OPERATING SYSTEMS: SESSION3

JANUARY 2023

TecnoCampus Escuela Superior Politécnica

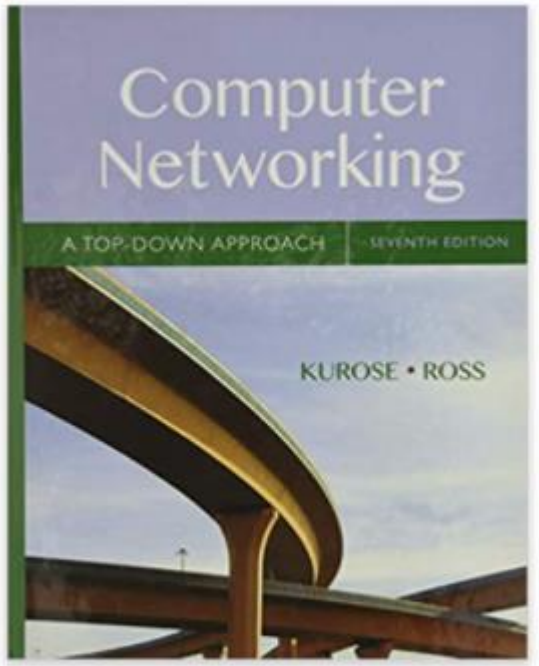
Centro adscrito a la upf. Universitat Pompeu Fabra Barcelona

TecnoCampus 10 años

SOCKETS

- How can we share information between different computers?

- First we need to **identify** A COMPUTER in a NETWORK



ISBN-13: 978-0133594140
ISBN-10: 9780133594140

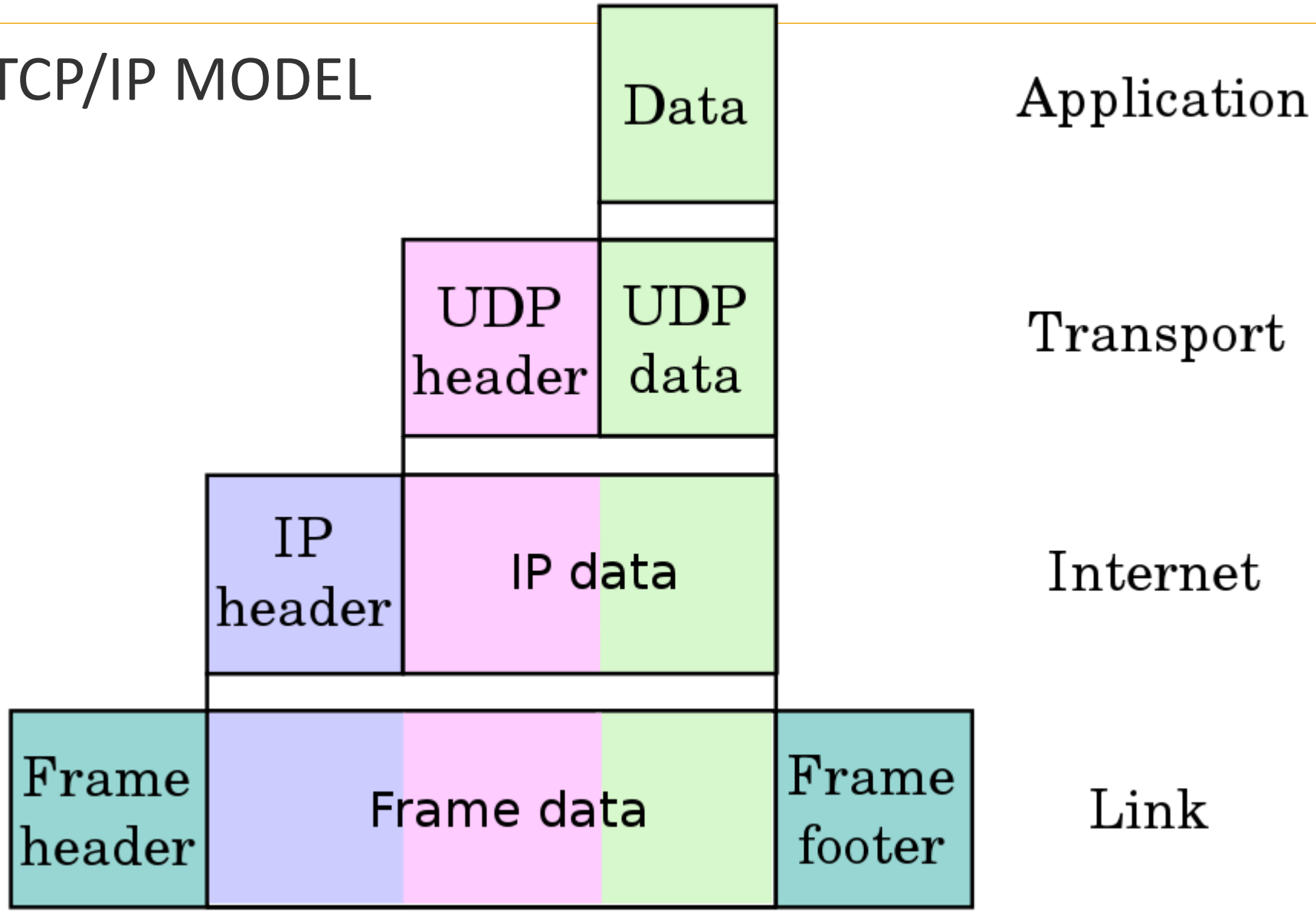
TCP/IP STACK

- Networking communication is split in layers (OSI MODEL)

OSI model

Layer			Protocol data unit (PDU)	Function[19]
Host layers	7	Application	Data	High-level APIs, including resource sharing, remote file access
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1	Physical	Bit, Symbol	Transmission and reception of raw bit streams over a physical medium

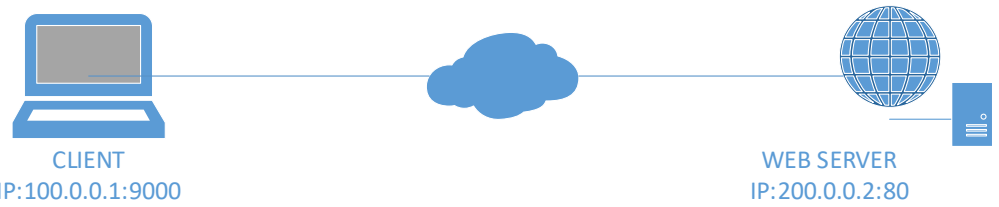
- TCP/IP MODEL


Data
UDP header
UDP data
IP header
IP data
Frame header
Frame data
Frame footer
Application
Transport
Internet
Link

TCP/IP STACK

- Using **TCP/IP** Model if a device must be connected to the INTERNET, that device NEEDS:
  - **IP ADDRESS**, for the layer INTERNET (3) connectivity using IP protocol
  - **PORT**, for the layer TRANSPORT (4) connectivity using a convenient TRANSPORT PROTOCOL
- If you want to call a friend at a company, then you need the phone number of the branch of the company, and once you are connected at the reception of the company, you must provide the EXTENSION number of your friend to be redirected correctly: the first number is the IP ADDRESS and the second is the PORT

- In the example:
  - SERVER will be available for HTTP CONNECTIONS AT THE IP 200.0.0.2 ON PORT 80
  - CLIENT will be asking for the HTTP CONNECTION FROM THE IP 100.0.0.1 ON PORT 9000



COMMUNICATIONS MODELS

- There are two ways of communicating devices:

1.- CONNECTION ORIENTED SOLUTION

2.- NON-CONNECTION ORIENTED SOLUTION

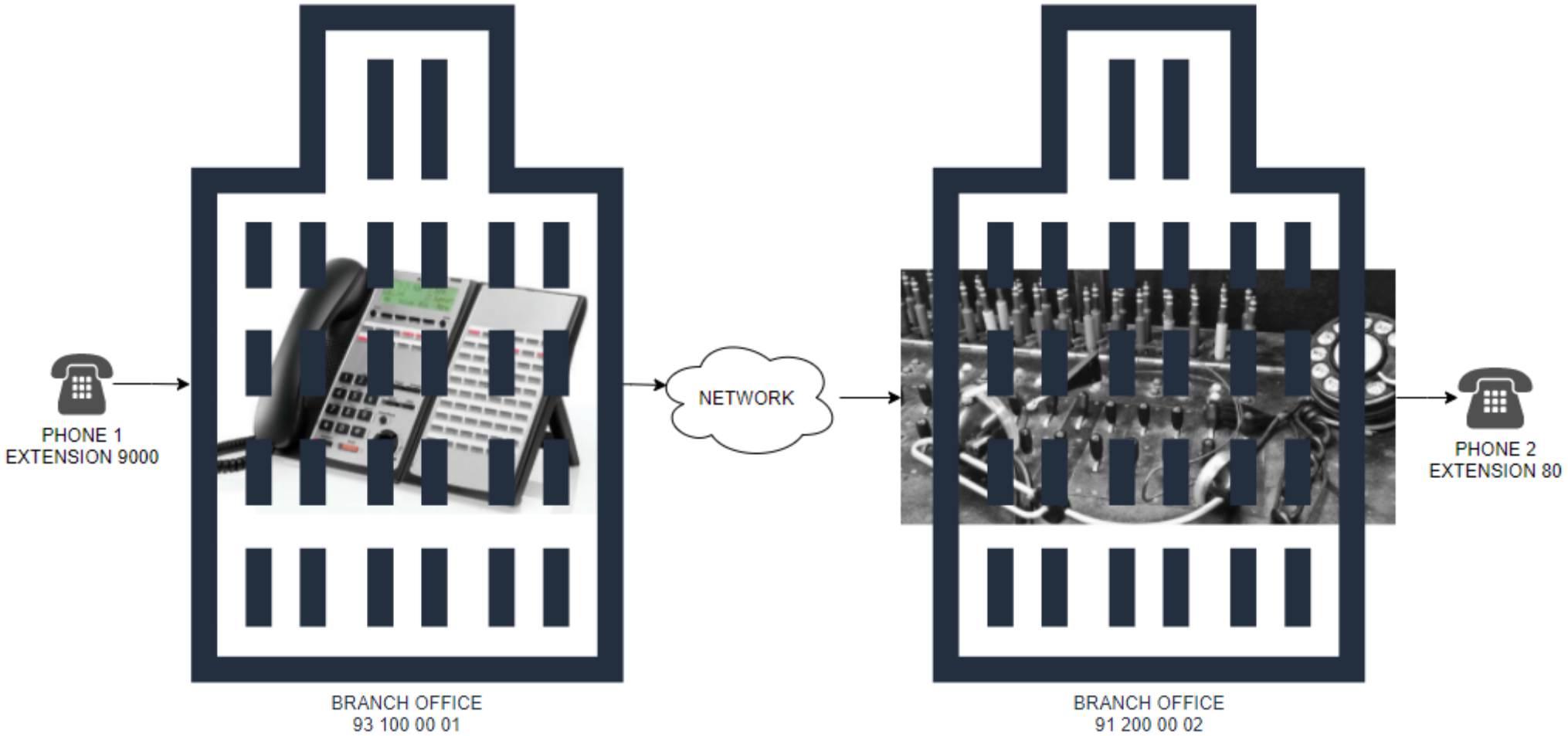
CONNECTION ORIENTED SOLUTION

- The procedure is similar to a PHONE CALL:

- 1: CONNECTION REQUEST---CONNECTION ESTABLISHMENT
- 2: COMMUNICATION
- 3: CONNECTION CLOSE

- Before having the possibility to talk to someone … you MUST ESTABLISH a CONNECTION between THE PEERS
- Once the connection is established, INFORMATION can be sent in between both sides
- Once the communication is done (all the information has been shared) PEERS MUST CLOSE THE CONNECTION

CONNECTION ORIENTED SOLUTION



CONNECTION ORIENTED SOLUTION

- In this solution. Once the connection is established PEERS know perfectly they are talking with the right PEER (the one that ACCEPTED the connection request/the one that HAS REQUESTED the connection establishment)
- You know all the “words” you send from one side will get the remote side

- The connection can be CLOSED by both sides OR JUST ONLY BY ONE OF THE SIDES: if it is closed in one side, the opposite site is not anymore able to talk… THERE IS NO ONE AT THE OTHER SIDE

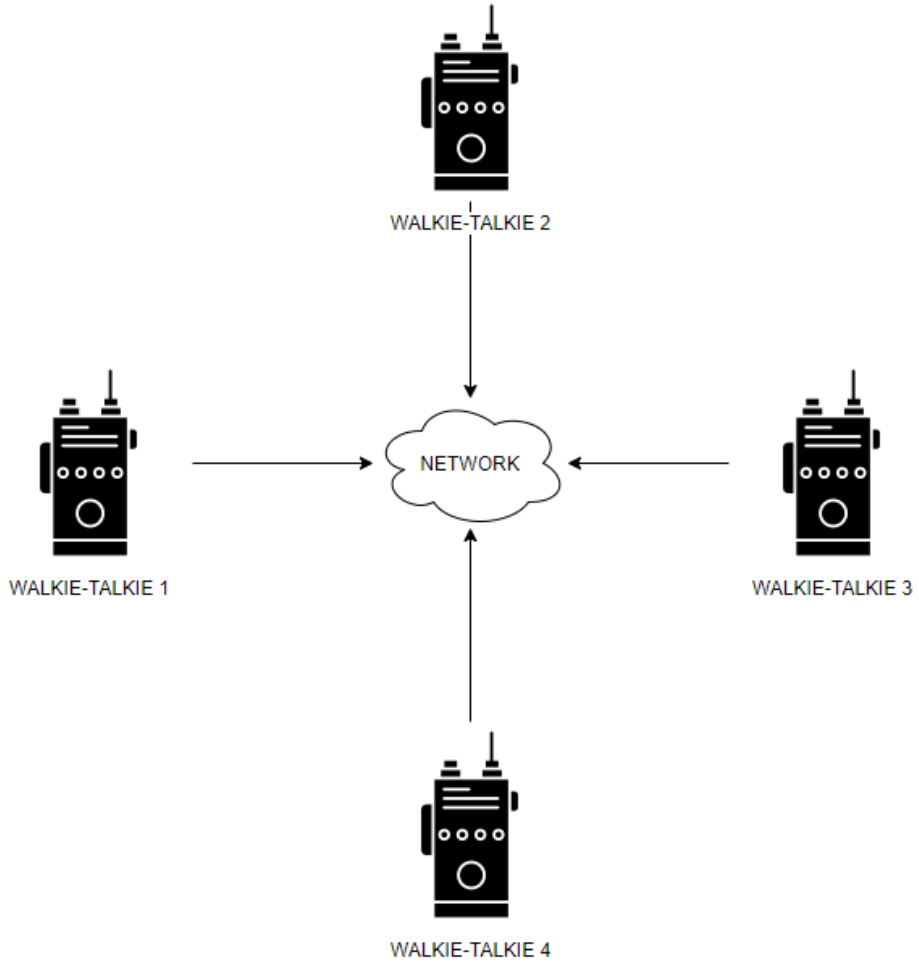
NON-CONNECTION ORIENTED SOLUTION

- The procedure is similar to a **PUSH-TO-TALK** SOLUTION:

- 1: we never have a CONNECTION in between PEERS
- 2: The TRANSMITTER will SEND INFORMATION to the RECEIVER, with the IP:PORT of the SERVER

- For every MESSAGE: you need to SEND THE SENDER INFORMATION (who is talking) AND THE RECEIVER INFORMATION (who you want to talk with)

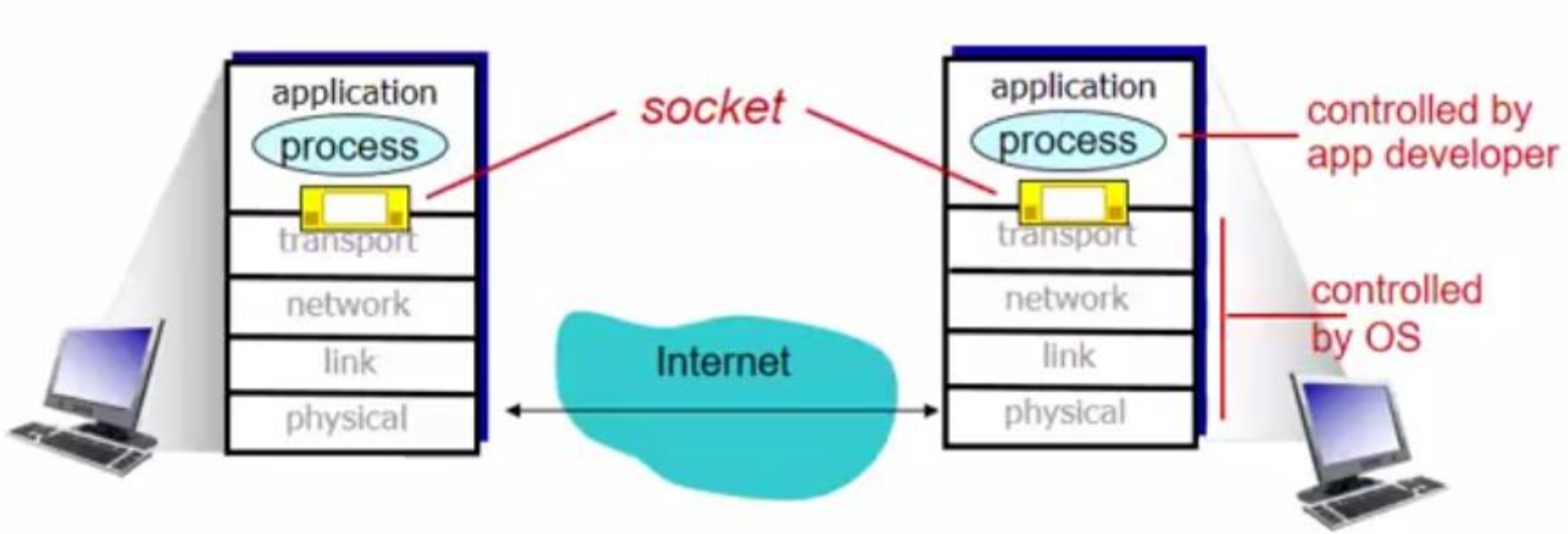
NON-CONNECTION ORIENTED SOLUTION



- For the OPERATING SYSTEM … the mechanism to have the ability to communicate with a peer on INTERNET … you must define a SOCKET:
  - It is a kind of PSEUDO FILE that represents a NETWORK "CONNECTION" (LINK)
  - As a LINK IT IS A **FULL-DUPLEX LINK**:
    - You can use the same SOCKET as a BIDIRECTIONAL LINK

SOCKETS

- The SOCKET is managed by the OPERATING SYSTEM
- The OS provides to the developer **SYSTEM CALLS** (API) to use the SOCKET
- The developer will program the CLIENT-SERVER APPLICATION



- If you take a look at BEEJ NETWORK PROGRAMMING GUIDE (https://beej.us/guide/bgnet/): EVERYTHING IN UNIX IS A FILE!

> Ok—you may have heard some Unix hacker state, "Jeez, *everything* in Unix is a file!" What that person may have been talking about is the fact that when Unix programs do any sort of I/O, they do it by reading or writing to a file descriptor. A file descriptor is simply an integer associated with an open file. But (and here's the catch), that file can be a network connection, a FIFO, a pipe, a terminal, a real on-the-disk file, or just about anything else. Everything in Unix *is* a file! So when you want to communicate with another program over the Internet you're gonna do it through a file descriptor, you'd better believe it.

- So to work with the SOCKET as with a FILE you need a **SOCKET DESCRIPTOR**:

- **SOCKET DESCRIPTOR**:
  - A kind of POINTER (index in a SOCKET TABLE) to point where you have all the relevant information for the socket:
    - INFORMATION IN QUEUE
    - MEMORY ADDRESS WHERE THE INFORMATION IS STORED
    - PERMISSIONS
    - …

SOCKET

- A SOCKET is identified (uniquely identified) if you have the **5-TUPLE PARAMETERS**:
- **PROTOCOL**: connection oriented, non-connection oriented
- **PEER 1**: IP1:PORT1
- **PEER 2**: IP2:PORT2

SOCKET: PROTOCOLS

- For INTERNET SOCKETS you can work with 2 PROTOCOLS:

- **1- TCP: TRANSMISSION CONTROL PROTOCOL**
  - CONNECTION ORIENTED SOLUTION
  - ERROR-FREE TRANSPORT PROTOCOL (RETRANSMISSION)

- **2- UDP: USER DATAGRAM PROTOCOL**
  - NON-CONNECTION ORIENTED SOLUTION
  - GUARANTEES-FREE TRANSPORT PROTOCOL

- On the SOCKET COMMUNICATION you have 2 PEERS

- 1 – The **SERVER**: The device that will accept the connection request, or the device that will receive the first message in a non-connection oriented solution(THE **CALLEE**)

- 2 – The **CLIENT**: The device that will request the connection to the SERVER or the device that will send the first message (THE **CALLER**)

SOCKET: SOCKET 5-TUPLE PARAMETERS CONFIGURATION

- You have to DEFINE the SOCKET on **BOTH SIDES**, in PARALLEL

- When REQUEST/ACCEPT is done or the MESSAGE is sent THE SOCKET WILL BE PROPERLY CONFIGURED IN BOTH SIDES

SOCKET: SOCKET 5-TUPLE PARAMETERS CONFIGURATION (TCP)

STEP (SERVER)	PROTOCOL	LOCAL-IP	LOCAL-PORT	REMOTE-IP	REMOTE-PORT
STEP1: PROTOCOL DEFINITION	TCP				
STEP 2: SOCKET BINDING: LOCAL PARAMETERS CONFIGURATION		SERVER-IP	SERVER-PORT		
STEP 2B: CONNECTION ACCEPT (READY TO ACCEPT CONNECTIONS)	TCP	SERVER-IP	SERVER-PORT	?	?
STEP7: CONENCTION REQUEST ACCEPTED	TCP	SERVER-IP	SERVER-PORT	CLIENT-IP	CLIENT-PORT

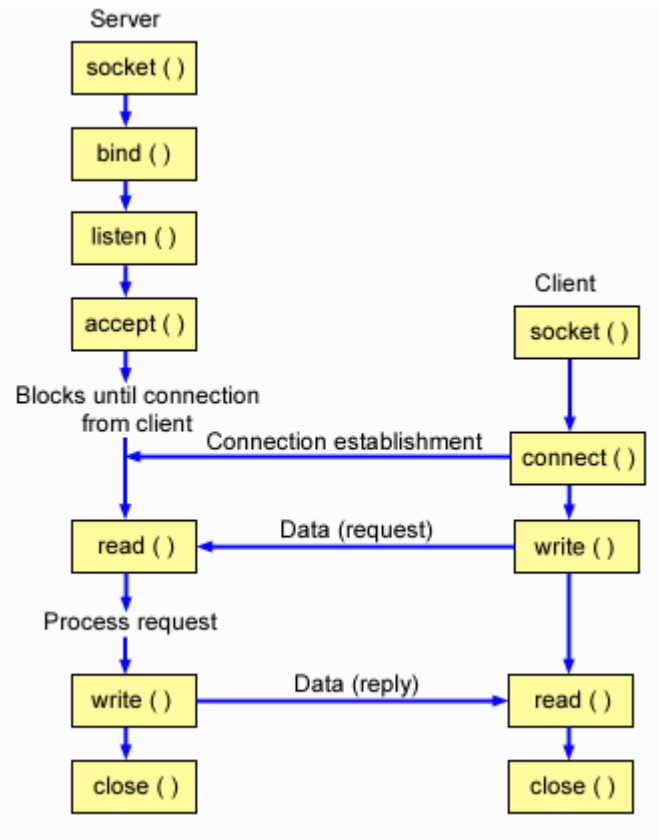
STEP (CLIENT)	PROTOCOL	LOCAL-IP	LOCAL-PORT	REMOTE-IP	REMOTE-PORT
STEP3: PROTOCOL DEFINITION	TCP				
STEP4: LOCAL PARAMETERS CONFIGURATION		CLIENT-IP	CLIENT-PORT		
STEP5: REMOTE PARAMETERS CONFIGURATION				SERVER-IP	SERVER-PORT
STEP6: CONNECTION REQUEST (ACCEPTED)	TCP	CLIENT-IP	CLIENT-PORT	SERVER-IP	SERVER-PORT

SOCKET: SOCKET 5-TUPLE PARAMETERS CONFIGURATION (UDP)

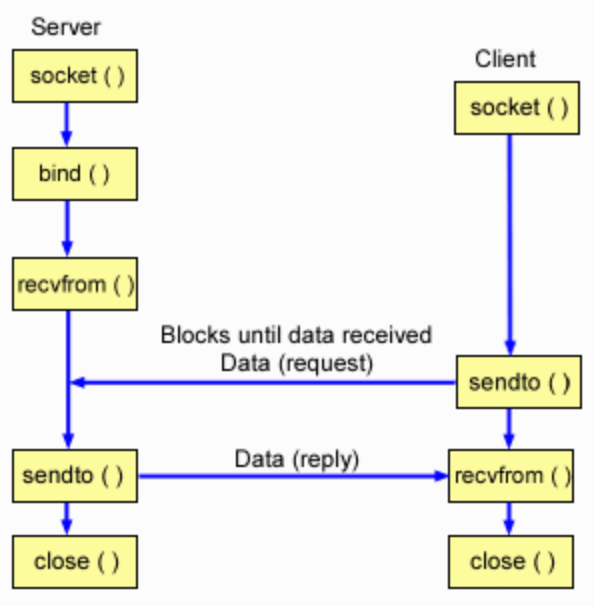
STEP (SERVER)	PROTOCOL	LOCAL-IP	LOCAL-PORT	REMOTE-IP	REMOTE-PORT
STEP1: PROTOCOL DEFINITION	UDP				
STEP 2: SOCKET BINDING: LOCAL PARAMETERS CONFIGURATION		SERVER-IP	SERVER-PORT		
STEP 2B: WAITING FOR A MEESAGE FROM PEER (READY TO READ MESSAGES)	UDP	SERVER-IP	SERVER-PORT	?	?
STEP7: MESSAGE RECEIVED FROM PEER	UDP	SERVER-IP	SERVER-PORT	CLIENT-IP	CLIENT-PORT

STEP (CLIENT)	PROTOCOL	LOCAL-IP	LOCAL-PORT	REMOTE-IP	REMOTE-PORT
STEP3: PROTOCOL DEFINITION	UDP				
STEP4: LOCAL PARAMETERS CONFIGURATION		CLIENT-IP	CLIENT-PORT		
STEP5: REMOTE PARAMETERS CONFIGURATION				SERVER-IP	SERVER-PORT
STEP6: SEND MESSAGE TO PEER	UDP	CLIENT-IP	CLIENT-PORT	SERVER-IP	SERVER-PORT

CONNECTION ORIENTED COMMUNICATION



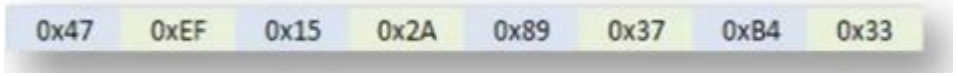
NON-CONNECTION ORIENTED COMMUNICATION



- **ENDIANNESS**:
  - The way information is written could be:
  - LITTLE ENDIAN/BIG ENDIAN
  - LITTLE ENDIAN:
    - DEC
    - INTEL
  - BIG ENDIAN:
    - MOTOROLA
    - TCP/IP PROTOCOL

1 BYTE DATA SEQUENCE:
0x47-0xEF-0X15-0X2A-0X89-0X37-0XB4-0X33

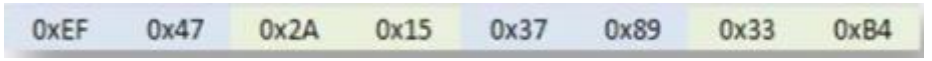
BIG ENDIAN



LITTLE ENDIAN

2 BYTES DATA SEQUENCE :
0x47EF-0X152A-0X8937-0XB433

BIG ENDIAN



LITTLE ENDIAN

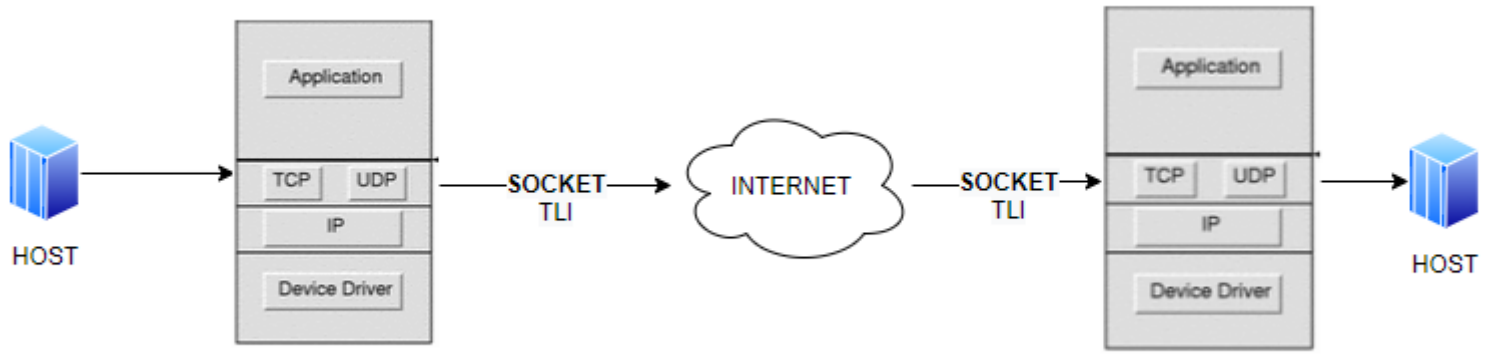
4 BYTES DATA SEQUENCE :
0x47EF152A-0X8937B433

BIG ENDIAN

LITTLE ENDIAN

ENDIANNESS ON SOCKETS

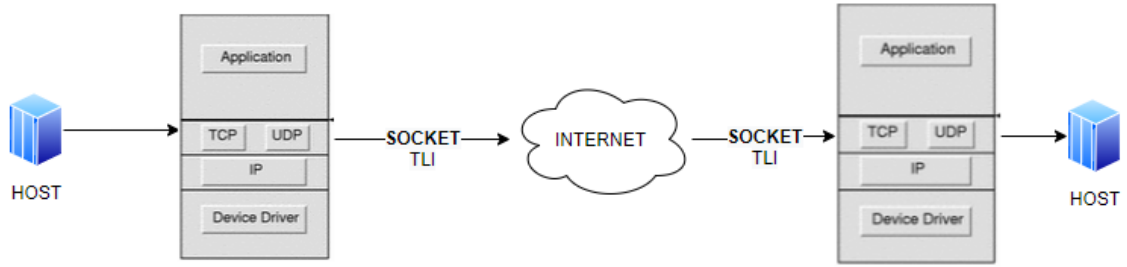
- When working with SOCKETS you must be CAREFUL with ENDIANNESS



HOST1	TLI 1	TLI 2	HOST2
LE	BE	BE	LE
BE	BE	BE	BE
LE	BE	BE	BE
BE	BE	BE	LE

ENDIANNESS ON SOCKETS

- When working with SOCKETS you must be CAREFUL with ENDIANNESS: SHORT—PORT, LONG—IP ADDRESS
- HTON→**htons**(), **htonl**()
- NTOH→**ntohs**(), **ntohl**()



HOST1	CONVERSION	TLI 1	TLI 2	CONVERSION	HOST2
LE	HTON()	BE	BE	NTOH()	LE
BE	HTON()	BE	BE	NTOH()	BE
LE	HTON()	BE	BE	NTOH()	BE
BE	HTON()	BE	BE	NTOH()	LE

C FUNCTIONS FOR SOCKETS: SOCKET: to have a SOCKET-FILE DESCRIPTOR

```
/* Try to create TCP socket */          int sock
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sock < 0) {
    err_sys("Error socket");
}
```

```
SOCKET(2)

NAME
       socket - create an endpoint for communication

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int socket(int domain, int type, int protocol);

DESCRIPTION
       socket()  creates  an  endpoint  for communication and returns a file descriptor that
       refers to that endpoint.  The file descriptor returned by a successful call  will  be
       the lowest-numbered file descriptor not currently open for the process.
```

C FUNCTIONS FOR SOCKETS:

```
/* Try to create TCP socket */
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sock < 0) {
    err_sys("Error socket");
}
```

INTERNET

TCP

```
void err_sys(char *mess) { perror(mess); exit(1); }
```

FAMILY: always **PF_INET** for INTERNET SOCKETS 'Protocol_Family INET (InterNET)'

TYPE: always **SOCK_STREAM** for CONNECTION-ORIENTED SOCKETS

PROTOCOL: always **IPPROTO_TCP** for CONNECTION-ORIENTED SOCKETS

```
SOCK_STREAM     Provides sequenced, reliable, two-way, connection-based byte streams.
                An out-of-band data transmission mechanism may be supported.
```

C FUNCTIONS FOR SOCKETS:

Name	Purpose	Man page
AF_UNIX	Local communication	unix(7)
AF_LOCAL	Synonym for AF_UNIX	
AF_INET	IPv4 Internet protocols	ip(7)
AF_AX25	Amateur radio AX.25 protocol	ax25(4)
AF_IPX	IPX - Novell protocols	
AF_APPLETALK	AppleTalk	ddp(7)
AF_X25	ITU-T X.25 / ISO-8208 protocol	x25(7)
AF_INET6	IPv6 Internet protocols	ipv6(7)
AF_DECnet	DECet protocol sockets	
AF_KEY	Key management protocol, originally developed for usage with IPsec	
AF_NETLINK	Kernel user interface device	netlink(7)
AF_PACKET	Low-level packet interface	packet(7)
AF_RDS	Reliable Datagram Sockets (RDS) protocol	rds(7), rds-rdma(7)
AF_PPPOX	Generic PPP transport layer, for setting up L2 tunnels (L2TP and PPPoE)	
AF_LLC	Logical link control (IEEE 802.2 LLC) protocol	
AF_IB	InfiniBand native addressing	
AF_MPLS	Multiprotocol Label Switching	
AF_CAN	Controller Area Network automotive bus protocol	
AF_TIPC	TIPC, "cluster domain sockets" protocol	
AF_BLUETOOTH	Bluetooth low-level socket protocol	
AF_ALG	Interface to kernel crypto API	
AF_VSOCK	VSOCK (originally "VMWare VSockets") protocol for hypervisor-guest communication	vsock(7)
AF_KCM	KCM (kernel connection multiplexor) interface	
AF_XDP	XDP (express data path) interface	

TCP

UDP

SOCK_STREAM	Provides sequenced, reliable, two-way, connection-based byte streams. An out-of-band data transmission mechanism may be supported.
SOCK_DGRAM	Supports datagrams (connectionless, unreliable messages of a fixed maximum length).
SOCK_SEQPACKET	Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each input system call.
SOCK_RAW	Provides raw network protocol access.
SOCK_RDM	Provides a reliable datagram layer that does not guarantee ordering.
SOCK_PACKET	Obsolete and should not be used in new programs; see packet(7).

Some socket types may not be implemented by all protocol families.

Since Linux 2.6.27, the type argument serves a second purpose: in addition to specifying a socket type, it may include the bitwise OR of any of the following values, to modify the behavior of socket():

SOCK_NONBLOCK	Set the O_NONBLOCK file status flag on the open file description (see open(2)) referred to by the new file descriptor. Using this flag saves extra calls to fcntl(2) to achieve the same result.
SOCK_CLOEXEC	Set the close-on-exec (FD_CLOEXEC) flag on the new file descriptor. See the description of the O_CLOEXEC flag in open(2) for reasons why this may be useful.

ADDRESS CONFIGURATION is FAMILY DEPENDENT

For INTERNET FAMILY:

- IP ADDRESS
- PORT

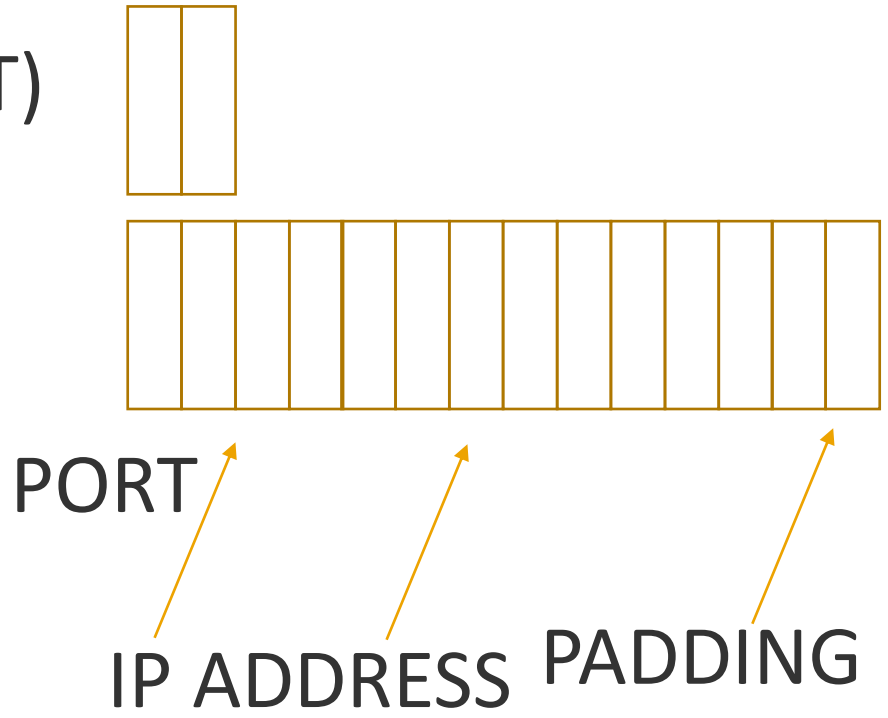
C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

SOCKADDR

```
struct sockaddr {
        u_short sa_family;              /* address family */
        char    sa_data[14];            /* up to 14 bytes of direct address */
};
```

FAMILY (U_SHORT)

DATA (CHAR[14])



ADDRESS CONFIGURATION is FAMILY DEPENDENT

For INTERNET FAMILY:

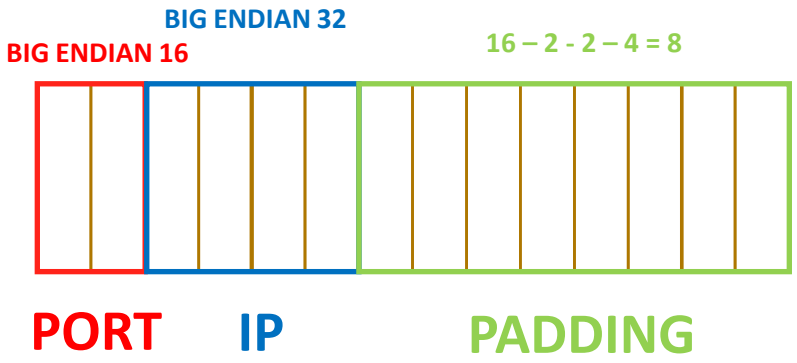
- IP ADDRESS
- PORT

C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

```
/* Structure describing an Internet (IP) socket address. */
#if   __UAPI_DEF_SOCKADDR_IN
#define __SOCK_SIZE__   16              /* sizeof(struct sockaddr)      */
struct sockaddr_in {
  __kernel_sa_family_t  sin_family;     /* Address family               */
  __be16                sin_port;       /* Port number                  */
  struct in_addr        sin_addr;       /* Internet address             */

  /* Pad to size of `struct sockaddr'. */
  unsigned char         __pad[__SOCK_SIZE__ - sizeof(short int) -
                        sizeof(unsigned short int) - sizeof(struct in_addr)];
};
```

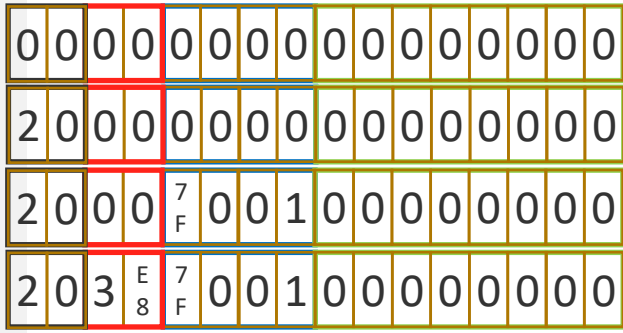
```
/* Internet address. */
struct in_addr {
    __be32  s_addr;
};
```



C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

```
struct sockaddr_in echoserver;
```

```
/* Set information for sockaddr_in */
memset(&echoserver, 0, sizeof(echoserver));        /* reset memory */
echoserver.sin_family = AF_INET;                   /* Internet/IP */
echoserver.sin_addr.s_addr = inet_addr(argv[1]);   /* IP address */
echoserver.sin_port = htons(atoi(argv[3]));        /* server port */
```




```
SO: ./tcp_client1b 127.0.0.1 hola 1000
[0]=0,[1]=0,[2]=0,[3]=0,[4]=0,[5]=0,[6]=0,[7]=0,[8]=0,[9]=0,[10]=0,[11]=0,[12]=0,[13]=0,[14]=0,[15]=0,
[0]=2,[1]=0,[2]=0,[3]=0,[4]=0,[5]=0,[6]=0,[7]=0,[8]=0,[9]=0,[10]=0,[11]=0,[12]=0,[13]=0,[14]=0,[15]=0,
[0]=2,[1]=0,[2]=0,[3]=0,[4]=7F,[5]=0,[6]=0,[7]=1,[8]=0,[9]=0,[10]=0,[11]=0,[12]=0,[13]=0,[14]=0,[15]=0,
[0]=2,[1]=0,[2]=3,[3]=E8,[4]=7F,[5]=0,[6]=0,[7]=1,[8]=0,[9]=0,[10]=0,[11]=0,[12]=0,[13]=0,[14]=0,[15]=0,
SO:
```

```
{
int cont;
char* pointer;

pointer = (char*)&echoserver;
for (cont = 0; cont < sizeof(echoserver); cont++)
    printf("[%d]=%X,",cont,(unsigned char)*pointer++);
printf("\n");
}
```

C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

```c
/* Set information for sockaddr_in */
memset(&echoserver, 0, sizeof(echoserver));       /* Reset memory */
{
  int cont;
  char* pointer;

  pointer = (char*)&echoserver;
  for (cont = 0; cont < sizeof(echoserver); cont++)
      printf("[%d]=%X,",cont,(unsigned char)*pointer++);
  printf("\n");
}
```

```c
echoserver.sin_addr.s_addr = inet_addr(argv[1]);  /* Server address */

  {
  int cont;
  char* pointer;

  pointer = (char*)&echoserver;
  for (cont = 0; cont < sizeof(echoserver); cont++)
      printf("[%d]=%X,",cont,(unsigned char)*pointer++);
  printf("\n");
}
```

```c
echoserver.sin_family = AF_INET;                  /* Internet/IP */

  {
  int cont;
  char* pointer;

  pointer = (char*)&echoserver;
  for (cont = 0; cont < sizeof(echoserver); cont++)
      printf("[%d]=%X,",cont,(unsigned char)*pointer++);
  printf("\n");
}
```

```c
echoserver.sin_port = htons(atoi(argv[3]));       /* Server port */

  {
  int cont;
  char* pointer;

  pointer = (char*)&echoserver;
  for (cont = 0; cont < sizeof(echoserver); cont++)
      printf("[%d]=%X,",cont,(unsigned char)*pointer++);
  printf("\n");
}

exit(1);
```

C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

```
struct sockaddr_in echoserver;
```

```
/* Set information for sockaddr_in */
memset(&echoserver, 0, sizeof(echoserver));      /* reset memory */
echoserver.sin_family = AF_INET;                 /* Internet/IP */
echoserver.sin_addr.s_addr = inet_addr(argv[1]); /* IP address */
echoserver.sin_port = htons(atoi(argv[3]));      /* server port */
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	7 F	0	0	1	0	0	0	0	0	0	0	0
2	0	3	E 8	7 F	0	0	1	0	0	0	0	0	0	0	0

```
The inet_addr() function converts the Internet host address cp from IPv4 numbers-and-
dots notation into binary data in network byte order. If the input is invalid, IN-
ADDR_NONE (usually -1) is returned. Use of this function is problematic because -1
is a valid address (255.255.255.255). Avoid its use in favor of inet_aton(),
inet_pton(3), or getaddrinfo(3), which provide a cleaner way to indicate error re-
turn.
```

STRING (“192.168.0.1”) ←

inet_addr()

INT (192 – 168 – 0 - 1)

```
uint32_t htonl(uint32_t hostlong);

uint16_t htons(uint16_t hostshort);

uint32_t ntohl(uint32_t netlong);

uint16_t ntohs(uint16_t netshort);
```

C FUNCTIONS FOR SOCKETS: ADDRESS CONFIGURATION

A **port number** is a **16-bit unsigned integer**, thus ranging from **0** to **65535**.

For **TCP**, port number 0 is reserved and cannot be used, while for **UDP**, the source port is optional and a value of zero means no port.

A **process** associates its input or output channels via an **Internet socket**, which is a type of **file descriptor**, associated with a **transport protocol**, an **IP address**, and a **port number**.

This is known as **binding**. A socket is used by a process to send and receive data via the network. The operating system's networking software has the task of transmitting outgoing data from all application ports onto the network and forwarding arriving network packets to processes by matching the packet's IP address and port number to a socket.

Applications implementing common services often use specifically reserved well-known port numbers for receiving service requests from clients. The **well-known ports** are defined by convention overseen by the Internet Assigned Numbers Authority (**IANA**).

Conversely, the client end of a connection typically uses a high port number allocated for short term use, therefore called an **ephemeral port**.

RANGE PORT:	WELL-KNOWN :	0→1023 (SYSTEM PORTS)
	NON-STANDARD SERVICES :	1024→4095 (49151)
	EPHEMERAL:	4096 (49152)→65535

C FUNCTIONS FOR SOCKETS: CONNECTION REQUEST

- From CLIENT point of view (for the 5-tuple parameters) we have:

  - THE PROTOCOL/FAMILY
  - THE REMOTE ADDRESS (IP+PORT)

  - DO WE NEED TO CONFIGURE THE LOCAL ADDRESS (IP+PORT)?

C FUNCTIONS FOR SOCKETS: CONNECTION REQUEST

- Do you need to KNOW YOUR PHONE NUMBER TO MAKE A CALL?
  - SO you DO NOT NEED TO CONFIGURE YOUR LOCAL ADDRESS TO REQUEST FOR A CONNECTION:
    - The OS will know the right LOCAL IP ADDRESS TO USE and will configure it by ITSELF
    - The OS will select a LOCAL PORT for the CLIENT (A RANDOM PORT NUMBER > 4096)