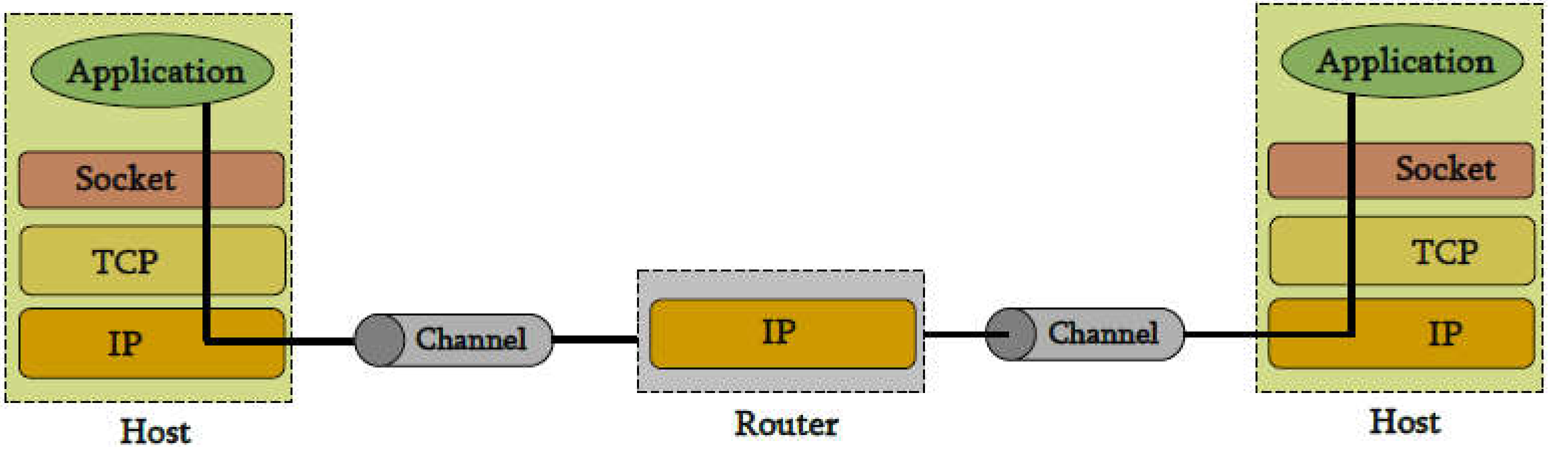
Podešavanje mreže i razvojnog okruženja

- TCP/IP, Telnet servisi pod WIN 10, CMD
- Automatsko konfigurisanje TCP/IP parametara
- Za pocetak isključen Firewall
- Ipconfig
- Ping
- Ako je potrbano isklljučenje ostalih anti-virus programa
- Telnet <adresa> 13
- Telnet <adresa> 17

Socket programming basics

Berkley Sockets

- Universally known as **Sockets**
- It is an abstraction through which an application may send and receive data
- Provide **generic access** to interprocess communication services
  - e.g. IPX/SPX, Appletalk, TCP/IP
- Standard API for networking

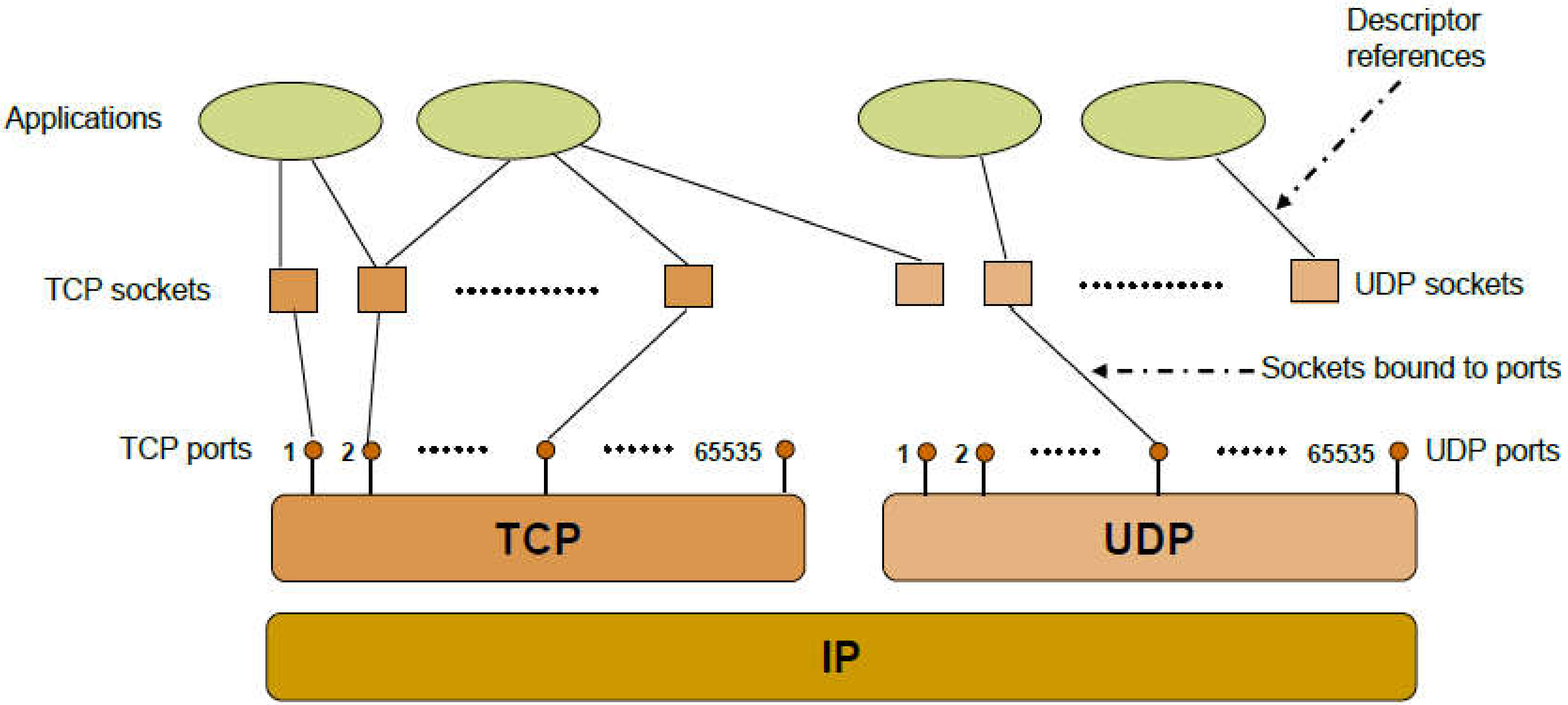


Sockets

- Uniquely identified by
  - an internet address
  - an end-to-end protocol (e.g. TCP or UDP)
  - a port number
- Two types of (TCP/IP) sockets
  - Stream sockets (e.g. uses TCP)
    - provide reliable byte-stream service
  - Datagram sockets (e.g. uses UDP)
    - provide best-effort datagram service
    - messages up to 65.500 bytes

- Socket extend the convectional UNIX I/O facilities
  - file descriptors for network communication
  - extended the read and write system calls

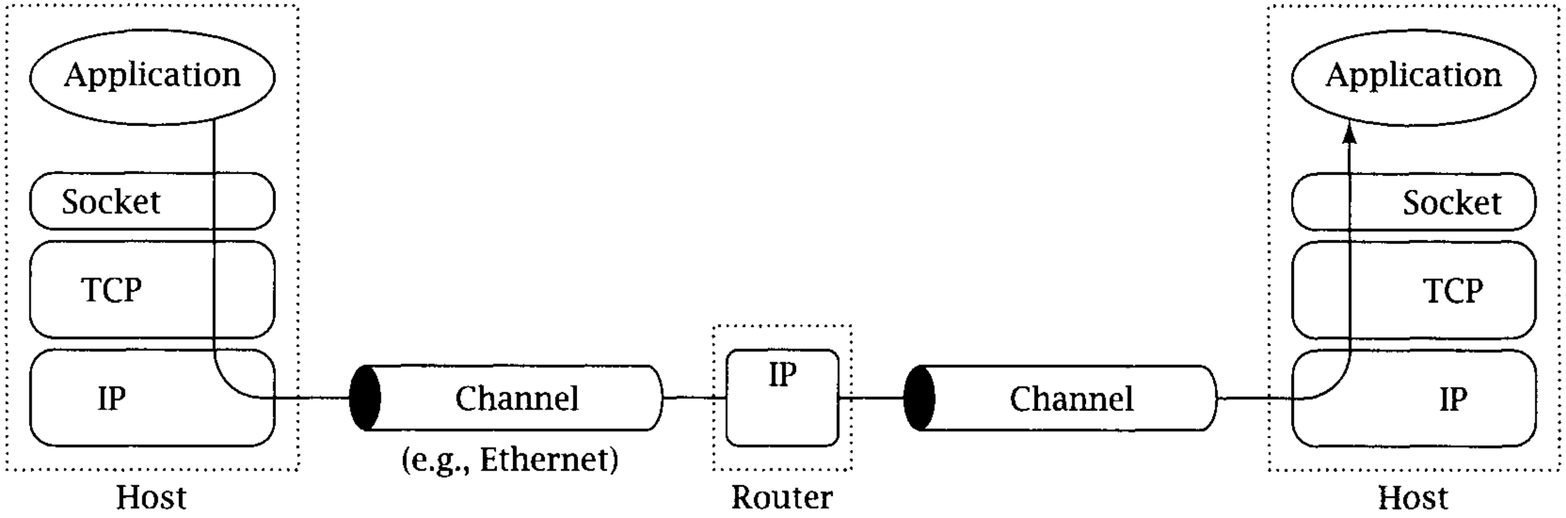
Sockets


Descriptor references
Applications
TCP sockets
UDP sockets
Sockets bound to ports
TCP ports
1
2
65535
1
2
65535
UDP ports
TCP
UDP
IP

Client-Server communication

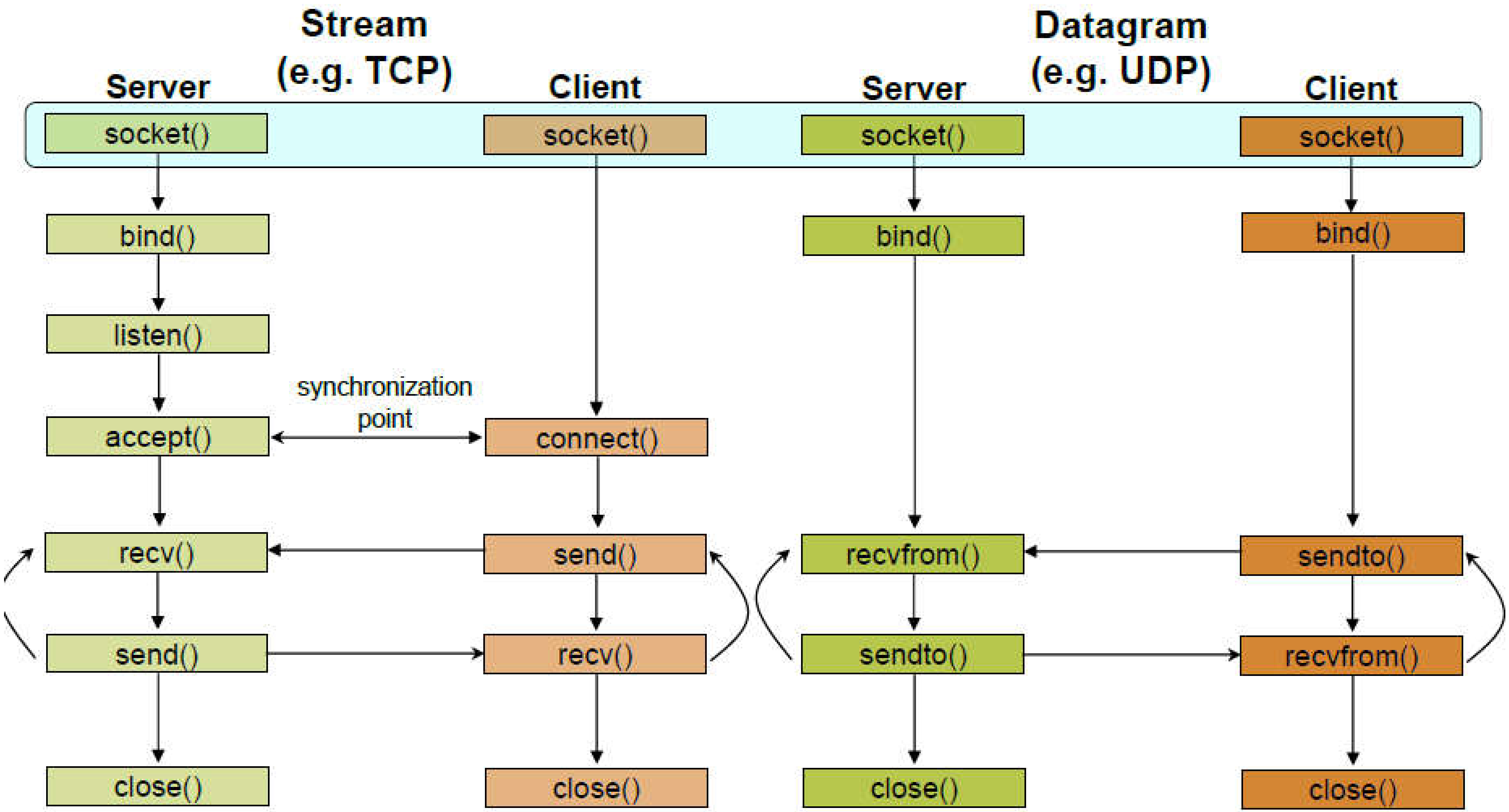
- Server
  - passively waits for and responds to clients
  - passive socket
- Client
  - initiates the communication
  - must know the address and the port of the server
  - active socket

IPv4 komunikacija



Sockets - Procedures

Primitive	Meaning
Socket	Create a new communication endpoint
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection


Stream
(e.g. TCP)
Datagram
(e.g. UDP)
Server
Client
Server
Client
socket()
socket()
socket()
socket()
bind()
bind()
bind()
listen()
synchronization point
accept()
connect()
recv()
send()
recvfrom()
sendto()
send()
recv()
sendto()
recvfrom()
close()
close()
close()
close()

Creating and Destroying socket

To communicate using TCP or UDP, a program begins by asking the operating system to create an instance of the socket abstraction. The function that accomplishes this is socket(); its parameters specify the flavor of socket needed by the program.

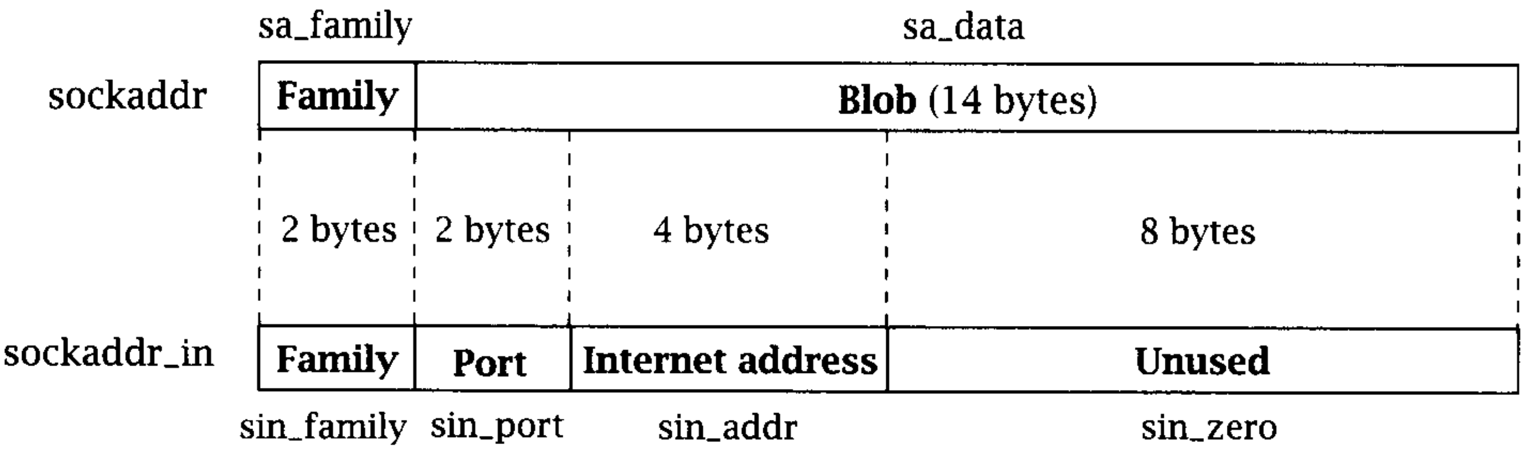
int socket(int protocolFamily, int type, int protocol)

int close(int socket)

• **Protocol family:** sockets API provides a generic interface for a large number of protocol families. PF_INET specifies a socket that uses protocols from the Internet protocol family.
• **type of the socket:** semantics of data transmission with the socket.The constant SOCK_STREAM specifies a socket with reliable byte-stream semantics, SOCK_DGRAM specifies a best-effort datagram socket.
• **Protocol**: end-to-end protocol to be used. For the PF_INET protocol family, we want TCP (identified by the constant IPPROTO_TCP) for a stream socket and UDP (identified by IPPROTO_UDP) for a datagram socket. Supplying the constant 0 as the third parameter requests the default end-to-end protocol for the specified protocol
• **Return**: a nonnegative value for success and -1 for failure

Specifying Addresses

```
struct sockaddr {
unsigned short sa_family;
char sa_data[14] ;
};
```



sa_family sa_data

sockaddr: **Family** | **Blob** (14 bytes)

2 bytes | 2 bytes | 4 bytes | 8 bytes

sockaddr_in: **Family** | **Port** | **Internet address** | **Unused**

sin_family sin_port sin_addr sin_zero

Specifying Addresses

```
struct in_addr
{
        unsigned long s_addr;
};
/* Internet address (32 bits) */

struct sockaddr_in
{
        unsigned short sin_family; /* Internet protocol (AF_INET) */
        unsigned short sin_port; /* Address port (16 bits) */
        struct in_addr sin_addr; /* Internet address (32 bits) */
        char sin_zero[8]; /* Not used */
}
```

TCP Client

The typical TCP client goes through four basic steps:

1. Create a TCP socket using socket().
2. Establish a connection to the server using connect ().
3. Communicate using send() and recv().
4. Close the connection with close().

int connect(int *socket, struct sockaddr *foreignAddress, unsigned int addressLength)*

- *socket* is the descriptor created by socket ().
- *foreignAddress* is declared to be a pointer to a sockaddr because the sockets API is generic; for our purposes, it will always be a pointer to a sockaddr_in containing the Internet address and port of the server,
- *addressLength specifies* the length of the address structure and is invariably given as sizeof(struct sockaddr_in).
- When connect () returns successfully, the socket is connected and communication can proceed with calls to send() and recv().

TCP Client

int send(int *socket,* ***const void *msg, unsigned int msgLength, int flags)***
int recv(int *socket, void *rcvBuffer, unsigned int bufferLength, int flags)*

The default behavior for send() is to block until all of the data is sent

•The *flags* parameter in both send() and recv() provides a way to change the default behavior of the socket call.

•Setting *flags to 0* specifies the default behavior, send() and recv() return the number of bytes sent or received or -1 for failure.

TCP Server

1. Create a TCP socket using socket().
2. Assign a port number to the socket with bind().
3. Tell the system to allow connections to be made to that port, using *listen* ( ) .
4. Repeatedly do the following:
    - Call accept () to get a new socket for each client connection.
    - Communicate with the client via that new socket using send() and recv().
    - Close the client connection using close().

-while the client has to supply the server's address to connect(), the server has to specify its own address to bind().

int bind(int *socket, struct sockaddr *localAddress, unsigned int addressLength)*

TCP Server

int listen(int *socket, int queueLimit)*

•*queueLimit:* parameter specifies an upper bound on the number of incoming connections that can be waiting at any time. The precise effect of *queueLimit* is very system dependent, so consult your system's technical specifications.)
•listen ( ) returns 0 on success and - 1 on failure.

The socket that has been bound to a port and marked "listening" is never actually used for sending and receiving.
it is used as a way of getting new sockets, one for each client connection; the server then sends and receives on the new sockets.
The server gets a socket for an incoming client connection by calling *accept ().*

int accept(int *socket, struct sockaddr *clientAddress, unsigned int *addressLength)*

TCP Server

accept() dequeues the next connection on the queue for *socket.* If the queue is empty*,* accept() blocks until a connection request arrives. When successful, accept() fills in the *sockaddr* structure, pointed to by *clientAddress,* with the address of the client at the other end of the connection,

addressLength specifies the maximum size of the *clientAddress* address structure and contains the number of bytes actually used for the address upon return.

If successful, accept() returns a descriptor for a new socket that is connected to the client. The socket sent as the first parameter to accept() is unchanged (not connected to the client) and continues to listen for new connection requests. On failure, accept() returns -1.

The server communicates with the client using send() and recv(); when ommunication is complete, the connection is terminated with a call to close().