capitolo 10

BOOTP & DHCP

RARP is useless because netmask, DNS address and Gateway address are not specified by the RARP server. So RARP protocol was replaced by BOOTP and DHCP protocols.

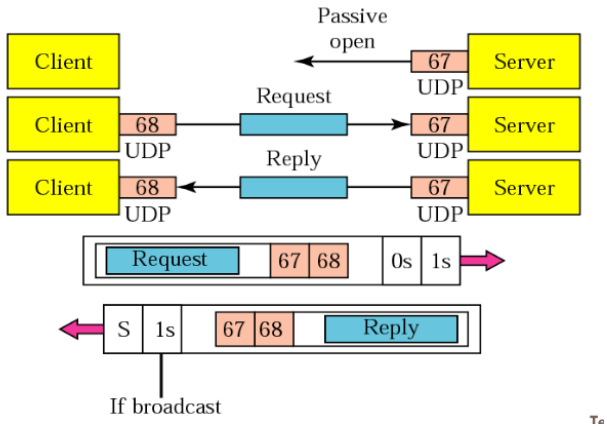
10.1 Bootstrap protocol (BOOTP)

It's based on a Client/Server protocol at application layer (*RFC951*). Client and server can be on different networks, so this requires a relay agent, that knows IP address of BOOTP server.
The connection is usually organized in this way:

- **Server listens on well-known port 67**
  clients must know beforehand where "to knock" to get a specific service

- **Client uses well-known port number 68**
  client typically uses ephemeral port numbers but in this case, it uses well-known port number, because client doesn't know its IP address.
  Hence the server replies with broadcast packet in which it's specified the port number of the client, so we are sure that the client with that port number is a BOOTP client listening for the IP address

If there are more than one request, all clients need to specify a unique identifier (Transition ID: it is usually a random number so the probability of collision is very low) to understand what the BOOTP reply is, not confusing its IP address with other IP addresses.



10.1.1 BOOTP packet format

1. **First part**

   - **Code**
     request/reply
   - **HW Type**
     identify the DLL protocol
   - **Hop count**
     max number of hops to reach BOOTP server
   - **Seconds**
     filled in by client, seconds since client started to boot
   - **Client IP**
     IP address of the requesting node (0.0.0.0 if unknown)
   - **Your IP**
     proposed IP address to be assigned to client in BOOTP-reply (0.0.0.0 in BOOTP-request because client doesn't know yet its IP address)

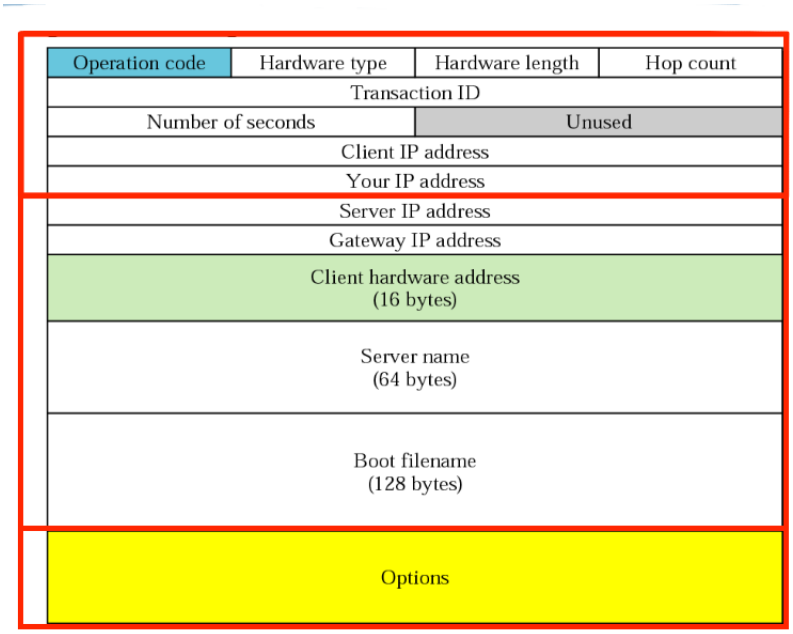
2. **Second part**

   - **Server IP address**
     255.255.255.255 in BOOTP-request (Broadcast request from client).
     IP address of BOOTP server in reply (or if Server isn't in local network)
   - **Gateway IP address**
     Set by 0.0.0.0 by client
     Set to IP address of BOOTP Relay in case of relay (if Gateway isn't in local network)
   - **Client HW address**
     MAC address of requesting client
   - **Server name (optional)**
   - **Boot filename**
     Path of configuration file containing all required info (netmask, DNS IP address,...)

3. **Options**
   they are not based on some standard, because they are optional. They were initially left to vendor-specific definition. Successively (*RFC 1048*) they were extended to support vendor-independent options with standard:

   - **Type-Length-Value (TLV) format**
     If TLV format is used, then the option field must start with the magic cookie

0x63825363	99.130.83.99



10.1.2 Limits of BOOTP

BOOTP only admits static addressing

- The BOOTP server contains a table with MAC addresses and IP addresses to be assigned to each MAC
- When a BOOTP request arrives, the server looks up the table and returns the corresponding IP address

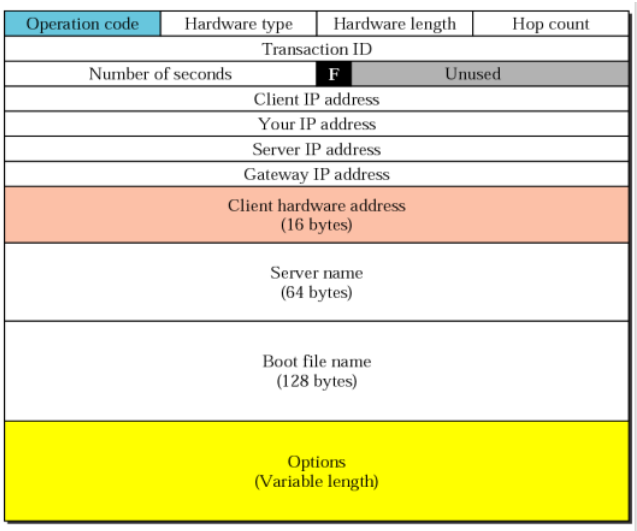
10.2 Dynamic Host Configuration Protocol (DHCP)

It's called also *RFC 2131* and it's based on application with client-server architecture. It's largely based on BOOTP but designed **to support dynamic address assignment**. DHCP messages are carried by UDP protocol and each message, sent by the client up to IP assignment, has:

- **Source IP address:** 0.0.0.0
- **Dest IP address:** 255.255.255.255
- **Source port:** 68
- **Dest port:** 67

10.2.1 DHCP packet format

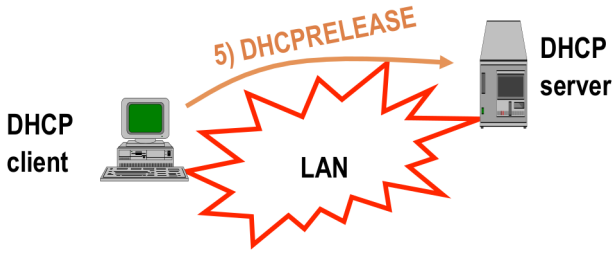
It's the same packet format of BOOTP but it has fixed **FLAG** that forces reply in broadcast.



10.2.2 Phases of DHCP

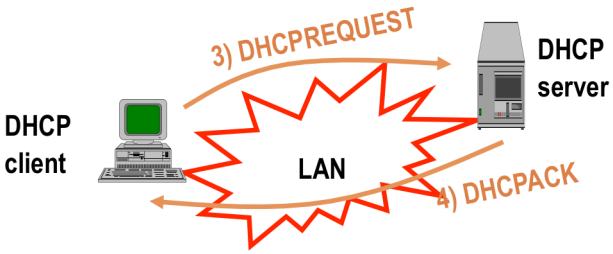
1. **DHCPDISCOVER & DHCPOFFER**

   - A host (client), that needs to configure its interface, broadcasts a **DHCPDISCOVER** message, containing its MAC address
   - The DHCP server intercepts the request and replies with a broadcast **DHCPOFFER** message that contains its ID and a proposed IP address with lease time

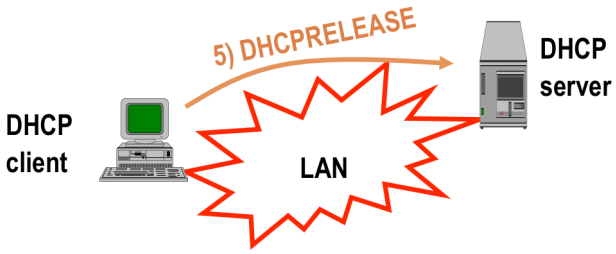


2. **DHCPREQUEST & DHCPACK**

   - The client can accept the offer by sending a broadcast **DHCPREQUEST** message that contains the ID of the DHCP server
   - Then the server associates the IP to the host and replies with a broadcast **DHCPACK** message that contains the path to fetch all needed information for interface configuration via FTP



3. **DHCPRELEASE**
   When the address is not longer needed (e.g., the interface is switched off), the host sends a (unicast) **DHCPRELEASE** to the server



The configuration parameters, that have to be assigned to a client, are:

- IP address
- Netmask
- Default Gateway
- DNS server

It is possible to have multiple DHCP server (Figure 10.1) to use DHCP Relay (works as BOOTP relay in Figure 10.2).

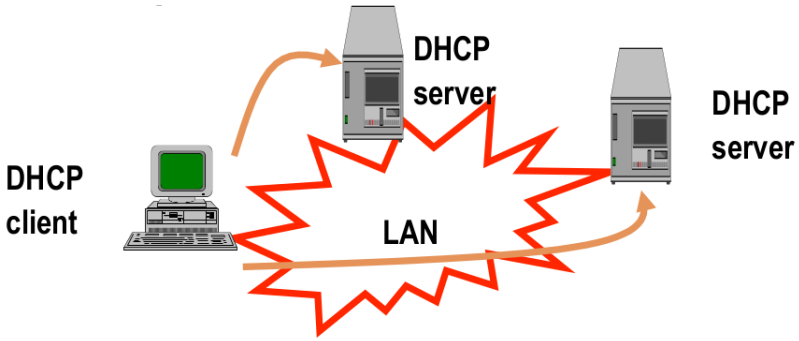


Figura 10.1: Example of multiple DHCP servers.

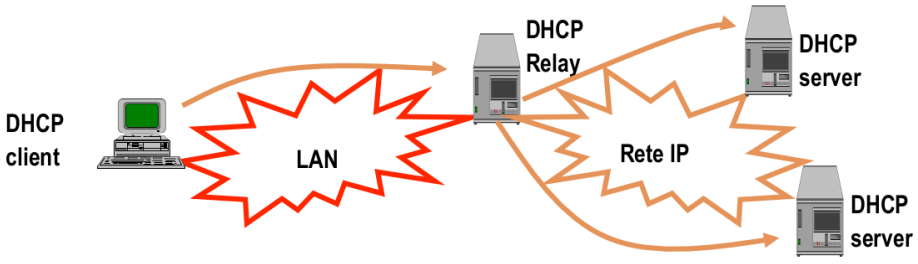
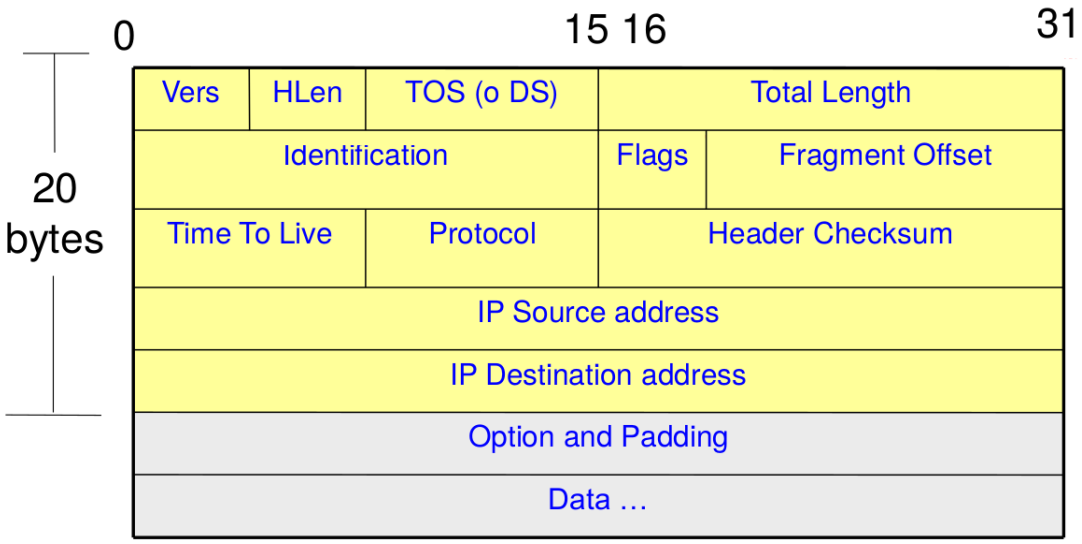


Figura 10.2: Example of DHCP relay.

Capitolo 11

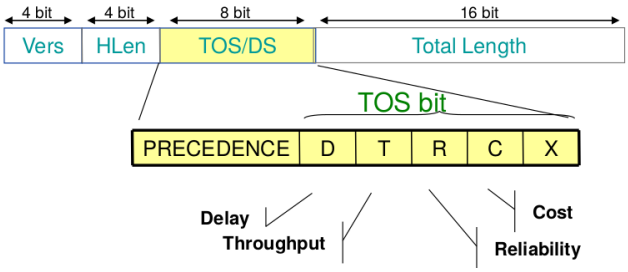
IP datagram



An IP datagram is a packet of IP protocol (IPv4 in this case), that is composed of many fields:

- **Header field (Mandatory of 20 bytes)**
  - **Vers**
    IPv4 or IPv6. It's at the beginning because the remain format of the datagram depends on version specified in this field
  - **IP Header Length (HLen)**
    Number of words of 4 Bytes that composes the header (Default: HLen=5 $\Rightarrow$ 20 byte)
  - **Type Of Service (TOS)**
    it's composed of two parts:
    * *PRECEDENCE*
      Priority 0-7 (Not used field by IPv4)
    * *TOS bits*
      composed of 4 bits. Some possible configuration are:

0000	normal (default)
0001	min cost
0010	max reliability
0100	max throughput
1000	min delay



- **Total Length**
  It's the total length of the datagram (# of byte) and it's needed to find out possible padding bytes added by MAC layer
- **Identification**
  used to uniquely identify fragments in which a IP datagram can be split
- **Flags (3 bits)**
  each bit has a particular meaning:
  - **Reserved**
    always set to 0
  - **Don't Fragment (DF)**
    if set to 1, it indicates that packet hasn't to be split by a device. If a device can't send the packet, without fragmentation, the IP fragment is dropped
  - **More Fragments**
    it indicates if this IP datagram is the last one of a series of packets in which a packet has been fragmented
- **Fragment Offset (13 bit)**
  It indicates the offset of a particular fragment (# of blocks of 8 bytes) with respect to the beginning of orginal IP datagram
- **Time To Live(TTL)**
  It's the number of maximum hops that can be done from the packet. This number decreases a unit at each forwarding. Default values of it depend on Operating System.
  If TTL becomes zero, the datagram is dropped and the router may send an ICMP message to IP source
- **Protocol**
  Upper layer protocol (ID of the protocol whose data is the IP payload)
  1 ⇒ICMP message
  6 ⇒TCP segment
  17 ⇒UDP datagram
- **Header Checksum**
  It covers only the IP Header. The header changes at every hop, because of TTL decreasing, checksum is recomputed and rewritten
- **IP Source & Destination addresses**
  For end-to-end addressing, Netmasks are NOT included because this info is provided by network administrators or DCHP servers and inserted into the routing table by routing algorithms

- **Option and Padding**
  The OPTION field may actually entail a number of Options of 6 different types. Each Option has the

typical Type-Length-Value structure (TLV).

TLV organized the options with some fields:

- **Code (8 bits)**
  divided in:
  - **Copy (1 bit)**

0	Copy only in first fragment
0	Copy into all fragments

  - **Class (2 bits)**

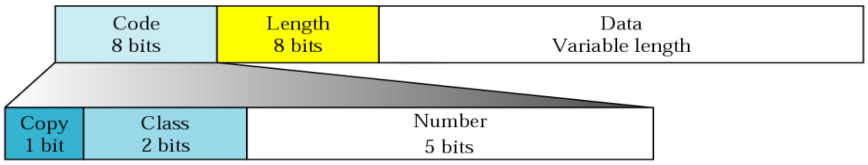
00	Datagram control
01	Reserved
10	Debugging and management
11	Reserved

  - **Number (5 bits)**

00000	End of option
00001	No operation
00011	Loose source route
00100	Timestamp
00111	Record route
01001	Strict source route

- **Length (8 bits)**
  Length of the option data field in bytes (skipped in single-byte options)
- **Data variable length**

- **Data**



11.1 Fragmentation

Every huge packet ("Elephant" or "Jumbo" frame) is divided into smaller packets at the source (Fragmentation), because if there is an error on one of them, it can be retransmitred (Using upper layer protocols) otherwise there can be errors on a frame of bigger dimension (more difficult to retransmit right and more time to transmit it).
To understand if a datagram should be split into smaller packets, a host watches the link on which it needs to forward the packet.
If bitrate of the link is lower than the size of the packet, it analyzes the value, specified in the datagram, for "Don't Fragment" field (DF).
If **DF=1** the packet will be dropped and router will send a ICMP message to the source (**Not mandatory**).
The maximum size of a packet across a link, is called **Maximum Transfer Unit (MTU)**.

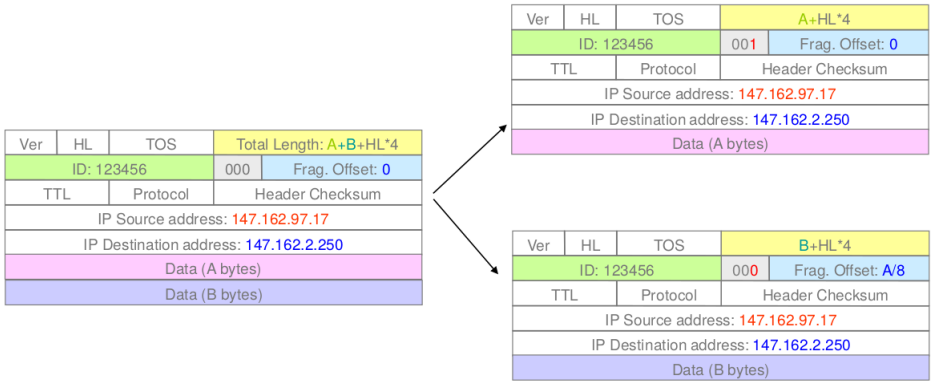
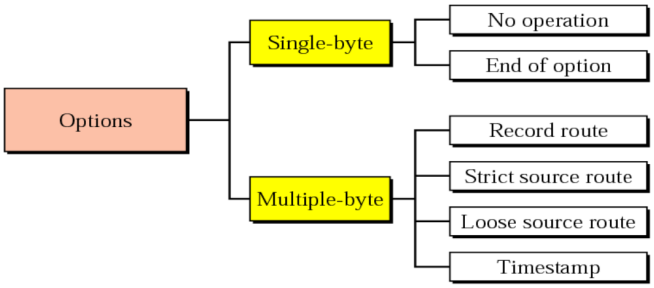


Figura 11.1: Example of fragmentation.

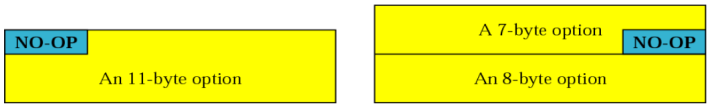
11.2 Different types of options



11.2.1 No-Operation options (NO-OP)

Code:1 ⇒ 00000001

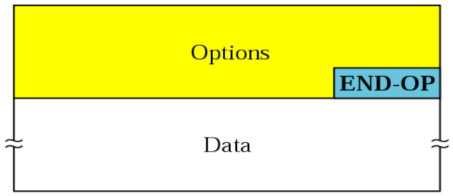
1. Used to align beginning of an option
2. Used to align the next option



11.2.2 End of Option (END-OP)

Code:0 ⇒ 00000000

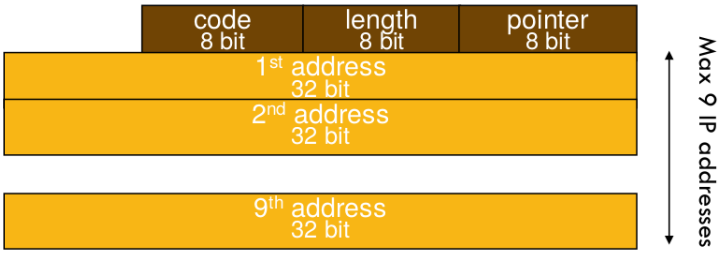
Used for padding of the end of an option.



11.2.3 Record route option

Code:7 ⇒ 00000111

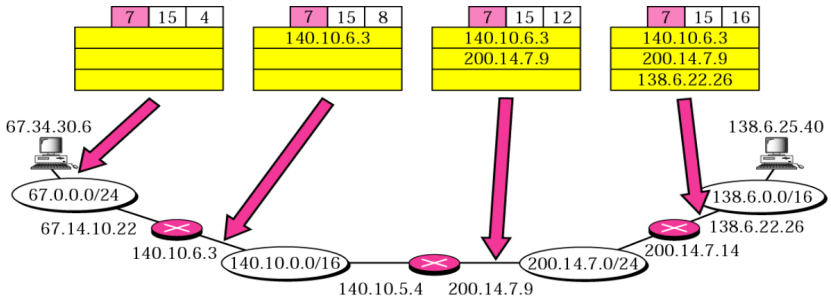
Used to track the path, followed by a datagram, from source to dst (Up to 9 hops).
Initially, all addresses are void and pointer gets the value 4 (first address field).
When the packet is forwarded by the router, new address will be updated and the pointer will point to the next IP address (pointer + size of IP address).

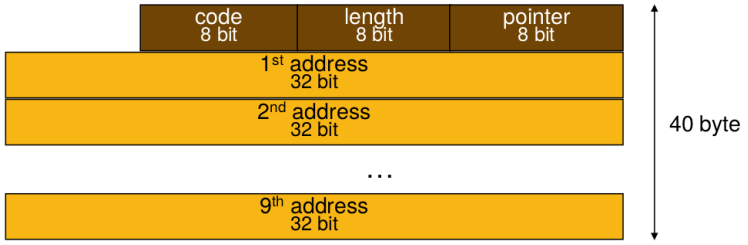


11.2.4 Strict source route

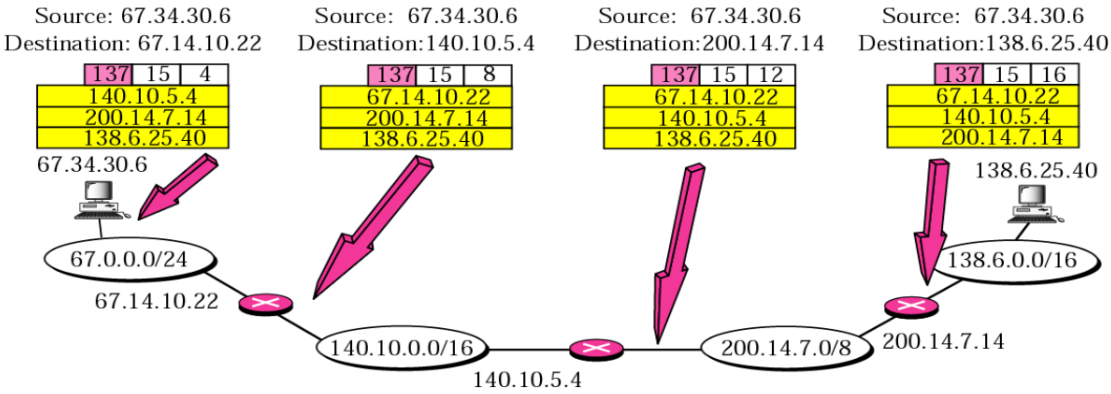
Code:137 ⇒ 10001001

It forces strict source routing forwarding. The path is decided by the source node: at the beginning, the option field contains the IP addresses of all the routers along the path from source to destination.
Each router replaces the IP address of the outgoing interface with the IP address of the incoming interface (for a reply packet, path information can be used again). Pointer is incremented by 4 at each hop as Record route option.





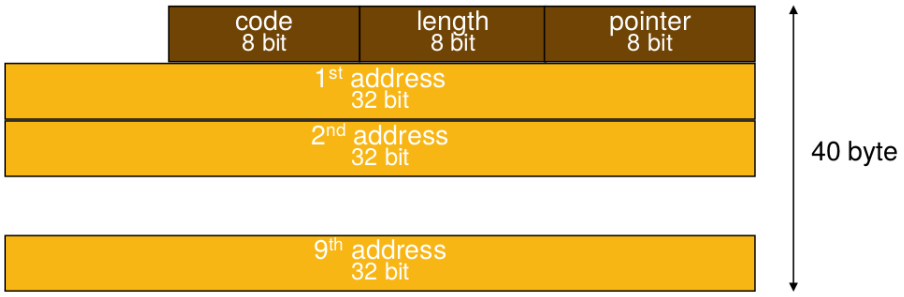
If next hop is unreachable, packet is discarded and error message (ICMP) is returned to the source.
This option is RARELY used in PRACTICE.



11.2.5 Loose source route

Code:131 ⇒ 10000011

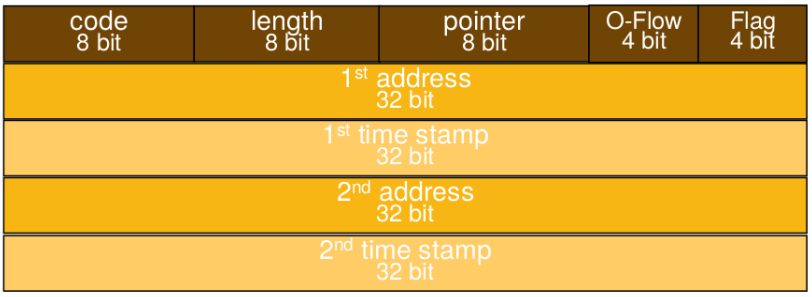
Same as Strict Source Routing but there can be some not specified hop.



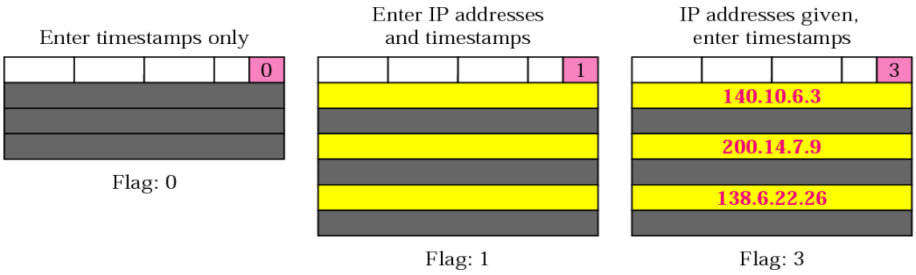
11.2.6 Time stamp

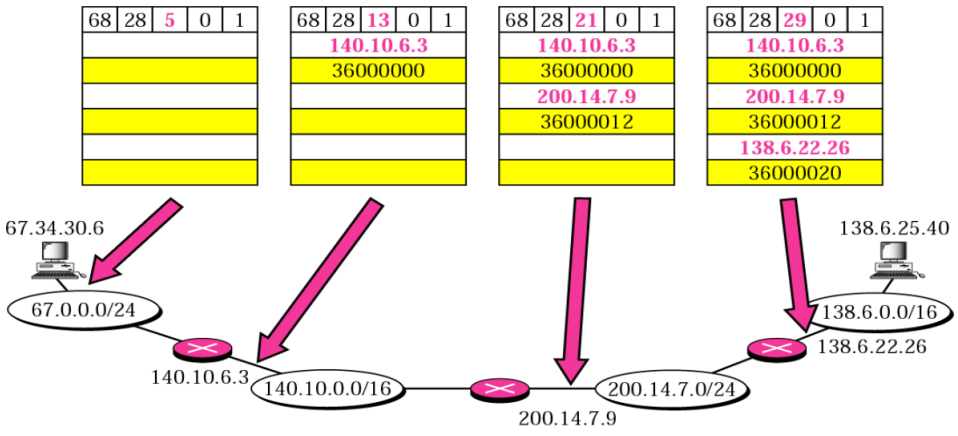
Code:68 ⇒ 01000100

It counts and records the absolute time, spent by the datagram in each router (in ms). For each IP address, its time stamp is specified.



- **Over-Flow (4 bits)**
  It counts the number of hops along the path, that could not add their timestamp, because the field was full
- **Flag(4 bits)**
  0 ⇒Whether IP addresses have not to be specified 1 ⇒Whether IP addresses must be entered together with timestamp 2 ⇒Whether IP addresses have already been entered by the source


Enter timestamps only
0
Flag: 0
Enter IP addresses
and timestamps
1
Flag: 1
IP addresses given,
enter timestamps
3
140.10.6.3
200.14.7.9
138.6.22.26
Flag: 3


68 28 5 0 1
68 28 13 0 1
140.10.6.3
36000000
68 28 21 0 1
140.10.6.3
36000000
200.14.7.9
36000012
68 28 29 0 1
140.10.6.3
36000000
200.14.7.9
36000012
138.6.22.26
36000020
67.34.30.6
138.6.25.40
67.0.0.0/24
138.6.0.0/16
138.6.22.26
140.10.6.3
140.10.0.0/16
200.14.7.0/24
200.14.7.9

Capitolo 12

ICMP

MAC packet contains IP packet and ICMP messages are embedded into IP datagrams. ICMP can also be seen as a protocol that makes use of IP.

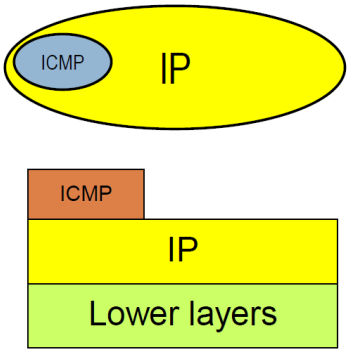


Figura 12.1: How ICMP is embedded in IP datagrams.

The main controls, made by ICMP, are:

- **Error management (passive)**
  - Destination unreachable
  - Time expired (TTL or fragment reassembly timer)
  - Data inconsistency
  - Flow control
- **Active mode**
  Echo + Echo Reply (ping Unix)

In the IP header, the field protocol takes value 1 and indicates that the payload is an ICMP message. Other header fields depend on the type of message that must to be generated.

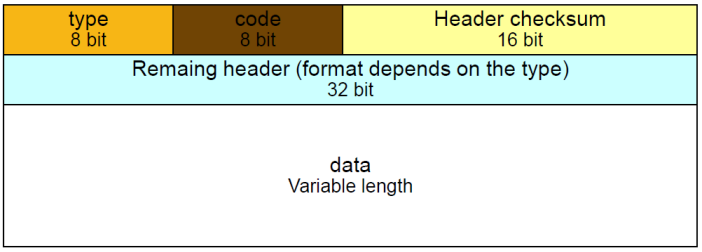


Figura 12.2: Format of ICMP message.

0	Echo reply
3	Destination unreachable
4	Source Quench
5	Redirect (change a route)
8	Echo request
11	Time exceeded
12	Parameter problem
13	Timestamp request
14	Timestamp reply
17	Address mask request
18	Address mask reply

Tabella 12.1: Type values.

12.1 Main rules of ICMP error messages

- No ICMP error message will be generated in response to a datagram carrying an ICMP error message
- No ICMP error message will be generated for a fragmented datagram that is not the first fragment
- No ICMP error message will be generated for a datagram having a multicast address
- No ICMP error message will be generated for a datagram having a special address such as 127.0.0.0 or 0.0.0.0.

NOTE: *No all routers generate ICMP messages.*

12.2 Types of ICMP messages

12.2.1 Echo

Echo-request and Echo-reply are used to check the reachability of hosts and routers.
Upon receiving an Echo-request, the ICMP entity of a device immediately replies with Echo reply.

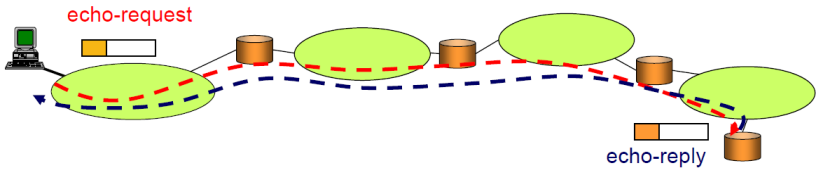


Figura 12.3: ECHO requests and replies in practice.

Code: $\Rightarrow 0$

Type: $\Rightarrow$ 8 request
$\Rightarrow$ 0 reply

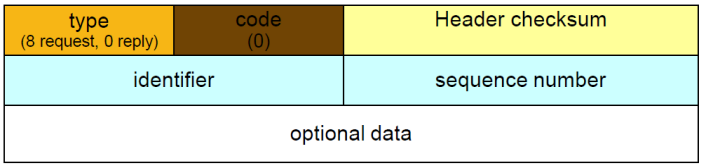


Figura 12.4: Format of ECHO message.

Other important fields of Echo messages are:

- **Identifier**
  Each **Echo** message has an identifier, defined in the **Echo request**, and replicated in the **Echo reply**.

- **Sequence number**
  Consecutive requests may have the same identifier and change from others for sequence number only. The sequence number is used to measure the RTT and count the number of lost bytes.

- **Optional data**
  The sender can add **Optional data** to the request message. The data will be replicated in the reply message.

The payload of Echo (IP datagram) is used to check the capacity of a link (RTT is bigger if the link has small bitrate).

12.2.2 Destination unreachable

When a packet is dropped, an error message is returned, through ICMP, to the source.

Type: $\Rightarrow$ 3

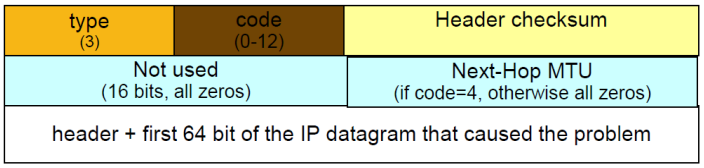


Figura 12.5: Destination unreachable message format.

The "code" field of the ICMP message refers to the type of error that has generated the message.

Code	Description	References
0	Network unreachable error.	RFC 792
1	Host unreachable error.	RFC 792
2	Protocol unreachable error. Sent when the designated transport protocol is not supported.	RFC 792
3	Port unreachable error. Sent when the designated transport protocol is unable to demultiplex the datagram but has no protocol mechanism to inform the sender.	RFC 792
4	The datagram is too big. Packet fragmentation is required but the DF bit in the IP header is set.	RFC 792
5	Source route failed error.	RFC 792
6	Destination network unknown error.	RFC 1122
7	Destination host unknown error.	RFC 1122
8	Source host isolated error. (Obsolete)	RFC 1122
9	The destination network is administratively prohibited.	RFC 1122
10	The destination host is administratively prohibited.	RFC 1122
11	The network is unreachable for Type Of Service.	RFC 1122
12	The host is unreachable for Type Of Service.	RFC 1122
13	Communication Administratively Prohibited. Administrative filtering prevents a packet from being forwarded.	RFC 1812
14	Host precedence violation. The requested precedence is not permitted for the particular combination of host or network and port.	RFC 1812
15	Precedence cutoff in effect. The precedence of datagram is below the level set by the network administrators.	RFC 1812

Tabella 12.2: Code values.

12.2.3 Time exceeded

It's generated when some packets are missing or don't reach the destination.

Type: $\Rightarrow 3$

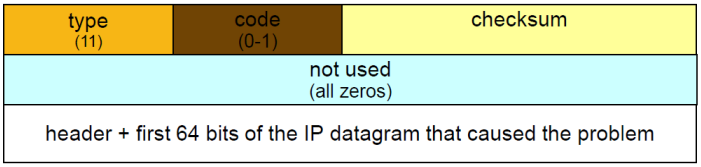


Figura 12.6: Time exceeded message format.

The main problems, that generate this message, are:

Code	Problem
0	Generated by a router when it decreases the TTL to 0
	Returned to the source of the IP datagram
1	Generated by the destination, when some fragments are missing, after the fragment reasembly timer expires

12.2.4 Parameter problem

It's generated when there are some wrong formats or unknown options.

Type: $\Rightarrow 12$

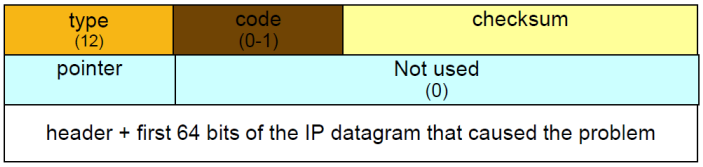


Figura 12.7: Format of Parameter problem message.

The main problems generated by this message are:

Code	Problem
0	If the header of an IP datagram contains a malformed field (violate format)
1	Used when an option is unknown or a certain operation cannot be carried out

12.2.5 Redirect

It's generated by a router to require the source to use a different router

$$\text{Type: } \Rightarrow 5$$
$$\text{Code: } \Rightarrow 0-3$$

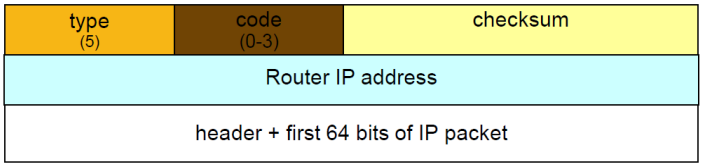


Figura 12.8: Format of Redirect message.

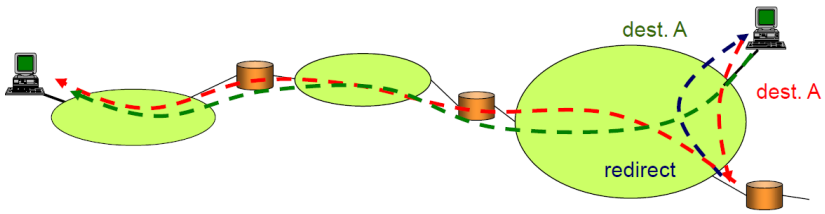


Figura 12.9: How Redirect messages are used.

12.2.6 Timestamp request e reply

It's used to exchange clock information between source and destination.

$$\text{Code: } \Rightarrow 0$$

- **Originate timestamp**
  inserted by the source
- **Receive timestamp**
  inserted by the destination right after receiving the ICMP message
- **Transmit timestamp**
  inserted by the destination just before returning the ICMP message

Type: ⇒ 13 request
⇒ 14 reply

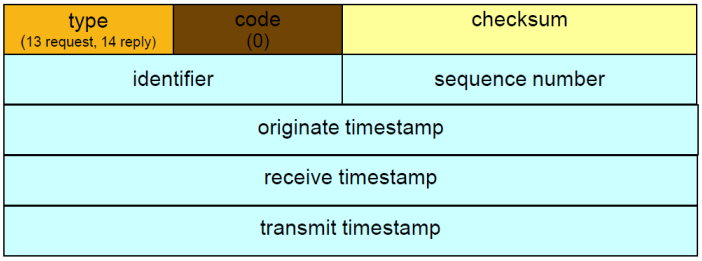


Figura 12.10: Format of Timestamp request and reply.

12.2.7 Address mask request and reply

It's used to ask for the netmask of a router/host.

Type: ⇒ 17 request
⇒ 18 reply

Code: ⇒ 0

- **Address mask**
  In the request message, it's void and it is populated by the device that replies to the request

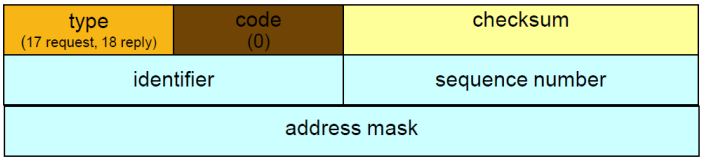


Figura 12.11: Format of Address mask request and reply.

Capitolo 13

IPv6

The temporary solutions to postpone the addresses exhaustion problem were:

- Classless Inter Domain Routing (CIDR)
- Network Address Translation (NAT)

IPv6 project was proposed as the ultimate solution to IP addresses exhaustion problem. On 1996, the first IPv6 (*RFC 1883*) was released.

13.1 IPv6 format

IPv6 address is composed of **128 bits**, represented as a succession of 8 groups of 4 hexadecimal digits separated by a colon.
There are some ways to write an address:

- Case insensitive

$$2001:0000:0000:BEEF:021C:B3FF:FEBF:6C74$$

- Leading zeros in a field is optional

$$2001:0:0:BEEF:21C:B3FF:FEBF:6C74$$

- Consecutive fields of zeros can be represented as :: , but only once in an address

$$2001::BEEF:21C:B3FF:FEBF:6C74$$

- A sequence of 4 bytes at the end of the address can also be written in decimal form, using dots as separators

$$2001::BEEF:21C:B3FF:\mathbf{192.168.1.1}$$

There are some types of destinations:

- **Unicast:** one to one
- **Multicast:** one to many
- **Anycast:** one to one of many (nearest one)
- **No Broadcast:** it's replaced by efficient multicast

The reachability scope of the address can be:

- **Node local**
  same node (loopback)
- **Link local**
  all interfaces connected to the same link (non routable)
- **Unique local**
  all interfaces in the same LAN (non routable outside)
- **Global unicast**
  public IPv6 addresses (fully routable)

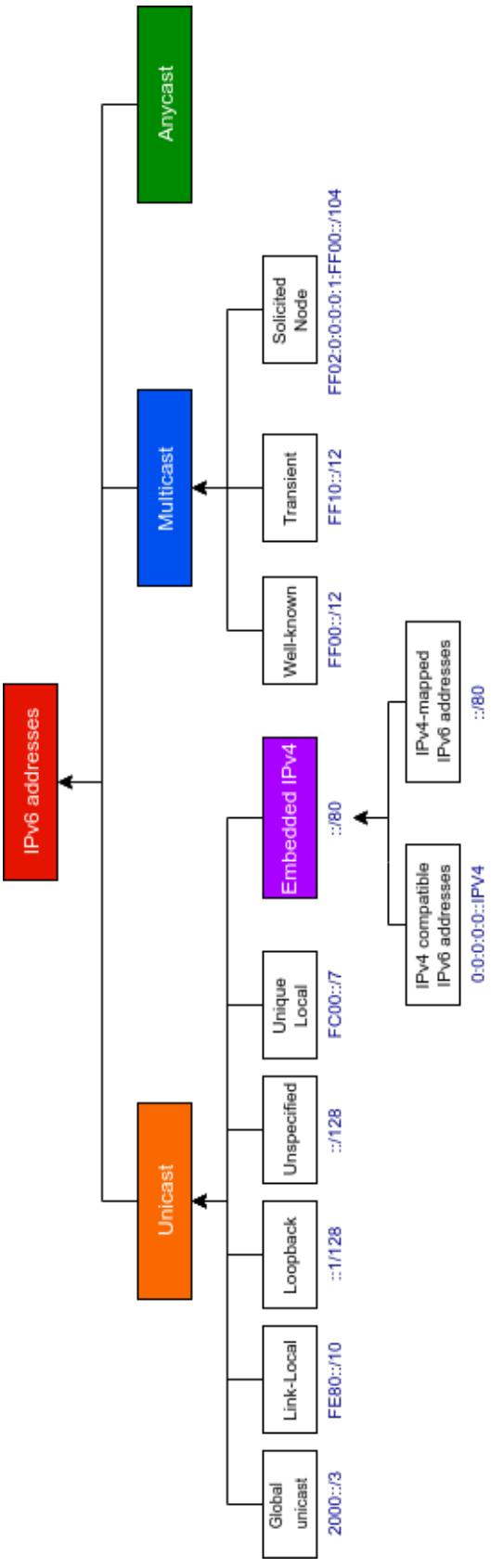


Figura 13.1: IP addresses types.

13.2 Global Unicast Address

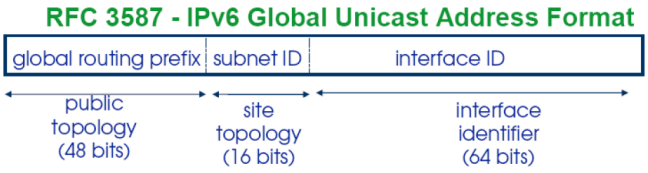


Figura 13.2: Global Unicast Address format.

The main fields of an IPv6 Global unicast Address are (Figure 13.2):

- **Global routing prefix**
  A (typically hierarchically-structured) value assigned to a site (a cluster of subnets/links)
- **Subnet ID**
  An identifier of a subnet within the site
- **Interface ID**
  the latest 64-field of unicast address may be assigned in several different ways:
  - Auto-configured from a 64-bit EUI-64
  - Auto-generated pseudo-random number
  - Assigned via DHCP
  - Manually configured

13.2.1 IEEE 64-bit Extended Unique Identifier (EUI-64)

It expands a 48-bit (MAC) address into a 64 bits interface identifier. The starting MAC address can either have:

- **Local scope**
  unique only locally, not globally ⇒ serial links
- **Global scope**
  globally unique ⇒ Ethernet MAC addresses

This method is composed of 2 steps:

1. **Step**
   - The 48-bit MAC address is expanded to 64 bit by stuffing FFFE after the first 16 bits of MAC address
   - FFEE can only appear in the middle of EUI-64 addresses obtained from 48-bit MAC addresses

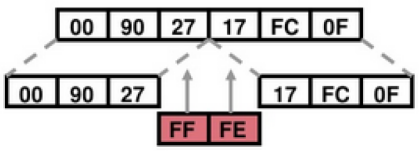


Figura 13.3: EUI-64 step 1.

2. **Step**
   The universal/local ("u" bit) is inverted (but also its meaning).

"U" Bit	MAC	EUI-64
0	Global	Local
1	Local	Global

Inverting this bit makes simpler the notation for the manually configuration of local scope identifier for a network administrator.

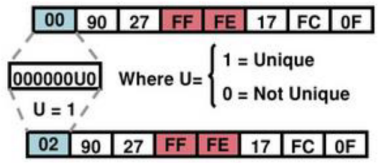


Figura 13.4: EUI-64 step 2.

13.3 Link local addresses

IPv6 unicast address that can be automatically configured on any interface using:

- link-local prefix FE80::/10 [1111 1110 10]
- followed by 54 bits equal to zero
- followed by the interface identifier (*EUI-64* format)

It can be used only on one-link (do not pass L3-switches) by specific protocols.
All IPv6 enabled interfaces have a link-local unicast address and it's used for automatic address configuration and neighbor discovery protocol (`https://www.cisco.com/c/en/us/support/docs/ip/ip-version-6-ipv6/113328-ipv6-lla.html`).

13.3.1 Special addresses

- **Unspecified**
  It's used as a place holder when no address is available (initial DHCP request, Duplicate Address Detection)

$$0:0:0:0:0:0:0:0/128 =_{::} /128$$

- **Loopback**
  It's the same as 127.0.0.1 in IPv4

$$0:0:0:0:0:0:0:1/128 =_{::} 1/128$$

13.3.2 Unique local IPV6 unicast (RFC 4193)

They are equivalent to IPv4 private addresses. They are unique at global level, but can actually be used only in private networks.
They are not expected to be routable on the global Internet but they are routable inside of a more limited area such as a site.
They may also be routed between a limited set of sites.

13.3.3 Embedded IPv4

There are two type of addresses:

- **IPv4-compatible IPv6 addresses**
  characterized by a prefix of all zeros (96 bits) and a suffix of 32 bits, taking the form of an IPv4-address. These addresses are IPv6 addresses that can be turned into IPv4 addresses by stripping off the first 96 bits.
  These addresses simplify the communication between IPv6-only and IPv4-only hosts. For the translation between IPv6 and IPv4, addresses would require an IPv4 address for each IPv6-only host, which is not much meaningful.
  Hence this assignment of addresses has been **deprecated**.

- **IPv4-mapped IPv6 addresses**
  characterized by a prefix of 80 zeros, followed by 16 ones (FFFF) and a suffix of 32 bits taking the form of an IPv4 address.
  These type of addresses are still valid and are used to represent an IPv4 address in an IPv6-compatible format.
  Any valid IPv4 address can be represented in an IPv6 only network adding the string **0:0:0:0:0:FFFF**, as a prefix, to the IPv4 address.

13.4 Multicast addresses

They are used for general multicast services and neighbor discovery.
They have different scope based on the fourth hexadecimal digit of the prefix (**scope**).

Figura 13.5: Multicast addresses format.

0	Reserved
1	Interface-Local Scope
2	Link-Local Scope
3	Reserved
4	Admin-Local Scope
5	Site-Local Scope
6	(unasigned)
7	(unasigned)
8	Organization-Local Scope
9	(unasigned)
A	(unasigned)
B	(unasigned)
C	(unasigned)
D	(unasigned)
E	Global Scope
F	Reserved

13.4.1 Multicast scopes

- **Scope 1: Interface-Local scope**
  spans only a single interface on a node and it's useful only for loopback transmission of multicast

- **Scope 2: Link-Local multicast scope**
  spans the same topological region as the corresponding unicast scope
- **Scope 4: Admin-Local scope**
  is the smallest scope that must be administratively configured, i.e., not automatically derived from physical connectivity or other non-multicast-related configuration
- **Scope 5: Site-Local scope**
  is intended to span a single site
- **Scope 8: Organization-Local scope**
  is intended to span multiple sites belonging to a single organization

13.4.2 Solicited-Node Multicast Address

Multicast address only valid within the local link. Every host interface has at least one such address associated. The address is created by taking the last 24 bits of an unicast address and appending them to the prefix **FF02:0:0:0:0:1:FF00::/104**.
It's used by Neighbour Discovery Protocol to find out the MAC address of a host with known IPv6 address. It has the same role of ARP in IPv4 but it avoids broadcast to all connected hosts.

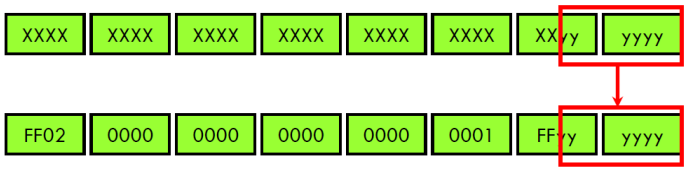


Figura 13.6: Solicited-Node Multicast Address format.

Some particular uses of Multicast addresses are:

- **All nodes multicast group**
  It's basically **local broadcast** $FF02::1/168 \Rightarrow$ all link-local IPv6 nodes
- **All routers multicast group**
  $FF01::1 \Rightarrow$ all routers address (interface local→loopback)
  $FF02::2 \Rightarrow$ all routers address (link local)
  $FF05::2 \Rightarrow$ all site-local routers (site local)

13.5 Anycast addresses

They are used for cover things like nearest DNS server, nearest DHCP server and similar dynamic groups. They can be used only as destination addresses.

13.6 IPv6 new features

- Stateless auto-configuration for hosts
- Native multicast support
- Link local addresses
- Jumbograms/Jumboframes
  packets are limited to 64KB, Jumbograms can be as large as 4GB (for high speed network)
- Interfaces can be identified by multiple addresses

- Network layer security
  IP security is built-in the protocol, unlike IPv4 built-on
- ICMPv6
  The ICMPv6 protocol is built-in the IPv6 protocol and it's provided as an extension header
- Mobility
  Mobile nodes can change their IPv6 dynamically

13.7 IPv6 Header

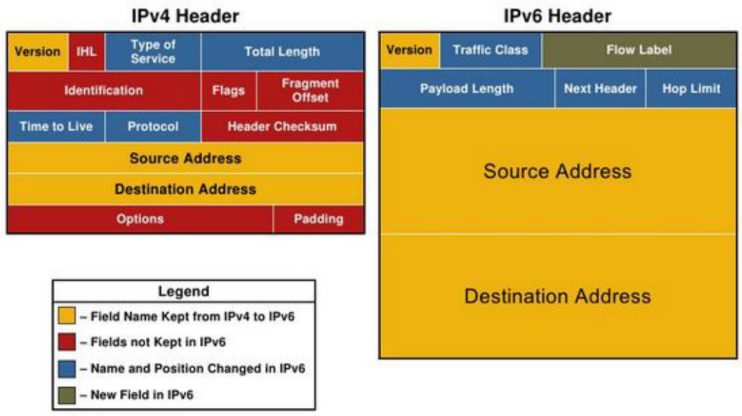


Figura 13.7: IPv4 vs IPv6.

13.7.1 IPv6 Extension Headers

- **Hop-by-hop options**
  Information that must be examined by every node along the path (e.g. Jumbograms)
- **Routing**
  Similar to IPv4 Loose Source and Record Route option
- **Fragment**
  Used to indicate to the destination how to reassemble the fragments (fragmentation can be performed only by the source node)
- **Authentication**
  Used to provide data integrity and authentication

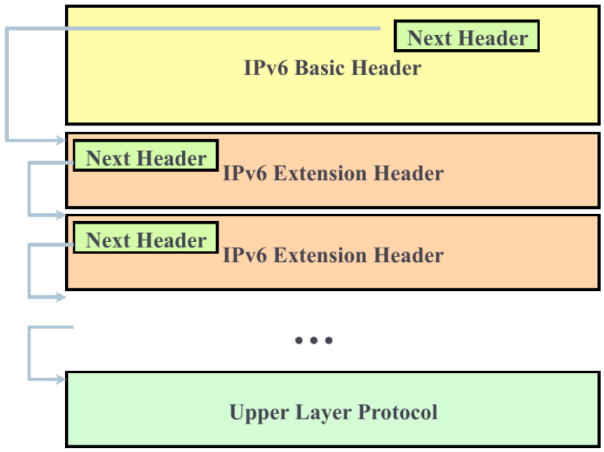


Figura 13.8: IPv6 extension Headers.

13.8 Hierarchical routing

IPv6 is designed to use a fully hierarchical routing approach, to reduce the size of the routing table maximally. Hence, this reduces the hardware memory needed and increases the lookup speed.
The Prefix length (Netmask) is also used, so the slash notation is used to indicate the number of bits in the IP address, used for routing.

Example: *2001:0DB8:100:BEEF:021C:B3FF:FEBF:6C74/48*

The entry in the routing table with the longest prefix, matches first.

13.9 ICMPv6

As in the previous version of the protocol, ICMP is used to monitor the state of the network and to send error reporting but there are new features:

- Neighbor Discovery Protocol (replaces ARP)
- Multicast Router Discovery (replaces Internet Group Management Protocol - IGMP)

Figura 13.9: ICMPv6 format.

Type	Message
1	Destination unreachable
2	Packet too big (MTU)
3	Time exceeded
4	Parameter problem
128/129	Echo Request/Echo Reply
133/134	Router Solicitation/Advertisement
135/136	Neighbor Solicitation/Advertisement

13.10 Neighbor Discovery Protocol

It replaces ARP and some ICMP functionalities. The packet must originate and terminate on the same link. It's used to:

- Discover other nodes on the link
- Determine link layer address of other nodes
- Address auto-configuration
- Duplicate Address Detection
- Find available routers and DNS servers
- Maintain node reachability information

It uses the following ICMPv6 messages:

- **Neighbor solicitation (NS)**
  Determine link-layer address of neighbor
- **Neighbor advertisement (NA)**
  Actively keep track of neighbor reachability
- **Router solicitation (RS)**
  Determine on-link routers and default route
- **Router advertisement (RA)**
  Send network information from routers to host

13.10.1 Router Solicitation

Nodes request routers to send router Advertisement immediately. The packet description is:

- **Source address** = Link-local address
- **Destination address** = Multicast Address All-Routers
- **ICMP type** = 133

13.10.2 Router Advertisement

Routers advertise periodically (max time between advertisements is between 4 and 1800 seconds). It specifies if statefull or stateless autoconfiguration must be used.
The packet description is:

- **Source address** = Router Link-local address
- **Destination address** = All Nodes Multicast Address
- **Data** = Prefixes, lifetime, default router, options
- **ICMP type** = 134

13.10.3 Neighbor Solicitation

It's sent by a node to determine link-layer address of a neighbor. It replaces the IPv4 **ARP request**.
The packet description is:

- **Source address** = Link-local address
- **Destination address** = Solicited-Node Multicast, Address or All Nodes Multicast Address
- **Data** = Link-Layer Address of source
- **Query** = Please send me your link-layer address
- **ICMP type** = 135

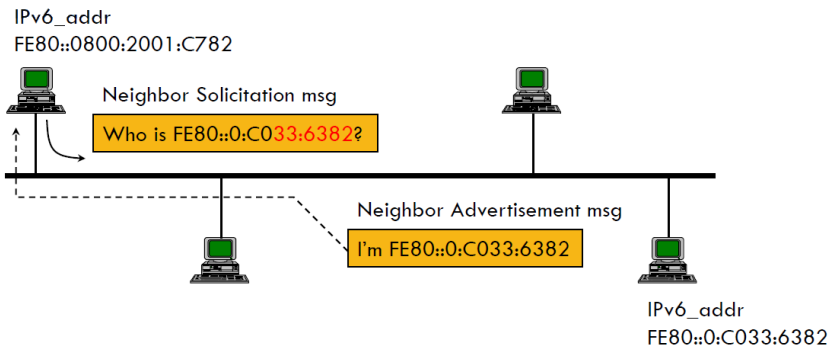
13.10.4 Neighbor Advertisement

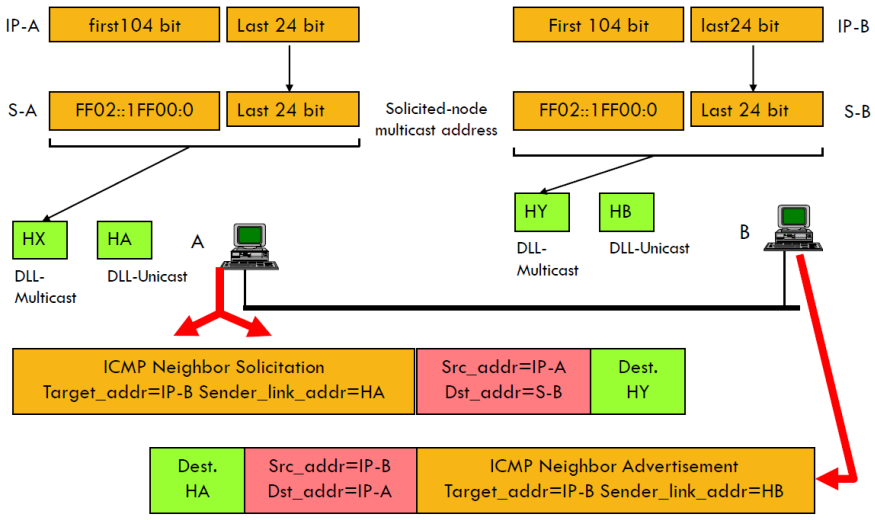
It's the response to a neighbor solicitation It replaces the IPv4 **ARP response**.
The packet description is:

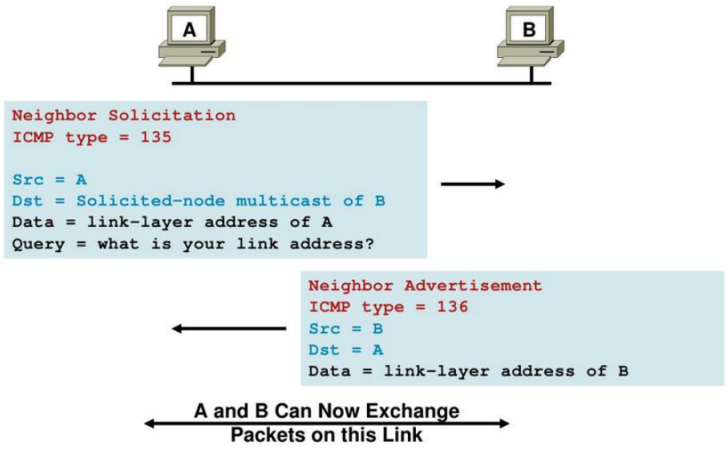
- **Source address** = Link-local address
- **Destination address** = Destination address of the NS request
- **Data** = Link-Layer Address of source
- **ICMP type** = 136

13.10.5 ICMPv6 Address Resolution

The Neighbor Solicitation is sent to the node-solicited multicast address, which can be determined even by the source node.
The Neighbor Advertisement is sent to the IPv6 of the source node.







13.10.6 Router discovery

13.11 Address autoconfiguration

Besides Link-local addresses, it is also possible to self-configure global unique addresses. There exists:

- **Stateful configuration (via DHCPv6)**
- **Stateless configuration (via ICMP)**

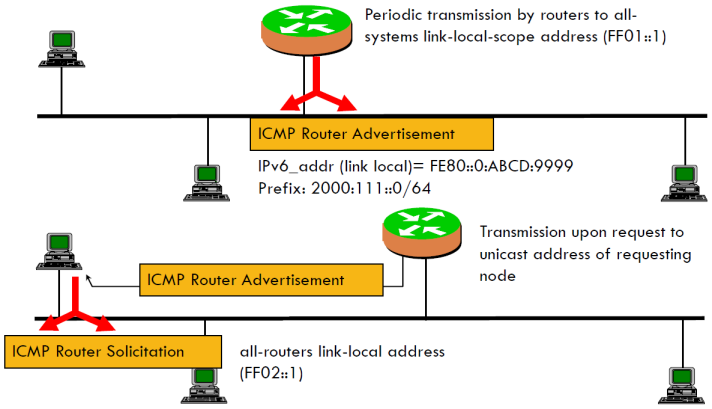


Figura 13.10: Router discovery.

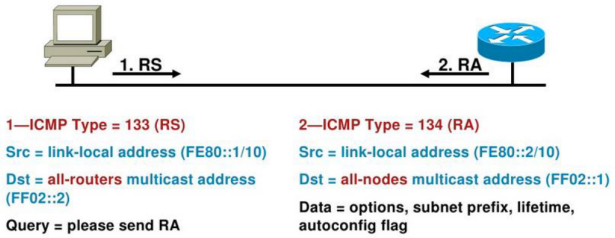


Figura 13.11: Router solicitation and advertising.

Prefix is announced by routers and host address can be obtained from 64-bit long PHY address.
The Auto-configuration process is composed of different phases:

- Host probes the local routers to get the address prefix
- Host generates the host identifier (given by link-layer address, EUI-64)
- The full address is obtained merging the two fields
- Host performs a Duplicate Address Detection to verify that the address taken is unique

13.11.1 Duplicate Address Detection (DAD)

The probability that 2 Ip addresses have the same address is very small but there is this protocol that controls the addresses.
After the node has computed the IPv6 address it joins the "solicited-node multicast" address and the "all hosts multicast" address.
A neighbor solicitation is sent setting the destination address with the tentative IPv6 address chosen.

The packet description is:

- **Source address** = Unspecified
- **Destination address** = Tentative address

The address is not unique if:

- A **Neighbor Advertisement** is received in response to the address solicitation with the tentative address as source address
- A **Neighbor Solicitation** is received with the tentative address as the destination address

13.12 IPv4-IPv6 Transition/Coexistence

The currently used techniques are:

- Dual stack
- Tunneling
- Translation

13.12.1 Dual stack

The dual stack technique uses a double IP layer in the protocol stack.
The devices, using the dual stack, can deliver both IPv4 and IPv6 packets to the other layers in a transparent fashion.

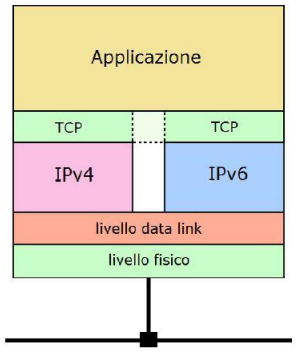


Figura 13.12: Dual stack format.

- **Pros**
  Simplest method to implement
- **Cons**
  - The device still need an IPv4 address
  - The processing load is heavier

13.12.2 Tunneling

The tunneling technique is the most adopted one.
Every IPv6 packet is embedded in an outer IPv4 packet and sent to the destination, through a point to point tunnel, over the IPv4 network or vice versa.

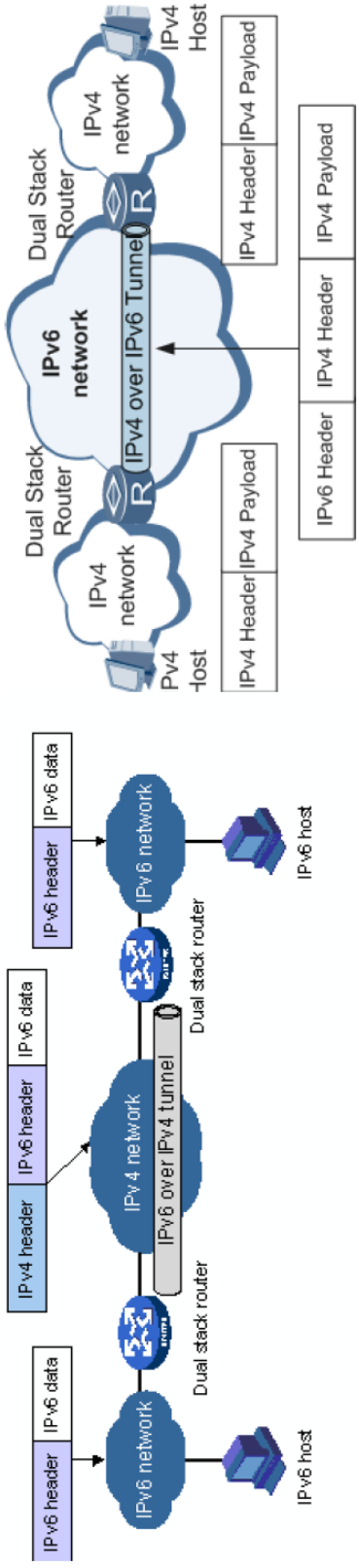


Figura 13.13: Tunneling example.

13.12.3 Network Address Translator – Protocol Translator (NAT-PT or NAT6to4)

It's a protocol that employs the classical IPv4 NAT concepts to translate an IPv6 address to an IPv4 address to let the IPv6 nodes communicate to the external network.

- An IPv6-only host can send a packet to a host in an IPv4-only network using the associated IPv4-mapped IPv6 address as destination address.
- The packet will reach the edge router (at the border of IPv6-only and IPv4-only networks), which will replace:
  - the source IPv6 address with one of the public IPv4 addresses with one of the public IPv4-addresses in the NAT pool
  - the IPv4-mapped IPv6 address with IPv4 address (taking the last 32 bits)

The reply packet will undergo the reverse address mapping.
This mechanism allows communication between IPv6-only client and IPv4-only servers, only when the conversation starts within the IPv6 cloud.
An IPv4-only host, instead, cannot contact an IPv6-only server.

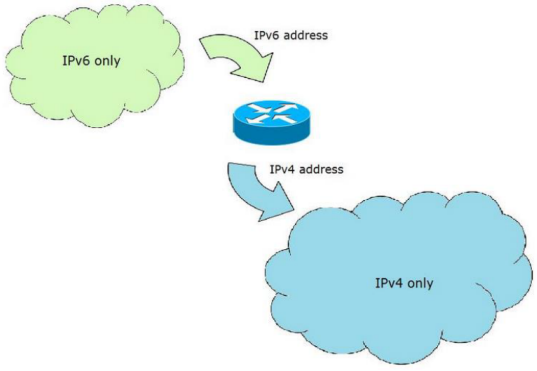


Figura 13.14: Example of connection between IPv6-only network and IPv4-only network.

13.13 IPv6 popularity

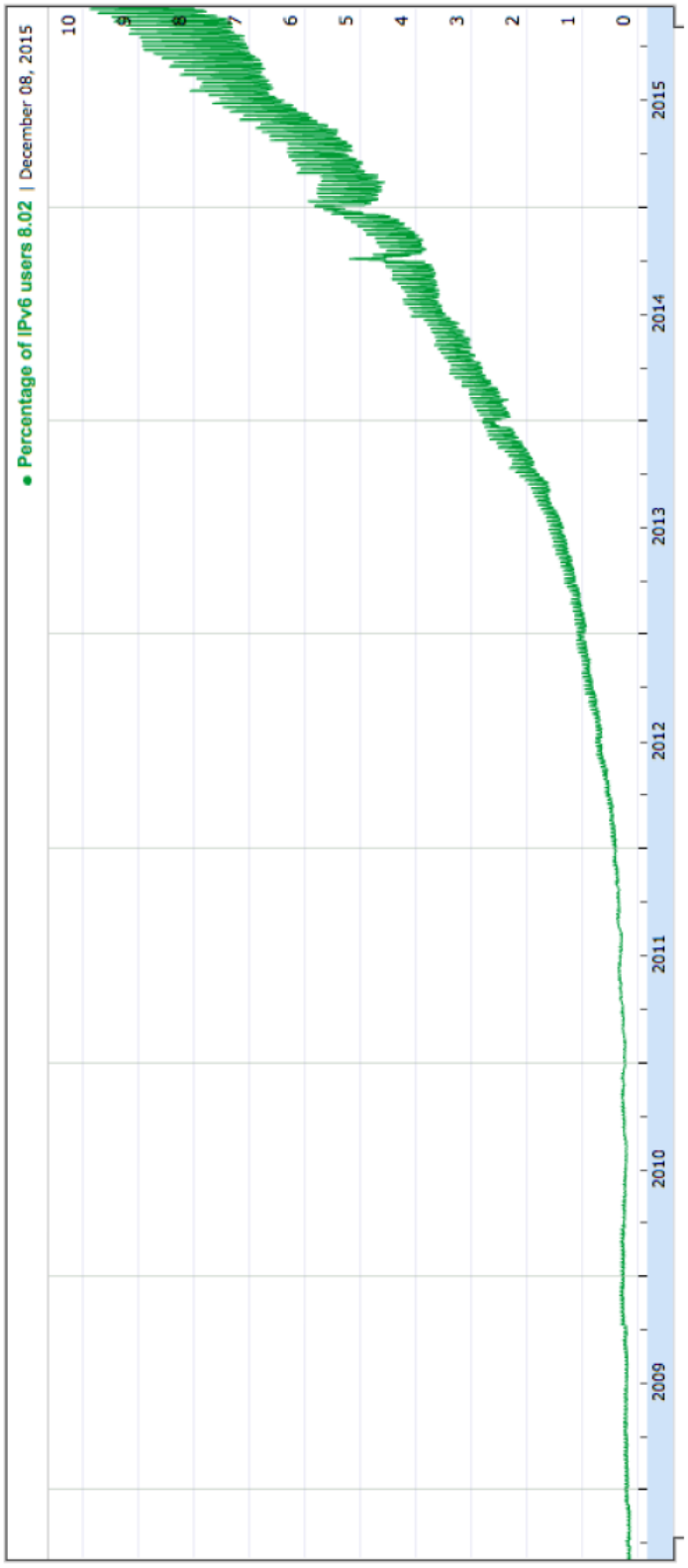


Figura 13.15: Number of users in the last few years.

Capitolo 14

Network Layer: routing

Routing consists of all the procedures that make possible to determine a path from a point to another one.
Routing operates at layer 3 (network), while bridging operates at layer 2.
The routing is hierarchical and based on:

- **Scale**
  To avoid explosion of the routing tables, a hierarchical partition of the gateway routers is necessary
- **Administrative autonomy**
  The company's requirement is to manage its routers inside its own network as needed
- **Autonomous System (AS)**
  The system is composed of a set of interconnected routers, all running the same routing algorithm and using a standard routing protocol to connect to routers in other ASs.
  There are two types of protocols:
  - **Intra-AS Routing Protocol**
    routing protocol used within an AS (example: RIP, OLSR)
    This is usually managed by only one ISP. An ISP can partition its own IP network into many interconnected ASs.
    The **Interior Gateways (IGs)**, in an AS, use one routing algorithm, which may be different for each AS.
  - **Inter-AS Routing Protocol**
    routing protocol used by an AS to connect to other ASs (Border Gateway Protocol, BGP).
    Border Gateways (BGs) which provide interconnection among distinct ASs.
    All BGs use the same routing algorithm: BG routing protocol (BGP)
    1. The BGs exchange messages to inform which addresses are reachable through their ASs
    2. The BGs notify the IGs of their own AS about this information, such that IGs know the BG to reach to forward packets addressed to other ASs

A ASN, that is a unique identifier, is assigned by IANA to each AS.
Each network can be seen by Inter-AS routing protocol as a single node.
The hierarchical organization of internet for the routing addresses assignation is composed of these organizations (from the biggest one to the smallest one):

$$\text{IANA} \Rightarrow \text{RIR} \Rightarrow \text{ISP} \Rightarrow \text{AS} \Rightarrow \text{networks}$$

	Inter-AS	**Intra-AS**
Policy	Control of traffic and who routes throught its net	Single Admin no policy decisions needed
Performance	policy dominates over it	focus on it

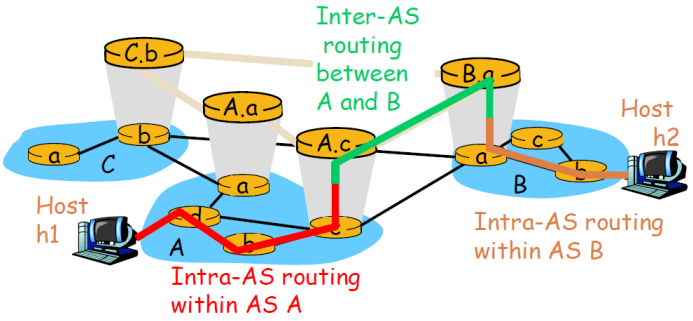


Figura 14.1: Example of Intra-AS and Inter-AS routing.

14.1 Intra-AS routing algorithm

14.1.1 Static vs dynamic

Static routing is based on static routing tables, that are set up manually.
This type of routing can't react automatically to network topology changes or router/link failures but it can be defined more (static) paths between pairs of nodes assigning different priorities to each pair in order to overcome problems in case of link failures.
Static routing is based on routing tables updated automatically watching the traffic loads or topology change.
Routing protocols have to maintain and disseminate routing information (control traffic,..).
Update of the routing tables in the nodes may not be homogeneous and it can cause risk of route instability during transient periods.
Finding a route in a network can be modeled as the problem of finding the "least cost path" in a graph:

- Router and networks ⇒ graph vertexes **V**
- Links ⇒ graph edges **E**
- Link cost ⇒ link distance,...

The least cost path is based on the choice of the costs.

14.2 Intra-routing

Distance Vector	Link State
Routing Information Protocol (RIP)	Open Shortest Path First (OPFS)
Intra Autonomous Systems	Intra Autonomous Systems
Algorithm: Bellman-Ford	Algorithm: Dijkstra
Forwarding of (a copy of) the whole table (distance vector)	Forwarding of just the link updates
It does not require the network topology	It does require the network topology
Frequent periodic updates: slow convergence	Event-based updates: fast convergence – (link state information)

14.2.1 Link State

It requires a global knowledge of the network topology.
Each node exchanges local information at global level (with all nodes in the network) during the definition of shortest path with Dijkstra's algorithm.
Its aim is to define shortest path in the graph in this way:

1. Each node needs to know its neighbours and to identify them (HELLO packets)
2. Compute the cost of direct links (E.g. ECHO packets to compute delays)
3. Create a packet with this information (Link State Packet)
4. **Flooding (sort of multihop broadcast)**
   Disseminate the packet to every other router (after any changes on the link states)
5. Locally reconstruct the complete topology of the entire network
6. Compute the least-path with Dijkstra algorithm
7. Once a router has received the Link State Packet from every other nodes, it can reconstruct the complete graph of the network

14.2.1.1 Dijkstra's algorithm

It finds the "least cost" path from node A to any other node.
For all $w, v \in V$, let

- $d(w, v)$ be the cost of the direct link between w and v, if it exists, and $d(w, v) = \infty$ otherwise
- $D(v)$ be the cost of the path from node A to node v, according to the least-cost path currently computed by A
- X be the list of nodes for which the least-cost path has already been found

Algorithm 1 Dijkstra's algorithm

```
1:  procedure DIJKSTRA(graphG)
2:      X ← {A}
3:      ∀v ∈ X, D(v) = d(A, v)
4:
5:      while |X| < |V| do
6:          Y ← V − X
7:          w ∈ Y : D(w ← min_{z∈Y}{D(z)}        ▷ Find node with least-cost
8:          X ← X, w                              ▷ Add w to X
9:          ∀z ∈ Y, D(z) ← min{D(z), D(w) + d(w, z)}   ▷ Update minimum distances of all nodes
10:     end while
11:
12: end procedure
```

There are different criteria on which the Dijkstra Algorithm can be defined. The cost of each link, that connects node v to node w, can be defined as a composition of:

- **R(w,v):** transmission rate of the link
- $\mathbf{T_p}$**(w,v):** propagation delay over the link from w to v
- $\mathbf{P_{ok}}$**(w,v):** probability of successful delivery of a packet across the link

There are some criteria on which we can define a problem for Dijkstra:

- **Shortest path (min # of hops)**
  cost of each link should be fixed to the same constant value $d(w,v) = 1$

- **Shortest delay path**
  to find paths with minimum e2e delay the cost of each link should relfect the time taken by a packet to cross the link:

$$d(w,v) = L(w,v)/R(w,v) + T_{\mathrm{p}}(w,v)$$

  L(w,v) = reference packet size (e.g., MTU of w-v link)

- **Max reliable path**
  The overall reliability of a path is given by the product of the packet delivery rate of each single path:
  Path **v** =**e1,e2,. . . ,ek** $\Rightarrow$ Reliability(v) = $\prod_i P_{ok}(e_i)$
  The max reliability path can be found by setting:

$$d(w,v) = -log(P_{ok}(w,v))$$

- **Max long-term throughput path**
  Maximum long-term throughput (MLTT) of a path **v**=**e1,e2,. . . ,ek** is given by the minimum rate of the links:

$$MLTT(v) = minR(e_i), e_i inv$$

  This cost function cannot be replicated by any additive distance metric. . . but we can get close to it by magnifying the gaps among the rates of the links

$$d(w,v) = 1/R(w,v) n \forall n > 1$$

  we amplify the weight of the bottleneck link in each path, so that the least-cost path will likely correspond to that with max long-term throughput

The complexity in the *worst-case* is $\mathbf{O(V^2)}$. It can be reduced to $\mathbf{O(E\ log(V))}$ with a more efficient management of the tables.

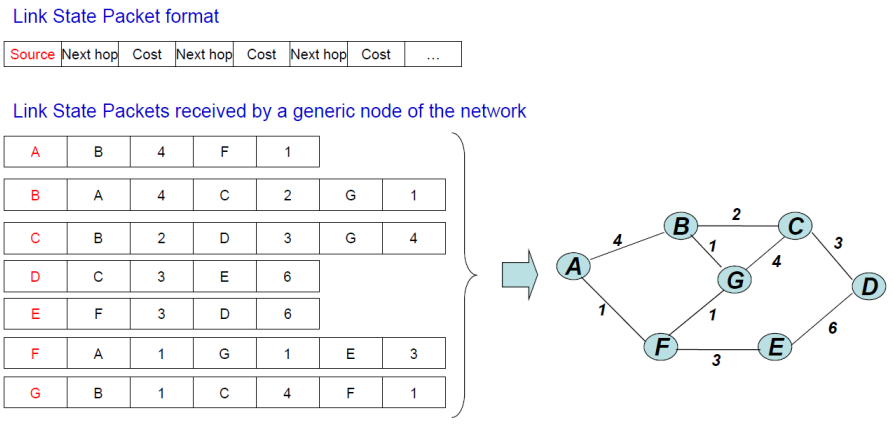


Figura 14.2: 1. Link State packets.

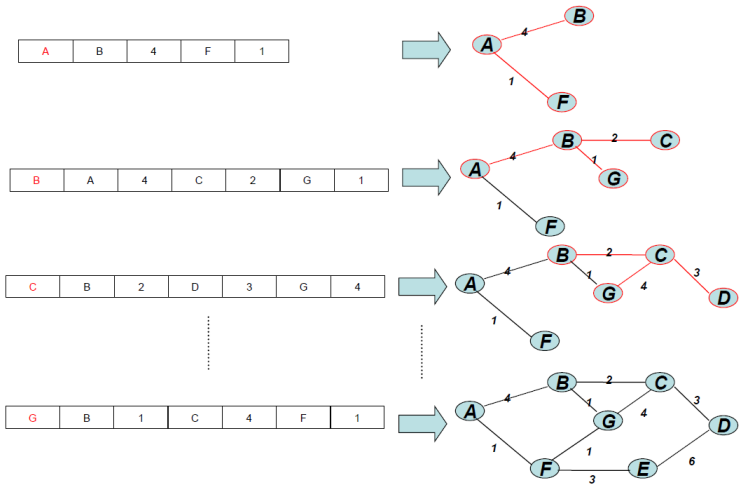


Figura 14.3: 1. Graph reconstruction.

Step	Φ	A	B	C	D	E	F	G
0	{A}	-	4-B	X	X	X	1-F	X
1	{A,F}	-	4-B	X	X	4-F	1-F	2-F
2	{A,F,G}	-	3-G	6-G	X	4-F	1-F	2-F
3	{A,F,G,B}	-	3-G	5-B	X	4-F	1-F	2-F
4	{A,F,G,B, E}	-	3-G	5-B	10-E	4-F	1-F	2-F
5	{A,F,G,B, E,C}	-	3-G	5-B	8-C	4-F	1-F	2-F

Figura 14.4: 1. Applocation of Dijkstra.

14.2.1.2 OSPF (Open Shortest Path First)

IG protocol for large AS. It was defined RFC 1247 and updated in RFC 1583.
There can be multiple routing tables for different ToS metrics and messages are directly encapsulated in IP packets.

OSPF areas

Area =Collection of linked networks, hosts, routers, all contained within an AS.
All areas inside an AS must be connected to Area 0 (**backbone**). Flooding of LS information is limited to each area (organizzation in hierarchical manner).
Some special routers (area border routers) summarize the topology information about their areas and send it to other areas through backbone.

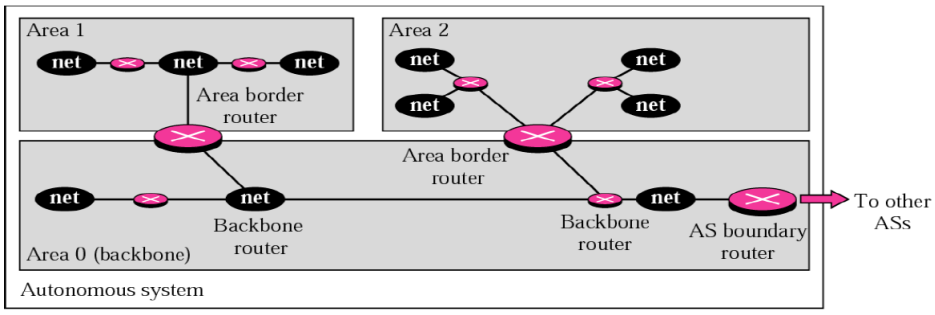


Figura 14.5: OSPF areas.

14.2.1.3 LSA (Link State Advertisement)

LSP (Link State Packet) is a packet that a router sends to other routers to construct the topology of the network. Each LSP is composed by several LSAs, that are messages(no packets) about the state of the link.

LSA Type 1: Router link

It's sent by each router and carries the IP address & Link Type of each link directly connected to the router. This LSA contains information about neighbouring routers.
The possible link types are:

1. **Point-to-point link to other router**
   specify the IP of the other router

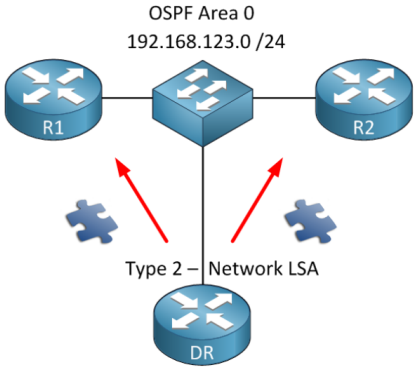
2. **Transient link**
   link to a network which allows transit traffic (at least 2 routers it's the first router). The message announces the IP address of the Designated Router (DR)

3. **Stub link**
   link to a network that does not admit transit traffic (stub network). The message announces the IP address of the stub network

LSA REMAINS WITHIN THE AREA

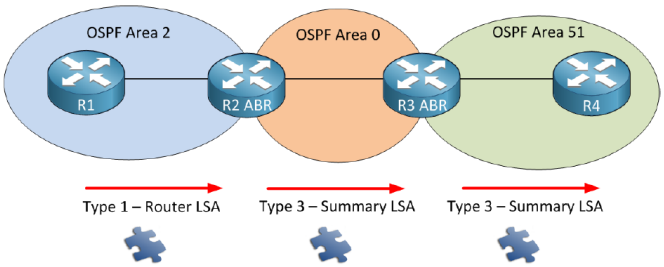
LSA Type 2: Network link

It's sent by the DR of a network to advertise itself.
The message carries the IP address of all the routers in the same network and netmask (e.g., R1, DR, R2 addresses & 255.255.255.0 netmask).
No costs are advertised because they will be sent by each router independently using the Router Link message.
LSA REMAINS WITHIN THE AREA



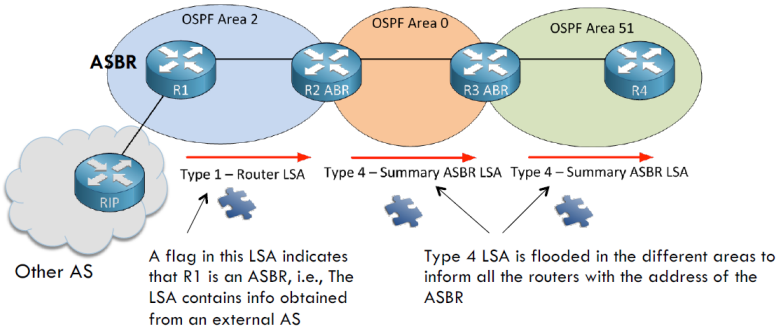
LSA Type 3: Summary link to network

Summary link to network. It's sent by an *area border router (ABR)* to advertise all the links collected by the backbone to an area, & vice versa.
It's used to communicate the reachability of other areas to internal routers.



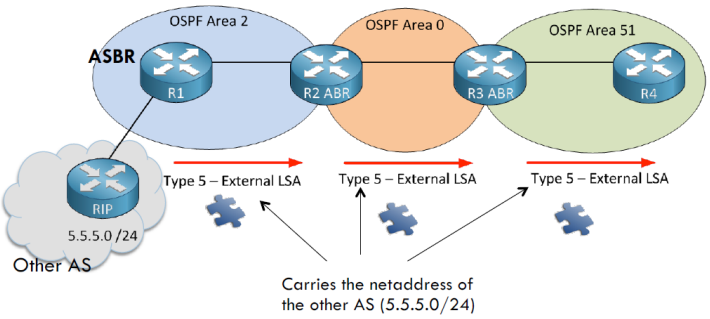
LSA Type 4: Summary link to AS

it's sent by an ABR to advertise the presence and the reachability of an AS Border Router (ASBR) (i.e., a router directly connected to other AS, in the different areas of the AS).



LSA Type 5: External link

It's sent by the ASBR (R1) to advertise the existence of a single network outside the AS to the backbone area of the current AS (to communicate external destination to routing domain).

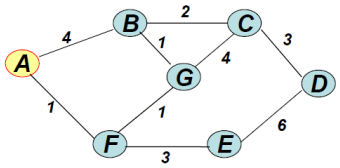


14.2.2 Distance vector

The least-cost path is computed in an iterative, distributed manner (Bellman Ford algorithm).
Each node holds:

- The list of its neighbouring nodes and the cost of the corresponding direct link
- The list of all the nodes of the network and the cost towards each of them (Distance Vector)
- The list of the nodes that represent the "next hop" in the current least-cost paths (forwarding table)

Each node exchange global information at local level (only with its neighbouring nodes). The global information is the cost for a node to reach another node (defined in its forwarding table) and shares its distance vector, generated from forwarding table, to neighbouring nodes.



Dest.	Distance	Next Hop
A	0	-
B	4	-
C	x	-
D	x	-
E	x	-
F	1	-
G	x	-

Distance Vector of A

Figura 14.6: Example of Distance vector of a node.

1. Initially, DV only contains neighbours at distance 1 hop:
   - HELLO packets to find the neighbors
   - ECHO packets to compute delays
2. Periodically or in case of DV changes, each node sends its own DV to its neighboring nodes only
3. When a node receives a DV from a neighbor, it updates its own DV using a distributed algorithm (Bellman-Ford)
4. If the DV of a node changes, then the node informs all its neighbors about its new DV
5. After a transient time, the DVs of the nodes converge to the actual least-cost paths

The update is done router by router but this process is very slow and there are different timers, set by routers, to do it:

1. Each router periodically sends a copy of its DV to its neighbors (**periodic timer**: random between 25s-35s)
2. If an entry is not updated within a certain time (**expiration timer**: 180 seconds) the cost to that destination is set to infinite
3. After another time (garbage collection timer: 120s), the entry is deleted

14.2.2.1 Bellman-Ford's algorithm

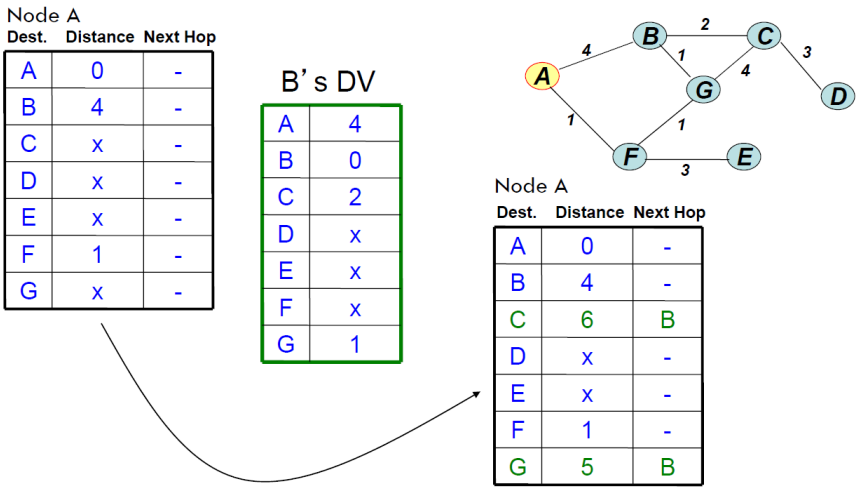
A node **A** receives the DV **"d"** from the neighbor **X**.

- For every destination , A updates the entry for w in the routing table according to the following rule n

$$D(A,w) = min\{D(A,w), D(A,X) + d(X,w)\}$$

- If the distance to w through X is shorter than the previous one, update the "next hop" field with X and the cost with

$$D(A,X) + d(X,w)$$



Count to infinity problem

The distance vectors are sent periodically to the neighbroughs. The nodes are not synchronized (they haven't the same clock).
The cost of the path to D grows to infinity but slowly.

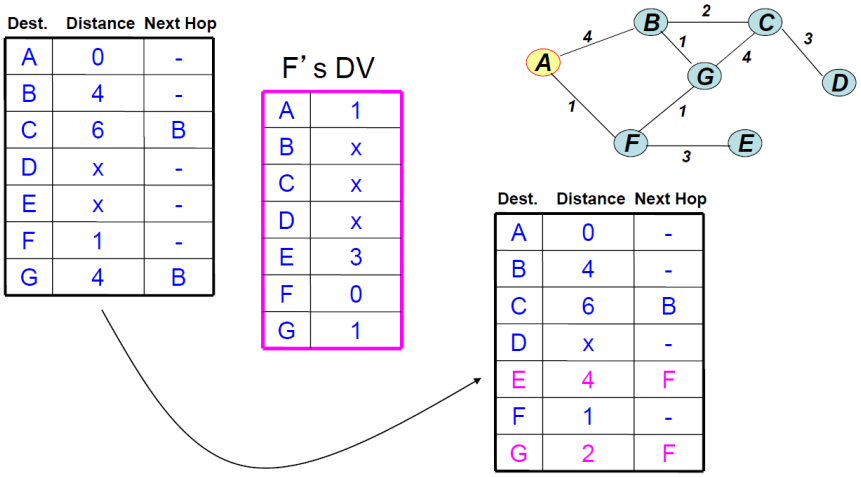
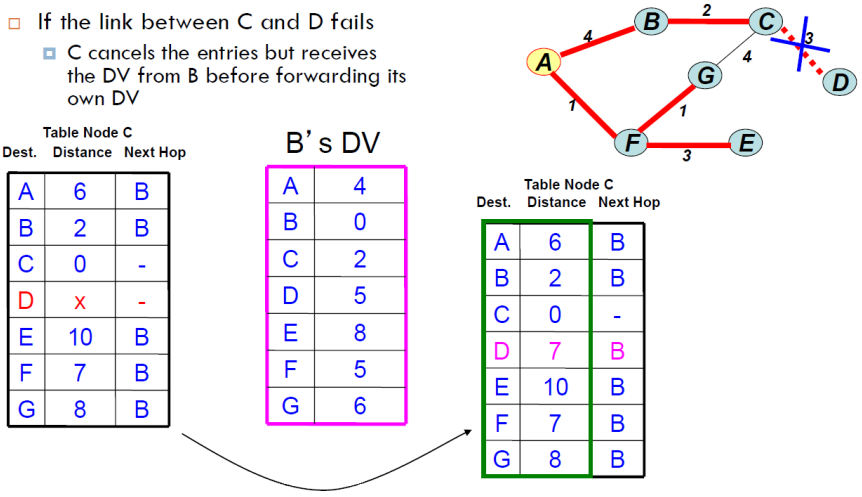
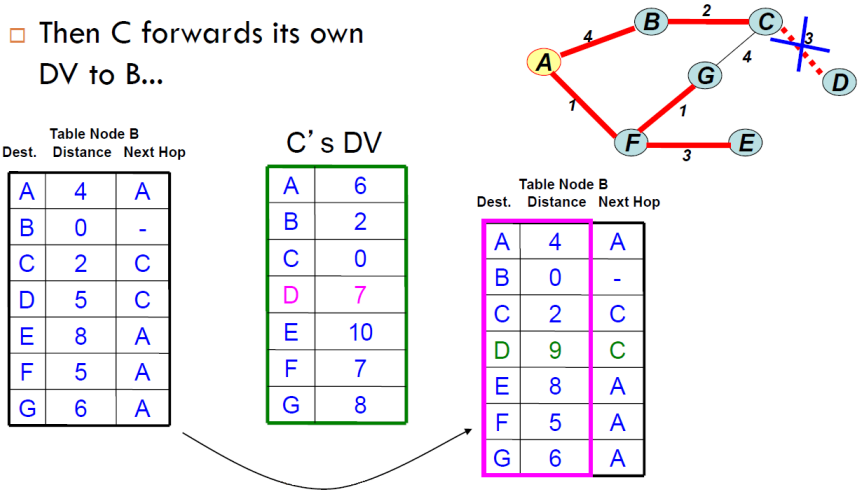


Figura 14.7: Example of Bellman-Ford updating.





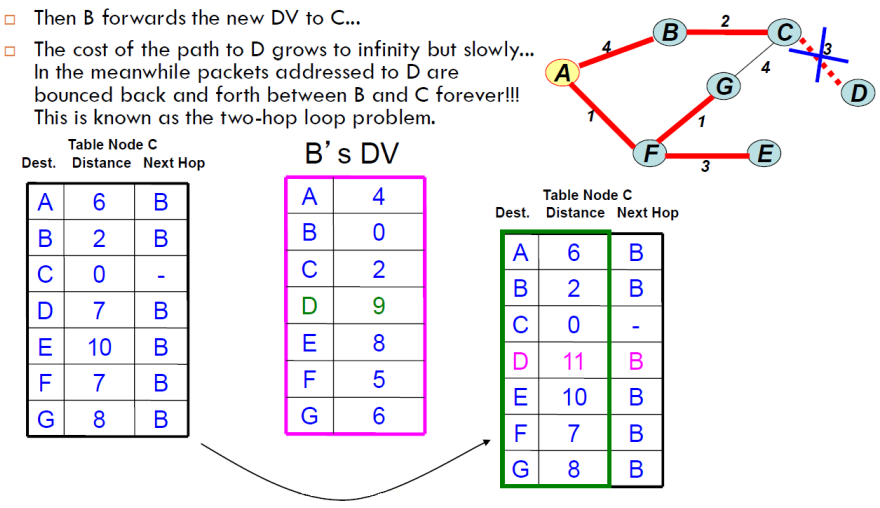


Figura 14.8: Example of count to infinity caused because the link between C and D fails.

Count to infinity solutions

- **Limited-infinity**
  It's fixed the max cost of reaching a node. If distance exceeds a max fixed integer value (E.g. 16), the cost of the path can be considered INFINITE.
  Drawback:this limits the max diameter of the network but anyway, there is a period in which the net is unreachable but the distance increases.

- **Split-horizon**
  Idea: forwards to each neighbor X only the DV entries that don't have X as next hop (E.g. B does not send to C the DV entry to D, since Next Hop is C).
  It prevents two-hop loop problems but NOT three hops or more.

- **Split-horizon & Poison-reverse**
  The node includes in its DV all the destinations but sets to infinity the distances to the nodes reachable through the links where it sends its DV.
  Receiving node knows that the path to that destination is through itself.
  (E.g. C does not know whether D is not advertised by B because of split-horizon or because it is not connected)

- **Hold down**
  When a router is advised that a network is not reachable, it ignores any incoming update for that destination for a certain time (60s).
  This prevents obsolete information to be considered as valid.
  Drawbacks:
  - it slows down reaction to topology changes
  - if a link is not available for only few seconds, the router block information streams to a node for more seconds

- **Triggered Update**
  Synchronous distance vector to nodes if a node is not available (topology changes are immediately notified and are differentiated from other notifications).
  Drawback: it can generate useless traffic if the link is down only for a short period (few seconds).

14.2.2.2 RIP (Routing Information Protocol)

RIP version 1

- *IG protocol*
  Designed at Berkeley (1982) and then defined in RFC 1058

- *Distance Vector*
  Bellman-Ford for the evaluation of the minimum cost paths

- *Cost metric*
  number of hops (hop count)

- *Maximum cost*
  limited to 15 hops (16 is considered infinity)

- *DV updates: periodic plus asynchronous*
  - **Periodic:** typically every 30 seconds
  - **Asynchronous:** any time network topology changes

- *Messages*
  RIP packets are encapsulated in UDP datagrams (assigned port: 520)

RIP version 2

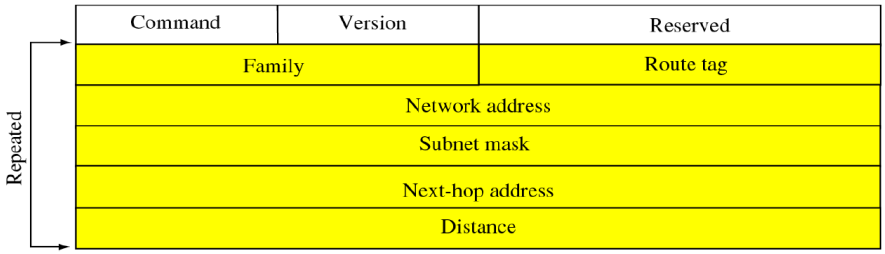


Figura 14.9: RIPv2 format.

Defined in *RFC 1723*.
There are some additional features (replacing the fields of RIPv1 that were filled with 0s with new fields):

- Connectivity information: route tag (ex. AS number or information derived from BGP) + next hop address
- **Authentication**
  Protection against unauthorized advertisements. No new fields are added, the code FFFF is entered instead:
  - *Authentication type (protocol used for authentication)*
  - *Authentication data*
- Classless routing (subnet mask)
- Multicasting: uses the address 224.0.0.9 (all RIP routers)

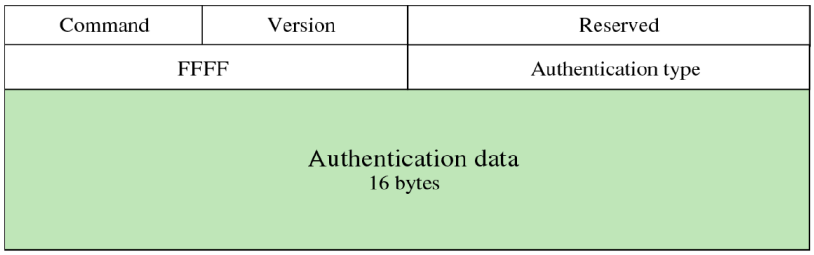


Figura 14.10: RIPv2 authentication.

14.3 Inter-AS routing

14.3.1 Border Gateway Protocol (BGP)

It's a protocol of External Gateway Protocol (EGP). There are two types of BGP protocol:

- **eBGP**
  is run by the AS border gateways only and used to propagate routing information between adjacent ASs. TCP connections for eBGP traffic are established between any pair of border gateways of different ASs that are directly connected.
- **iBGP**
  is run by the AS border gateways within a same AS and is used to propagate reachability information regarding other ASs.
  Propagation of routes could be made using interior gateway routing protocols (OSPF, RIP), but due to the potentially huge number of routes, this practice is strongly discouraged.
  TCP connections for iBGP are established between any pair of boarder routers in the same AS, even if they are not directly connected.

BGP enforces policies by choosing paths from multiple alternatives and controlling advertisement to other AS's. Some examples of BGP Policies are:

- **A multi-homed AS refuses to act as transit**
  Limit path advertisement
- **A multi-homed AS can become transit for selected AS's only**
  Only advertise paths to some AS's
- **An AS can favor or disfavor traffic transit from certain ASs**
  e.g, advertising longer than needed paths to other destinations

BGP messages are:

- **Open**
  It announces AS ID and specifies hold timer (interval between keep_alive or update messages, zero interval implies no keep_alive)
- **Keep_alive**
  It's sent periodically (but before hold timer expires) to peers to ensure connectivity. It's sent in place of an UPDATE message.
- **Notification**
  Used for error notification, that closes immediately TCP connection.
- **UPDATE message**
  it can contain:
  - List of withdrawn routes
  - Network layer reachability information (List of reachable prefixes)
  - Path attributes (origin, path, metrics)

The path selection criteria are attributes and (policy) information. **Examples:** hop count, policy considerations (preference for AS, presence or absence of certain AS), path origin and link dynamics.

The BGP is designed to:

1. Provide hierarchy that allows scalability
2. Allow enforcement of policies related to structure

The mechanisms, to do it, is **path vector-scalable**, hides structure from neighbors, detects loop quickly.

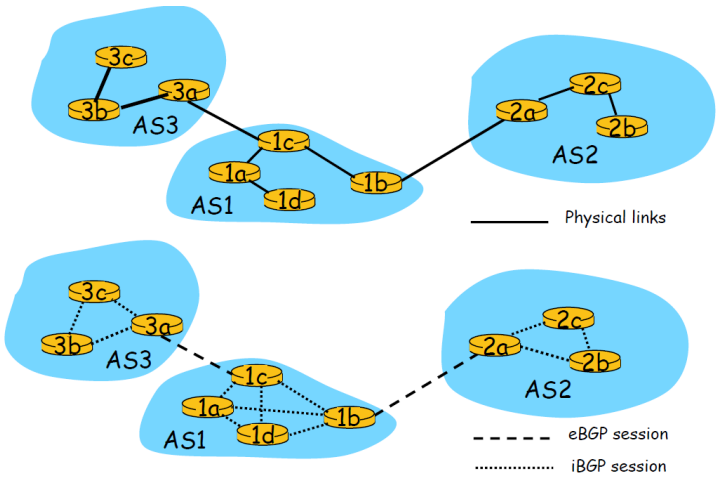


Figura 14.11: Example of eBGP & iBGP connections.

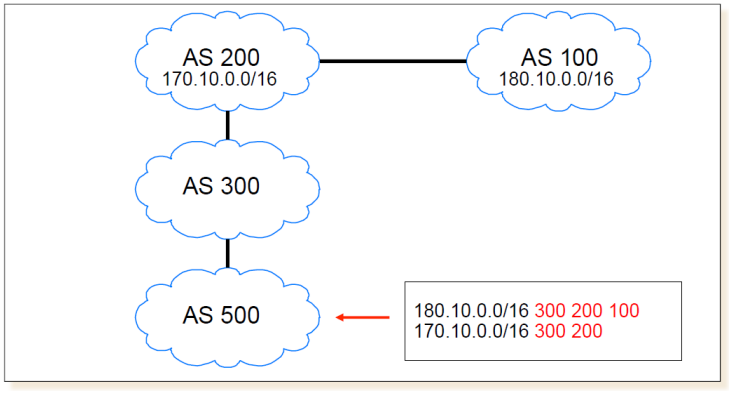


Figura 14.12: Example of AS_Path: List of traversed AS's.

14.3.2 Distance Vector with Path

It's the algorithm used for Inter-AS routing. Each routing update carries a Path Vector (PV) that specifies the entire path from source AS to each destination AS.
When AS gets route, it checks if AS already in path:

- **If yes, then**
  reject route→loop-free paths (no count to infinity problem!)
- **If no, then**
  checks whether current path cost is larger than new one
  - **If yes, then**
    add self and (possibly) advertise route further
  - **If no, then**
    keep current routing table

Path Vector (PV) contains the whole inter-AS path to reach the destination AS. BGP advertises to neighbors only those routes that it uses

- Consistent with the hop-by-hop Internet paradigm
- e.g., AS1 cannot tell AS2 to route to other AS's in a manner different than what AS2 has chosen

Network	Next router	Path
N01	R01	AS2,AS5,AS7,AS12
N02	R07	AS4,AS13,AS6,AS9
N03	R09	AS11,AS12,AS8,AS6
...	...	...

Figura 14.13: Example of Path vector.

Capitolo 15

Transport layer

There are two main protocols defined in this layer:

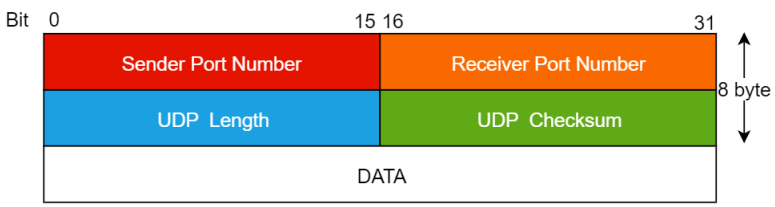
- **TCP**
  reliable, in-order delivery with control
- **UDP**
  unreliable, no-ordered delivery without control

Each application is associated to a 16 bit port number. When a process is started, the operating system assigns a port number and creates an input and an output queues.
This assignment is unique so if you try to add another process with same port and protocol the Operating System returns an error.
Messages, generated by the process, are sent to the output queue to be fetched by transport service entities for transmission.
Service transport entities copy arriving messages addressed to the specific port number to the associated input queue, where they are read by the intended application.
If input queue overflows, or the dst port number is not associated to any active application, an ICMP message of type "port unreachable" is generated .

15.1 User Datagram Protocol (UDP)

It's **connectionless** protocol (no connection establishment between sender and receiver) which can add delay. It's a simple protocol because there isn't a connection state to sender, receiver and it's composed by small segment header.
There isn't congestion control and retransmission in case of packet loss hence the connection is faster than TCP. For example, in DNS server, it's used a very small (small packets and in a short time period) UDP connection. One of the main application of UDP is multimedia transmission, in which we don't care about the delay of the network or packet loss but we continue to transmit packets anyway. There aren't rate limitations because of no throttling due to congestion and flow control mechanisms and no retransmission policy.

15.1.1 UDP packet format

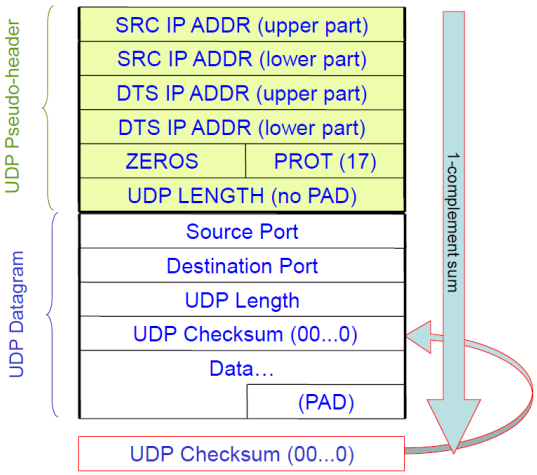


UDP packets are named **"UDP datagram"**. Its header is 8 bytes long and composed of:

- **Sender port number**
- **Receiver port number**

7	**Echo**	Echoes a received datagram back to the sender
9	**Discard**	Discards any datagram that is received
11	**Users**	Active users
13	**Daytime**	Returns the date and the time
17	**Quote**	Returns a quote of the day
19	**Chargen**	Returns a string of characters
53	**Nameserver**	Domain Name Service
67	**Bootps**	Server port to download bootstrap information
68	**Bootpc**	Client port to download bootstrap information
69	**TFTP**	Trivial File Transfer Protocol
111	**RPC**	Remote Procedure Call
123	**NTP**	Network Time Protocol
161	**SNMP**	Simple Network Management Protocol
162	**SNMP**	Simple Network Management Protocol (trap)

- **UDP length**
  datagram length (header+dati) in bytes
  - *Min:* 8
  - *Max:* 65535 bytes, but limited to 65,515 bytes to leave space for 20 byte IP header
- **UDP checksum** (optional)
  The sender:
  1. treats segment contents as sequence of 16-bit integers
  2. checksum: addition (1's complement sum) of segment contents
  3. sender puts checksum value into UDP checksum field



The computation of the UDP checksum is made by pre-pending **pseudo-header** information from IP layer, that includes source and destination IP addresses.
This computation is made to improve control in smaller networks, of which you don't know the topology and the organization.
Checksum is set to zero for computation and then updated with computed and returned value.

If you don't want to use checksum field, then the field is filled with zeros.
If the checksum is used, but its value turns out to be zero, it is replaced with all 1s, which is equivalent to the all zero word in 1's complement sum.

Now a better algorithm is used: Fletcher. This guarantees an error-detection capacity:

- All single errors
- All double errors
- Error bursts shorter than 16 with probability 0.99992
- Error bursts longer than 16 with probability 0.9985

15.2 Transport Control Protocol (TCP)

It's full-duplex data streaming in which connection is identified by the couple <**Src Socket, Dst Socket**> and it's bi-directional (full duplex) connection.
It's **connection oriented**, because the connection is managed through three phases:

- **Connection set-up:**
  three-way handshake
- **Data transmission:**
  byte stream (as if a dedicated circuit were established between source end destination)
- **Connection close:**
  close the connection (actually, no in-network resources need to be released, only resources at end-nodes)

15.2.1 TCP segment

Any TCP/IP data or ack packet is called "segment". For every segment we can define **Maximum Segment Size(MSS)**, that is the size of the largest segment that can be transmitted by the sender (SMSS) and receiver (RMSS), in bytes, and without counting TCP and IP headers and options.
Hence we can define Full-sized segment as segment with size equal to the MSS. TCP header is composed by

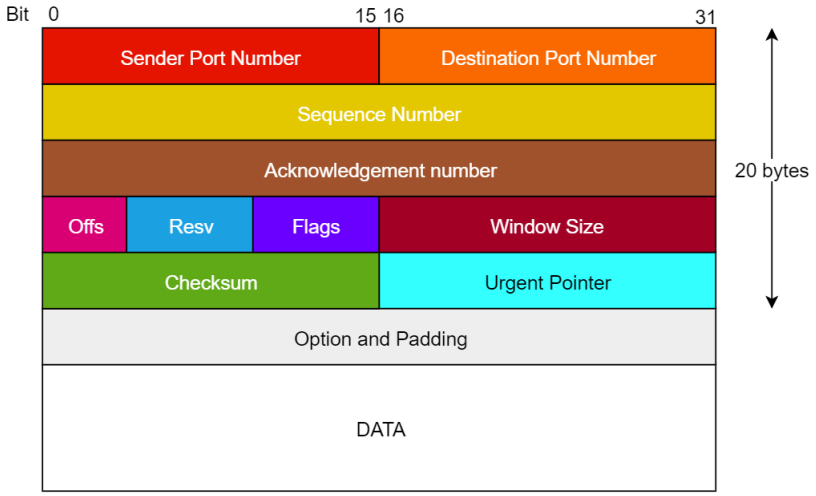


Figura 15.1: TCP segment format.

some fields:

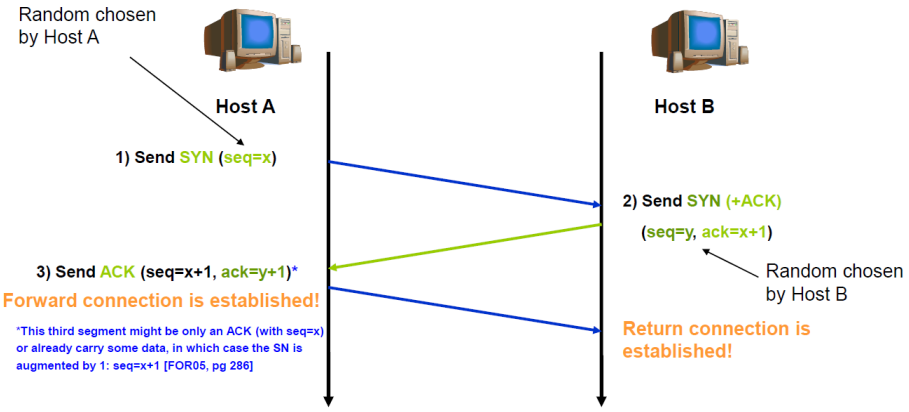
- **Source port**

7	**Echo**	Echoes a received datagram back to the sender
9	**Discard**	Discards any datagram that is received
11	**Users**	Active users
13	**Daytime**	Returns the date and the time
17	**Quote**	Returns a quote of the day
19	**Chargen**	Returns a string of characters
20	**FTP, Data**	File Transfer Protocol (data connection)
21	**FTP, Control**	File Transfer Protocol (control connection)
23	**TELNET**	Terminal Network
25	**SMTP**	Simple Mail Transfer Protocol
53	**DNS**	Domain Name Server
67	**BOOTP**	Bootstrap Protocol
79	**Finger**	Finger
80	**HTTP**	Hypertext Transfer Protocol
111	**RPC**	Remote Procedure Call

- **Destination port**
- **Sequence number (32 bit)**
  Number of the first byte in the segment with respect to the beginning of the connection
- **Acknowledgment number(32 bit)**
  Number of the next byte expected by the receiver (this field is valid if ACK bit in Flag field is set)
- **Offs (4 bit): Offset**
  Beginning of the data field (in multiples of 32 bits) from the end of header without options (length of the OPTION field, expressed in multiples of 32 bits)
- **Res (6 bit)**
  Reserved for future use
- **Flags (6 bit)**
  - **URG: Urgent Field Valid**
  - **ACK: Acknowledgment Field Valid**
    If set, the packet will be an ACK
  - **PSH: Push Request**
    If set, segment is immediately sent by the sender and delivered to upper layer at receiver (no buffering delay)
  - **RST: Reset Connection** If set, restart the connection immediately
  - **SYN: Initial Synchronization** If set, segment is sent for the initially synchronization of the TCP connection
  - **FIN: Last Segment**
    If set, segment is sent for the closing of the TCP connection
- **Window Size (16 bit)**
  available buffer space at the receiver side [bytes]
- **Checksum (16 bit)**
  it's computed in the same way of UDP connection checksum
- **Urgent Pointer (16 bit)**
  it's the last byte of the data block that needs to be passed to the application right after reception (no buffering delay at receiver) and signaled as **"urgent" (URG)**.
  It's not often used.

- **Options (variable)**
  it can be used by TCP peer entities to exchange control info (e.g. MSS, MTU,..)

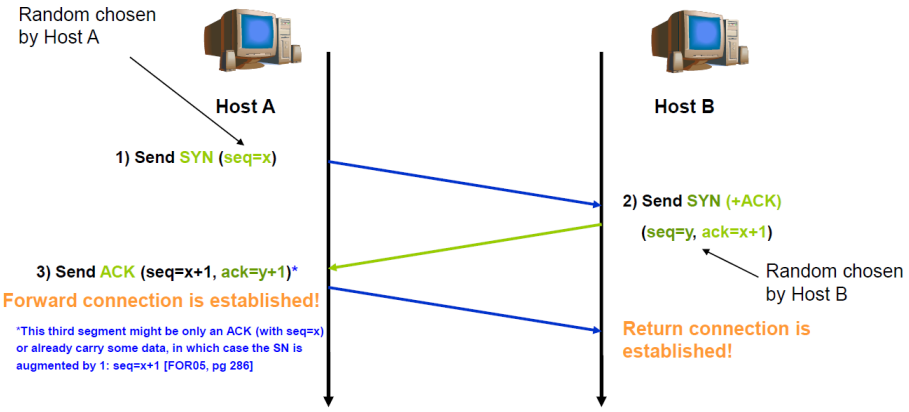
15.2.2 Connection set-up



Three-way handshaking

1. The client sends a packet with SYN=1 and identified by a SN=X (X defined in a random way by client)
2. The server replies with a packet with SYN=1 and ACK=1 and identified by SN=Y (Y defined in a random way by server) and ACKN=X+1
3. The client replies with ACK packet with ACKN=Y+1 and:
   - **SN=X** if doesn't carry data
   - **SN=X+1** if carries some data

15.2.3 Bidirectional transmission



15.2.4 Connecting closure

There are two possible approaches:

- **Three-way handshaking (most common)**

1. TCP client sends FIN segment (FIN flag in TCP header is set)
   FIN does consume a sequence number because it needs to be ACKed
2. TCP server gets the FIN e replies with FIN+ACK
3. TCP client sends ACK and closes the session in both directions

- **Four-way handshaking with half-close**

1. TCP client sends FIN segment with SQN=x' (FIN flag in TCP header is set)
2. TCP server gets the FIN e replies with ACKN=x'+1 and sends the rest of that data which has request previously (TCP client can reply only with ACK packets with same SQN=x' and can't sent new data).
3. When TCP server finishes to send packets, it sends the last packet with SQN=z and FIN flag
4. TCP client sends ACK with ACKN=z+1 and server immediately understands that connection must be closed and closes it (because in the previous step server has sent FIN message and doesn't have packet with SN=z+1)

15.2.5 Retransmission Time Out (RTO)

It's the maximum time that establish if a packet is delivered or not. RTT is largely variable with the extension and type of networks. RTO can't be fixed otherwise:

- **If RTO«RTT**
  retransmit packets that were successfully delivered
- **If RTO»RTT**
  each packet loss has strong impact on throughput

15.2.5.1 Adaptive RTO

It is one of the possible solutions to the definition of the RTO. It is based on:

1. **Estimate RTT statistics**
   - Mean→mRTT
   - Standard deviation→sRTT
2. **Set RTO based on estimated RTT statistics**
   - "pretend" RTT is a normal rv with mean mRTT and std sRTT and fix RTO such that Pr[RTT>RTO] is small
   - RTO = mRTT + 4 sRTT

15.2.5.2 RTO in pratice (*RFC 6298*)

mRTT: moving average mean RTT estimate
sRTT: moving average RTT standard deviation estimate
aRTT: actual RTT measurement
K = 4
beta = 1/4
alpha = 1/8
G* = granularity (typically 1s)

The algorithm is based on:

1. Start with $RTO \geq 1$ seconds (typically 1^{st})

2. When you get the first aRTT value, the host MUST set
   mRTT←aRTT
   sRTT← $\frac{aRTT}{2}$
   RTO←mRTT + K*sRTT

3. When a new aRTT' measurement is available, a host MUST set
   mRTT←(1 - alpha)*mRTT + alpha*aRTT'
   sRTT←(1 - beta)*sRTT + beta*|mRTT - aRTT'|
   RTOideal←mRTT + max(G,K*sRTT)
   RTO←min(max(60,MaxRTO^S), max(1$^\#$,RTOideal))

15.2.5.3 Karn's algorithm

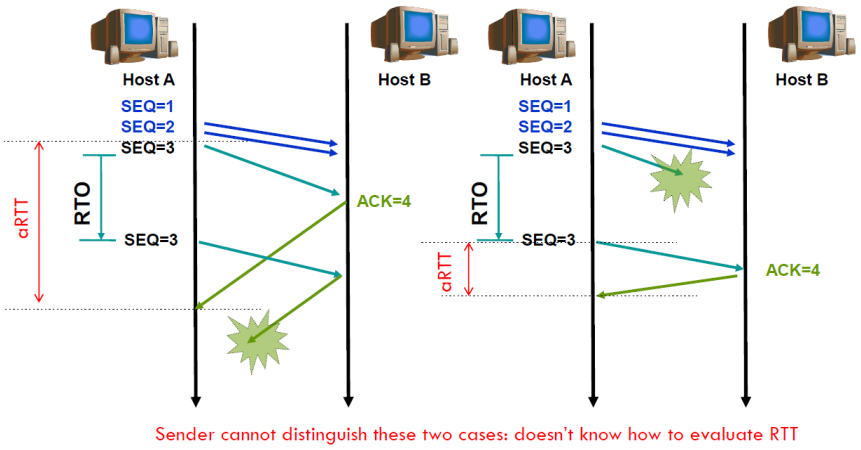


Figura 15.2: Ambiguous case in computation of RTO

The Figure 15.2 shows an ambiguous case in the computation of RTO with previous algorithm.
The **first solution** can be to choose RTT only for non-retransmitted segments.
The problem of this solution is that if actual RTT suddenly increases (e.g. because of a change of routing path)RTO cannot be updated because initially RTO is too small, and all packets are dropped ad it should be retransmitted.

The definitive solution can be found in **Karn's algorithm** that is based on:

- Only consider RTT of non-retransmitted segments
- **Exponential Backoff** double RTO at each retransmission, until RTT is adjusted and RTO can be computed in the usual manner
- Use this RTO also for new segments until a valid RTT measurement is available
- When unambiguous ACK is received, reset RTO using standard formula

15.2.5.4 RTO management

The following is the RECOMMENDED algorithm for managing the retransmission timer in RFC 6298:

1. Every time a packet containing data is sent (including a retransmission), if the timer is not running, start it running so that it will expire after RTO seconds (for the current value of RTO)
2. [...]

3. When an ACK is received that acknowledges new data, restart the retransmission timer so that it will expire after RTO seconds (for the current value of RTO)

4. Retransmit the earliest segment that has not been acknowledged by the TCP receiver

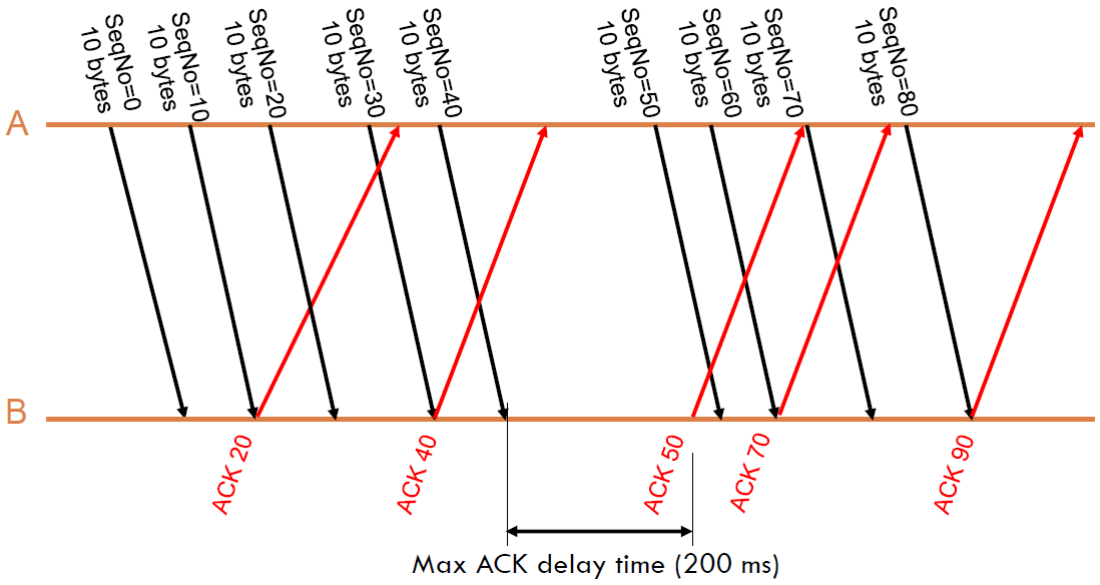
5. The host MUST set $RTO \leftarrow min(RTO * 2, MaxRTO)$

6. Start the retransmission timer, such that it expires after RTO seconds (for the value of RTO after the doubling operation outlined in 5)

7. If the timer expires awaiting the ACK of a SYN segment and the TCP implementation is using an RTO less than 3 seconds, the RTO MUST be re-initialized to 3 seconds when data transmission begins (i.e., after the three-way handshake completes)

15.2.6 Delayed Acknowledgement

Our goal is to try to exploit piggy backing. Wait for some time before returning an only-ACK packet.
Delay ACK i usally 200 ms long (with max 500 ms). If in the meantime new data become ready for transmission, ACK and data are sent together (piggybacking).
Because of delayed ACKs, an ACK is often observed for every other segment.



Event at receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Waits up to 500ms for next segment. If no next segment, sends ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately sends single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq #. Gap detected	Immediately sends duplicate ACK, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediate sends ACK, provided that segment starts at lower end of gap

15.2.7 Flow Control

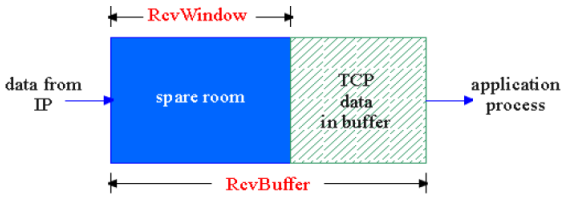
We suppose TCP receiver discards out-of-order segments. hence spare room in buffer in bytes is

$$RcvWindow = RcvBuffer - [LastByteRcvd - LastByteRead]$$

The receiver advertises spare room by including value of **RcvWindow** in segments

- When buffer is full, an ACK with zero-length RcvWindow field is returned to sender
- New ACK is generated when the buffer is (even partially) emptied by receiver application

Sender limits unACKed data to RcvWindow to guarantee receive buffer doesn't overflow.



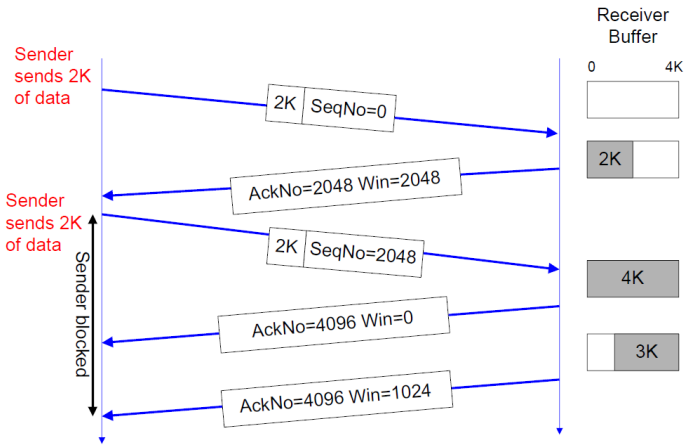


Figura 15.3: Example of receiver buffer working.

15.2.7.1 Deadlock

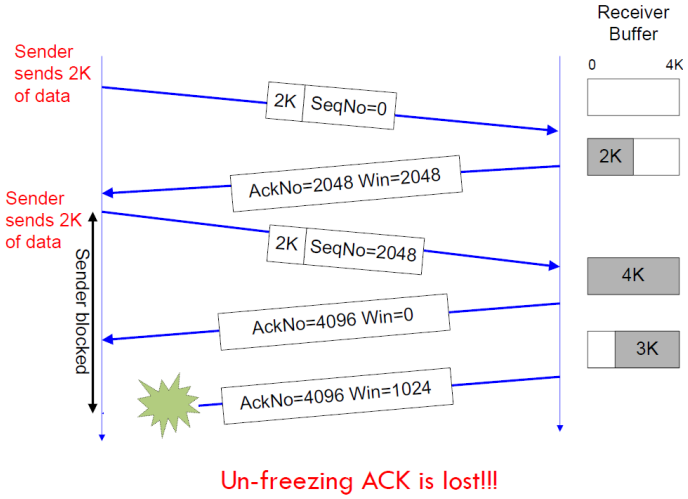


Figura 15.4: Example of deadlock problem.

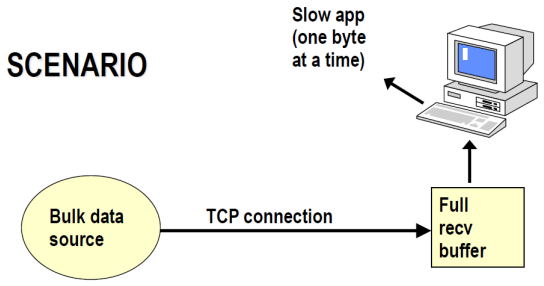
When *RcvWindow=0* (blocked sender), sender starts a **persist timer** (initially 500ms but depends on implementation).
Persist timer is restarted any time a segment with *RcvWindow=0* is received.
If persist timer expires, sender transmits "probe" segment of 1byte payload.

- **If receiver is still full**

  1. byte is dropped
  2. no ACK is returned
  3. sender double persit timer (up to 60 seconds)

- **Otherwise**
  receiver sends ack with actual available window

15.2.7.2 Silly window syndrome



This problem was discovered by David Clark (MIT, 1982) and is caused by a **slow receiver**.
It is solved, by preventing receiver to send a window unpdate for 1 byte.

- **Updating rule**
  1.receiver buffer can handle a whole MSS
  or 2.half received buffer has emptied(if smaller than MSS)

- **Sender rule(supplement rule)**
  sender can wait for sending data when window is low

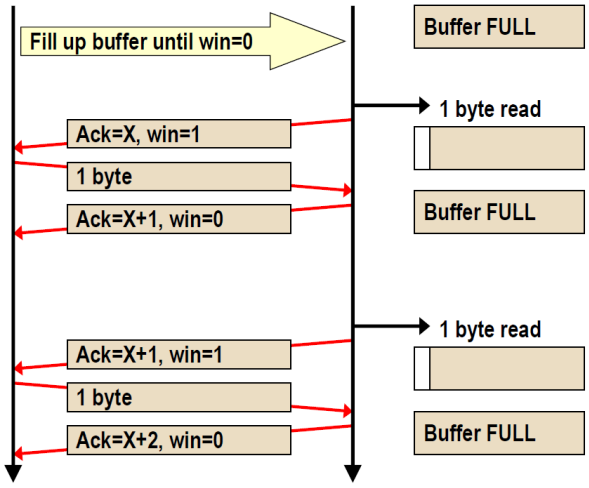


Figura 15.5: Silly window syndrome: slow receiver.

15.2.7.3 Tiny datagram

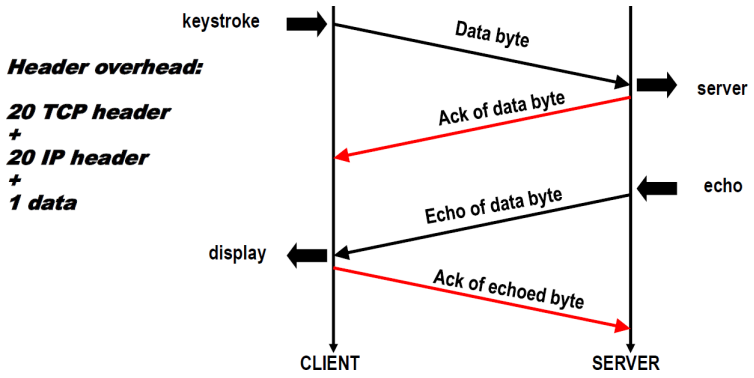


Figura 15.6: Tiny datagram problem: slow sender.

It is caused by a **slow sender**.

Algorithm 2 Nagle's algorithm

```
1: if data in transmission buffer < MSS and there is unconfirmed data still in the pipe then
2:     wait for ACK
3: else
4:     send data immediately
5: end if
```

15.2.7.4 Pipeline

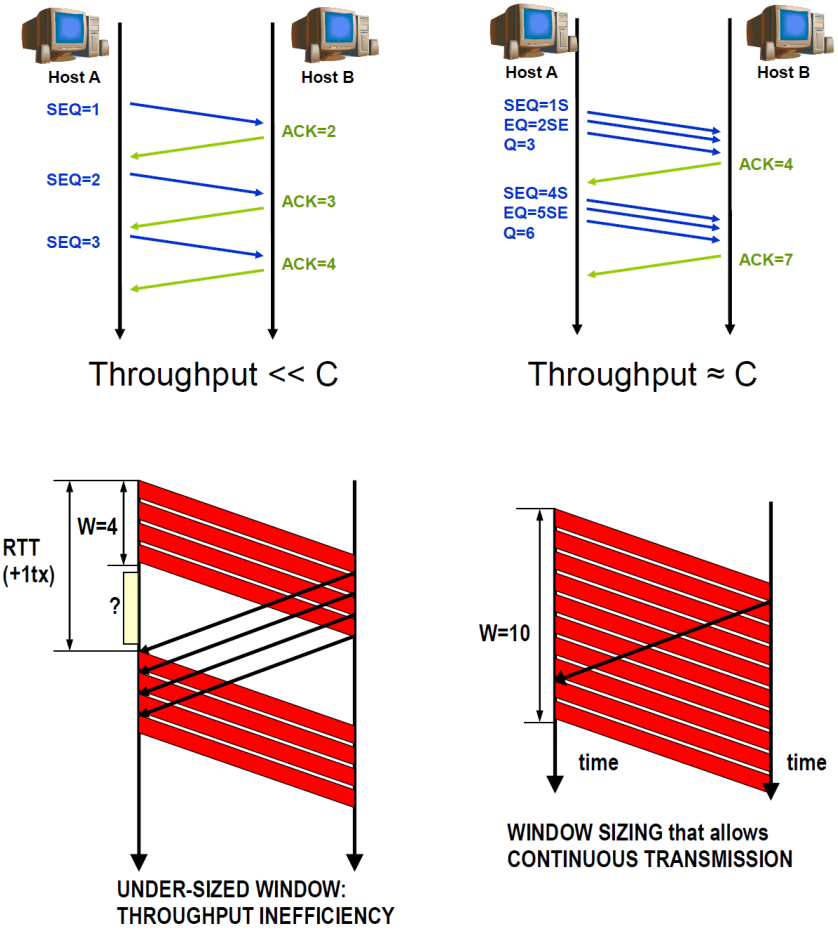
TCP performance depends on the product of the transfer rate and the round-trip delay. TCP performance problems aris when $bandwidth * delay$ is large. This happens in **LFN: "long, fat network"** (pronounced "elephan(t)").

Examples:

- DS1-speed satellite channel has a $bandwidth * delay$ of 1Mbit or more (100 outstanding TCP segments of 1200 bytes each)

$bandwidth * delay =$ amount of data that fill the pipe
buffer space required at sender and receiver to
obtain maximum throughput (keep the pipeline full)

- Terrestrial fiber-optical with a cross-country delay of 30 ms at a DS3 bandwidth (45Mbps) also exceeds 1Mbit



$$W\frac{MSS}{C} > RTT \Rightarrow W * MSS > RTT * C$$

If I don't do anything, the MWS is limited to the size defined by protocol and not real capacity of the device.

NETWORK	RTT (ms)	rate (kbps)	BxD (bytes)
Ethernet	3	10.000	3.750
T1, transUS	60	1.544	11.580
T1 satellite	480	1.544	92.640
T3 transUS	60	45.000	337.500
Gigabit transUS	60	1.000.000	7.500.000

Figura 15.7: Long Fat Network.

The TCP header uses a 16 bit field to report the receive window size to the sender. The largest window that can be used is 216 65K bytes but it is not enough to fill the pipe of LFNs.

"Window Scale" TCP option solves the problem. This option may be sent in an initial <**SYN**> segment (i.e., a segment with the SYN bit on and the ACK bit off).
It may also be sent in a <**SYN,ACK**> segment, but only if a Window Scale option was received in the initial <**SYN**> segment.
A Window Scale option in a segment **without a SYN bit** should be **ignored**.

15.2.8 Congestion control

Core network nodes don't run TCP but use only IP packets. So e2e TCP border nodes can control directly congestion slowing traffic but this doesn't happen in core networks.
First, we give some definitions:

- **RWND: Receiver Window**
  The most recently advertised receiver window
- **IW: Initial Window**
  Size of the sender's congestion window after the three-way handshake
- **LW: Loss Window**
  Size of the congestion window after sender detects loss using its retransmission timer
- **RW: Restart Window**
  Size of the congestion window when TCP restarts transmission after an idle period
- **FLIGHT WINDOW SIZE**
  Amount of data that has been sent but not yet acknowledged
- **DACK: Duplicate ACK segment**
  An ACK segment with the following characteristics
  1. The receiver of the ACK has outstanding data ($flightsize > 0$)
  2. The ACK does not carry any data
  3. SYN and FIN flags are both off
  4. ACK number is equal to the greatest ACK received so far
  5. The advertised WINDOW in the ACK segment equals the last received

Sender keeps a **"congestion window"** (**cwnd**) which defines how many bytes can be transmitted before stopping to wait for ACKs.
When cwnd is full, new bytes can be transmitted only upon reception of ACK that confirms the delivery of oldest bytes in the congestion window and makes the window slide forward (**Self-clocking**).

15.2.8.1 Slow Start/ Congestion Avoidance

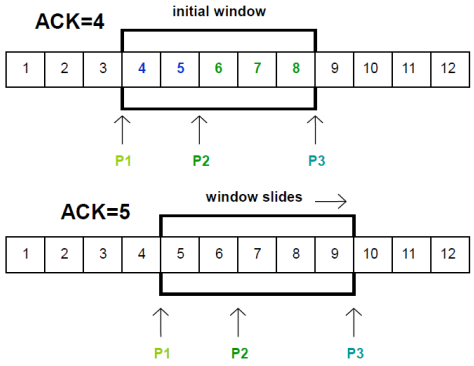
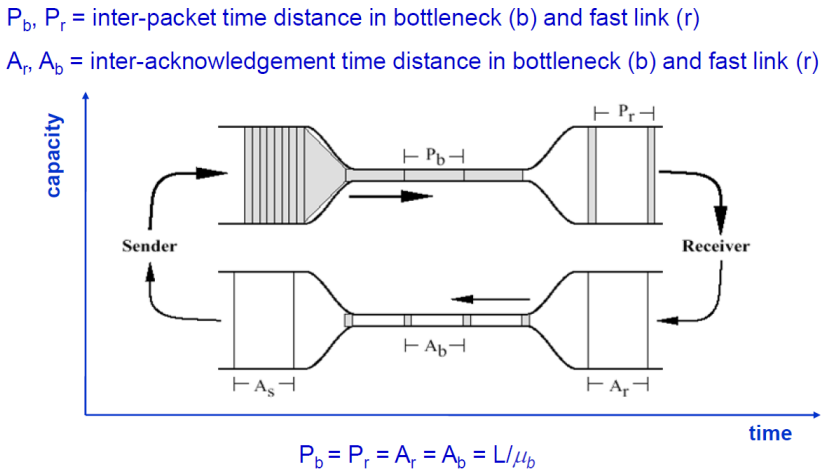


Figura 15.8: Sliding Window.



Consider a connection with n-hops, having rates μ_i $[\frac{bit}{s}]$ $\forall i = 1, 2, \ldots, n$.
Let $\mu_b = \min \mu_i$ be the bottleneck rate, and the pipe-capacity (in # of packets of size MSS) $W_p = \min_{i=1,\ldots,n} m_i \frac{RTT}{MSS}$.
If allowed window is equal to W_p, we have max throughput.
We don't know exactly RTT and μ_b at the beginning because we haven't yet estimate them.
If W_p ere known, TCP cannot transmit all segments in a batch since this may yield congestion at bottleneck.

Solution algorithm is composed of two phases:

1. *Slow Start* (**RFC 5681**)

   - At connection set up, start cwnd= 2 MSS packets (or greater)
   - For each received ACK, add 1 MSS to the cwnd

   Hence command doubles at every RTT. The comman window grows exponentially over time.

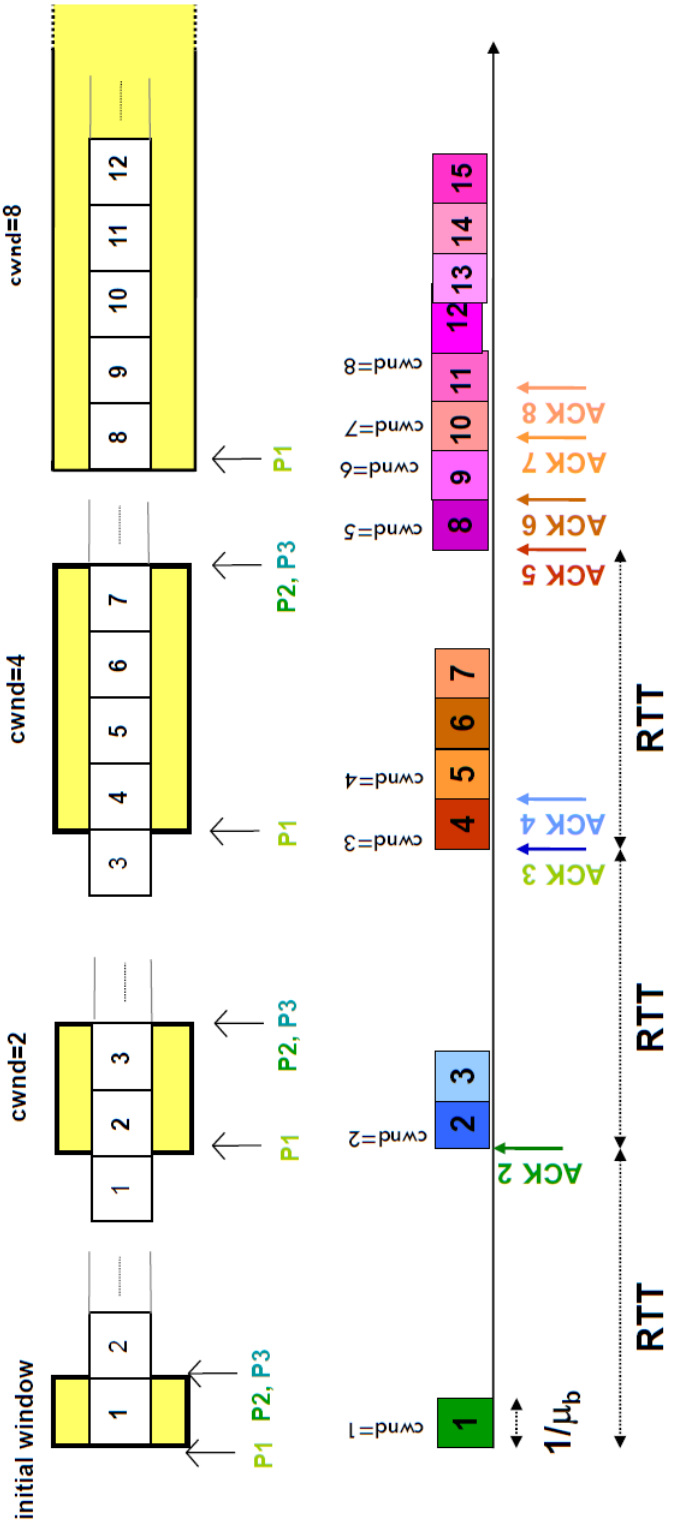


Figura 15.9: Example of Slow Start.

2. *Congestion avoidance*

   - When $cwnd > ssthresh$ TCP slows down the rate of increase of cwnd
   - 1^{st} **mode**At every ACK, cwnd is incremented by $\frac{MSS^2}{cwnd}$

     Example
     strarting from $cwnd(i)$ and receiving two more ACKs

$$cwnd(i+2) = cwnd(i+1) + \frac{MSS^2}{cwnd(i+1)} =$$

$$= cwnd(i) + \frac{MSS^2}{cwnd(i)} + \frac{MSS^2}{cwnd(i) + \frac{MSS^2}{cwnd(i)}} = cwnd(i) + 2\frac{MSS^2}{cwnd(i)}$$

     In a RTT the sender will receive k=cwnd/MSS ACKs

$$cwnd(i+k) = cwnd(i) + k\frac{MSS^2}{cwnd(i)} = cwnd(i) + \frac{cwnd(i)}{MSS}\frac{MSS^2}{cwnd(i)} = cwnd(i) + MSS$$

   - 2^{nd} **mode**Count the number of bytes ACKed since the last increment of cwnd (**SMSS**)

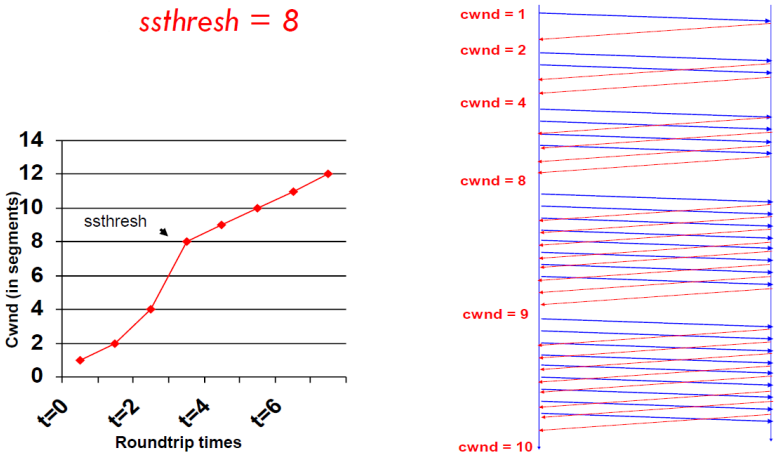


Figura 15.10: Congestion window and Slow start.

15.2.9 Reaction to segment loss

Congestion avoidance ends when a segment is lost. Different versions of TCP reacts in different ways to such an event:

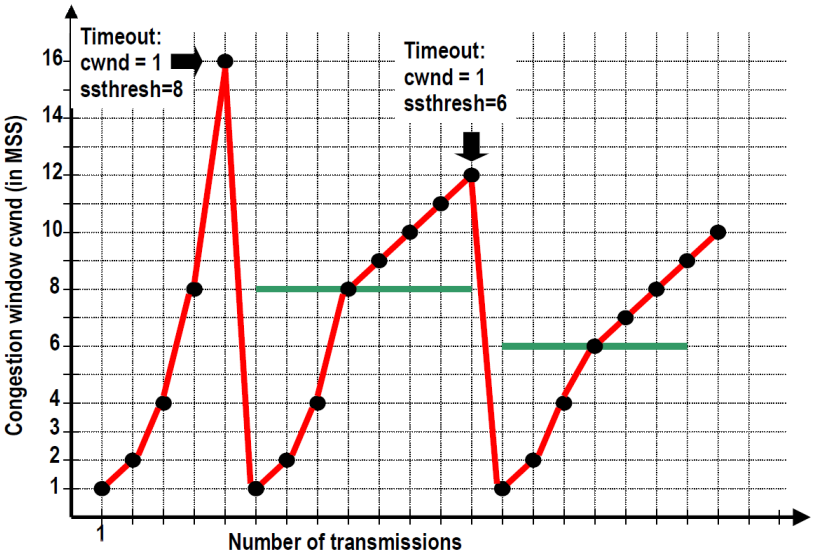
- **Classic TCP (TCP-old Tahoe)**
  Slow start & congestion avoidance after each RTO
- **TCP-Reno**
  Fast retransmit(after 3 DACKs) + Fast recovery
- **TCP-Tahoe**
  As old-Tahoe, but retransmission triggered also by 3 DUPACKs (fast retransmit)
- **TCP Sack**
  Selective acknowledgment

All the TCP protocols are based on the consideration that the receiver doesn't know and doesn't take care of which TCP protocol was used.

15.2.9.1 TCP-old Tahoe)

When there isn't ACK and RTO expires, this situation means that the segment is lost.
To react to this situation, we must decrease the flow:

1. $ssthresh = max(\frac{FlightSize}{2},\ 2 * SMSS)$
2. $cwnd = SMSS$ (size of the Loss Window)
3. Retransmit the first not-ACKed segment using slow start
4. Restart Retransmission Timer with $RTO = min(2 * RTO, 3)$
5. When $cwnd > ssthresh$ enters congestion avoidance



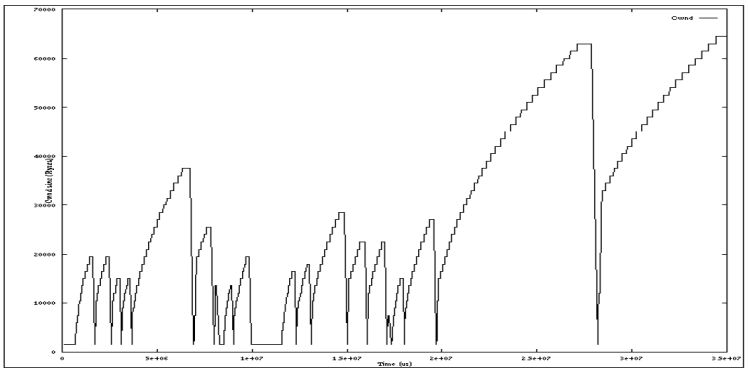


Figura 15.11: Example of evolution of cwnd.

15.2.9.2 TCP-Reno

It consists in two phases:

1. **FAST RETRANSMIT algorithm**

   - *If duplicate acks (DACK) are received*
     packets are bien delivered out of order
   - *if 3 ACKs are received for the same segment*
     the segment has been likely lost so retransmit it right away without waiting for RTO to expire

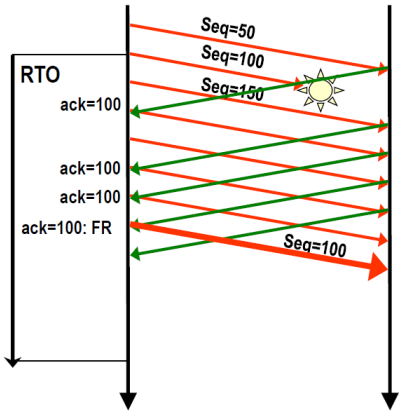


Figura 15.12: Fast retransmitt.

2. **FAST RECOVERY (RFC 6582)**

   (a) Write down the SN of the last segment sent (**SN***)

   (b) Set $cwnd = \frac{cwnd}{2}$, $ndup = 3$, $tmpW = cwnd + ndup * MSS$

   (c) For each other DUPACK

   - $ndup = ndup + 1$ e $tmpW = cwnd + ndup * MSS$
   - send new data, if allowed by $tmpW$

   (d) Upon receiving recovery ack

   - If $ack \leq SN*$ (partial recovery)
     retransmit segment with $SN = ack$, set $ndup = 0$, stay in fast recovery
   - If $ack > SN*$ (full recovery)
     set $ndup = 0$, exit in fast recovery

15.2.9.3 TCP SACK (Selective acknowledgment)

The receiver can acknowledge non-continuous blocks of data (SACK 0-1023, 1024-2047), so multiple blocks can be sent in a single segment.
Main characteristics of TCP Sack:

- Enters fast recovery upon 3 duplicate ACKs
- Sender keeps track of SACKs and infers if segments are lost. Sender retransmits the next segment from the list of segments that are deemed lost.

15.2.10 Standard TCP backdraws and limits

Standard TCP versions (STCP, for short) work fine with low bitrate small RTT connections, i.e. small bandwidthdelay product (BDP).
The main limits of TCP:

- **Fairness**
  long RTT flows that share bottleneck links with short RTT flows experience much lower long term throughput. The throughput is proportional to the inverse of RTT.

- **Performance in LFN**
  If RTT is long (in elephant transmission), if you are using alone the links, you don't use the traffic very well because in the avoidance phase, the MTT increases slowly.

15.2.11 New TCP proposals & upadates

The main new TCP proposals are:

- **TCP Vegas: Emission Control**
  Estimate available connection throughout using ACKs timing, and adjust CW accordingly

- **TCP Westwood: Bandwidth estimation**
  Similarly to Vegas, it estimates the available throughput but this information is used to set CW only after retransmissions

- **TCP cubic**
  New version which is quickly spreading

Main updates to TCP were introduced in **RFC 3390**:

- Specifies an increase in the permitted upper bound for TCP's initial window from one or two segment(s) to between two and four segments.

$$CWinit = min(4 * MSS,\ max(2 * MSS,\ 4380\ bytes))$$

- **OPTIONAL** one can decide to start with a larger CW

- After RTO, CW must be set to **1 MSS**

15.2.11.1 TCP Cubic

It's a TCP protocol that improves TCP standard protocol, making some changes in the congestion avoidance phase:

- Decreasing of cwnd in case of 3DUPACKs or RTO

- cwnd increase in congestion avoidance is cubic rather than linear

In connection in which BDP is lower the cubic works well. In connection with bigger BDP, Cubic is better.

1. **Efficency**
   CUBIC is more aggressive than STCP in big BDP networks by using a cubic window increase function in terms of the elapsed time from the last congestion event that speeds up recovery after failure and keeps throughput rather high and stable for longer periods (**HRX08**)

2. **Fairness to Standard TCP**
   The aggressiveness of CUBIC mainly depends on W_max (it's smaller in small BDP networks)

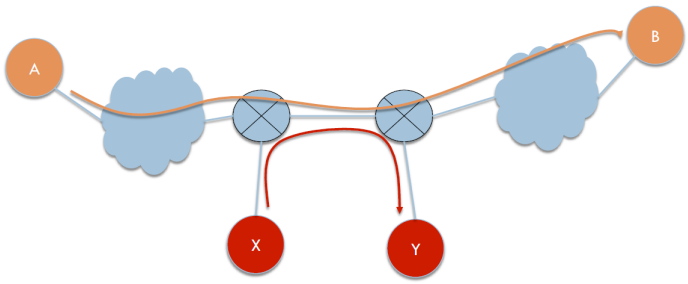


Figura 15.13: Fairness issues.

3. **Fairness**
   CUBIC's window increase rate is independent of RTTs (outside of the TCP-friendly region) because flows with different RTTs have similar congestion window sizes under steady state when they opearate outside the TCP-friendly region
   Throughput is inadvertially proportional to RTT (Standard TCP has a throughput invertially proportional to RTT square)

4. **Scalability & convergence speed**
   After congestion events CUBIC sets cwnd to $0.7 * Wmax$ while Standard TCP uses $0.5 * Wmax$.
   Cubic TCP can host multiple flows and make a better use od link capacities. However, new flows need more time to push aback existing ones, so that convergence speed is lower.

Cubic cwnd increase function

CUBIC [*HRX08*] uses a cubic window increase function in terms of the elapsed time from the last congestion event(misuration of it will be explained next).
This function increases rapidly the size of cwnd at the beginning of the congestion avoidance.
If the cwnd doesn't see packets loss in the stable center part of cubic evolution,(for values of command window less than BWmax), then the cwnd size will increase quickly.
After 3 DUPACKs (or explicit congestion notification in routers that implement it):

1. W_{max} =current value of cwnd
2. cwnd becomes $W_{cubic}(0) = beta_{cubic} * W_{max} = 0.7 * W_{max}$
3. Start a time "t" that measures the time elapsed since the beginning of the current congestion avoidance phase
4. congestion avoidance phase is started, but **cwnd** increase now follows a cubic function in time (see next slide)

Cubic cwnd increase function uses the following window increase function:

$$W_{cubic}(t) = C\ (t - K)^3 + W_{max}$$

where $k = \sqrt[3]{W_{max} \frac{1 - beta\_cubic}{C}}$:

- **C**
  constant that determines the aggressiveness of window increase in high BDP networks
- **t**
  elapsed time from the beginning of the current congestion avoidance
- **k**
  time period that the above function takes to increase the current window size to $\mathbf{W_{max}}$
- **beta_cubic (set to 0.7)**
  shrinking factor applied to cwnd after 3DUPACKS

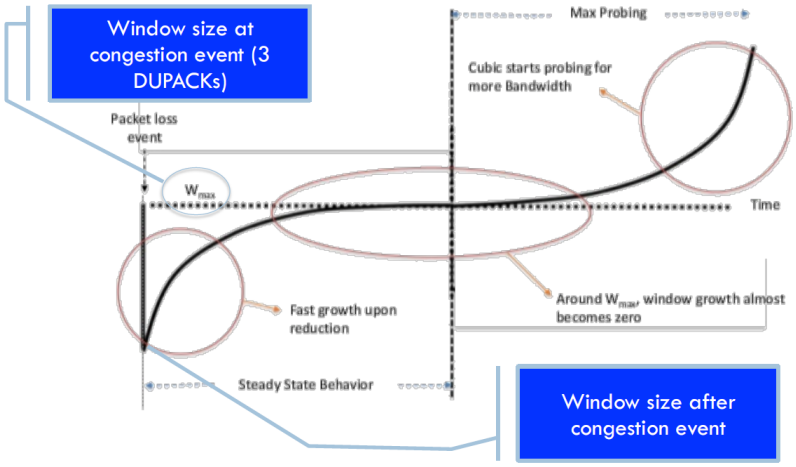


Figura 15.14: Cubic cwnd increase function.

Algorithm 3 CUBIC congestion avoidance regions

$W_{est}(t) = W_{max} * beta_{cubic} + 3\frac{1-beta_{cubic}}{1+beta_{cubic}}\frac{t}{RTT}$

if $cwnd < W_{est}(t)$ **then**
    TCP-Friendly Region
    $Cwnd := W_{est}(t)$
    Performance as good as STCP (with small BDP)
else
    if $cwnd < W_{max}$ **then**
        Concave region
        $cwnd = cwnd + \frac{W_{cubic}*(t+mRTT)-cwnd)}{cwnd}$
    else
        Convex region
        $cwnd = cwnd + \frac{W_{cubic}*(t+mRTT)-cwnd)}{cwnd}$
    end if
end if

15.2.12 TCP metrycs

Throughput

- **if** $cwnd < BDP$

$$"instantaneous"\ throughput = \frac{cwnd}{RTT}$$

- **if** $cwnd = BDP$

$$throughput = \frac{BDP}{RTT} = \frac{RTT * C_{bottleneck}}{RTT} = C_{bottleneck}$$

- **if** $cwnd > BDP$

$$throughput = C_{bottleneck}$$
$$cwnd = BDP + h * MSS$$

where $h * MSS$ is the number of bytes queued at Bottleneck.

15.2.12.1 Delivery time

Given the maximum number of bytes that can fit simultaneously in the connection $W_{b0} = W_p + B$, where B is the size of bottleneck buffer.

Slow start

$s_k = \frac{cwnd(k)}{RTT}$ throughput
$S_k = \frac{Y_k}{k*RTT}$ total number of bytessent up to *k-th* RTT Where k_s can be computed as follow: $W_s = W_0 2^{k-1} \Rightarrow$

$$\begin{aligned} &k = 1 && cwnd(k) = W_0 \\ &k = 2 && cwnd(k) = 2W_0 \\ &... \\ &k = k_s - 1 && cwnd(k) = W_0 2^{k-1} \end{aligned}$$

$W_s = \frac{W_0 2^k}{2} \Rightarrow k_s = \lceil log_2 \frac{2W_s}{W_0} \rceil$

For $\mathbf{k<k_s}$:
$Y_k = W_0 + 2W_0 + ... + 2^{k-1}W_0 = W_0 \sum_{l=0}^{k-1} 2^l = W_0 \frac{2^k - 1}{1} = W_0(2^k - 1)$

During the $k_s - th$ RTT we have $\mathbf{cwnd=W_s}$, so:
$Y_{k_s} = W_0(2^{k_s} - 1) \Rightarrow$ if Message **M** has $MSS * M \leq Y_{k_s} \Rightarrow$**Slow Start**

Congestion avoidance

Given $h_p = \frac{W_p - W_s}{MSS}$ and $cwnd(k_s + l) = W_s + l * MSS$

For $k_s \leq k \leq k_s + h_p$, such that $k = k_s + h$:
$Y_{k_s+h} = Y_{k_s} + h * W_s + \sum_{l=1}^{h} l * MSS = Y_{k_s} + h * W_s + MSS \frac{(h+1)h}{2}$

If Message **M** has $Y_{k_s} \leq MSS * M \leq Y_{k_s+h_p} \Rightarrow$**Congestion avoidance (No saturation)**

Saturation region

bytes delivered in a RTT=W_p
In the istant in which saturation zone begins: $cwnd(k_s + h_p) = W_p$
First RTT in saturation zone:
$cwnd(k_s + h_p + 1) = W_p + MSS \Rightarrow T_{increase} = \frac{cwnd}{W_p} RTT$

$\forall l$ (that is the packets enqueued in buffer) we can define:

$$h_l = \#\text{ RTTs to reach cwnd size with } l \text{ RTTs in saturation}$$

$h_l = W_p + l * MSS = \sum_{j=0}^{l-1} \frac{W_p + j*MSS}{W_p} = l + \frac{MSS}{W_p} \frac{l(l-1)}{2}$

Hence, we can compute h_{b0} of the figure as h_l in the case of full buffer (for Buffer of size B):

$$h_{b0} = B + \frac{MSS}{W_p} \frac{B(B-1)}{2}$$

After h rounds in saturation we have $Y_{k_s+h} + h * W_p$. Hence, after h_{b0} RTTs in saturation zone, we have:

$$Y(w_0, w_s) = Y_{k_s+h} + h_{b0} * W_p$$

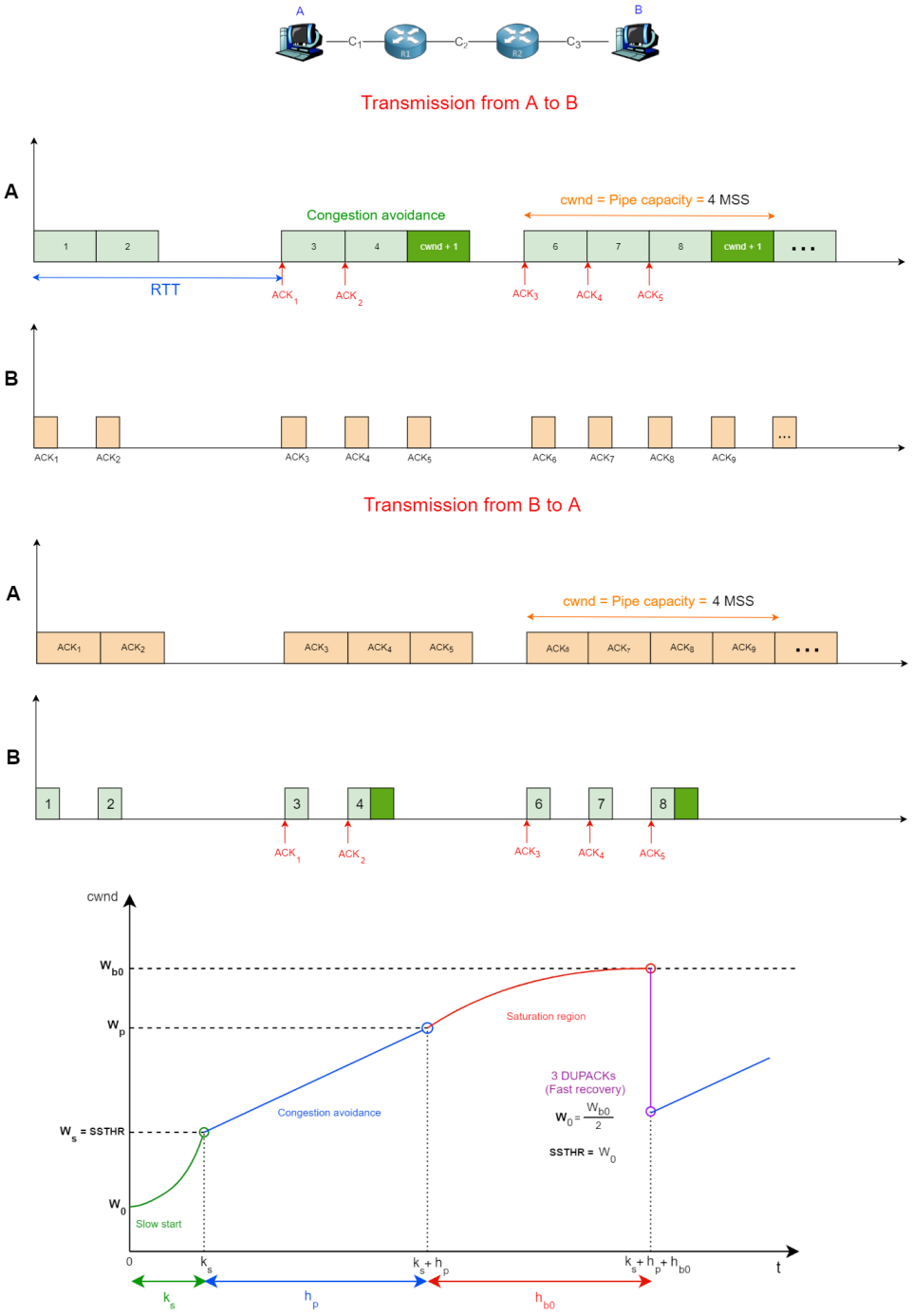


Figura 15.15: Example of TCP evolution of congestion window with saturation.

Capitolo 16

Application layer

The design of an application at application layer is based on 3 possible architectures:

- **Client-Server**
  clients don't communicate directly. It is used in Whatsapp in which there is a central server that send messages only after a request.

- **P2P**
  Highly scalable but more difficult to manage

- **Hybrid of client-server and P2P**
  There are some examples of this architecture:

  - **Napster**

    1. File transfer P2P
    2. File search centralized
       * Peers register content at central server
       * Peers query same central server to locate content

  - **Instant messaging (e.g., Skype)**

    1. Chatting between two users is P2P
    2. Presence detection/location centralized
       * User registers its IP address with central server when it comes online
       * User contacts central server to find IP addresses of buddies

Application	**Application layer Protocol**	**Transport layer Protocol**
e-mail	SMTP *[RFC 2821]*	TCP
remote terminal access	Telnet *[RFC 854]*	TCP
Web	HTTP *[RFC 2616]*	TCP
file transfer	FTP *[RFC 959]*	TCP
streaming multimedia	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Skype, Vonage,Dialpad)	typically UDP
Name resolution	DNS *[RFC 920]*	typically UDP

16.1 Domain Name System (DNS)

A DNS is a system with a tree structure that analizes input application address (analized at the root of the tree) and finds its IP address (Figure 16.1).
The main levels of this tree are grouped into:

- **Top level domains**
  contains all the last domain (after the last point) grouped into two typologies:
  - Generic domain
  - Country domain

If a application layer finds its domain address in the previous domain levels then the research goes on, analysing the other parts of the address (from right to left) and searching each time a domain between children of that node.
When the entire address is analysed, the related IP address is returned by DNS and specified in children of found domain.

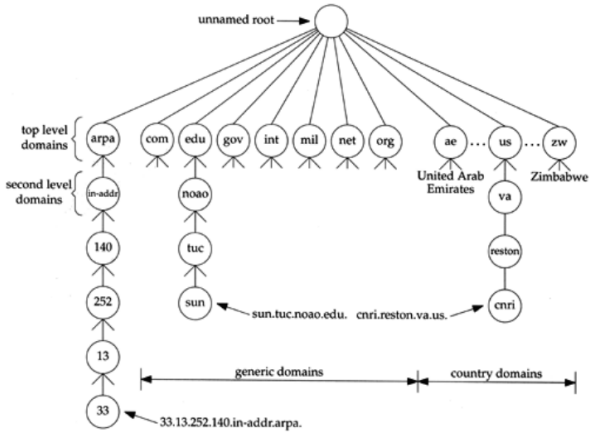


Figura 16.1: Organization of DNS.

It's designed as an Application-layer protocol: host, routers, name servers communicate to resolve names (address/name translation).
It's a distributed **database** implemented in hierarchy of many name servers.
The main services implemented by DNS, are:

- Hostname to IP address translation
- Host aliasing
- Mail server aliasing
- Load distribution

The main reasons of not centralized of DNS:

- **Single point of failure**
  more probability to collision if there is a single server
- **Traffic volutme** too much traffic on server
- **Distant Centralized Database**
- **Maintenance**
  it's more difficult to guarantee the correct function of the server if the traffic to it is high(too much clients requests to a single server)

16.1.1 Distributed, Hierarchical Database

Hierarchical organization of DNS service database is based on some types of DNS servers:

1. **Root name servers**
   knows all top-level domain servers of the Internet

2. **Top-level domain (TLD) servers**
   responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jt,...

3. **Authoritative DNS servers**
   organization's DNS servers, providing authoritative hostname to IP mappings for organization's servers (e.g., Web and mail).

Each level knows only IP address of DNS services of upper and lower for a level in the hierarchy.
Sometimes there are local DNS server that don't belong to hierarchy. When a host makes a DNS query, query is sent to its local DNS server.

16.1.1.1 Iterative resolution

An **iterated query** sent by local DNS server is used to contact DNS server that replies with name of DNS server to contact.
First contacted server doesn't know the name, but specifies you to ask another server (of the next level in hierarchical organization), as in example of Figure 16.2.
The management of the address resolution is given to the local DNS server. This reduces the load of the traffic between the main DNS server.

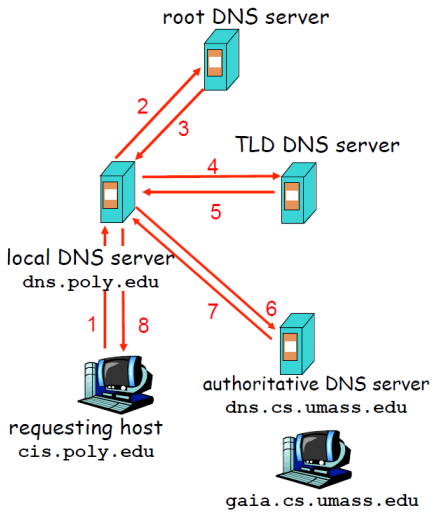


Figura 16.2: Host at **cis.poly.edu** wants IP address for **gaia.cs.umass.edu**

16.1.1.2 Recursive resolution

An **iterated query** is used to contact DNS server.
A DNS server, with respect to its functions in iterative resolution, doesn't reply with name of next DNS server but forwards request, as in example of Figure 16.3.
It increases the load for local server and upper layer DNS servers because each of them must send one or more request packets and must wait for one or more reply packets.
An advantage of this method is that all the servers cache are updated with addresses of Authorative DNS server and this can be useful in many cases.

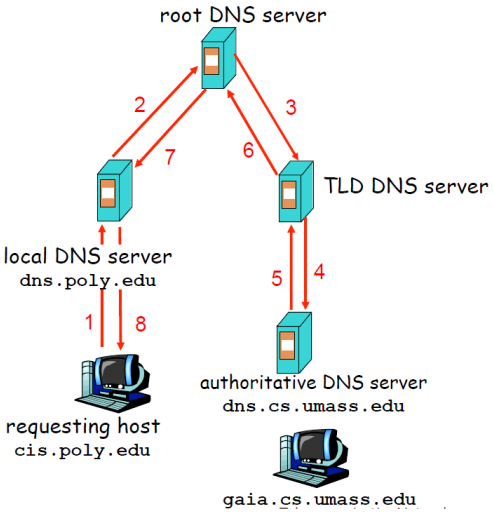
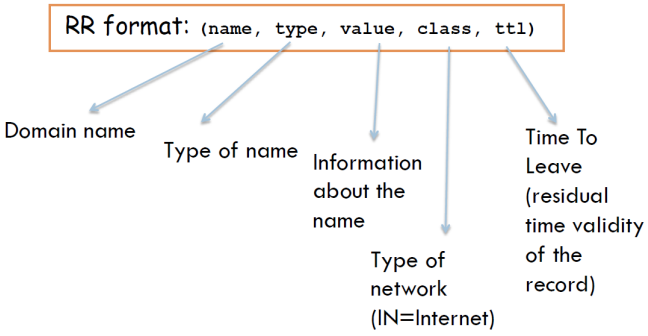


Figura 16.3: Host at **cis.poly.edu** wants IP address for **gaia.cs.umass.edu**



16.1.2 DNS record

- **A**
  - `name` is hostname
  - `value` is IP address
- **NS**
  - `name` is domain (e.g. foo.com)
  - `value` is hostname of authoritative name server for this domain
- **CNAME**
  - `name` is alias name for some "canonical" (the real) name *www.ibm.com* is really *servereast.backup2.ibm.com*
  - `value` is canonical name
- **CNAME**
  - `value` is name of mailserver associated with name

16.1.2.1 Inserting records into DNS

To register new name *networkuptopia.com* at a **registrar** (e.g., Network Solutions), this one needs to:

1. provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
2. Registrar inserts two RRs into the **com Top-Level Domain (com TLD)** server, as in Figure

type	Other parameters	
A	`name`	hostname
	`value`	IP address
NS	`name`	domain (e.g. foo.com)
	`value`	hostname of authoritative name server for this domain
CNAME	`name`	alias name for some "canonical" (the real) name *www.ibm.com* is really *servereast.backup2.ibm.com*
	`value`	canonical name
CNAME	`value`	name of mailserver associated with name

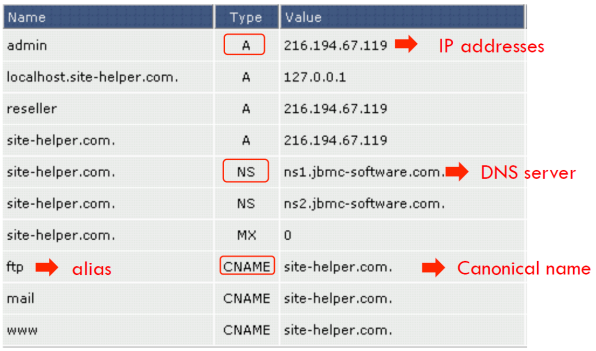


Figura 16.4: Example of the content of DNS database.

16.1.3 DNS protocol messages

Name	Type	Value
networkutopia.com	NS	dns1.networkutopia.com
dns1.networkutopia.com	A	212.212.212.1

Figura 16.5: Example of two RRs inserted into the **com TLD.**

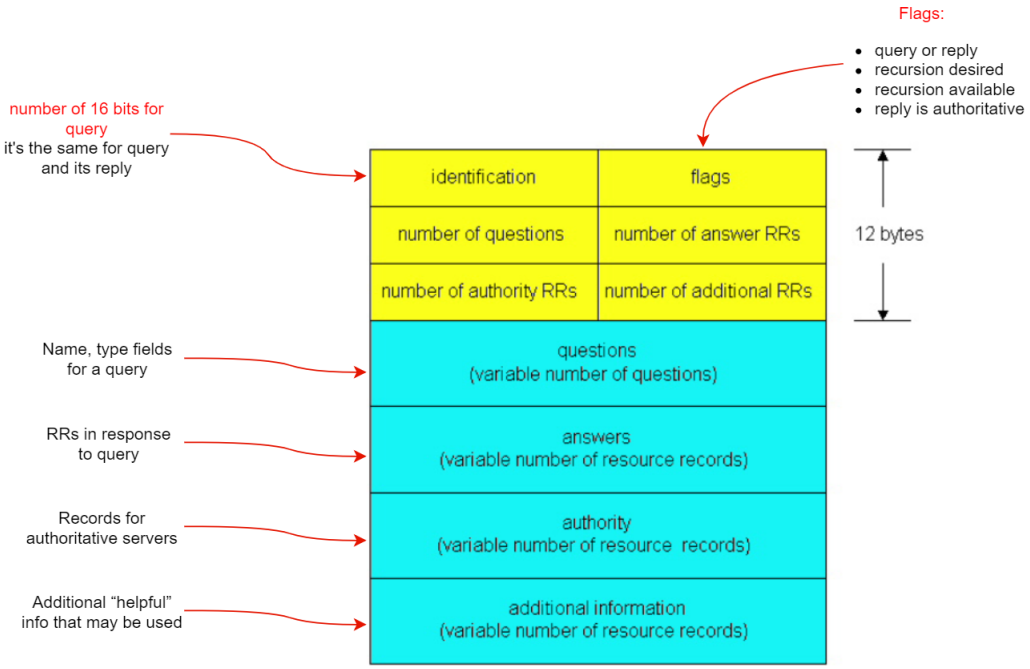


Figura 16.6: DNS protocol message (query and reply have the same message).

16.2 telnet

It's a **client-server program** that allows a user on the client side to log into a remote server site and use the services available (Figure 16.7).
It requires credentials to access the remote server but it does not support encryption. All data (included login&password) are transmitted in plaintext, so it's going to be replaced by **Secure Shell (ssh)**.

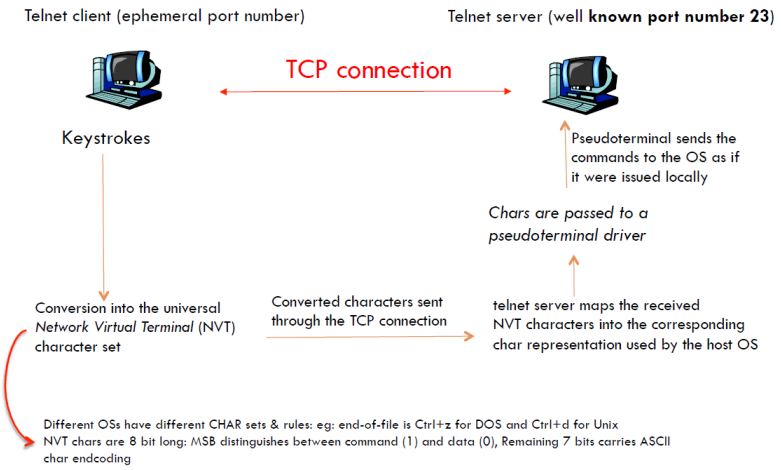


Figura 16.7: How Telnet works.

16.3 HTTP

Web page consists of objects (HTML file, JPEG image, Java applet, audio file,...). Web page must contain a base HTML-file which includes several referenced objects.
Each object is addressable by a **URL (Uniform Resource Locator)**.

```
http://www.someschool.edu/someDept/pic.gif
```

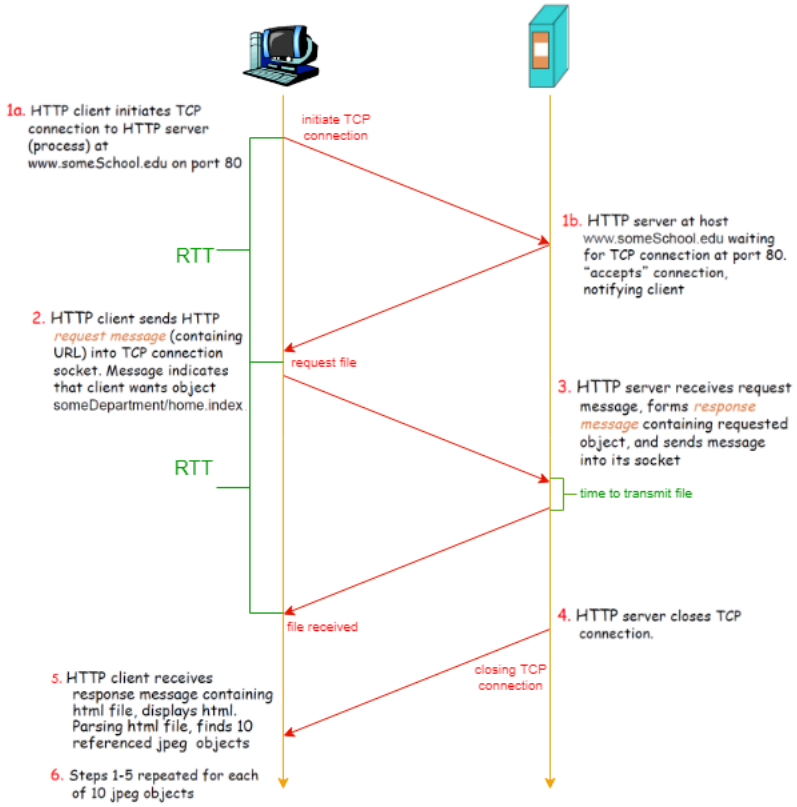
Hypertext Transfer Protocol (HTTP) is a web's application layer protocol, based on client-server model.
It uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages exchanged between browser (HTTP client) and Web server (HTTP server) *TCP connection closed*

HTTP is "stateless" connection because server maintains no information about past client requests.

16.3.1 Nonpersistent HTTP

At most one object is sent over a TCP connection (used by **HTTP/1.0**).
It requires 2 RTTs per object, so browsers often open parallel TCP connections to fetch referenced objects.


1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80
initiate TCP connection
1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client
RTT
2. HTTP client sends HTTP request message (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index
request file
3. HTTP server receives request message, forms response message containing requested object, and sends message into its socket
RTT
time to transmit file
file received
4. HTTP server closes TCP connection.
closing TCP connection
5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects
6. Steps 1-5 repeated for each of 10 jpeg objects

16.3.2 Persistent HTTP

Multiple objects can be sent over single TCP connection between client and server (HTTP/1.1 uses persistent connections in default mode).
Server leaves connection open after sending response and subsequent HTTP messages between same client/server sent over open connection.

- **Persistent without pipelining**
  client issues new request only when previous response has been received (one RTT for each referenced object)
- **Persistent with pipelining**
  client sends requests as soon as it encounters a referenced object (as little as one RTT for all the referenced objects). It's default in **HTTP/1.1**

16.3.3 HTTP request message

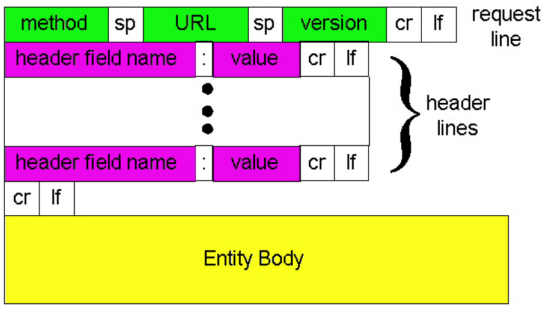


Figura 16.8: General format of HTTP request message.

16.3.4 Cookies

Cookies were introduced to save information about users because HTTP is a stateless protocol.
Server creates a cookie (random string of numbers) and assign it to the customer that wants to connect to it. This number can be considered as a ID and it's stored in the sever database and in the device of the client (Figure 16.9).
This cookie is used by web server to understand that you are the same user, understanding that it's your ID and tracking you.
It's used by server to remember part of your history. The cookies can bring authorization, shopping carts,recommendations and user session state (Web e-mails).

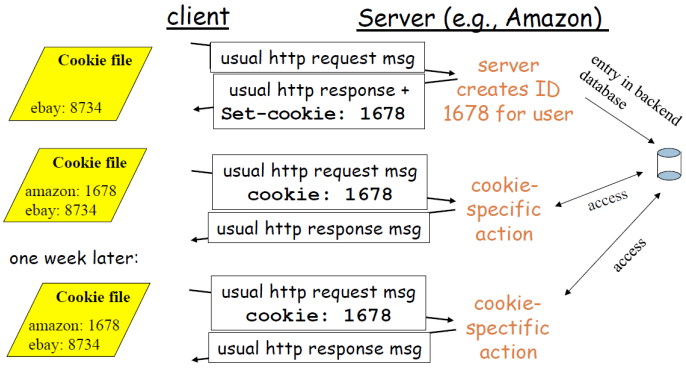


Figura 16.9: How cookies work.

16.3.5 Web caching

User sets browser (Web access via cache). Then, browser sends all HTTP requests to cache:

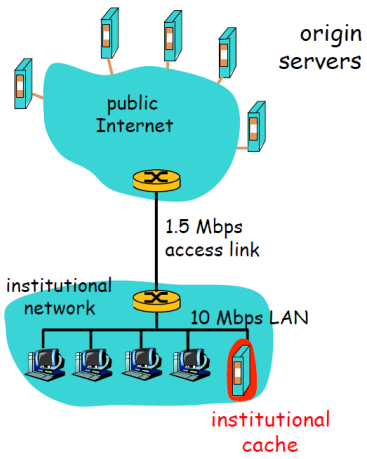
- **If object is in cache**
  cache returns object
- **Else**
  cache requests object from origin server, then returns object to client

Cache acts as both client and server and it's installed by ISP. The goals are:

1. Reducing response time for client request
2. Reducing traffic on an institution's access link
3. Internet dense with caches enables "poor" content providers to effectively deliver content (but so does P2P file sharing)

Access by istutional cache is made by ADSL (Asymetric) or XDSL (Cross), that provide common cache to many device to guarantee more efficency because more Web contents are uploaded from different devices.
The **hit rate** of an installed cache specifies how much percentage of request will be satisfied immediately by cache and the rest of it will be satisfied by origin server.

Hit rate .4 ⇒ 40% requests will be satisfied almost immediately
⇒ 60% requests satisfied by origin server



16.3.6 Dynamic Adaptive Streaming over HTTP (DASH)

An example of most used application-layer protocol is **Dynamic Adaptive Streaming over HTTP (DASH)**.
This protocol is used nowadays in video streaming and was designed with the creation of smarthphones. Before that moment, streaming was based on the download of all the video as a unique stream.
In modern devices connection, it's more difficult to guarantee a homogeneus and efficent connection, because for every device it changes moving from one place to another one (changing cells).
With DASH protocol, video is split in chunks (usually with duration of 1s) composed of different segments of original video (Figure 16.10). If size of each chunk is very small (if you choose a small duration of it), using HTTP you can download very quicklythe object a web page (video in this case).
Errors can be considered negligible, because if some segments of a chunk are missing, the user doesn't see the difference because the chunk is very short in time.
This type of protocol can be used to guarantee a quality that changes with the quality of the connetion, because every chunk is transmitted separately.

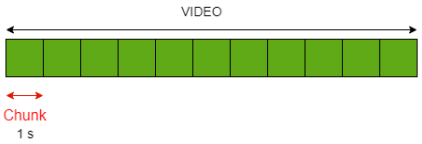


Figura 16.10: Example of division in chunks of a video.

16.4 Mail Services

It's composed of three major components:

- User agents
- Mail servers
- Simple Mail Transfer Protocol (SMTP)

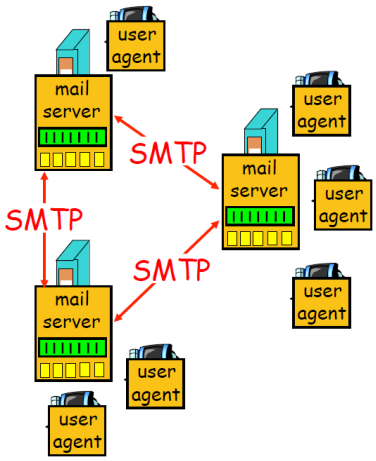


Figura 16.11: Mail servers organization.

16.4.1 Simple Mail Transfer Protocol (SMPT)

It's a **Message Transfer Agent: RFC 2821** that uses TCP to reliably transfer email message from client to server at port **25**. There is a direct transfer, sending server to receiving server and it's composed of three phases:

1. handshaking
2. transfer of messages
3. closure

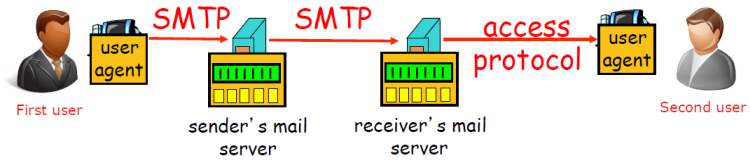
The command interaction is made by ASCII text command and response is composed by status code and phrase. The messages must be written in 7-bit ASCII.

16.4.1.1 How SMTP works

1. *First user* uses UA to compose message "to" **second_user@someschool.edu**
2. *First user*'s UA sends message to her mail server using SMTP; message placed in message queue
3. Client side of SMTP opens TCP connection with *Second user*'s mail server

4. SMTP client sends *First user*'s message over the TCP connection
5. *Second user*'s mail server places the message in *Second user*'s mailbox
6. *Second user* invokes his user agent to read message (using Message Access Agent, e.g., POP3, IMAP)

16.4.2 Mail access protocols



There are several mail access protocols:

- **POP: Post Office Protocol [RFC 1939]**
  1. authorization between agent and server
  2. download of emails on the device
- **IMAP: Internet Mail Access Protocol [RFC 1730]**
  1. more features than **POP**
  2. manipulation of stored messages on server (messages are kept in the server)
- **HTTP**
  Hotmail, Yahoo! Mail, etc.

POP3	IMAP
1^{st} `mode:"download and delete"`	>Keep all messages in one place (server)
user cannot re-read e-mail if he changes client	>Allows user to organize messages in folders
2^{nd} `mode:"download and keep"`	>IMAP keeps user state across sessions
copies of messages on different clients	(names of folders and mappings
	between message IDs and folder name)
stateless across sessions	

16.5 P2P file sharing

Example of how it works

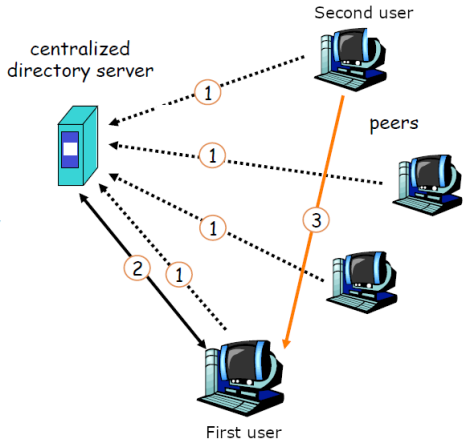
1. **First user** runs P2P client application on his notebook computer
2. He Intermittently connects to Internet and gets new IP address for each connection
3. He asks for *specific_file*
4. Application displays other peers that have copy of Hey Jude
5. **First user** chooses one of the peers (**Second user**).
6. File is copied from **Second user**'s PC to **First user**'s notebook through `HTTP`
7. While **First user** downloads, other users uploading from **First user**.

First user's peer is both a Web client and a transient Web server. This permit to obtain a highly scalable system.

16.5.1 P2P with centralized directory

It was the orginal **Napster** design:

1. When peer connects, it informs central server about its IP address and its content
2. **First user** queries for a *specific_file*
3. **First user** requests file from **Second user**



Problems with this type of P2P:

- **Single point of failure**
  If the server is down, no requests can be successfully completed
- **Performance bottleneck**
  There is high traffic for a single server (server with centralized directory)
- **Copyright infringement**
  Everyone can access to contents from everyone

16.5.2 Query flooding: Gnutella

It's fully distributed (no central server) and has public domain protocol. It has this structure:

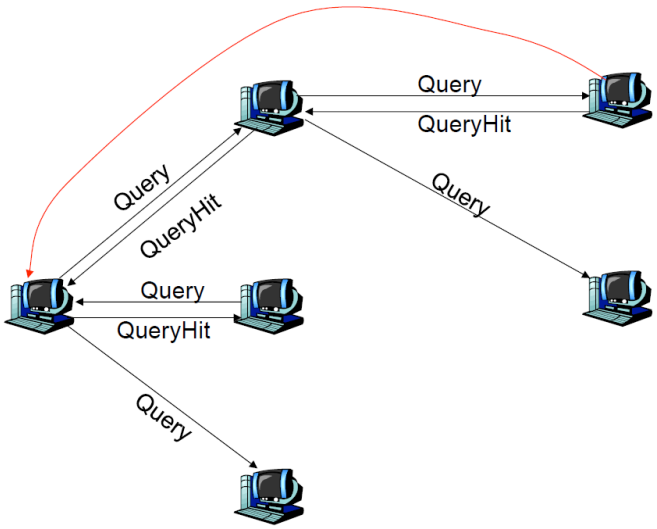
- edge between peer X and Y if there's a TCP connection (edge os not a physical link)
- all active peers and edges is overlay net

It's based on the use of queries:

1. **Query** message sent over existing TCP connections
2. peers forward **Query** message
3. **QueryHit** sent over reverse path

Peer joining in Gnutella follows these steps:

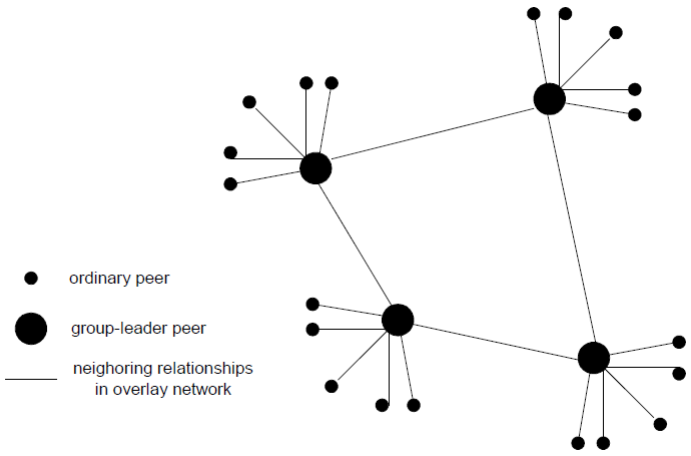
1. Joining peer X must find some other peer in Gnutella network: use list of candidate peers
2. X sequentially attempts to make TCP with peers on list until connection setup with Y
3. X sends Ping message to Y; Y forwards Ping message
4. All peers receiving Ping message respond with Pong message
5. X receives many Pong messages. It can then setup additional TCP connections (the transfer is based on HTTP protocol)



16.5.3 Exploiting heterogeneity: KaZaA

The networks becomes with this structure:

- Each peer is either a group leader or assigned to a group leader
  - TCP connection between peer and its group leader
  - TCP connections between some pairs of group leaders
- Group leader tracks the content in all its children



Querying in KaZaA follows these steps:

1. Each file has a hash and a descriptor
2. Client sends keyword query to its group leader
3. Group leader responds with matches (for each match: metadata, hash, IP address)
4. If group leader forwards query to other group leaders, they respond with matches
5. Client then selects files for downloading (HTTP requests using hash as identifier sent to peers holding desired file)

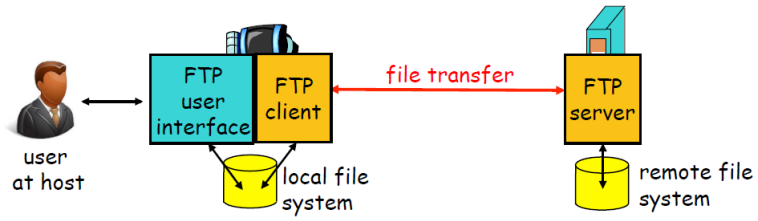
There are many KaZaA tricks:

- Limitations on simultaneous uploads
- Request queuing
- Incentive priorities
- Parallel downloading

16.6 File Transfer Protocol (FTP)

It permits to transfer file to/from remote host (RFC 959). It's based on **client/server model**:

- **client**
  side that initiates transfer (either to/from remote)
- **server**
  remote host



The FTP has separate control and data connections and works following these steps:

1. FTP client contacts FTP server at port 21, specifying TCP as transport protocol
2. Client obtains authorization over control connection
3. Client browses remote directory by sending commands over control connection
4. When server receives a command for a file transfer, the server opens a new TCP data connection to client
5. After transferring one file, server closes connection
6. Server opens a second TCP data connection to transfer another file
   - Client issues "passive open" on ephemeral port
   - Client sends port number to the server using PORT command
   - Server issues active open using well known port 20 & ephemeral
7. Control connection: **"out of band"**
8. FTP server maintains "state": current directory, earlier authentication

Capitolo 17

Quality of Service

The main principles for QoS Guarantees are:

1. **Classification & marking**
   packet marking needed for router to distinguish between different classes; and new router policy to treat packets accordingly

Where:	⇒	source or network ingress
Why:	⇒	discriminate packets that need to be treated differently (e.g., based on type of source, sender, destination,...)
What:	⇒	after classification, marking tools can set an attribute of a frame or packet to specific values

2. **Isolation: scheduling and policing**
   provide protection (isolation) for one class from others

   - **policing**
     force source adherence to bandwidth allocations (avoiding applications send more than what declared)

Where:	⇒	network ingress
Why:	⇒	check whether traffic conforms agreements and act accordingly
What:	⇒	mark, remark, or drop packets/requests

   - **Scheduling**

Where:	⇒	each QoS-enabled node
Why:	⇒	guarantee isolation among traffic classes
What:	⇒	determine servicing order in case of congestion

3. **High resource utilization**
   While providing isolation, it is desirable to use resources as efficiently as possible

4. **Call admission**
   flow declares its needs, network may block call (e.g., busy signal) if it cannot meet needs. It's given by the fact that a link can not support traffic demands beyond link capacity.

Where:	⇒	connection ingress
Why:	⇒	offer optimization techniques to network administrators
What:	⇒	shaping, fragmentation, compression,...

17.1 Classification

The process of identifying flows of packets and grouping individual traffic flows into aggregated streams (e.g. classes) such that actions can be applied to those streams (e.g. policing, shaping, scheduling).
There are four types of classification:

1. **Implicit Classification**
   watching *incoming interfaces*
2. **Simple classification (single-field classification)**
   e.g., Ethernet802.1Q Class-of-Service field, IP destination address, TCP/UDP port number, ...
3. **Complex Classification (multi-field classification)**
   e.g., some combination of route prefixes, IP protocol and UDP/TCP ports
4. **Deep packet inspection/stateful inspection** it's used when it's complex to classify applications

Marking (a.k.a. colouring) is the process of setting the value of a dedicated packet field so that the traffic can easily be identified later

- IP Differentiate Service Code Points (DSCP)
- EXP field in MPLS frames

Traffic marking can be applied unconditionally or evaluating how much the network is loaded.

17.1.1 Token Bucket Algorithm

The simplest Token Bucket Algorithm mechanism is a policy method that analyzes the traffic defined by two parameters:

- **r:** token generation rate
- **b:** bucket size (# of tokens it can contain)

Tokens are saved up during idle periods (queued up to the max size of the bucket) and consumed by packet transmissions. For example one byte can cost and consume one token.
Let X be the number of tokens in the bucket and B the packet size.

17.1.1.1 One Rate Two Colors (1R2C)

We define marking method **One Rate Two Colors (1R2C)** as:

- $X \geq B$ →packet is marked as "conform"
- $X < B$ →packet is marked as "exceed"

Suppose traffic specifications (TS) say

Average rate = λ_a		
Peak rate = λ_p	⇒	**Token arrival rate:** λ_a
Max Burst duration = T_p		**Bucket capacity:** $(\lambda_p - r)T_p$
		bucket will not be empty before period T_p

Example:
$R_1 = 100\frac{pck}{s}$ $L = 500B$

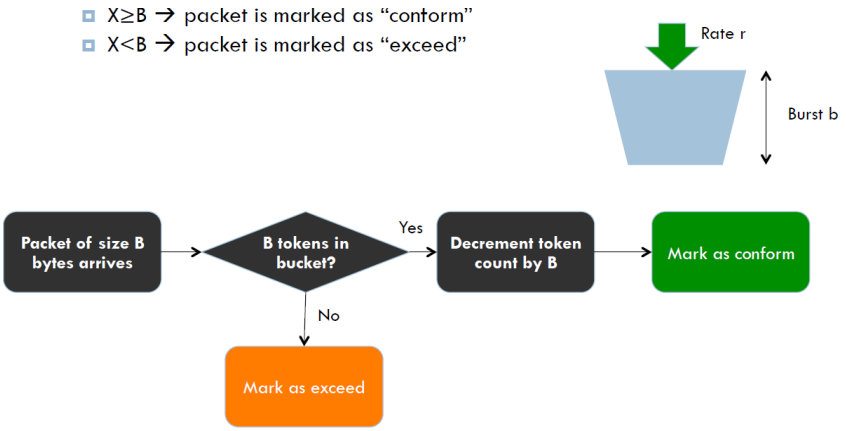


Figura 17.1: Flow chart of algorithm 1R2C.

If the transmission of packet is at rate λ_p
fraction of packets marked as "conformed": $\frac{r}{\lambda_p}$
fraction of packets marked as "exceed": $1 - \frac{r}{\lambda_p}$

$R_1 = 500\frac{kbit}{s}$
$D = 2s$
$r = avarage\, rate = lim_T^{\inf} \frac{1}{T}(R_1 T + \lambda T(R_2 - R_1)D)$

If buckets contains b token, which are consumed at a rate $R_2 \frac{token}{s}$ and produced at rate $r\frac{token}{s}$ then the time to empty the bucket is $\frac{b}{R_2 - r} = D \Rightarrow b = D(R_2 - r)$.
If the rate is small,the avarage rate is small and then bucket capacity is small. Bucket is used because we want more flexibility and to transmitt at rate that is greater than what you can transmitt. Packets are queued watching the size b.

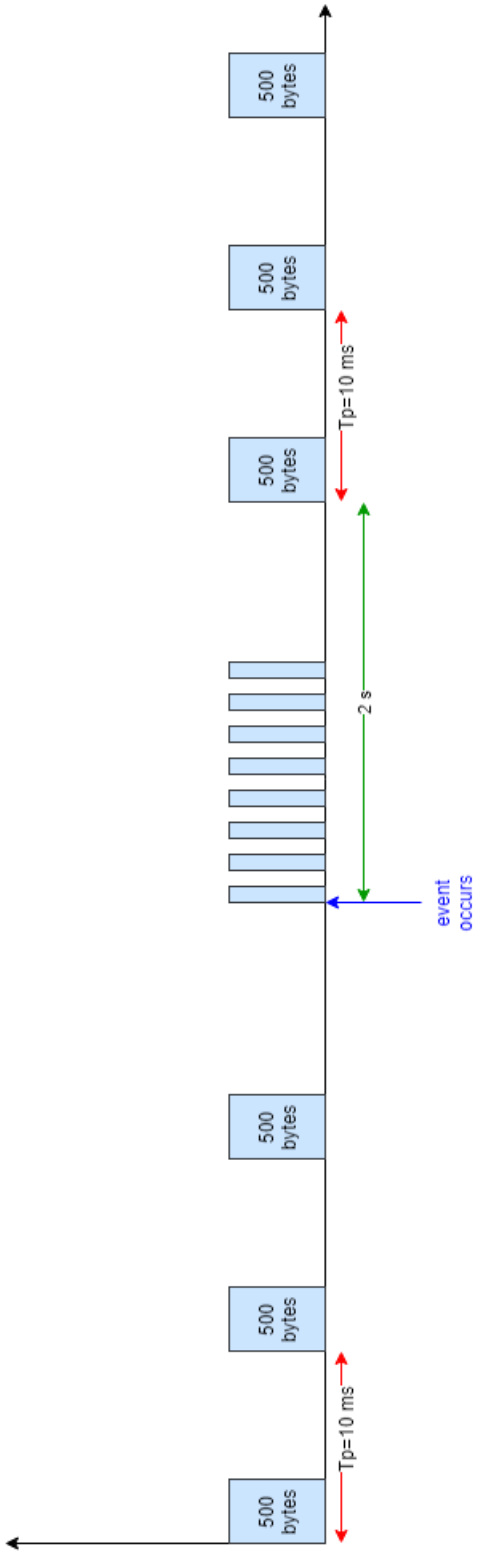


Figura 17.2: How to compute bucket size in 1R2C.

17.1.1.2 Two-rate three color (2R3C)

We can define marking method **Two-rate three color (2R3C)**. In this case Customer and ISP agree upon (*RFC 2698*):

- *Committed Information Rate (CIR)*
  Minimum required transmit rate
  If I have $CIR = 1\frac{Mbit}{s}$ from ISP, this means that bucket must be made to guarantee this rate using previous formulas
- *Committed Burst Size (CBS)*
  Period of bursting traffic permitted when CIR is not fully exploited for a period
- *Peak rate (PIR) or Excess Information Rate (EIR)*
  Maximum required transmit rate. The same consideration can be applied as done for CIR
- *Peak Burst Size (PBS)*

Two consecutive bursts generate exceed packets. Long idle times do not allow for longer bursts (tokens arriving to a full bucket are dropped out).

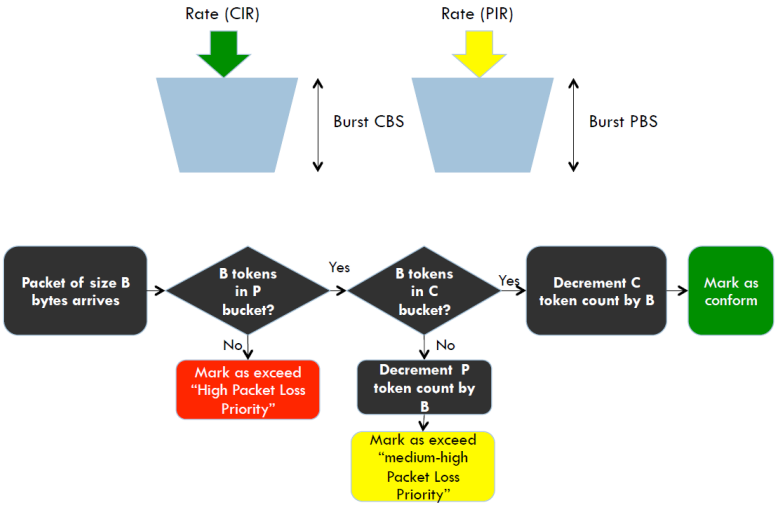


Figura 17.3: Flow chart of algorithm 2R3C.

17.2 Traffic shaping

Traffic shaping is a set of rules that describe a flow's traffic characteristics.

- "shapes" the traffic before it enters the network, so that characteristics are amenable to provide QoS guarantees
- control the rate at which packets are sent (not just how many)

The main purposes of traaffic shaping are:

- the way of a flow to describe its traffic to the network
- the network can determine if the flow should be allowed in
- the network can periodically monitor the flow's traffic

17.2.1 Burstiness & Bandwidth Allocation

- **Classification of sources - data, voice and video**
  Data sources are generally bursty, whereas voice and video are naturally continuous and bursty if compression is used

- **Multimedia sources can be roughly divided into**

  1. **Constant bit rate (CBR) traffic** it's often from continuous sources. It's strongly periodic, strongly regular stream characteristics.
     It needs peak-rate allocation of bandwidth for congestion free transmission.
  2. **Variable bit rate (VBR) traffic** it's often from bursty sources. It's strong periodic and weakly regular stream characteristics (period emission of variable-size packets (MPEG)).
     It's weakly periodic and strongly regular stream characteristics (neven emission of fixed-size packets (online gaming)).
     Average rate of transmission can be a small fraction of the peak rate. Peak rate allocation would result in underutilization and average rate allocation would result in losses/delays.

17.2.2 Motivations of using traffic shaping

1. Admission control and scheduling alone are insufficient
   - users may attempt to exceed the rates specified at the time of connection establishment (possibly inadvertently)
   - QoS is negotiated for average rate, but bursts occur during transmissions, hence resource allocation can be unbalanced
   - Need constant watch over traffic associated with each application
2. Traffic policing checks whether input conforms to declared values, but doesn't prevent bursts

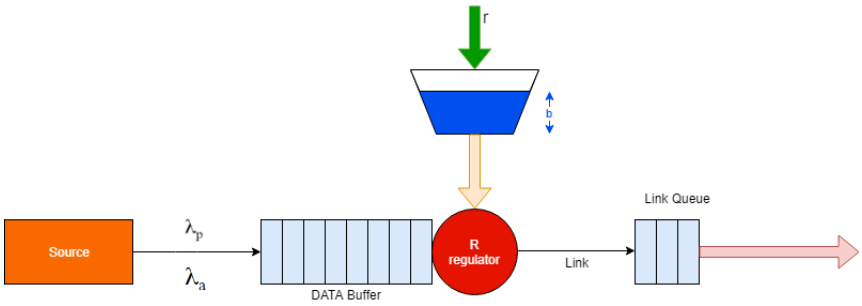
17.2.3 Token Bucket for shaping bursty traffic

Sending a packet of size B tokens ($B < b$) there are three possible scenarios:

1. *Token bucket is full*
   the packet is sent and B tokens are removed from the bucket

2. *Token bucket is empty*
   the packet must wait for B tokens to drip into the bucket before being sent

3. *Token bucket is partially full (β tokens in the bucket)*
   If $B \leq \beta$ then the packet is sent immediately; otherwise it must wait for the remaining $B - \beta$ tokens before being sent



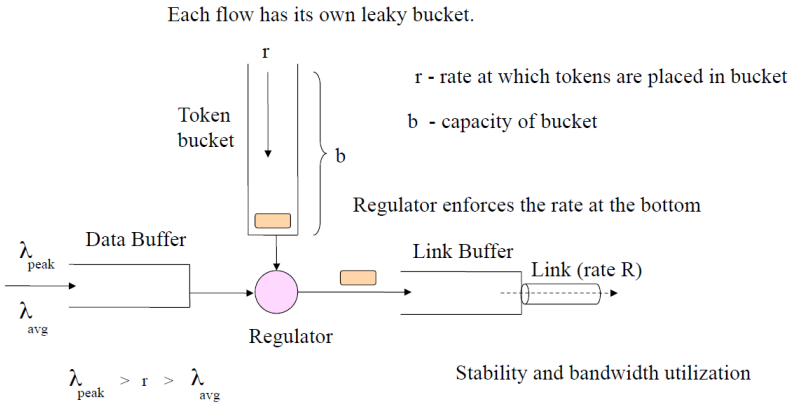


Figura 17.4: Token Bucket for shaping of bursty traffic.

17.2.3.1 Maximum burst length

1. **If $\lambda_p < R$ fitered packets are immediately transmitted over the link**

   - Tokens are generated at rate r and consumed at rate λ_p
     Bucket empties at a rate equal to $\lambda_p - r$
   - While bucket still holds tokens, packets are filtered at rate λ_p
     packets arrive at the link at rate λ_p
   - When bucket empties, packets are filtered at rate r packets arrive at the link at rate $r \Rightarrow$burst is over

   Maximum burst length equals bucket depletion time d
   Starting with a full bucket, the depletion time is given by

   $$b + rd = \lambda_p d \Rightarrow d = b/(\lambda_p - r)$$

2. **If $\lambda_p > R$**
   same result as previous case

17.2.3.2 Maximum burst period

Assume the source violates the TB spec concerning max burst period, and keep generating packets at peak rate even once the TB is emptied. In this case, data buffer is incrementally filled with packets. Let's see what happens step by step.

1. **Step** Data buffer empty, token bucket full
   Source starts generating packet at peak rate $\lambda_p < R$

   - Packets are filtered at rate λ_p, consuming tokens held in the bucket
   - Data Buffer remains empty

2. **Step** Token bucket empties after a time period d

   - Packets get filtered at rate $r < \lambda_p$, consuming each generated token
   - Data Buffer starts filling up at rate $\lambda_p - r$

3. **Step** Data Buffer is full
   Filling the data buffer takes a time $\frac{B}{(\lambda_p - r)}$

Therefore, Data Buffer overflows at time

$$t_2 = d + \frac{B}{\lambda_p - r} = \frac{b}{b}/(\lambda_p - r) + \frac{B}{\lambda_p - r} = \frac{b+B}{\lambda_p - r}$$

17.2.3.3 Maximum packet delay

- If the source complies using Token Bucket spec (*burst duration* $< d$)
  - $\lambda_p < R$
    there is not queuing and packet delay due to TB is *zero*
  - $\lambda_p > R$
    pcks get queued at the link buffer. Last packet undergoes largest delay
    1. The maximum number of packets queued at the link
    $$Q = (\lambda_p - R)d = (\lambda_p - R)\frac{b}{\lambda_p - R}$$
    2. Packets are sent at rate **R**
    $$d_{max} = \frac{Q}{R} = \frac{b\lambda_p - R)}{(\lambda_p - r)R}$$
- If the source doesn't comply using Token Bucket spec (*burst duration* $\geq d$)
  the maximum delay is given by the waiting time at the data queue, when it is full
  $$d_{max} = \frac{B}{r}$$
  1. Data Buffer gets filled at rate $\lambda_p - R$ (once the bucket is emptied)
  2. Data buffer system time $\frac{B}{r}$
  3. Once passed the data buffer, link buffer holds **Q2** packets.
     After the bucket empties, link starts getting rid of the queued packets with a rate equal to $R - r$
     $$Q2 = Q - \frac{B}{\lambda_p - r}x(R - r) = \frac{(\lambda_p - R)b}{\lambda_p - r} - \frac{B(R - r)\lambda_p}{r(\lambda_p - r)}$$

17.2.4 Leaky Bucket

The leaky bucket was developed by J. Turner (1986, Washington Univerity) and enforces a constant output rate (average rate) regardless of the burstiness of the input:

- It does nothing when input is idle
- It queues packets when input rate is larger than token rate
- It drops packets if excess rate lasts too long

The host injects one packet per clock tick onto the network. This results in a uniform flow of packets, smoothing out bursts and reducing congestion.

- When packets are the same size (as in ATM cells), the one packet per tick is okay
- For variable length packets though, it is better to allow a fixed number of bytes per tick

Leaky bucket does smooth out bursts of traffic, hence guaranteeing outgoing traffic rate never exceeds the leaky rate ρ.
The average outgoing rate is, hence, always less than or equal to ρ

- **CBR** sources with *rate* $\leq \rho \Rightarrow$go through
- **VBR** bursty sources with *average rate* $\leq \rho \Rightarrow$get clamped

17.2.4.1 Leaky Bucket vs Token Bucket

Leaky Bucket forces bursty traffic to smooth out, Token Bucket permits burstiness but bounds it:

1. flow never sends more than $b + \tau * r$ tokens worth of data in an interval τ
2. long-term transmission rate will not exceed r

With Token Bucket, a packet can only be transmitted if there are enough tokens to cover its length in bytes.

17.2.5 Composite Shaper

- **Token Bucket**
  makes it possible to source to get "credits" if it transmits less than committed rate
- **Leaky Bucket**
  smoothes out peaksof rate, guaranteeing that leakage rate must be larger than token generation rate

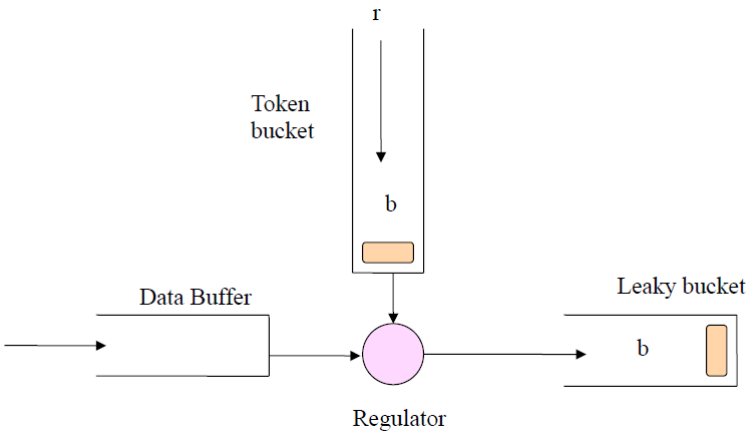


Figura 17.5: Example of composite shaper.

Capitolo 18

IEEE 802.x

It's a project promoted by a consortium of companies under the support of IEEE(Institute for Electrical and Electronics Engineers). These standards for LANs were recognized by other standardisation companies, like ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission.
These standards were identified by some numbers (Figure 18.1):

- **802.1:** Higher layer LAN protocols
  It describes the general architecture of project `IEEE 802.x`.
  It introduces the subdivision of the Data Link Layer (DLL) of the OSI model into two different sublayers:
  - Logical Link Control (LLC)
  - Medium Access Control (MAC)
- **802.2:** Logical Link Control (LLC)
  It specifies the services and the interfaces provided by the LLC layer (protocols and interfaces for the management of the logical connections in a LAN)
- Mac Protocols
  - **802.3** (Ethernet)
  - **802.4** (Token Bus)
  - **802.5** (Token Ring)
  - **802.6** (MAN)
  - **802.11** (Wireless LAN)
  - **802.15** (Wireless PAN)

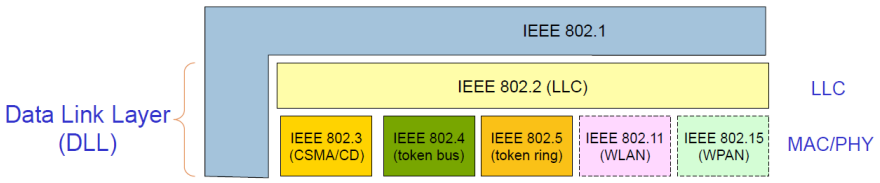


Figura 18.1: IEEE 802.x Standards Family

18.1 Ethernet

The DIX consortium formed by Digital Corporation, Intel Corporation, Xerox Corporation develops the Ethernet standard.
IEEE 802.3 differs in some marginal aspects but Commercial Network Interface Cards implement both standards.
The main features of ethernet protocol are:

- **Connection less**
  a frame is sent whenever ready
- **No flow control**
  sender can overwhelm receiver with frames hence excess frames are silently dropped. Loss recovery is delegated to upper layers
- **No use of ACK**
  frames contain a 32-bits CRC (or FCS) field for error detection but ARQ is not natively implemented at MAC layer for bit errors. *If CRC fails, the frame is silently dropped by the receiver*
- **Retransmission in case of collision**

18.1.1 Ethernet frame

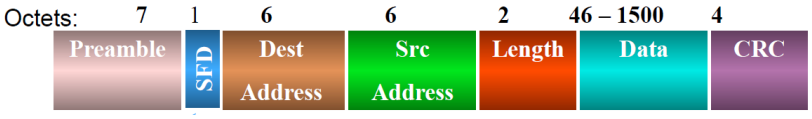


Figura 18.2: IEEE 802.x Standards Family

It's organized in many parts (Figure 18.2):

- **Preamble** (7 bytes)
  for the bit synchronization at the receiver. This filed is actually added by the PHY layer
- **Start Frame Delimiter** (1 byte)
  The preamble ends with the sequence of 8 bits `10101011` for the frame synchronization at the receiver (flag). SFD is actually added at PHY layer
- **Destination address** (6 bytes)
  destination MAC address
- **Source address** (6 bytes)
  source MAC address

Unicast	**0**xxx...
Multicast	**1**xxx...
Broadcast	**111...1**

- **Length** (2 bytes)
  DATA length in bytes (without preamble and PADding bytes). The most significant bit is always set to 0 (to distinguish the frame from that of Ethernet), hence theoretical Max Length: 215 bytes = 32 Kbytes.
- **Data** (46-1500 bytes)
  PDU at the LLC layer (+ PAD to reach the minimum data length of 46 byte, i.e., frame length of 64 bytes = 512 bits excluded preamble and SFD).
  The MAXIMUM frame length is used to:

  1. Avoid one station monopolizes the bus for a long time
  2. Bound the packet error probability
  3. Reduce buffering requirements

  The MINIMUM frame length is used to:

  1. Reduce overhead

2. To guarantee collision detection
   The time required to be sure that the transmission is not collided is 2tpthat is two times the maximum propagation delay so $L > R2t_p$
   t_p, time taken by the signal to propagate between the two farthest endpoints of the link

- **Cyclic Redundancy Check (**4 bytes**)**
  error detection

18.1.2 Classification of Ethernet cables

$$\mathbf{X}Base\mathbf{Y}$$

- **X** is transmission data rate in Mbit/s
- *Base* baseband modulation
- **Y**
  - **if it's a number**
    it specifies the approximate maximum distance between any two nodes ("segment") in *100 m*
  - **it is a letter**
    it specifies the transmission medium

The main examples are:

- **Ethernet at 10 Mbit/s**
  - **10Base5**
    Network at 10 Mbit/s over thick coaxial cable (Figure 18.3)
  - **10Base2**
    Network at 10 Mbit/s over thin coaxial cable (Figure 18.4)
  - **10Base-T**
    Network at 10 Mbit/s over unshielded twisted pair (UTP) cable (Figure 18.5)
  - **100Base-T**
    Network at 100 Mbit/s over unshielded twisted pair (UTP) cable (Figure 18.6). It can be used to create star topology with LAN switch at the center.
    1. It supports full-duplex transmission between the station and the LAN Switch
    2. Automatic learning of MAC address/output port mapping
       * Read source MAC address of incoming packets and map to incoming switch port
       * If destination address is unknown, forward through all (other) ports
       * Provides a sort of layer-2 routing
    3. Buffer frames to the same destination and forward them sequentially, avoiding collisions
  - **10Base-F** Network at 10 Mbit/s over optical fiber cable

- **Gigabit Ethernet**
  There are different physical layer standards:
  - **1000BASE-FX**
    optical fiber
  - **1000BASE-T**
    twisted pair cable
  - **1000BASE-CX**
    balanced copper cable (almost obsolete)

  There is no longer CSMA/CD, so it doesn't need minimum frame length (it doesn't need to perform CD). It doesn't need to limit the max cable length but it only depends on cable attenuation.

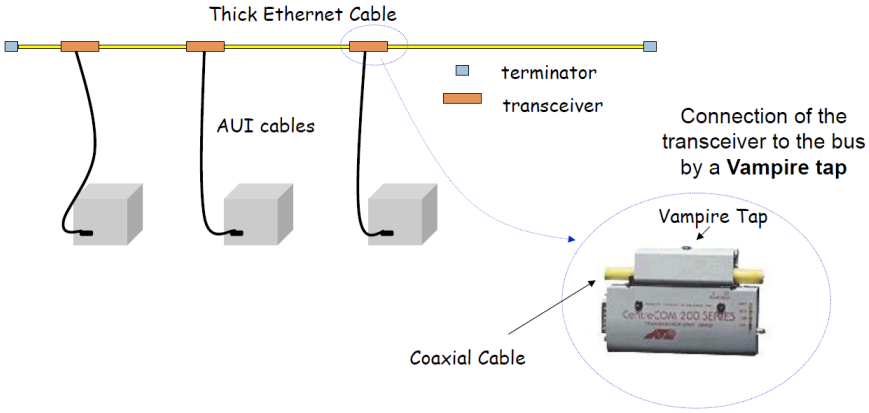


Figura 18.3: Thicknet - 10Base5

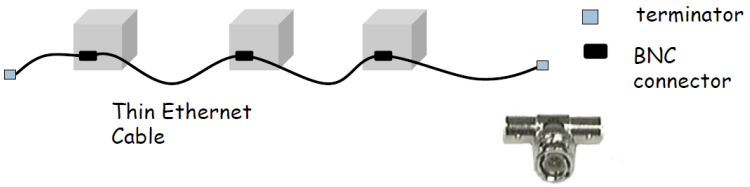


Figura 18.4: Thinnet - 10Base2

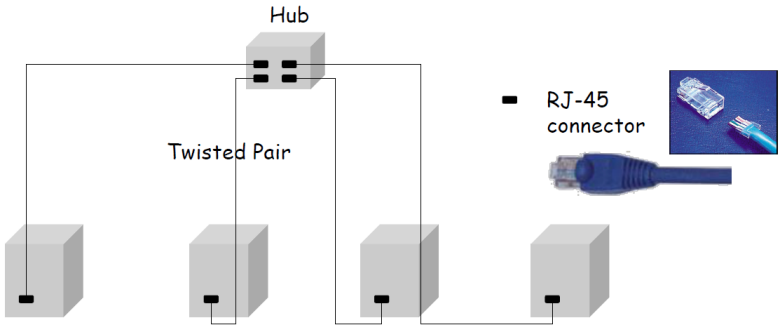


Figura 18.5: Ethernet over twisted pair - 10BaseT

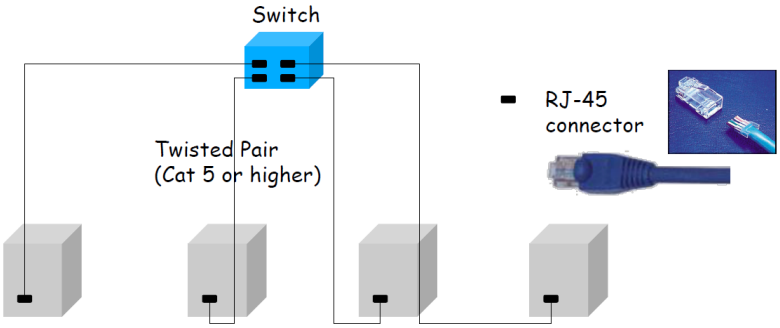


Figura 18.6: Fast Ethernet over twisted pair - 100BaseT

18.2 Wireless LAN

The bands used in industrial application (Industrial-Scientific-Medical, ISM) are at **2.4-2.4835 GHz** (*80MHz*) and **5.725–5.875 GHz** (*150 MHz*)
The band at **2.4 GHz** can be divided into 14 partially overlapping channels of 22 MHz each (channels **1,6,11** and 2,7,12 are orthogonal each other).

WLAN modes of operation are:

- **Independent basic service set (IBSS)**
  There is no access point, it's composed by indipendent or autonomous devices (clients).
  The three parameters for IBSS participation are:
  - **SSID**
    Name of the service set used to identify the WLAN and for device segmentation

Figura 18.7: SSID

  - RF channel
  - Security configuration

- **Basic service set (BSS)**
  It's foundation of distributive WLANs:
  - Access point (AP)

Pros	Cons
Easy to configure	Limited range
Inexpensive to deploy	No centralized admin
	Not scalable
	Difficult to secure

– Client devices

It's defined by the same parameters of **IBSS**.

Pros	Cons
Flexible in various situations, homes, SOHO, small enterprises	Additional hardware costs (APs etc.)
Scalable	Require site survey
Centralized admin	Must connect to a larger infrastructure
Security parameters	Technical expertise

- **Extended service set (ESS)**
  It's used for one or more connected BSS. It requires a layer two or layer three devices for connectivity.

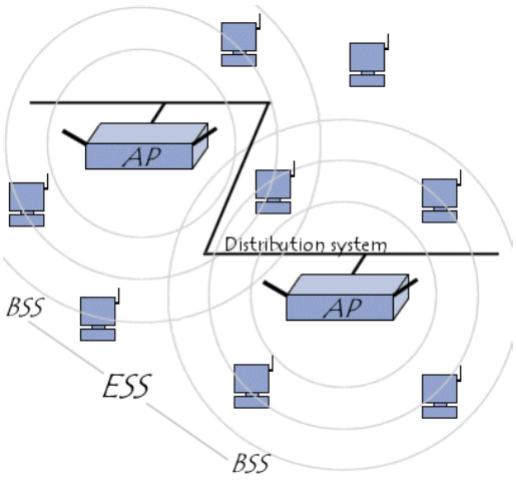


Figura 18.8: ESS

18.2.1 Frame types

- **Management frames**
  beacon, probe request, authentication, association request, association response
- **Control frames**
  - **RTS (request to send)**
  - **CTS (clear to send)**
  - **ACK**
- **Data frames**
  carry the payload

18.2.2 CSMA (Carrier Sense Multiple Access)

It's done to prevent the collision during the transmission at MAC level. The **CSMA**, performs Clear Channel Assessment (CCA):

- If channel is sensed **idle**
  it starts transmission
- If channel is sensed **busy**
  wait for the channel to become idle again, then enter backoff
  - Set the backoff timer to this value to a random value in the set $\{0, 1, \ldots, W\}$
  - Decrease the timer at each "slot" for each "idle" slot
  - Freeze the countdown when the channel becomes busy
  - When backoff timer clears, send the packet

Collision (Figure 18.9)

If multiple stations start transmitting during the same time slot.

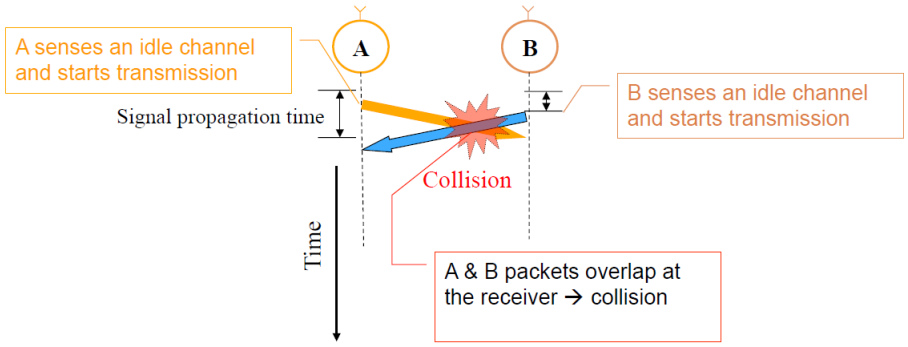


Figura 18.9: Collision

Hidden node problem (Figure 18.10)

A and B are mutually hidden (out of reach), but both within the coverage range of C.

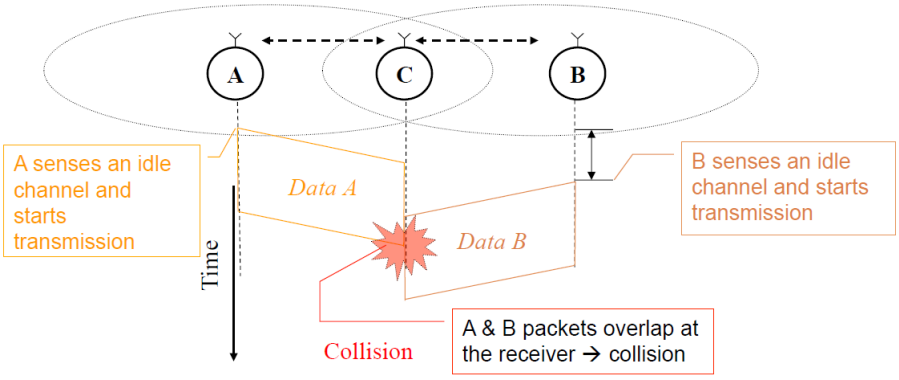


Figura 18.10: Hidden node

18.2.3 Collision Avoidance (RTS/CTS)

1. Sender sends a short **Request-To-Send** (**RTS**) to book the channel
2. Receiver replies with **Clear To Send** (**CTS**)
   Both RTS and CTS carry the time interval required for data exachange
3. Nodes overhearing either RTS or CTS set **Network Allocation Vector** (**NAV**) and refrain from transmission for that time interval

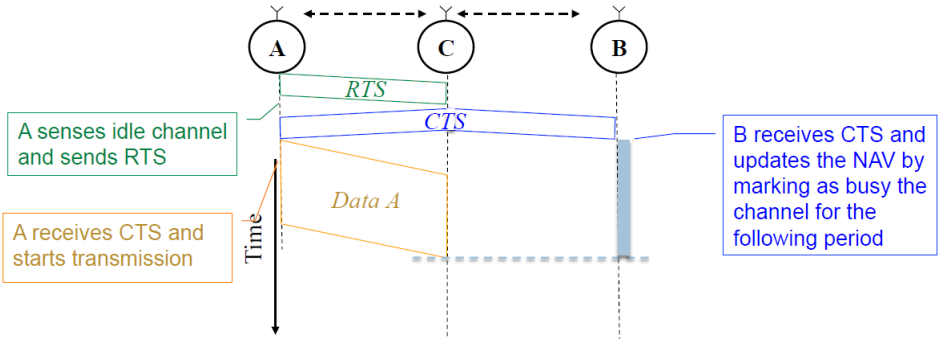


Figura 18.11: RTS/CTS

Control messages are sent at lower rate to guarantee that all devices see the collision (increasing time of transmission). If RTS is very small, probability of collision is smaller than the collision of 2 packets.

18.3 Bluetooth

The units are connected into small networks, called **piconets** (Figure 18.12). Each piconet can host 2 to 8 active devices (many more in sleep mode).
A unit acts as **Master**, the others as *Slave*. Master manages the channel access by using a poling algorithm.

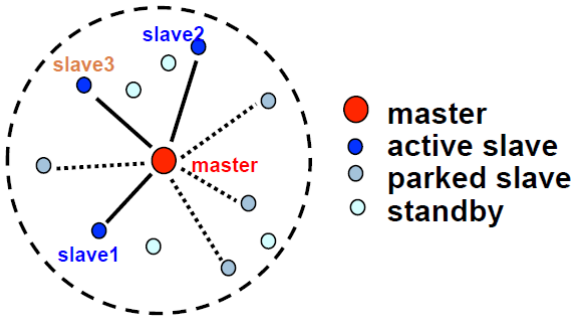


Figura 18.12: Bluetooth piconet

Full-duplex communication is obtained by means of **Time Division Duplex** (**TDD**) (Figure 18.13):

- Time is divided in consecutive slots of 625 μs each
- **Downlink (Master-to-slave) transmissions**
  start from odd slots
- **Uplink (Slave-to-Master) transmissions**
  start on even slots

Each transmission occurs on a different 1-MHz RF channel, according to a **frequency-hopping** pattern that is different for each piconet, decided according to master address (Figure 18.14).

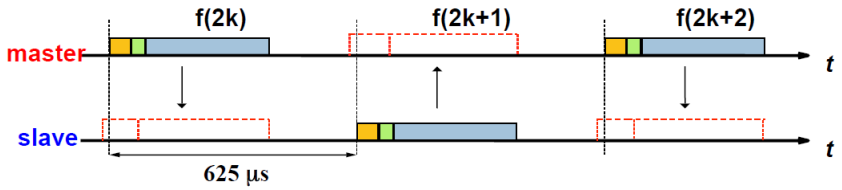


Figura 18.13: Full Duplex transmission channel

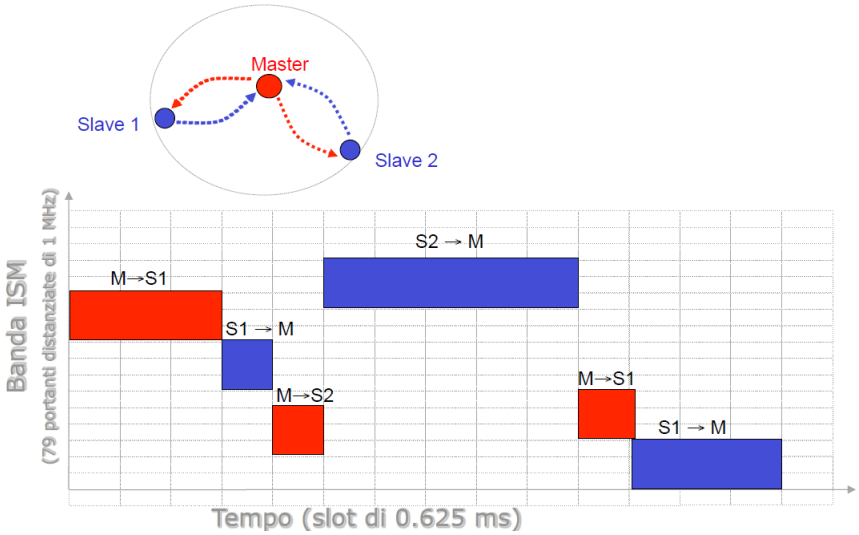


Figura 18.14: Example of frequency hopping

18.3.1 Voice and Data connections

There are two type of connections (Figure 18.15):

- **Synchronous Connection Oriented (SCO)**
  - Synchronous, symmetric, connection oriented service
  - MAC is deterministic: slots are reserved to voice traffic at regular time intervals
- **Asynchronous Connection-less (ACL)**
  - Packet oriented, asymmetric, asynchronous
  - MAC is polling-based

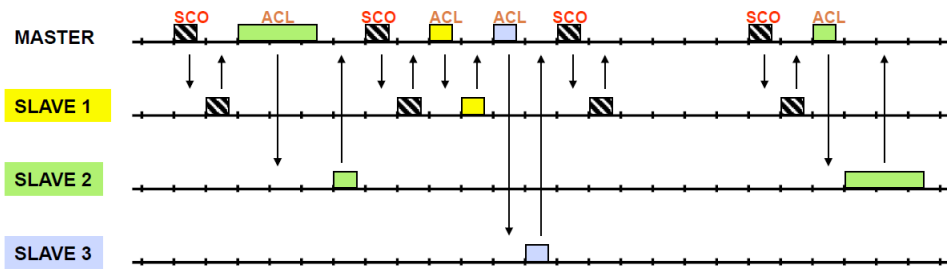


Figura 18.15: Voice and data connections

Packets can be 1, 3 or 5 slot long. Carrier frequency (i.e., channel) does not change during the transmission of a multislot packet. Multislot packets reduce overhead due to header and guard time ($\approx 220\ \mu s$).

Capitolo 19

Netkit

19.1 Simulation vs. emulation

Emulation and simulation systems put at user's disposal a virtual environment that can be exploited for tests, experiments, measures

- **simulation systems**
  aim at reproducing the performance of a real-life system (latency time, packet loss, etc.)
  e.g.: ns, real, ...
- **emulation systems**
  aim at accurately reproducing the functionalities of a real-life system (configurations, architectures, protocols), with limited attention to performance

19.2 What is Netkit?

Netkit has three main components:

- **set of tools and commands**
  used to easily set up a virtual computer network (most commands are implemented as scripts)
- **a ready-to-use filesystem**
  exploited as a pattern for creating the file system of each vm (most commonly used networking tools are already installed in this filesystem)
- **a User Mode Linux (UML) kernel**
  used as kernel for the virtual machines. Each emulated network device is a virtual linux box (based on UML kernel).
  The linux os is shipped with software supporting most of the network protocols. Hence, any linux machine can be configured to act as a bridge/switch or as a router.

19.3 User mode linux

User-mode linux is a linux kernel (inner part of the linux os) that can be executed as a user process on a standard linux box. It is also a user-mode linux process called **virtual machine (vm)**.
The linux box that hosts a virtual machine is called **host machine (host)**. Several virtual machines can be executed at the same time on the same host. Withs normal host, you can create no more than 10 vms.
Each virtual machine has:

- **a console**
  a terminal window
- **a memory**
  "cut" into the memory of the host

- **a filesystem (stored in a single file of the host filesystem)**
  File System of host machine is connected to file system of vms.
  It permit to create a relation between real and virtual world in the machine. Each vm has 1 or more network interface, connected to a collision.
  `/hosthome` special folder that corresponds to the home of the real system.
- **(one or more) network interfaces**
  - each network interface can be connected to a (virtual) collision domain
  - each virtual collision domain can be connected to several interfaces

  Each collision domain is equal to a **hub**.

For each vm, when i use vstart command, a file `name_host.disk` is created inside machine file system, and this file contains the configuration parameters of the vm.
There is a process running for each vm and for every collision domain.

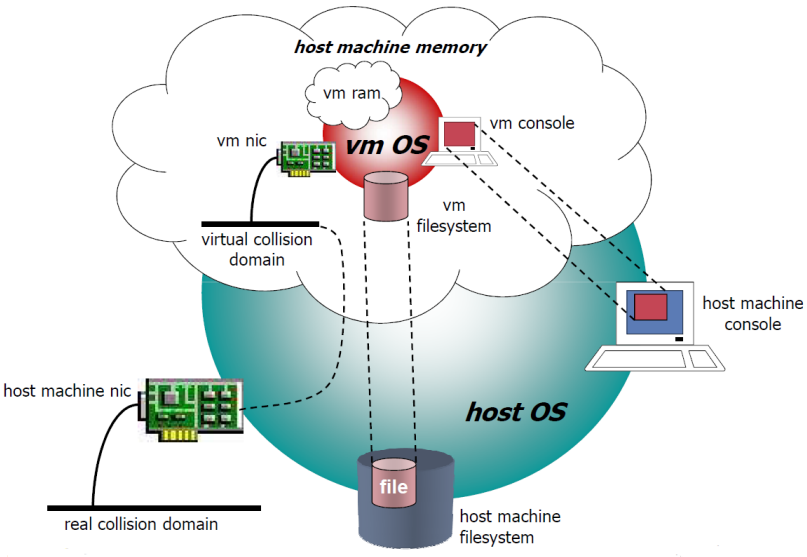


Figura 19.1: How real and virtual OSs cooperate.

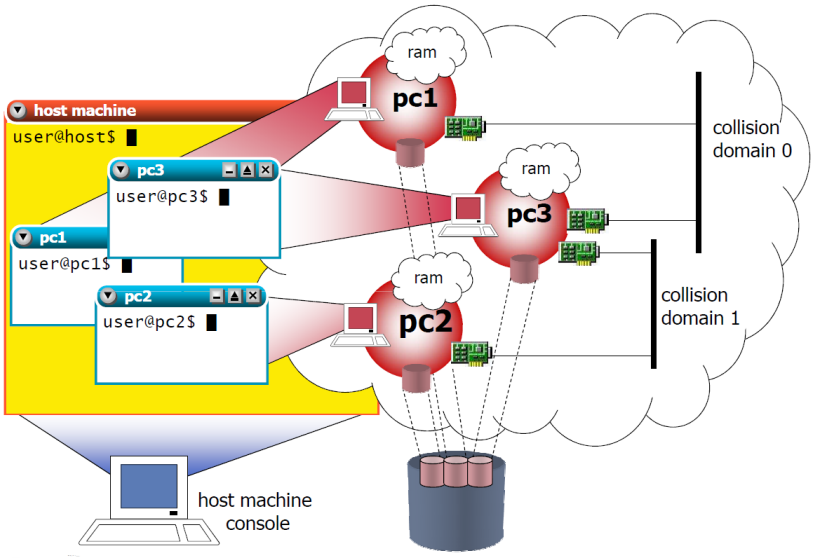


Figura 19.2: VMs shells.

19.4 Netkit commands

Netkit provides users with two sets of commands:

- **v-prefixed commands (vcommands)**
  act as low level tools for configuring and starting up single virtual machines. They allow to startup virtual machines with arbitrary configurations (memory, network interfaces, etc.).

`vstart`	starts a new virtual machine
`vlist`	lists currently running virtual machines
`vconfig`	attaches network interfaces to running vms
`vhalt`	gracefully halts a virtual machine
`vcrash`	causes a virtual machine to crash
`vclean`	cleans up all netkit processes and configuration settings on host

- **l-prefixed commands (lcommands)**
  provide an easier-to-use environment to set up complex labs consisting of several virtual machines. Thanks to them, it becomes easy setting up complex labs consisting of several virtual machines.

`lstart`	starts a netkit lab
`lhalt`	gracefully halts all vms of a lab
`lcrash`	causes all the vms of a lab to crash
`lclean`	removes temporary files (also filesystems **.disk**) from a lab directory after the lab has been halted/crashed
`linfo`	provides information about a lab without starting it
`ltest`	allows to run tests to check that the lab is working properly

19.5 Preparing a lab

A netkit lab is a set of preconfigured virtual machines that can be started and halted together. It may be implemented in (at least) two ways:

- **by writing a single script lab-script**
  that invokes vstart for each virtual machine to be started
- **by setting up a standard netkit lab**
  that can be launched by using the lcommands (recommended)

19.5.1 A netkit lab as a single script

A script (e.g., **lab-script**) invokes **vstart** with some options to start up each virtual machine ().
By using the –**exec** option of **vstart**, the same script can be invoked inside vms (e.g., in order to automatically configure network interfaces). A check inside **lab-script** can be used to test if we are in the real host or inside a vm.

19.5.2 Netkit labs using lcommands

A standard netkit lab is a directory tree containing:

- **A** *lab.conf* **file**
  describes the network topology and the settings of the vms that make up a lab.
  Inside it, there is a list of `machine[arg]=value` assignments, where:

```
vstart pc1 --eth0=0 --eth1=1 --exec=this_script
vstart pc2 --eth0=0 --exec=this_script
vstart pc3 --eth0=1 --exec=this_script
if [ `id -u` == "0" ]; then
       case "$HOSTNAME" in
              pc1)
                     ifconfig eth0 10.0.0.1 up
                     ifconfig eth1 10.0.0.2 up;;
              pc2)
                     ifconfig eth0 10.0.0.3 up;;
              pc3)
                     ifconfig eth0 10.0.0.4 up;;
       esac
fi
```

Figura 19.3: Example of lab as a script.

- `machine` is the name of the vm (e.g., pc1)
- if `arg` is an integral number (say `i`), then value is the name of the collision domain to which interface `ethi` should be attached
- if `arg` is a string, then it must be the name of a `vstart` option and `value` is the argument (if any) to that option

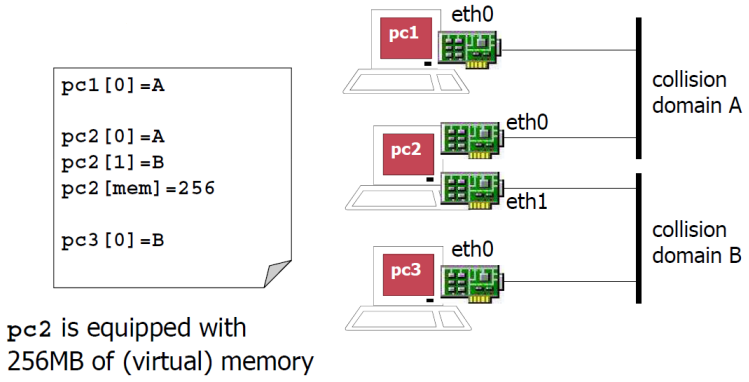


Figura 19.4: Example of lab as a script.

There can be other optional assignments:

- `machines="pc1 pc2 pc3..."`
  explicitly declare the virtual machines that make up the lab
- Descriptive information displayed when the lab is started
  - `LAB_DESCRIPTION`
  - `LAB_VERSION`
  - `LAB_AUTHOR`
  - `LAB_EMAIL`
  - `LAB_WEB`

- **A set of subdirectories**
  each subdirecory contains the configuration settings for each virtual machine.
  Netkit starts a virtual machine for each subdirectory, with the same name of the subdirectory itself.
  the contents of subdirectory vm are mapped (=copied) into the root (/) of vm's filesystem.

- *.startup* **and** *.shutdown* **files**
  describe actions performed by virtual machines when they are started or halted.

Shell scripts that tell virtual machines what to do when starting up or shutting down. They are executed inside virtual machines.
Upon startup, a vm named **vm_name** runs:

- `shared.startup`
  defines actions common to every vm at startup of the lab
- `vm_name.startup`
  defines actions executed by the vm **vm_name** at startup of the vm

```
ifconfig eth0 10.0.0.1 up
/etc/init.d/zebra start
```

Upon shutdown, a vm named **vm_name** runs:

- `shared.shutdown`
  defines actions common to every vm at shutdown of the lab
- `vm_name.shutdown`
  defines actions executed by the vm **vm_name** at shutdown of the vm

- **[optionally] a** *lab.dep* **file**
  describe dependency relationships on the startup order of virtual machines. Multiple virtual machines can boot at once as parallel startup (**-p** option of `lstart`)

```
pc3: pc1 pc2
```

- **[optionally]** *a_test* **directory**
  containing scripts for testing that the lab is working correctly. The scripts inside the directory, with extension **.test**, are executed thanks to command `ltest`.

19.6 Zebra/Quagga

Routers (i.e., devices that run routing protocols) in netkit are virtual machines that run a specific piece of software that implements routing protocols, **Zebra/Quagga**.
Quagga is a software that implements several routing protocols

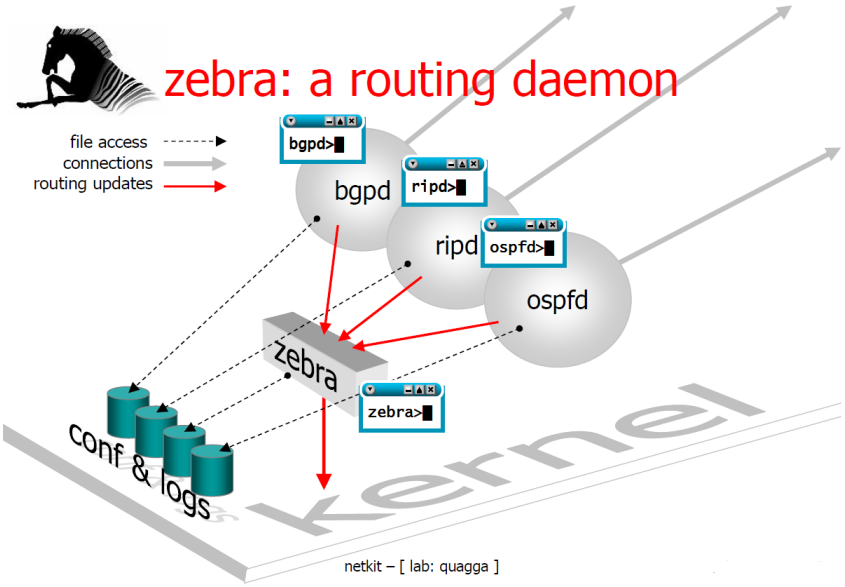
- rip (v1 and v2)
- ospf (v2 and v3)
- is-is
- bgp

Quagga: "a fork of GNU Zebra [that] aims to build a more involved community around Quagga than the current centralised model of GNU Zebra" (zebra development stopped at release 0.95a and quagga superseded zebra).

19.6.1 Zebra configuration files

When zebra is started, each daemon checks these files to read the starting configuration.

```
pc1:~# cd /etc/zebra/
pc1:/etc/zebra# ls
bgpd.conf               ospfd.conf              vtysh.conf
bgpd.conf.sample        ospfd.conf.sample       vtysh.conf.sample
bgpd.conf.sample2       ripd.conf               zebra.conf
daemons                 ripd.conf.sample        zebra.conf.sample
ospf6d.conf             ripngd.conf
ospf6d.conf.sample      ripngd.conf.sample
```



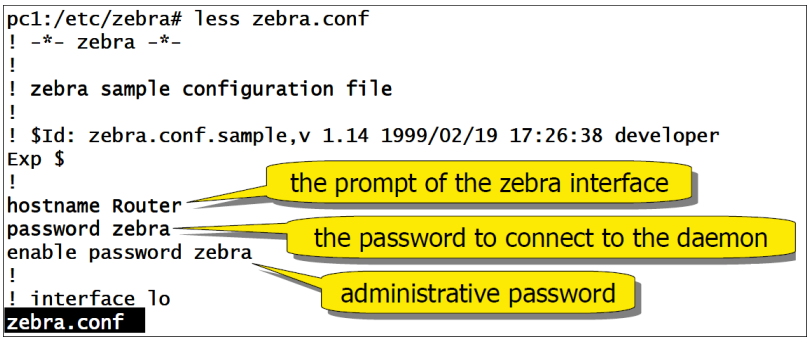

```
pc1:/etc/zebra# less daemons
# This file tells the zebra package
# which daemons to start.
# Entries are in the format: <daemon>=(yes|no|priority)
# where 'yes' is equivalent to infinitely low priority, and
# lower numbers mean higher priority. Read
# /usr/doc/zebra/README.Debian for details.
# Daemons are: bgpd zebra ospfd ospf6d ripd ripngd
zebra=yes
bgpd=no
ospfd=no
ospf6d=no
ripd=yes
ripngd=no
daemons (END)
```

the zebra main daemon will be started

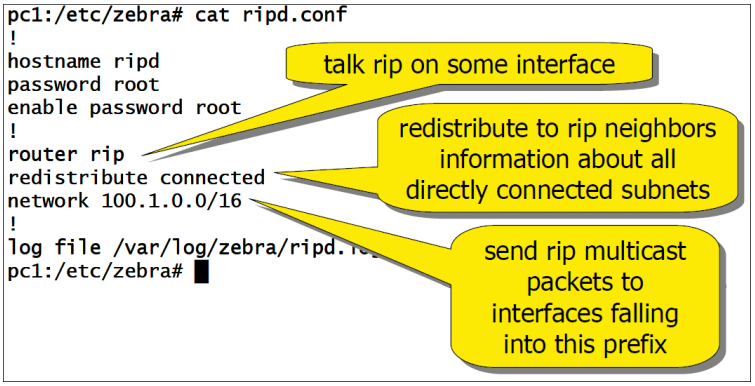
the rip daemon will be started too

19.6.1.1 deamons file

19.6.1.2 zebra.conf



19.6.1.3 ripd.conf



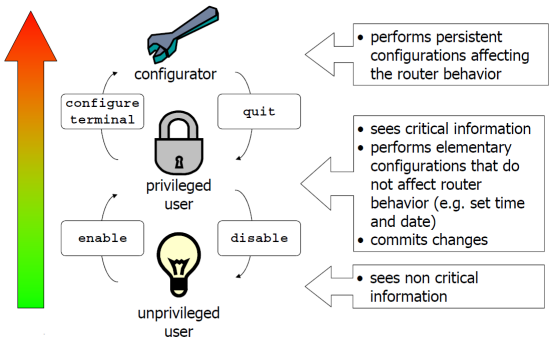
19.6.2 Connection to the main zebra deamon

To establish the connection to the deamon, we need to use the command `telnet localhost zebra` inside a vm.

```
r3:~# telnet localhost zebra
Trying 127.0.0.1...
Connected to r3.
Escape character is '^]'.

Hello, this is Quagga (version 0.99.10).
Copyright 1996-2005 Kunihiro Ishiguro.


User Access Verification

Password: zebra
zebra>
```

we are unprivileged users

19.6.3 Privileges on a router




```
zebra>
  echo       Echo a message back to the vty
  enable     Turn on privileged mode command
  exit       Exit current mode and down to previous mode
  help       Description of the interactive help system
  list       Print command list
  quit       Exit current mode and down to previous mode
  show       Show running system information
  terminal   Set terminal line parameters
  who        Display who is on vty
```

Figura 19.5: Help window generated by typing '?'.

Capitolo 20

Firewall

A **firewall** is a device that implements an access control policy between networks (tipically between one considered safe and another one considered unsafe - as Internet).
The term firewall can be used in many applications:

- **Packet filters**
- **application layer firewalls**
- **proxy servers**

20.1 Packet filters

The main operation done by a packet filter of "inspecting" all the incoming, outgoing and traversing packets allowing to "pass" the firewall only the packets that match a set of creteria.
The main types of firewall are:

1. **Stateless packet filters**
   inspection is based on *the protocol type, the source and destination addresses and the source and* destination ports.
   - If packets don't match criteria:
     - **packet dropped**
       silently discarded
     - **packet rejected**
       discarded but with an error message sent back to the source
   - Otherwise, packets will pass the firewall
2. **Stateful packet filters**
   inspection is based on *same basic operations of a stateless one but inspection capability reach the transport layer.*
   It has the ability to:
   - keep track of previously seen packets
   - determine if the current packet is part of an already existing "conversation", is the beginning of a new one or nothing of the previous
   - speed up the packet processing allowing the current packet to pass the firewall without further elaboration when it is part of an already existing "conversation"

     "Conversation" state

   - can be kept not only for packets that use **connection-oriented** as **TCP** (information in TCP header parameters)

- in **connectionless** as **UDP** the "connection" state is obtained wathing the time between two UDP packets. Two UDP packets are considered part of the same conversation if the *second is the reply to the first* (source and destination addresses and ports of the second are switched in comparison to those of the first) and the time interval between them is less then some value.

20.2 iptables

iptables is a tool in user-space that acting on its kernel-space counterpart (NetFilter) let us implement a stateful packet filter (with network address and port translation) on Linux machines.

Netfilter is composed by four *tables* that contain *chains* (predefined or user-defined):

- `filter`
- `nat`
- `mangle`
- `raw`

A firewall of this type splits the network traffic it is part of in three categories(Figure 20.1 and 20.2):

- **input stream**
  ncoming packets that have as their "true" final destination processes local to the firewall itself
  1. a packet enters one of the network interface of the firewall
  2. **PREROUTING** chain of the table **nat** tipically used for destination network address and port translation (DNAT) operations
  3. it's time to route the packet
     happens after DNAT operations take place so that the packet contains its "true" destination address.
  4. the packet enters the **INPUT** chain of the `filter` table. Here is where filtering happens
  5. if allowed the packet reaches its destination (a local process)
- **output stream**
  all outgoing packets originated from processes local to the firewall itself
  1. local process generate a packet
  2. the routing process decides which output interface
  3. the packet enters the **OUTPUT** chain of the `filter` table where filtering happens
  4. the packet enters **POSTROUTING** table of the `nat` table tipically used for source network address and port translation (SNAT) operations
  5. packet exits on one of the network interfaces
- **forward stream**
  all the incoming packets that traverse the firewall itself
  1. the packet enters one of the firewall network interfaces
  2. enters the **PREROUTING** chain of the `nat` table used for destination network address and port translation (DNAT) operations
  3. it's time to route the packet.
     DNAT operations happens before routing decision and filtering
  4. the packet enters the **FORWARD** chain of the `filter` table where all filtering happens
  5. if allowed the packet reaches the **POSTROUTING** chain of the nat table tipically used for source network address and port translation (SNAT) operations.

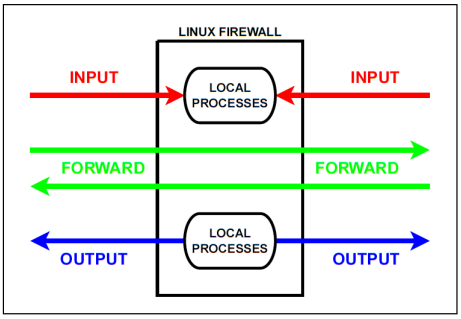


Figura 20.1: Linux firewall network streams

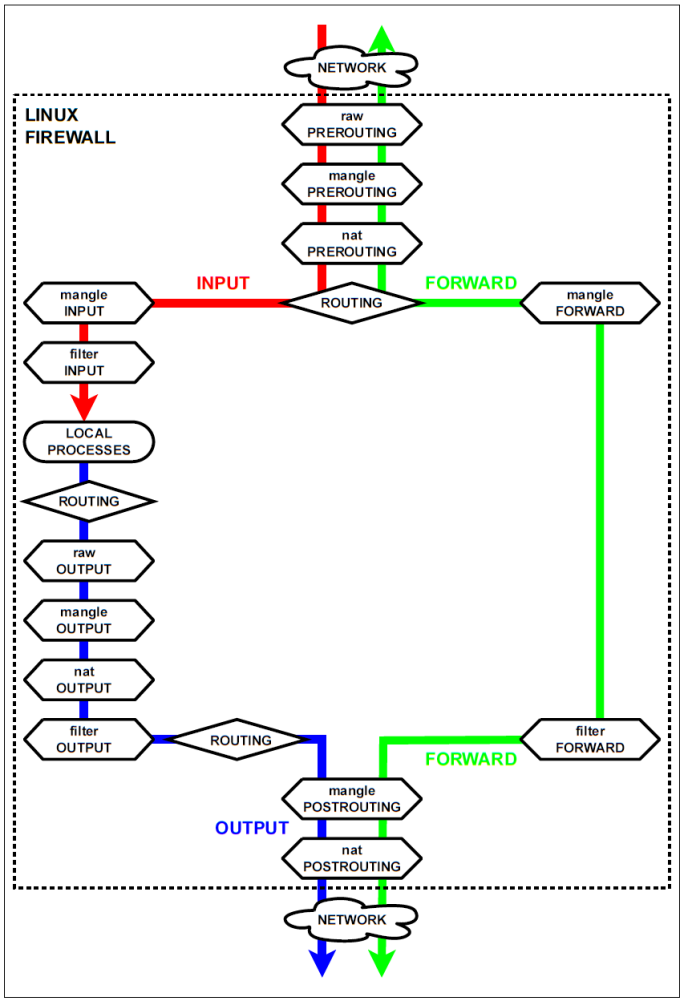


Figura 20.2: Packet stream diagram in details

20.2.1 filter table

It's used for basic packet filtering operations.
The predefined chains are:

- `INPUT`
- `OUTPUT`
- `FORWARD`

Every chain contains a list of rules each defining a matching criteria and an action (or target).
The fate of the packet is decided by the first matching rule. If there's no matching rule at all the default policy of the chain is applied.

The most common targets are:

- `ACCEPT` to accept the packet
- `DROP` to silently discard it
- `REJECT` to discard it but with an error message sent back to the packet sender
- `user-defined-chain` to redirect the packet to a user-defined chain
- `RETURN` to return from the user-defined chain

20.2.2 nat table

It's used for network address and port translation operations and contains the following predefined chains:

- `PREROUTING`
- `OUTPUT`
- `POSTROUTING`

Every chain contains a list of rules (a matching criteria and a target) but the target are packets (in particular addresses and port) manipulation operation for the acceptance of a pakcket.
The most common targets are:

- `DNAT` to change the destination address and port of a packet
- `SNAT` to change the source address and port of a packet
- `MASQUERADE` to change the source address and port of a packet exactly as SNAT but in a scenario where the public IP address of the firewall can change over time
- `REDIRECT` to redirect the packet to the firewall itself

20.2.3 mangle table

It's used for specific packet alteration (such as the Time to Live or the Type of Service fields) and contains the following predefined chains:

- `PREROUTING`
- `INPUT`
- `OUTPUT`
- `FORWARD`
- `POSTROUTING`

20.2.4 raw tablle

It's used for *setting up exceptions to the "conversation" tracking system* and contains the following predefined chains:

- `PREROUTING`
- `OUTPUT`

20.3 iptables usage

Each rule is inserted in a chain by an iptables instruction.

The main possible operations are:

- to append a rule to the chain:
  `iptables [-table table-name] -append chain-name matching-criteria -jump target`
- to delete a rule from the chain:
  `iptables [-table table-name] -delete chain-name matching-criteria -jump target`
- to insert a rule in a specified position of the chain:
  `iptables [-table table-name] -insert chain-name [rulenum] matching-criteria -jump target`
- to replace a previously inserted rule with a new one:
  `iptables [-table table-name] -replace chain-name rulenum matching-criteria -jump target`
- to show the current content of a chain: `iptables [-table table-name] -list [chain-name]`
- to empty a chain from all its rules:
  `iptables [-table table-name] -flush [chain-name]`
- to create a user-defined chain:
  `iptables [-table table-name] -new-chain [chain-name]`
- to delete an empty user-defined chain:
  `iptables [-table table-name] -delete-chain [chain-name]`
- to modify the default policy of a chain:
  `iptables [-table table-name] -policy chain-name target`
- to rename a user_defined chain:
  `iptables [-table table-name] -rename-chain old-chain-name new-chain-name`

The most common matching criteria are:

- the input interface (eth0, eth1, ... and lo for the loopback one):
  `-in-interface interface-name`
- the output interface (eth0, eth1, ... and lo for the loopback one):
  `-out-interface interface-name`
- the source address of the packet:
  `-source ip-address`
  `-source network-address/netmask`
- the destination address of the packet:
  `-destination ip-address`
  `-destination network-address/netmask`
- the packet protocol (icmp, udp, ...):
  `-protocol protocol-name`

For each supported protocol there are further matching criteria:

- • for the icmp protocol, for example, it is possible to specify the ICMP message type (echo-request, echo-reply, ...):
  `-protocol icmp -icmp-type message-type`
- for the udp and tcp protocol, it is possible to specify packet source and/or destination ports:
  `-protocol udp -source-port port-number`
  `-protocol udp -dest-port port-number`
- for the tcp protocol, it is possible to specify the values of any of the TCP flags (SYN, ACK, FIN, RST, URG, PSH, ALL, NONE): `-protocol tcp -tcp-flags list-of-examined-flags list-of-set-flags`

  For example:

```
--protocol tcp --tcp-flags SYN,ACK,FIN,RST SYN
```

The main matching criteria based on **"conversation" state**:

- allow to match packets starting a new "conversation"
  `-match state -state NEW`
- allow to match packets belonging to an already established "conversation"
  `-match state -state ESTABLISHED`
- allow to match packet starting a new "conversation" that is related to an already established one (e.g. FTP protocol)
  `-match state -state RELATED`
  where `RELATED` state doesn't exist for the TCP protocol.

The most common targets for rules contained in filtering chains are: The most common targets for rules

`-jump ACCEPT`	accept the packet
`-jump DROP`	drop silently the packet

`-jump REJECT [-reject-with type]`	where type can be something like
	`icmp-net-unreachable`
	`icmp-host-unreachable`
	`icmp-portunreachable`
	`icmp-proto-unreachable`

`-jump LOG -log-level level` `-jump LOG -log-prefix prefix`	A packet exits a filtering chain only if it matches a rule with **ACCEPT**, **DROP** or a **REJECT** target. The **LOG** target allows many diffent option

contained in natting chains are:

- DNAT to perform destination network address and port translation operations:
  `-jump DNAT -to-destination ipaddr[:port]`
- SNAT to perform source network address and port translation operations:
  `-jump SNAT -to-source ipaddr[:port]]`
- MASQUERADE to perform source network address and port translation operations in all that situations where the public IP of the firewall is dynamically assigned: `-jump MASQUERADE`

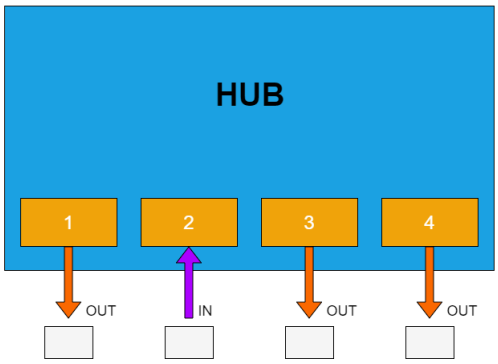
Capitolo 21

Real network devices

21.1 Hub vs Switch

There are two devices that forward packets into the network:

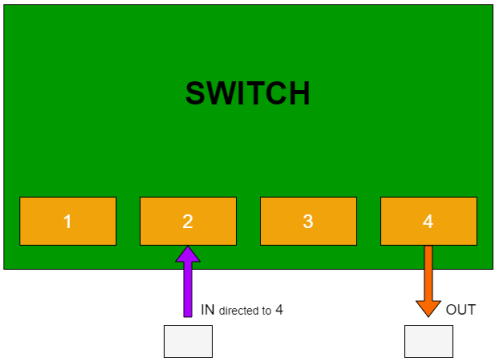
- **Hub**
  A packet that reaches the device is copied and forwarded to the ports of all networks that are connected to the hub.
  In hubs use, there is no routing but only broadcasting. Infact, hubs work on layer 2 and don't do IP identification.



(a) Function scheme

(b) Real Hub device

- **Switch**
  A packet that reaches the device is copied and forwarded to the port of the network that permit the packet to arrive to the destination specified by the source device.



(a) Function scheme

(b) Real Switch device

21.2 Cables

There are two main types of cable to create a link between two network devices:

- **Patch cables**
- **Crossover cables**

21.2.1 Patch cables

Patch cables are electrical or optical cables used to connect ("patch in") one electronic or optical device to another for signal routing.
Devices of different types (e.g., a switch connected to a computer, or a switch to a router) are connected with patch cords.
It is usually produced in many different colors so as to be easily distinguishable, and is relatively short, perhaps no longer than two metres.
Types of patch cords include:

- microphone cables
- headphone extension cables
- XLR connector
- Tiny Telephone (TT) connector
- RCA connector
- 1/4" TRS phone connector cables (as well as modular Ethernet cables)
- thicker
- hose-like cords (snake cable) used to carry video or amplified signals.

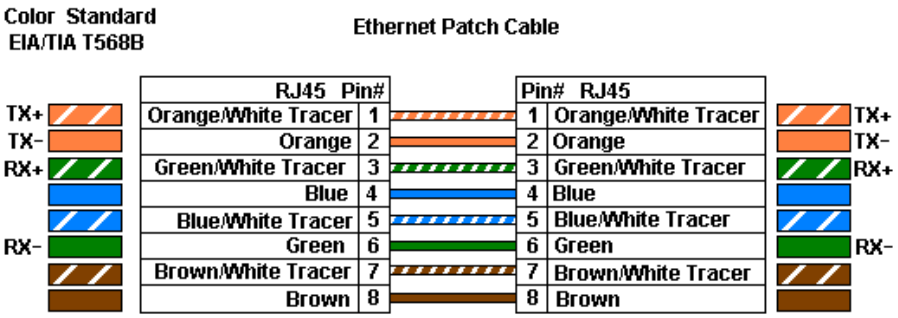


Figura 21.1: Pin connection in Patch cable

21.2.2 Crossover cables

Crossover cables connects two devices of the same type. The crossing wires in a cable or in a connector adaptor allows:

- connecting two devices directly, output of one to input of the other
- letting two terminal (DTE) devices communicate without an interconnecting hub knot, i.e. PCs
- linking two or more hubs, switches or routers (DCE) together, possibly to work as one wider device

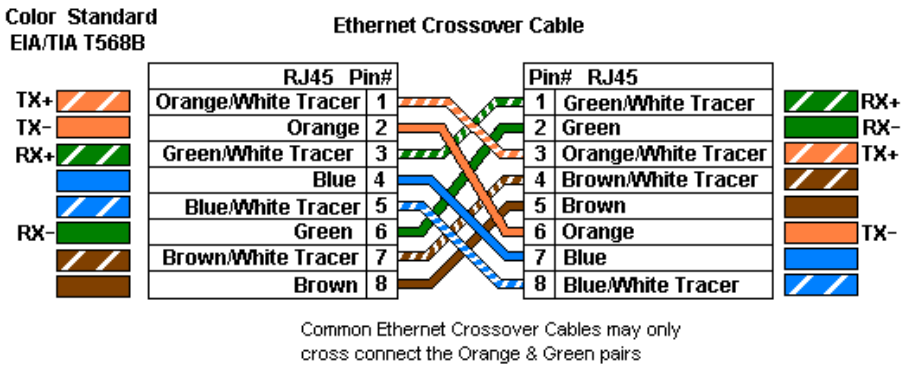


Figura 21.2: Pin connection in Crossover cable

21.3 CISCO router

Cisco Systems, Inc. (CISCO) is a multinational technology conglomerate headquartered in San Jose, California, in the center of Silicon Valley, that develops manufactures and sells **networking hardware**, **telecommunications equipment** and other **high-techonology services and products**.

Figura 21.3: CISCO router 1841

21.3.1 CISCO command modes

There are 5 main command modes[1]:

- **User EXEC**
  commands that application gives you at the opening of the app
- **Privileged EXEC**
  commands that application gives you with the registration as user root. It's used to view and change system parameter of a Cisco router
- **Global Configuraion**
  It's used to modify system wide con guration parameters.

  routerXX#`configure terminal`

  which return the following prompt:

[1] `http://www.cisco.com/en/US/docs/ios/12_0/configfun/command/reference/fun_r.html`

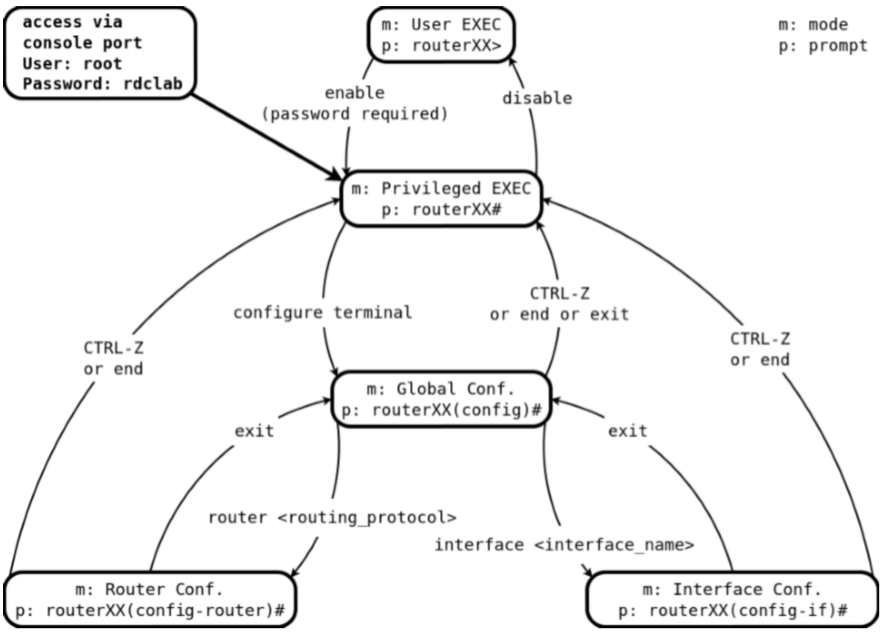


Figura 21.4: Commands modes

routerXX(config)#

- **Router Configuration**
  It's used to make changes to a network interface

  routerXX(config)# `router <routing protocol>`

  which return the following prompt:

  routerXX(config-router)#

- **Interface Configuration**
  It's used to make changes to a network interface

  routerXX(config)# `interface FastEthernet 0/0`

  where FastEthernet is the name of the network interfaces in CISCO routers.
  This command returns the following prompt:

  routerXX(config-if)#

21.3.2 Configuration of CISCO router

To access a Cisco router from one of the PCs, connect the serial port (`/dev/ttyS0`) of the PC (**PCXX**) to the console port of the Cisco router (**routerXX**) via a serial cable.
Then, use the kermit command to establish a remote terminal connection to the router.

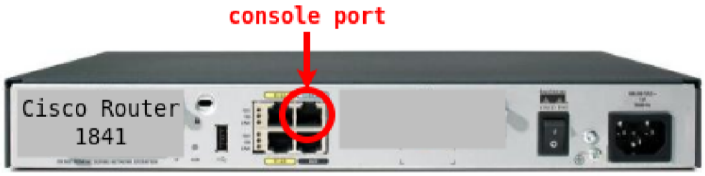


Figura 21.5: Console port

- **On pcXX connect the cisco router following these steps**
  1. Open a termial window and start kermit by typing

     `kermit`

  2. This display the prompt:

     C-kermit>

  3. Select the ttyS0 serial port by typing:

     C-kermit>`set line /dev/ttyS0`

  4. Disable the carrier detection signal by typing:

     C-kermit>`set carrier-watch off`

  5. Connect to the router by typing:

     C-kermit>`connect`

- **Connect as user root** Now you should see the following prompt (*Privileged EXEC prompt*):

  routerXX#

- **Configuration of IP interfaces**
  Once the IP address is set, you need to write **no shutdown** command because in CISCO routers, you

`ip routing`	Enable ip routing
`no ip routing`	Disable ip routing
`show ip route`	Display the contents of the routing table
`show ip cache`	Display the routing cache
`ip route-cache`	Enables route caching (Default is enabled)
`no ip route cache`	Disable route caching
`ip route <destination> <netmask> <gw_address>`	Adds a static route
`ip route <destination> <netmask> <interface>`	Adds a static route
`no ip route <destination> <netmask> <gw_address>`	Deletes a static route
`no ip route <destination> <netmask> <interface>`	Deletes a static route

need to create electrical configuration of the address otherwise changes to IP address don't appear.

```
Router1# configure terminal
Router1(config)# no ip routing
Router1(config)# ip routing
Router1(config)# interface FastEthernet 0/0
Router1(config-if)# ip address <ip_address> <netmask>
Router1(config-if)# no shutdown
Router1(config-if)# interface FastEthernet 0/1
Router1(config-if)# ip address <ip_address> <netmask>
Router1(config-if)# no shutdown
Router1(config-if)# end
```

Capitolo 22

Tools

22.1 traceroute

provides delay measurement from source to router along end-end Internet path towards destination.

```
traceroute [options] IP_address|Mnemonic
```

If the mnemonic name of a site is used (e.g. google.com), traceroute does two preliminary steps:

1. It sends ARP request for DNS server (to understand what is its IP address)
2. It translates Mnemonic name with its IP address (using DNS)

For all **i** (Figure 22.4):

- sends packets starting from setting TTL=1
- router **i** will reply with ICMP TimeExceeded message when TTL becomes equal to 0
- sender measures the Round Trip Time: time interval between transmission and reply
- traceroute increases TTL at each iteration, since the packets doesn't reach the destination

It uses ICMP TimeExceeded messages to discover the from source to destination and make some measurements.

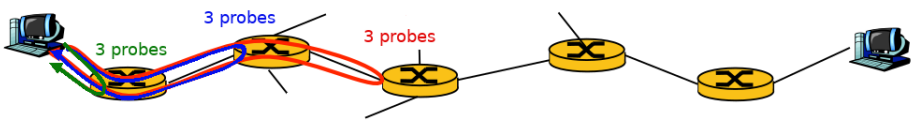


Figura 22.1: How traceroute works.

```
$ traceroute fhda.edu
traceroute to fhda.edu (153.18.8.1), 30 hops max, 38 byte packets
1 Dcore.fhda.edu (153.18.31.254) 0.995 ms 0.899 ms 0.878 ms
2 Dbackup.fhda.edu (153.18.251.4) 1.039 ms 1.064 ms 1.083 ms
3 tiptoe.fhda.edu (153.18.8.1) 1.797 ms 1.642 ms 1.757 ms
```

Figura 22.2: Example of found host.

```
$ traceroute mhhe.com
traceroute to mhhe.com (198.45.24.104), 30 hops max, 38 byte packets
1 Dcore.fhda.edu (153.18.31.254) 1.025 ms 0.892 ms 0.880 ms
2 Ddmz.fhda.edu (153.18.251.40) 2.141 ms 2.159 ms 2.103 ms
3 Cinic.fhda.edu (153.18.253.126) 2.159 ms 2.050 ms 1.992 ms
  ...
16 * * *
17 * * *
.....
```

Figura 22.3: When traceroute doesn't receive response.

22.2 netstat

The netstat command displays network status and protocol statistics.
You can display the status of TCP and UDP endpoints in table format, routing table information, and interface information.
Netstat displays various types of network data depending on the command line option selected.

```
netstat [-m] [-n] [-s] [-i | -r] [-f address_family]
```

The most frequently used options for determining network status are: To understand if an address represents

`-route, -r`	Display the kernel routing tables
`-interfaces, -i`	Display a table of all network interfaces
`-statistics, -s`	Display summary statistics for each protocol

a broadcast in the same network, you have to watch Gateway coloumn and find the MAC broadcast address (all `f`).

22.3 tcpdump

Dump traffic (all the packets) on a network.

```
tcpdump [-B buffer_size] [-c count] [-C file_size] [-e] [-F file] [-i interface] [-I] [-n] [-r file] [-x] [-xx]
                                        [-X] [-XX] [expression]
```

22.3.1 Packet sniffing with wireshark

Packet sniffing is a tecnique based on analysis and collection of packets accross a network link. This technique is considered illegal but it's carried out by ISPs, governments, AD-companies and hacker because the knowledge of sent packets can be used to modify them and add other information (malwares, adds,...) to the packets stream.
Packet sniffing can be done with command **tcpdump**, executed by sender or receiver of packets (in this case packets are sent throught the command **ping**).
Packet sniffing usually can be done only in local network, otherwise it can be considered as interception because strangers can't see the content of packets elaborated by nodes.
The **wireshark** programm can be used to analyze recorded packets by **tcpdump**, previously saved in a file with extension **.pcap**, in which each line is a packet (ARP,ACK,TCP/UDP,...). You can see encapsulation of

`-B buffer_size`	Set the OS buffer size to `buffer_size` [KiB]
`-c count`	Exit after receiving `count` packets
`-C file_size`	Before writing a raw packet to a savefile, check wheter the file is currently larger than `file_size` and, if so, close the current savefile and open a new one
`-e`	Print the link-layer header on each dump line
`-F file`	Use file as input for the filter expression
`-i interface`	Listen on `interface`
`-I`	"Monitor mode" (supported on IEE 802.11 Wifi interfaces and some Operating Systems)
`-n`	Don't convert addresses to names
`-r file`	Read packets from `file` (created before with `-w`)
`-w file`	Write the raw packets to `file` rather parsing and printing them out
`-x`	In addition to headers, print data of each packet
`-xx`	In addition to headers, print data and link level header in hex of each packet
`-X`	In addition to headers, print data of each packet in hex and ASCII
`-XX`	In addition to headers, print data and link level header in hex and ASCII of each packet
`expression`	Selects which packets will be dumped (packets that have `expression=true`)

`-m ttl`	Set the IP Time To Live for outgoing packets. You can only set TTL for request pkts, reply pkts use their default TTL
`-s packetsize`	Set size of payload of packets (# bytes) (Default size depends on Operating System)
`-D`	Set the Don't Fragment bit

original packets and all the headers introduced by a layer.

22.4 ping

It sends Echo request to a host and waits for its Echo reply in order to understand if that host can be reachable.

```
ping [options] Ip_address|Mnemonic_name
```

If there is a `Mnemonic_name`, ping translates it like 22.1.

The main useful options are: At the end of the session of ping, the command prints total number of transmitted packets by source, total number of packets received by destination and a statistichal analysis of RTT (its mean, standard deviation, maximum and minimum),

22.4.1 MTU

It's the max size of IP datagram that can cross the path without being fragmented (avoid fragmentation for no overhead).

$$minMAXpayloadsizeofDLL(i)i \in path$$

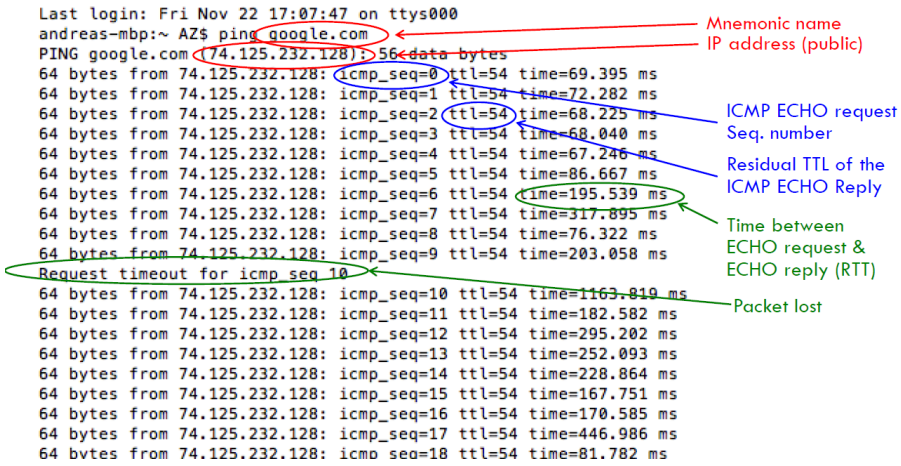


Figura 22.4: Example of ping output.

Estimation of MTU:

1. use `-D` option
2. increase packets size verifying that ECHO reply is received
3. When there is no reply, the previous size of packets was MTU

22.5 ifconfig

It's used to configure a network interface. Without option, it will displays all the available interfaces with their IP addresses, netmasks, etc..

`ifconfig`

The command used to assign an IP address to an interface is:

`ifconfig` Interface IP_address `netmask` Netmask `broadcast` IP_broadcast `up`

`up` turns on the settings of the IP.

The options to manage an IPv6 address are: Files correlated with ifconfig output are:

`add addr/prefixlen`	Add an IPv6 address to an interface
`del addr/prefixlen`	Remove an IPv6 address from an interface

- `/proc/net/dev`
  contains information about network interfaces and their traffic
- `/proc/net/if_inet6`
  contains information about net addresses

The configuration of an interface can be made by appending the following lines to the `/etc/network/interfaces` file like in the following example.

```
auto eth0
iface eth0 inet static
  address 10.0.0.1
  netmask 255.255.255.0
```

Listing 22.1: Example of intefaces file contents

22.6 route

It's used to define the routing table of a node.
The command to add a new entry in the table is:

```
route [-A family |-4|-6] add [-net|-host] target [netmask Nm] [gw Gw] [[dev] If]
```

The command to delete a entry in the routing table of a node is:

```
route [-A family |-4|-6] del [-net|-host] target [gw Gw] [netmask Nm] [[dev] If]
```

The command to show the routing table is:

```
route [-A family |-4|-6]
```

-A inet	**-4**
-A inet6	**-6**

Tabella 22.1: Family of addresses

-net target	the target is a network
-host target	the target is a host
netmask NM	when adding network, to specify the netmask to be used.
gw GW	next hop in routing table
dev If	forces the route to be associated with the specified device the interface is of the node in which you're defining routing table

Examples:

route add -net 127.0.0.0 netmask 255.0.0.0 metric 1024 dev lo

adds the normal loopback entry, using netmask 255.0.0.0 and associated with the "lo" device (assuming this device was previously set up correctly with ifconfig(8)).

route add -net 192.56.76.0 netmask 255.255.255.0 metric 1024 dev eth0

adds a route to the local network 192.56.76.x via "eth0". The word "dev" can be omitted here.

route del default

deletes the current default route, which is labeled "default" or 0.0.0.0 in the destination field of the current routing table.

route del -net 192.56.76.0 netmask 255.255.255.0

deletes the route. Since the Linux routing kernel uses classless addressing, you pretty much always have to specify the netmask that is same as as seen in **'route -n'** listing.

route add default gw GW

adds a default route (which will be used if no other route matches). All packets, using this route, will be gatewayed through "GW".

route add -net 192.57.66.0 netmask 255.255.255.0 gw 0000000ipx4

This command adds the net "192.57.66.x" to be gatewayed through the former route to the SLIP interface

22.7 arp

It manipulates or displays the kernel's IPv4 network neighbour cache. Without options, the command print the current content of the ARP table.

The command to delete a ARP table entry, is:

```
arp -d address
```

The command to set up a new table entry, is:

```
arp -s address hw addr
```

where `hw addr` is dependent on the hardware class.

22.8 telnet

It's a program for testing connection to a remote host that uses commands sent as plain text, without encryption. One of the main aims of using **telnet** is testing if a port is open.

```
telnet [domain_name or ip] [port_number]
```
or
```
telnet [domain_name or ip][:port_number]
```

```
telnet 192.168.0.10 25
```

Listing 22.2: Example 1

```
telnet rpc.acronis.com 443
```

Listing 22.3: Example 2

To have access to a remote host, you can type:

```
telnet -l [username] [domain_name or ip][:port_number]
```

where `-l [username]` indicates the login on remote host as `username`.

22.9 ssh

It's a program for logging into a remote machine and executing commands. It provides secure encrypted communications between two untrusted hosts over an insecure network.
To log into a remote computer `[domain_name or ip]`, type the following command at a shell prompt:

```
ssh [domain_name or ip]
```

If you access into remote device, the server will be added to your list of known hosts (`~/.ssh/known_hosts`).

To log into a remote computer `[domain_name or ip]` with a different `username`:

```
ssh [username]@[domain_name or ip]
```
or
```
ssh -l [username] [domain_name or ip]
```

Once you access into remote host, you can tyoe whatever you want on its shell, but if you want to type only a command, you can add it on the same line of previous ones, after them.

```
ssh sample.ssh.com ls /tmp/doc
```

22.10 lynx

Lynx is a fully-featured World Wide Web (WWW) client for users running cursor-addressable, character-cell display devices (e.g., vt100 terminals, vt100 emulators running on Windows 95/NT or Macintoshes, or any other "curses-oriented" display).
It will display hypertext markup language (HTML) documents containing links to files residing on the local system, as well as files residing on remote systems running Gopher, HTTP, FTP, WAIS, and NNTP servers.
Current versions of Lynx run on Unix, VMS, Windows 95/NT, 386DOS and OS/2 EMX.

```
lynx [options] [path or URL]
```

22.11 dig

`dig (domain information groper)` is a flexible tool for interrogating DNS name servers.
It performs DNS lookups and displays the answers that are returned from the name server(s) that were queried. Most DNS administrators use dig to troubleshoot DNS problems because of its flexibility, ease of use and clarity of output. Other lookup tools tend to have less functionality than dig. Although dig is normally used with command-line arguments, it also has a batch mode of operation for reading lookup requests from a file.
A brief summary of its command-line arguments and options is printed when the -h option is given. Unlike earlier versions, the BIND 9 implementation of dig allows multiple lookups to be issued from the command line.

```
dig [domain_name or ip]
```

22.12 HTTP request methods

A method specifies the operation theat the client requests to server.

Method	HTTP version	Changes in HTTP/1.1
GET	*1.0 e 1.1*	Request for parts of objects
HEAD	*1.0 e 1.1*	Use of conditional headers for request
POST	*1.0 e 1.1*	Management of connection and transmission of messages
PUT	*1.1 (1.0 non standard)*	
DELETE	*1.1 (1.0 non standard)*	
OPTIONS	*1.1*	Extendibility
TRACE	*1.1*	Extendibility
CONNECT	*1.1*	Future use

- **GET**
  It's used to request a resource to the server (message made only by header part).

  ```
  GET /path\_of\_resource
  ```

  It's a secure method that can be:

  - **Absolute**
    request without other parameters
  - **Conditional**
    if the resource uses criteria specified in header (*If-Match, If-Modified-Since, If-Range, ...*)
  - **Partial**
    if a resource is subpart of another stored resource

- **HEAD**
  It's another version of GET, used mostly for control and debugging. The server replies only with headers of required resource and it doesn't send payload.
  It's secure method used to verify:

  - **URL validity**
    the resource exists and has length different to 0
  - **URL accessibility**
    the resource is reachable by the server and it doesn't need authentication procedures
  - **Control with respect to the content of cache**
    the resource isn't changed with respect to the resource stored in cache

- **POST**
  It's used to send information from client to server:

  - updates an existing resource
  - gives data to input
  - sends data in the payload of a request (payload od GET)

  It'sa an unsecure method. Server can replies in 3 different ways:

`200 OK`	data have been received by the specified resource and answer was given to it
`201 Created`	data have been received but the resource didn't exist so it has been created
`204 No content`	data have been received and given to the specified resource but answer wasn't given to it

- **PUT**
  It's used to transmitt information from client to server, creating or substituting specified resource.
  The difference between PUT and POST is:

  - l'URL di **POST** identifies the resource that will manage information sent by the request
  - l'URL di **PUT** identifies the resource sent by the request. It's the resource that we would like to obtain by using GET with the same URL

  It doesn't give security of access control or locking.

22.13 SMTP commands

After the connection with `tlcnet servername 25` if non-encrypted port or `tlcnet servername 2525` if port 25 is filtered (port 2525 is opened on all SiteGround servers) or `tlcnet servername 465` for SMTP securely, you can use **basic SMTP commands**:

- `HELO` (Hello)
  The client sends this command to the SMTP server to identify itself and initiate the SMTP conversation. The domain name or IP address of the SMTP client is usually sent as an argument together with the command (e.g. *"HELO client.example.com"*).

301 Moved Permanently	Redirection: moved resource
304 Not Modified	Not modified resource
400 Bad Request	Request message not understood by server
401 Unauthorized	The resource requires authentication
403 Forbidden	Forbidden access
404 Not Found	Resource doesn't exist
412 Precondition Failed	access to the target resource has been denied
500 Internal Server Error	Unexpected error that doesn't permit required service
505 HTTP Version Not Supported	Required version of HTTP is not supported by server

Tabella 22.2: HTTP request codes

If-Match: <etag_value>, <etag_value>, ... Succeeds if the ETag of the distant resource is equal to one listed in this header
If-None-Match: <etag_value>, <etag_value>, ... Succeeds if the ETag of the distant resource is different to each listed in this header If-None-Match: * * is a special value representing any resource
If-Modified-Since: <day-name>, <day> <month> <year> <hour>:<minute>:<second> GMT Succeeds if the Last-Modified date of the distant resource is more recent than the one given in this header
If-Unmodified-Since: <day-name>, <day> <month> <year> <hour>:<minute>:<second> GMT Succeeds if the Last-Modified date of the distant resource is older or the same than the one given in this header
If-Range: <day-name>, <day> <month> <year> <hour>:<minute>:<second> GMT If-Range: <etag> Similar to If-Match, or If-Unmodified-Since, but can have only one single etag, or one date. If it fails, 206 Partial Content response

Figura 22.5: All the possible conditions of HTTP requests.

- `MAIL FROM`
  Specifies the e-mail address of the sender. This command also tells the SMTP server that a new mail transaction is starting and makes the server to reset all its state tables and buffers etc.
  This command is usually sent as the first command after the identifying and login process.

- `MAIL, SEND, SOML` and `SAML`
  they start mail transfer

- `RCPT TO`
  Specifies the e-mail address of the recipient. This command can be repeated multiple times for a given e-mail message in order to deliver a single e-mail message to multiple recipients.

- `DATA`
  Starts the transfer of the message contents (body text, attachments etc). After that the DATA command has been sent to the server from the client, the server will respond with a 354 reply code.
  After that, the message contents can be transferred to the server. When all message contents have been sent, a single dot (".") must be sent in a line by itself.

- `RSET` (Reset)
  If the RSET command is sent to the e-mail server the current mail transaction will be aborted.

- `VRFY` (Verify)
  This command asks the server to confirm that a specified user name or mailbox is valid (exists).
  If the user name is asked, the full name of the user and the fully specified mailbox are returned.

- `NOOP` (No operation)
  Does nothing else than makes the receiver send an OK reply. The main purpose is to check that the server is still connected and is able to communicate with the client.

- `QUIT`
  Asks the server to close the connection.

```
S: 220 smtp.server.com Simple Mail Transfer Service Ready
C: HELO client.example.com
S: 250 Hello client.example.com
C: MAIL FROM:<mail@samlogic.com>
S: 250 OK
C: RCPT TO:<john@mail.com>
S: 250 OK
C: DATA
S: 354 Send message content; end with <CRLF>.<CRLF>
C: <The message data is sent>
C: .
S: 250 OK, message accepted for delivery: queued as 12345
C: QUIT
S: 221 Bye
```

You can use also **Extended SMTP (ESMTP) Commands**:

- `EHLO` (Extended Hello)
  Same as HELO but tells the server that the client may want to use the **Extended SMTP (ESMTP)** protocol instead

- `AUTH` (Authentication)
  The AUTH command is used to authenticate the client to the server. The AUTH command sends the clients username and password to the e-mail server.
  AUTH can be combined with some other keywords as PLAIN, LOGIN and CRAM-MD5 (e.g. AUTH LOGIN) to use different login methods and different levels of security.

  ```
  S: 220 smtp.server.com Simple Mail Transfer Service Ready
  C: EHLO client.example.com
  S: 250-smtp.server.com Hello client.example.com
  S: 250-SIZE 1000000
  S: 250 AUTH LOGIN PLAIN CRAM-MD5
  C: AUTH LOGIN
  S: 334 VXNlcm5hbWU6
  C: raffaele
  S: 334 UGFzc3dvcmQ6
  C: password_di_raffaele
  S: 235 2.7.0 Authentication successful
  ```

  where **"VXNlcm5hbWU6"** is the BASE64 encoded text for the word **"Username"** and **"UGFzc3dvcmQ6"** is the BASE64 encoded text for the word **"Password"** in the example above.

- `STARTTLS` (Start Transport Layer Security) E-mail servers and clients that uses the SMTP protocol normally communicate using plain text over the Internet.
  The communication often goes through one or more routers that is not controlled or trusted by the server and client.
  This communication can be monitored and it is also possible to alter the messages that are sent via the routers.

  To improve security, an encrypted TLS (Transport Layer Security) connection can be used when communicating between the e-mail server and the client.
  TLS is most useful when a login username and password (sent by the AUTH command) needs to be encrypted.
  TLS can be used to encrypt the whole e-mail message, but the command does not guarantee that the whole message will stay encrypted the whole way to the receiver; some e-mail servers can decide to send the e-mail message with no encryption.
  But at least the username and password used with the AUTH command will stay encrypted. Using the STARTTLS command together with the AUTH command is a very secure way to authenticate users.

```
S: 220 smtp.server.com Simple Mail Transfer Service Ready
C: EHLO client.example.com
S: 250-smtp.server.com Hello client.example.com
S: 250-SIZE 1000000
S: 250-AUTH LOGIN PLAIN CRAM-MD5
S: 250-STARTTLS
S: 250 HELP
C: STARTTLS
S: 220 TLS go ahead
C: EHLO client.example.com *
S: 250-smtp.server.com Hello client.example.com
S: 250-SIZE 1000000
S: 250-AUTH LOGIN PLAIN CRAM-MD5
S: 250 HELP
C: AUTH LOGIN
S: 334 VXNlcm5hbWU6
C: adlxdkej
S: 334 UGFzc3dvcmQ6
C: lkujsefxlj
S: 235 2.7.0 Authentication successful
C: MAIL FROM:<mail@samlogic.com>
S: 250 OK
C: RCPT TO:<john@mail.com>
S: 250 OK
C: DATA
S: 354 Send message, end with a "." on a line by itself
C: <Message data is sent>
S: .
S: 250 OK, message accepted for delivery: queued as 12345
C: QUIT
S: 221 Bye
```

where the second `EHLO` on line **10** is used to start the communication again but this time the communication will be encrypted until the `QUIT` command is sent.

- `SIZE`
  The SMTP server can inform the client what is the maximum message size and the client can inform the SMTP server the (estimated) size of the e-mail message that will be sent.
  The client should not send an e-mail message that is larger than the size reported by the server.
  Normally there is no problem if the message is larger than the size informed by the client to the server.

```
S: 250 SIZE 1000000
C: MAIL FROM:<mail@samlogic.com> SIZE=500000
```

- `HELP`
  This command causes the server to send helpful information to the client, for example a list of commands that are supported by the SMTP server.

220	SMTP Service ready
221	Service closing
250	Requested action taken and completed
251	The recipient is not local to the server, but the server will accept and forward the message
354	Start message input and end with <**CRLF**>.<**CRLF**> This indicates that the server is ready to accept the message itself (after you have told it who it is from and where you want it to go)
503	If sent commands, during transfer, are inserted out of order and so SMTP server doesn't change its state

Tabella 22.3: SMTP request codes

22.14 POP3 commands

After the connection with `tlcnet servername 110` if non-encrypted port or `tlcnet servername 995` POP3 securely, you can use **basic SMTP commands**:

- `USER <username>` It's used to insert the username of the account
- `PASS <password>`
  It's used to insert the password of the account
- `STAT`
  displays the number of messages currently in the mailbox and the size in bytes
- `LIST`
  used to get a summary of messages where each message number is shown with the size in bytes next to it
- `RETR message_number`
  used to retrieve a particular message, where the first message number is 1
- `DELE message_number`
  Use the DELE command to delete a particular message
- `RSET`
  You can reset the session to its initial state using the RSET command. This will undelete all messages deleted using DELE.
- `TOP message_number number_of_lines`
  Not all mail servers support the POP3 TOP command but if they do it is useful for only retrieving part of the message. It works like RETR but you also specify the number of lines of the message body you would like returned. All the headers will always be returned.
- `QUIT`
  Once you have completed your POP3 converstion simply log off by using the POP3 QUIT command

```
C: telnet <servername> 110
S: +OK POP3 mail.domain.net v2003.83 server ready
C: USER myusername
S: +OK User name accepted, password please
C: PASS mysecretpassword
S: +OK Mailbox open, 3 messages
C: STAT
S: +OK 3 5467
C: LIST
S: +OK Mailbox scan listing follows
S: 1 1823
S: 2 1825
S: 3 1819
S: .
C: DELE 1
S: +OK Message deleted
C: RSET
S: +OK Reset state
C: TOP 1 10
S: +OK Top of message follows
S: --- all message headers ---
S: --- first 10 lines of body ---
S: .
C: QUIT
S: +OK Sayonara
```

22.15 IMAP commands

After the connection with `tlcnet servername 143` if non-encrypted port or `tlcnet servername 993` for IMAP securely, you can use commands:

- `LOGIN [username] [password]`
  Identifies the client to the server and carries the plaintext password authenticating this user
- `LIST [flags] [folder separator] [search term]`
  Returns a subset of names from the complete set of all names available to the client
- `SELECT [mailbox]`
  Selects a mailbox so that messages in the mailbox can be accessed
- `FETCH [mail number] body[header]` and `FETCH [mail number] body[text]` and `FETCH [first]:[last] flags`
  Retrieves data associated with a message in the mailbox
- `LOGOUT`
  Informs the server that the client is done with the connection

22.16 FTP commands

After the connection with `ftp IP_address`. Commands are sent as ASCII text over control channel, the main commands are:

- `USER username`
  inserts the username of the user that wants to connect itself
- `PASS password`
  inserts the password of the user that wants to connect itself
- `LIST`
  return list of files in current directory
- `RETR`
  filename retrieves (gets) file
- `STOR filename`
  stores (puts) file onto remote host

331	Username OK, password required
125	data connection already open; transfer starting
425	Can't open data connection
452	Error writing file

Tabella 22.4: FTP request codes

22.17 DNS files

- `/etc/resolv.conf`
  contains name of default DNS server to which queries are sent automatically

```
nameserver 192.168.0.11
search lugroma3.org
```

Listing 22.4: Example of content of /etc/resolv.conf

- `/etc/bind/named.conf`
  configuration on the name servers specifies associations between zones and name servers

```
.......
zone "." {
type hint;
file "/etc/bind/db.root";
};
.......
// add entries for other zones below here
zone "lugroma3.org" {
type master;
file "/etc/bind/db.org.lugroma3";
};
```

Listing 22.5: Example of content of /etc/bind/named.conf

- `db.name_DNS`

  configuration on the name servers specifies information about the root name servers.It's composed by resource records with form:

  `<domain> <class> <type> <rdata>`

  - <**domain**>
    the record owner (domain to which the record refers)
  - <**class**> usually IN (=Internet system); may be HS (=hesiod) or CH (=chaos)
  - <**type**>

A	**a host address.**
A6	**an IPv6 address.**
AAAA	**Obsolete format of IPv6 address**
AFSDB	(x) location of AFS database servers. Experimental.
CERT	holds a digital certificate.
CNAME	identifies the canonical name of an alias.
DNAME	for delegation of reverse addresses. Replaces the domain name specified with another name to be looked up. Described in RFC 2672.
GPOS	Specifies the global position. Superseded by LOC.
HINFO	identifies the CPU and OS used by a host.
ISDN	(x) representation of ISDN addresses. Experimental.
KEY	stores a public key associated with a DNS name.
KX	identifies a key exchanger for this DNS name.
LOC	(x) for storing GPS info. See RFC 1876. Experimental.
MX	**identifies a mail exchange for the domain. See RFC 974 for details.**
NAPTR	name authority pointer.
NSAP	a network service access point.
NS	**the authoritative nameserver for the domain.**
NXT	used in DNSSEC to securely indicate that RRs with an owner name in a certain name interval do not exist in a zone and indicate what R
PTR	**a pointer to another part of the domain name space.**
PX	provides mappings between RFC 822 and X.400 addresses.
RP	(x) information on persons responsible for the domain. Experimental.
RT	(x) route-through binding for hosts that do not have their own direct wide area network addresses. Experimental.
SIG	("signature") contains data authenticated in the secure DNS. See RFC 2535 for details.
SOA	**identifies the start of a zone of authority.**
SRV	information about well known network services (replaces WKS).
TXT	text records.
WKS	(h) information about which well known network services, such as SMTP, that a domain supports. Historical, replaced by newer RR SRV.
X25	(x) representation of X.25 network addresses. Experimental

Figura 22.6: Available record types.

  - <**rdata**>
    record data (depends on the record type)

```
.              IN   NS   ROOT-SERVER.
ROOT-SERVER.   IN   A    192.168.0.5
```

Listing 22.6: Example of content of /etc/bind/db.root (DNS root)