Practical Network Defense

NETWORKING 101

What is the Internet?

We define the **Internet** as a **hierarchical and interconnected network of networks**.
The hierarchical system can be divided as follows:

- *Internet backbone:* connections that are very far away, even thousands of km, and that interconnects the main ISPs' backbones
- *ISP backbone:* connections that spread across a large geographical space, like countries, and provide connection to local organizations
- *Organization backbone:* connections that spread across a limited geographical space, like cities, and provide connections to local networks
- *LANs and end systems*

We also say that connection can be **public** or **private**, depending on the scope.

We define a set of **internet standards**, an agreed set of rules on some specific implementation's aspects, that were created to ensure properties like compatibility.

Internet architecture

The connection to the internet that we're receiving at home passes through a lot of different components, and they are the main core of the network:

- *Root DNS Server:* external to all global ISPs, is an independent service paid by ISPs to allow them to provide their services in the network
- *ISP backbone:* internal network of an ISP, used to forward packets to their subscribers and interconnects other ISPs
- *ISP:* the Internet Service Provider is the company you're paying to handle all the packets that you've generated and that are destined to you
- *LAN:* the Local Area Network is a network made of devices that belong to the same domain, meaning that they can reach one to each other without passing through the main network

We also say that the Internet network can be divided in two main parts:

- *Edge:* all devices that are not part of the main infrastructure, like end users, public servers, data centers, etc…
- *Core:* all devices that contribute to maintaining the internet alive, like ISPs

In particular, the **core** of the network is a full-mesh topology network that interconnects all ISPs' backbones one to another, in order to reach every destination in the world.
This network is made of routers that work together to compute the most efficient path when routing a packet from its source to its destination, and they are all interconnected using optic fiber links which are very fast.
Also, since the network is constantly changing, there are several distributed algorithms that adapt the *routing tables* according to the internet's conditions, meaning that they are generated and maintained in real time.

ISPs and IXPs

We can distinguish **Internet Service Providers** in the different categories:

- *Tier 1:* they are the main core of the network and provide connectivity all around the world thanks to their full-mesh topology that interconnect them, only a dozen
- *Tier 2:* they provide connectivity among entire regions, and rely on tier-1 ISPs to send packets in other regions that are not under their control
- *Tier 3:* they provide connectivity among small geographical areas, and they are the company that end users pay to access the internet

When a certain organization does not know how to route a packet towards its destination, they usually forward it to an upper-tier ISP since they have a much wider area of connectivity, but doing this operation is not free and comes at a high price (especially with respects to tier-1 ISPs).
To avoid the cost of asking a global-scope ISP to route a packet, **Internet Exchange Points** were born as a cheaper solution that also allows for a more complex and complete interconnectivity between ISPs.

The IXP is, essentially, a company that functions as a *switch* among different ISPs, allowing different regions or countries to interconnect one to another without the need of an upper-tier ISP as intermediate.
Since the operation of switching is relatively cheap, many ISPs pay to have a direct line inside these companies.

Protocols

We define **protocol** the set of rules that defines a specific service.
Usually, they define:

- *procedure rules:* the type of messages to be exchanged as well as their syntax and semantic, in which sequence they need to be exchanged and which operation to perform with respect to messages and events
- *message format:* the format, size and coding of the message
- *timing:* time to wait between any event

One of the most used strategies in implementing protocols is **modularization**, where all functions are splitted in several smaller *layers*: this allows for an easier maintenance of each functionality, since each layer can be developed independently without disturbing the other, and it also helps in hiding the entire implementation details.

Examples of protocols are *packet switching*, where every packet takes an independent route toward its destination, and all *network congestion* and *flow control* mechanisms.

Local Area Networks

The **local area network** is a network used by organizations or end users where all devices can talk to each other without traversing the public network, and they rely on edge routers for any connection that is outside of the local scope.

Access layer

The **access layer** provides connectivity among stations or nodes in the same network, meaning they can directly communicate one to another, and is constituted by networks with endpoints of the same local management.

Ethernet protocol
One of the oldest methods of cabled connection is via **ethernet**.
Each host in an ethernet network has a *Network Internet Card* with a (generally) fixed so called *Media Access Control* address, or *MAC*.
This address is 48 bits long and **uniquely** identifies the host inside the network, and it is very important because each host will only process packets that are intended for it.

The packets have a fixed format, that can be described as follows:
- preamble (7 bytes)
- start frame delimiter (1 byte)
- destination IP address (6 bytes)
- source IP address (6 bytes)
- type (2 bytes)
- data (46-1500 bytes), also the only field of the packet which has variable size
- frame check sequence (4 bytes)

In the ethernet protocol, all hosts are connected via a shared *transmission system*, that acts as if they were all connected to the same medium: this is usually achieved with the usage of *switches* or *hubs*, meaning all hosts connects their ethernet cable to these devices and are interconnected one to another.

An ethernet network constitutes a **broadcast domain** where, ideally, all frames sent in the same domain are potentially received by all the hosts that belong to it.
In reality, *switches* segment the network to limit the explosion of packets in a network, so only broadcast messages are effectively replicated.
To do this, all switches have a *MAC Table*, that correlates a source MAC addresses and the port from which a packet from that address was received: given this information, the device will replicate the packet only on the segment of the network where the destination MAC address is.

This type of network makes high use of broadcast messages, so even if we split large networks is several *broadcast domains*, it still remains inefficient.
To solve this, we can define a *logical* division of the networks using IP as the *distribution layer*.

IP Protocol
While ethernet is based on a fixed *physical* address, which remains the same wherever you go and that could be changed but it is not advised to do so, the **Internet Protocol** is based on *logical addresses*.
Each host assigns an IP address to its NIC, meaning that it depends on the network that is originating the connection, and such addresses can be used to identify and reach other networks and hosts.
Furthermore, each network has a *default gateway*, which is the main device, and it allows all the packets that are originated inside the network to be routed toward their external destination.

This protocol also acts as a **distribution layer**, meaning that it interconnects local networks one to another but also *Autonomous Systems*.

IP Addresses
An IP address is made of 32 bits, organized in four octets, separated by dots for human readability.
Given a certain address, it can essentially be divided in two parts:
- *network part:* sets of bits that define the network which it belongs to
- *host part:* set of bits that define the computer belonging to the address

To know whether two certain IP addresses belongs in the same network, we can use the **subnet mask**, which is the property that defines the boundary between the network portion and the host portion of the address.
To do this, we can perform the following operations:
1. translate both addresses and the netmask in binary values
2. perform the AND bit-wise between the netmask and each address separately
3. if the AND returns the same values on both addresses, they are in the same network, otherwise they are not

When using IP addresses, we first need to distinguish between **local** and **remote** addresses:

- *local address:* used when the destination is in the same network, meaning there is a direct connection or at least a path that interconnects them without the need of passing through the public network
- *remote address:* used when the destination is NOT in the same network, meaning you need to forward the packet to the default gateway, which will then determine where the packets need to be forwarded in order to reach its destination.

The next division defines the **type** of the address:

- *unicast:* packets that are being sent to a single destination host
- *broadcast:* packet that are being sent to every host on a network or subnet
- *multicast:* packets that are being sent to a subgroup of host on a network or subnet

Then we define the **allocation class** of IP addresses:

- *Class A:* 8 bits of netmask and 24 bits for host addresses, from 0.0.0.0 to 127.255.255.255
- *Class B:* 16 bits of netmask and 16 bits for host addresses, from 128.0.0.0 to 191.255.255.255
- *Class C:* 24 bits of netmask and 8 bits for host addresses, from 192.0.0.0 to 223.255.255.255
- *Class D:* multicast addresses, from 224.0.0.0 to 239.255.255.255
- *Class E:* reserved addresses, from 240.0.0.0 to 255.255.255.255

Lastly, we differentiate between **routable** and **non-routable** addresses:

- *routable:* addresses that are unique on the internet and that can be reached by any public router
- *non-routable:* private addresses (for example assigned by a VPN) that should not be publicly reachable, and every packet destined to a private network that manage to reach the public network should be dropped for security and privacy reasons

In particular, *non-routable* addresses are defined in specific IP ranges:

- 10/8 prefixes: from 10.0.0.0 to 10.255.255.255.255
- 172.16/12 prefixes: from 172.16.0.0 to 192.31.255.255
- 192.168/16 prefixes: from 192.168.0.0 to 192.168.255.255

Given an IP address and the number of bits used by the netmask, we can compute various information.
Let's assume the address 192.168.8.0/24, for example:

1. since 24 bits are taken from the netmask, we can define the *network part* which is 192.168.0
2. the remaining 32 - 24 = 8 bits are reserved for the *host part*
3. given 2^8 = 255 addresses to assign, 2 of them will be reserved for special purposes, therefore:
    a. 192.168.0.0 is the *network address*
    b. 192.168.0.1 to 192.168.0.254 are *host addresses*
    c. 192.168.0.255 is the *broadcast address*

Special types of IP addresses

Using only 8 bits for each octet, meaning a range between 0-255, gives poor flexibility with respect to the aforementioned *classes*.
To solve this problem, we define the *Classless Inter-Domain Routing*, which allows for a **classless** notation for the netmask: we only need to specify how many bits of the 32-bit total define the *network address*, all the remaining bits will automatically be used for *host addresses*.

Another particular type of networks are **point-to-point links**, meaning that only two devices belong to the network, and using a 30 bits netmask is a waste since the *broadcast* address will have the same functionality as the *unicast* address.
To solve this, 31 bits prefixes have been defined.

NETWORK TRAFFIC MONITORING

The communication between hosts in a network is organized in different tasks, and each one of them is assigned to a **layer**.
The task of a **layer** involves the exchange of messages that follow a set of rules define by a **protocol**, in the following way:

- offers a service to the upper *layer*
- exploits a service offered by a lower a lower *layer*

When sending data, each layer will add some *protocol information* and ulterior data that will be used by the layer below.
The bottom layer, which is the *physical layer*, will then send the data over the physical medium to the destination.

On the contrary, when receiving data, each layer will read the appropriate *protocol information* and forward the data to the layer above until it reaches the *application layer*, also called "top of the stack".

ISO/OSI Model

The most common used model is the **ISO/OSI**, which defines 7 layers:

- *Application Layer:* generates data that needs to be sent outside and process incoming data
- *Presentation Layer:* manipulates the data to present it in some way (ex. encryption, encoding, etc…)
- *Session Layer:* defines a communication session with the destination's application
- *Transport Layer:* responsible for the delivery of the data, distinguishes between several instances running (maybe of the same application) on the host machine and responsible to re-arrange fragmented packets
- *Network Layer:* realizes the routing mechanism
- *Data-Link Layer:* realizes the local network communication
- *Physical Layer:* responsible for converting the data in the most appropriate form to send it through the physical medium

Ideally, we can also provide the following alternate explanation of the following layers:

- *Transport Layer:* gives the illusion of a direct end-to-end connection between two processes on two different systems
- *Network Layer:* realizes the transfer of data between any two nodes in the network
- *Data-Link Layer:* realizes the transfer of data between two nodes that are directly connected, via direct cable or shared medium

TCP/IP Model

The **TCP/IP** model is a "simplified" version of the *ISO/OSI* model, more focused on the network part, and it defines 4 layers:

- *Application Layer:* contains the application, presentation and session layers of the ISO/OSI model, essentially represents all the operations that are managed by the application in a single block
- *Transport Layer:* same as above
- *Internet Layer:* same as the network layer above
- *Network Access Layer:* contains the data-link and physical layer of the ISO/OSI model, essentially represents all the operation needed to physically send a packet in the network in a single block

Examples of protocol for each layer

- *Application Layer:* SMTP, HTTP (internet name), FTP, etc…
- *Transport Layer:* TCP and UDP ports (65535), each one having a different application behind
- *Internet/Network Layer:* IP, ICMP, etc…
- *Data-Link Layer:* Ethernet and Wi-Fi (MAC address), ARP, etc…

Client-Server Communication
When exchanging data between two hosts, there must be a 1:1 mapping between the respective layers.
The source's *application layer* inserts the data and all the information needed to reach the destination address, that will later be used by both the *transport layer* and *network layer*.
Then, the source's *link layer* inserts all the information about the next-hop needed to reach the destination, its own MAC address and the MAC address of the next-hop.
Lastly, the source's *physical layer* sends the packet in the network, according to the medium and the destination specified.

It's important to observe that the communication happens **logically** on the *application layer*, while the real exchange happens thanks to all the other layers.

Encapsulation
In the TCP/IP protocol, when creating a new packet, each layer will add its own set of information that will then be used by the destination to forwards the packet to the application correctly.
It works in the following way:

1. The *application layer* generates the data, adds its *header* and then forwards it to the lower layer
2. The *transport layer* adds the source's port, the destination's port, adds its header and then forwards it to the lower layer
3. The *internet layer* adds the source's IP and destination's IP, computes the routing methods and the MAC address of the next-hop, adds its header and then forwards it to the lower layer

On the other hand, when receiving a packet, each layer will read its own reader and remove it before forwarding the packet to the upper layer.

IPv4 Packet Structure

- *Version* (4 bit): specifies the IP protocol's version (in this case 0100 -> 4)
- *Internet Header Length* (4 bit): specifies the dimension of the header up until the data field starts, which has a minimum length of 20 bytes
- *Type of Service* (8 bit): used to prioritize certain types of traffic to provide better use experience
- *Total length* (16 bit): specifies the total dimension of the packet (header + data)
- *Identification* (16 bit): used from fragmentation to uniquely identify a fragment, it depends on the MTU
- *Flags* (3 bit): used for special purposes
  - Bit 0: reserved, has to be zero
  - Bit 1: "Don't fragment" bit
  - Bit 2: "More fragment" bit
- *Fragment Offset* (15 bit): see *Identification*
- *Time To Live* (8 bit): specifies how many hops a packet can pass through before dying, in which case an ICMP packets is generated to advise the source that its packets have been dropped before reaching the destination
- *Protocol* (8 bit): specifies the protocol used by the communication, TCP or UDP
- *Header Checksum* (8 bit): used to detect corrupted packets
- *Source Address* (32 bit)
- *Destination Address* (32 bit)
- *Options* (variable size, multiple of 32)
- *Data* (variable size)

Ports
Each host in a network has 65535 ports to send/receive packets, which are managed by both the OS and the applications that are running on the system.
When talking about **source ports**, they are randomly chosen by the OS in the range of *ephemeral ports* (>49152), while **destination ports** are application specific.
There are also *well-known ports*, which are ports that have been pre-assigned for standard internet applications, like:

- 25: SMTP
- 80: HTTP
- 143: IMAP
- etc…

TCP and UDP
The **TCP** and **UDP** are the two main protocols used by the *transport layer*, and although they both implement the operation of sending a packet, they do it in very different ways.

The **TCP** protocol is *connection-oriented*, which means that is built in such a way that the delivery of the packet is guaranteed.
It implements control flow mechanism that track down all the packets that are sent in the channel, advising the source of retransmitting a packet if needed, and also allowing the flow to be adjusted so that packets are not dropped due to congestion.
Furthermore, in case of fragmentation, it is able to recognize all fragments that belong to a certain packet and rearrange them to recover the original data.
Services that rely on TCP are, for example:

- *FTP* on ports 20 and 21
- *SSH* on port 22
- *Telnet* on port 23
- *SMTP* on port 25
- *HTTP* on port 80
- *IMAP* on port 143
- etc…

The **UDP** protocol is *connectionless*, which means that the delivery of a packet is not guaranteed, and it work in a best-effort manner.
It does not implement any form of control over the packet that are delivered and those who are not, meaning that you cannot know which packets have been delivered and which are lost, unless you talk with the destination address.
Services that rely on UDP are, for example:

- *DNS* on port 53
- *DHCP* on ports 67 and 68
- etc…

The difference between the two is also reflected in their headers, where the *TCP header* has a much larger size and control information, while the *UDP header* contains the bare minimum needed to send a packet.

TCP reliable connection mechanism
To implement its *connection-oriented* property, this protocol uses a mechanism called **3-Way Handshake**, where both hosts first ensure that a proper connection has been set up before start transmitting.
The pattern is the following:

1. the first host sends a packet containing the SYN flag set to 1, without any data, and chooses as *sequence number* a random value ensuring that hijacking the connection will be very hard
2. the second host answers with a packet containing the SYN and the ACK flags set to 1, chooses a random value as *sequence number* and sets the *ack* value to the sequence number received in the first step incremented by one
3. the first host finally answers with a packet containing the ACK flag set to 1, then it uses both the sequence number and ack value received in the second step as responses, specifically setting the *sequence number* equal to the ack incremented by one and the *ack* equal to the sequence number incremented by one.

DNS

The **DNS** protocol is a service to get the IP address starting from a *human-friendly* domain.
It uses a *hierarchy* system of servers, where each one of them is responsible in resolving a certain part of the address, and the query is performed in a *recursive* manner.

Each hostname can be divided into different parts that, altogether, form the final address the client will type in its browser's search bar.

To **resolve** the IP address of a certain hostname, the pattern is the following:

1. the host sends a request to the *local DNS resolver* for a certain domain name
2. assuming the domain is not available locally, the host asks its ISP for the external DNS server to query
3. the DNS server starts a recursive query to the following servers:
    a. first it will query the *root DNS server*, which contains information about which top-level domain (TLD) server needs to be queried next
    b. then it will query the *TLD server* provided before, which will solve the rightmost section of the domain (.org, .com, .net, …), and provide the server that contains all the IPs that end with said TLD
    c. lastly, it will query the *authoritative DNS server* provided before, which will return the IP address of the domain requested
4. the IP address resolved is returned to the ISP's DNS server, which will store in inside the DNS cache, and send it to the host that initially requested this address (which also stores the IP inside its cache)

CAPTURING PACKETS
Packets that are flowing through the network can be captured, which means storing them even if not directed to our host.
Some examples of tool that allow this are:
- *dumpcap*
- *Wireshark*
- *tcpdump*
- *Netflow*
- *Zeek*

Wireshark
Wireshark is a tool that not only allows for capturing packets, but can also *analyze* and decode the content of captured packets.
To do this, all the data captured from a network interface is "dissected" in all its fundamentals parts, which are then interpreted and visualized in the context of the recognized protocol.

To capture packets, the network interface must be in **promiscuous mode**, and this is because otherwise all packets that were not intended for our host would not be seen.
In case of *wireless* networks, the interface must be in **monitor mode**, but in this case we are also able to capture packet that are destined to different APs.

The operation of capturing packets is usually done for troubleshooting reasons or to analyze the status of the connection, but it can also be used to monitor a conversation between two devices.

The logic behind the *analysis* made by this tool is that as packets in the network are processed by different layers, we can use different "dissectors" specialized in processing all the information regarding a certain layer.
Given this, when a packet is captured, it is analyzed with a bottom-up approach by every dissector implemented.

In particular, the *protocol* of a packet can be detected in two ways:
- *directly*, if the frame has a field that states which protocol it is encapsulating
- *indirectly,* using heuristics based on combinations of port/protocol

Wireshark filters
When capturing packets, especially on a network that is heavily used, we need to focus only on the packets that we are interested in: to do this, we can specify **filters** that perform exactly this operation.

The **capture filters** are used to limit the traffic that is being captured, which means we are specializing which packet to keep and which can be forwarded or processed without additional overhead.
The main syntax of this filter is based on the *Berkeley Packet Filter Syntax*, where we can specify the following options:
- *protocol*
    - ether, tcp, udp, ip, ip6, arp
- *direction*
    - src, dst
- *type*
    - host, port, net, portrange
- *operators*
    - less, grater, gateway, broadcast, and (&&), or (||), not(!)

The **display filters** are used to display only a portion of all the packets that have been captured, which means that in this case no packet have been lost.

Packet capturing techniques

Assuming we are in a *wired network*, using promiscuous mode has its limitation in the sense that we are only able to see packets that are passing through our **segment** of the network.

To solve this problem, we can use other silent techniques like:

- *physical tap:* we can physically interconnect the cables in such a way that all traffic will be first redirected through our segment of the network, then the traffic goes along its original path
- *port mirroring:* by using a "switched port analyzer" on a switch, we are able to replicate on our host all the destinations ports that are normally used for traffic, therefore every packet will be also forwarded to our segment of the network

There also are more "aggressive" approaches, like:

- *ARP cache poisoning:* unsolicited ARP replies to steal IP addresses
- *MAC flooding:* fill the CAM of the switch to make it acting as a hub
- *DHCP redirection:* using a "rouge" DHCP server that exhausted all its pool of IP addresses, it pretends to be the default gateway of the network via its new DHCP request

How to prevent packet capture

- **Dynamic address inspection**
  This operation can be performed manually by inspecting the binding between IP-MAC, dropping all invalid packets, or performed autonomously by switches using the *Dynamic Address Resolution Inspection* protocol
- **DHCP snooping**
  This operation can be performed manually by inspecting traffic generated by a specific port, which can then be disabled, or performed autonomously by switches by implementing a list of trusted ports and by inspecting the binding between IP-MAC

IPv6

The **IPv6** protocol is not a "new" protocol, since it was developed mid to late 1990s, but it was created using the 10+ years of experience given by IPv4 with the objective of solving the most critical problem of internet: the exhaustion of IP addresses that began in 2011 and reached critical levels in 2019.
While IPv4 has a 32-bit address space for a memory address, written in binary, IPv6 uses a 128-bit address space written in hexadecimal, obtaining a number of total possible addresses that is around 340 *undecillions*!

Obs. In between IPv4 and IPv6 there also was an experimental protocol developed to provide *quality of service* for real-time multimedia applications, such as video and voice, and it was called **Internet Stream Protocol** *(ST)* and informally also known as IPv5

Benefits of IPv6

This new protocol does not only introduce a much larger address space, but it also has some other benefits.
One of them is being able to ensure end-to-end reachability without the need of private addresses and NAT, although some people think that having a public and globally reachable address is less secure.
It also allows for *stateless configuration*, better *QoS* and mobility support.

IPv6 Address Notation

An IPv6 address is expressed in *hexadecimal* values, divided in groups of 16 bits separated by colons (:).
There also are some rules that can be applied to the address in specific situation, allowing for a much easier readability:

1. **Omitting leading 0s**
   Given a single group of 16 bits, all leading zeroes can be omitted from the representation
   Ex. 2001:0DB8:0001:1000:00A2:ABCD:0EF0:0A05 -> 2001:DB8:1:1000:A2:ABCD:EF0:A05

2. **Double Colon**
   To simplify the address even further, any single and contiguous string of one (or more) 16-bit groups consisting of only zeroes can be represented with a double colon (::)
   This rule can only be applied once, with the following restrictions:
   - If there are multiple possible reductions with different lengths, then only the *longest one* should be abbreviated
     Ex. 2001:0DB8:1000:0000:0000:AD4F:0000:0120 -> 2001:DB8:1000::AD4F:0:120
   - If there are multiple possible reductions with same lengths, then only the *first appearance* should be abbreviated
     Ex. 2001:0DB8:1000:0000:0000:AD4F:0000:0000 -> 2001:DB8:1000::AD4F:0:0

IPv6 address types

The IPv6 addresses can be divided in different categories, usually depending on the bits that form the first group of 16, but a very distinctive feature of IPv6 is that it does not define any *broadcast* addresses, but instead uses *multicast* addresses to perform the same functionality.

IPv6 Unicast Addresses

The **unicast** addresses are always related to an end-to-end communication, meaning they refer to a single receiving host.
A packet generated in this situation can have the following addresses:

- **source address:** global unicast address *(GUA)* or link-local
- **destination address:** any unicast, multicast or anycast address

Global Unicast Address
The **global unicast address** *(GUA)* is a *globally unique* and *routable* address in the network, and its similar to public IPv4 addresses.
The prefix for GUAs starts from 2000::/3, which means that the range varies from 2000::/3 to 3FFF:FFFF:FFFF:[…]:FFFF.

A *GUA* address structure can be divided in several parts:
- *001:* they are the first 3 bits of every GUA address
- *global routing prefix:* it represents the prefix or the network portion of the address assigned by the provider, such as an ISP, and its length is equivalent to the subnet mask for an IPv4 address
- *subnet ID:* field handled by the client for subnets allocation
- *interface ID:* it identifies the interface on the subnet, and it is equivalent to the host part for an IPv4 address

In most cases, GUA addresses has a netmask of /64 bits, where the other 64 are left as the host parts of the network, and the *subnet ID* has a number of bits that is between the end of the *global routing prefix* and the start of the *interface ID*.
To be more specific, we can divide the 128 bits in the following manner:
- N bits are used for the *global routing prefix*, where 3 of them are reserved by the first 3 bits 001
- M bits are used for the *subnet ID*
- the remaining 128-N-M bits left are assigned to the *interface ID*

The usual division is N=48, M=16 and 64 bits left to be assigned to the hosts in the network.

Except under specific circumstances, all end users have a GUA, and every interface of a host can have multiple IPv6 addresses assigned on the same or different networks.

Link-Local Unicast
The **link-local unicast address** is an address that is *unique* for the local network, and every IPv6 device on a local network must have (at least) one of this type of address.
A packet generated in this situation can have the following addresses:
- **source address:** always a link-local unicast address
- **destination address:** link-local unicast, multicast or anycast address

The prefix for link-local addresses starts from FE80::/10, which means that the range varies from FE80::/10 to FEBF:FFFF:FFFF:[…]:FFFF.

When creating a link-local address, it can happen in two ways:
- *automatically*
    - concatenation of FE80 and the *interface ID* part of the address, which can be configured either using the EUI-64 protocol or by choosing 64 random bits
- manually

The EUI-64 protocol can be used to compute the *interface ID* part of a link-local address in the following way:
1. it starts from the MAC address of the interface, which can be divided in two parts
    a. 24 bits given by the vendor of the interface
    b. 24 bits that uniquely identify the device
2. between the two parts of the MAC address, which means right in the middle, the bits FFFE are added
3. lastly, the second to last bit of the first octet is flipped

The problem with EUI-64 is that, since it is based on the MAC address of the interface, the *interface ID* part of the address will be the same on any network joined.
Using a randomly generated number increases the privacy.

Obs. These addresses are used to communicate with other devices in the local network and *should NOT* be used outside of it, meaning they are not routable from other networks.

Stateless Address Autoconfiguration
The **SLAAC** allows the router to send to each host only the *prefix* of the network, then each host chooses autonomously the IP address that is going to be assigned to each interface.

To do this operation, we use the **ICMPv6 Neighbor Discovery** protocol, that defines 5 different packet types:

- *Router Solicitation/Advertisement Message:* used to perform dynamic address allocation
- *Neighbor Solicitation/Advertisement Message:* used to perform address resolution, similar to ARP in IPv4
- *Redirect message:* similar to ICMP in IPv4, used to perform router-to-device communication

Address Resolution

- **IPV4**
  1) The sender broadcasts an `ARP_Request` for the destination's IP, asking for its MAC address
  2) The receiver, upon receiving a message with destination IP its own address, will respond to the sender with an `ARP_Reply` containing its MAC address
  3) Lastly, the sender stores the info received inside its *ARP cache*
- **IPv6**
  1) The sender sends a *multicast* `Neighbor_Solicitation` for the destination's IP, asking for its MAC address
  2) The receiver, upon receiving a message with destination IP its own address, will respond to the sender with a `Neighbor_Advertisement` containing its MAC address
  3) Lastly, the sender stores the info received inside its *Neighbor cache*

The protocols are pretty similar, with slight differences in the packets sent in the network, but IPv6 is more efficient because the packets are sent over *ethernet* which helps in reducing the traffic in the network.

Dynamic Address Allocation
The **dynamic address allocation** refers to the *DHCP* protocol of IPv4, but in IPv6 we have different options while implementing this mechanism.

All implementations start with a host that sends a multicast `Router_Solicitation` message to the default gateway of the network, but how it will respond to this request depends:

- **SLAAC without DHCPv6 (stateless)**
  In this implementation, the router has all the information needed for a new host to correctly join the network, which means:
  - network prefix
  - prefix length
  - default gateway
  - domain and DNS server list (optional)

  The router will then send a *multicast* `Router_Advertisement` to all IPv6 devices in the net, who will process the information receive and create a new address applying EUI-64 or randomly choosing an *interface ID*.
  Since the protocol is **stateless**, which means there is no mechanism that is maintaining a mapping of all active addresses, the host that just acquired a new address has to make sure that it is not a duplicate: the *Duplicate Address Detection* mechanism solves this problem.
  Essentially, the host computes a new IPv6 devices and then sends a *multicast* `Neighbor_Solicitation` with destination its own IP address, so that:
  - if it receives a response, it means that there is already another host that is using that IP address
  - otherwise, if no response is received, the address in unique

- **SLAAC + Stateless DHCPv6 (stateless)**
  This implementation is similar to the previous one, except that there are DHCPv6 servers inside the network that provide other useful information not managed by the router.
  This time, when the router responds with its *multicast* `Router_Advertisement` to the host, the message will also contain two special flags:
    - *O Flag:* if the "option" flag is set to 1, it means that other informations (such as DNS addresses) are handled by the DHCPv6 servers
    - *M Flag:* if the "managed" flag is set to 1, it means that the network is entirely managed by the DHCPv6 servers

  This implementation sets the flags O=1 and M=0, which means the host will then:
    1) send a *multicast* DHCPv6`_Solicit` to all DHCPv6 servers
    2) receive a *unicast* DHCPv6`_Advertise` from one of the servers
    3) send a *multicast* DHCPv6`_Information_Request` to all DHCPv6 servers
    4) receive a *unicast* DHCPv6`_Reply` from one of the servers

  Again, after receiving all the information needed to generate a valid IPv6 address, the host has to choose how to create a new *interface ID*, but in this case it will use the random method.

- **DHCPv6 Addressing (stateful)**
  This implementation relies entirely on the DHCPv6 servers inside the network, while the router only provides its own *link-local* address to be used as the default gateway.

  Here the flags are O=0 and M=1, which means the host will then:
    5) send a *multicast* DHCPv6`_Solicit` to all DHCPv6 servers
    6) receive a *unicast* DHCPv6`_Advertise` from one of the servers
    7) send a *multicast* DHCPv6`_Information_Request` to all DHCPv6 servers
    8) receive a *unicast* DHCPv6`_Reply` from one of the servers

DHCPv6 Prefix Delegation Process

When we receive the *prefix* of our network from the ISP, assuming we're using IPv4, the information is sent to the public IPv4 address of our border gateway, which will then use it to create the internal network.
This implies the usage of **NAT**, which translate the private internal IP addresses so that they are not routable from the outside.
In IPv6, on the other hand, there is complete reachability without the need of NAT.

First, the border gateway of our network receives an IPv6 prefix address.
Then, similarly to any other IPv6 client, it dynamically gets a new address using one of the 3 implementations mentioned before.
Lastly, the router asks for a separate IPv6 prefix that will be used for the internal network.