Sockets : what they are and how they work

A **socket** is a ***software object*** that **allows the sending and receiving of data, between remote hosts** (over a network) **or between local processes** (Inter-Process Communication).
More precisely, the concept of socket is based on the Unix file Input/Output model, therefore on the ***open*** , ***read*** , ***write*** and ***close operations*** ; in fact, it is used in the same way, adding parameters useful for communication, such as **addresses** , **port numbers** and **protocols** .
Communicating local and remote **sockets form** a **pair** , consisting of the client and server **address** and **port** ; there is a logical connection between them.
Operating systems usually provide APIs to allow applications to control and use network sockets .
We can see sockets as **intermediaries** between the **application** and **transport layers** in the TCP/IP stack . In fact, the function of sockets is to **address processes** .

Format of a Socket Address

Since there can be many processes on the interlocutor systems, it is necessary to have a way to precisely **address** the process with which one is communicating. For this purpose,
ports are used : numbers that identify the processes in execution.
The interlocutors then store **the address and port** of the other party in a **socket address** , formed as follows:

- IP address: 32 bit;
- Port number: 16 bit.

Port numbers have been defined by the IANA , and are divided into:

- Well-known (reserved for specific protocols): 20, 23, 25, 80, 110,… ;
- Do not use: 0;
- Reserved for well-known processes : 1-255;
- Reserved for other processes: 256-1023;
- Other applications: 1024-65535;

Normally the port numbers assigned to processes (not well-known) are at the discretion of the operating system. The operating system is said to assign *ephemeral* ports .

Socket families

The types of **protocols** used by **socket** define its **family** (or domain). We can distinguish, for example, two important families:

- ***AF_INET*** : communication between remote hosts , via the Internet;

- ***AF_UNIX*** : local inter-process communication, on Unix machines. This family is also called *Unix Domain Socket* .

Socket Types

Within the family we can distinguish the **socket type** , depending on the connection mode. We have:

- **Stream sockets** : connection-oriented , based on reliable protocols such as TCP or SCTP;
- **Datagram socket** : non-connection-oriented (connectionless), based on the fast but unreliable UDP protocol;
- **Raw socket** (raw IP): the transport layer is bypassed, and the header is accessible to the application layer.

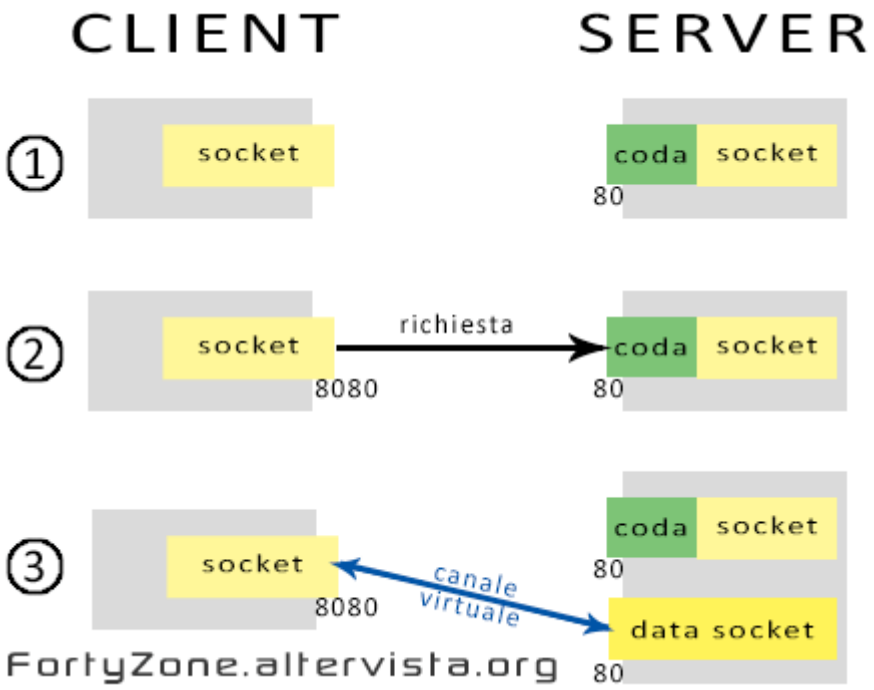
Stream Socket

Being based on transport layer protocols such as **TCP , they ensure reliable** , **full-duplex** , **connection-oriented** communication with a variable-length ***byte stream*** .

Communication through this socket consists of these phases:

#1 – Creating Sockets

Client and server create their respective **sockets** , and the **server listens** on a **port** .



Since the server can create multiple connections with different clients (but also with the same one), it needs a **queue** to handle the various requests.

#2 – Connection request

The **client** makes a **connection request** to the server.
Note that we can have two different port numbers, because one could be dedicated only to outgoing traffic, the other only to incoming traffic; this depends on the host configuration . In essence, it is not a given that the local port of the client coincides with the remote port of the server
The **server** receives the request and, if it is accepted, a **new connection is created** .

#3 – Communication

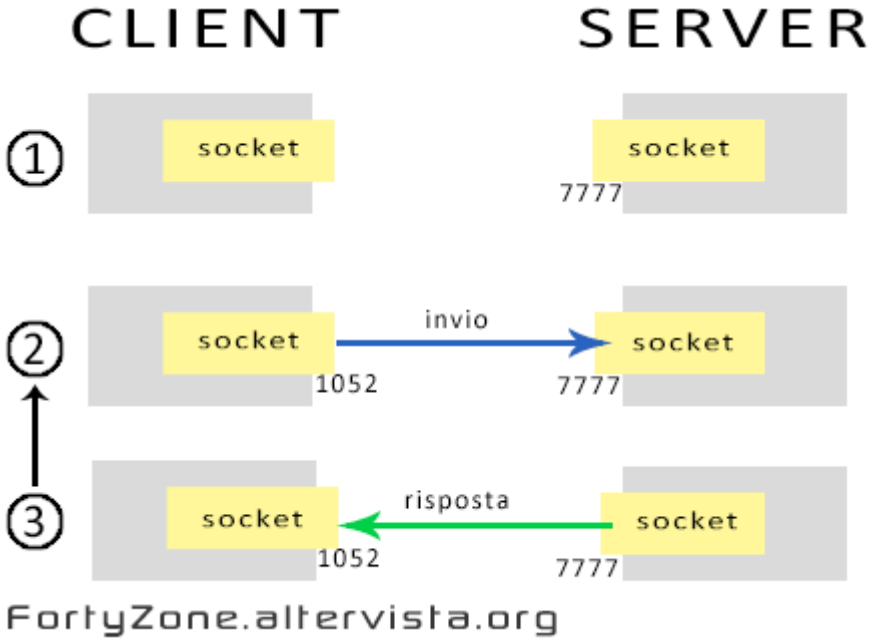
Now the client and server communicate through a **virtual channel** , between the socket of the former and a new one of the server, created specifically for the data flow of this connection: **data socket** .
Consistent with what was mentioned in the first phase, the server creates the data socket because the first one is used exclusively for request management. It is possible, therefore, that there are many clients communicating with the server, each towards the **data socket** created by the server for them.

#4 – Closing the connection

Since TCP is a connection-oriented protocol, when there is no longer a need to communicate, the client communicates this to the server, which **de-instantiates the data socket** . The connection is then closed.

Datagram Socket

They are based on **UDP** , a transport layer protocol that guarantees low latency communications (ideal for video chats, for example), at the expense of data reliability. In fact, there is no flow control (datagram ordering and error control).



Since UDP is not a connection-oriented protocol, there is no connection phase seen above, but the client communicates directly with the server, whenever it wants.

#1 – Creating Sockets

As in the previous type, client and server create their respective **sockets** , and the **server listens on** a port **.** The server socket , this time, does not need a queue, since incoming and outgoing data does not need to be confined within a connection, so **communication with different clients takes place on the same interface** .

#2 – Sending data

The **client** sends datagrams directly to the server. The same applies here as previously stated about port numbers.

#3 – Server Response

The **server** sends the client a possible **response** . The communication is therefore made up of a **loop that lasts as long as there is data to send** and, obviously, as long as the hosts are reachable.