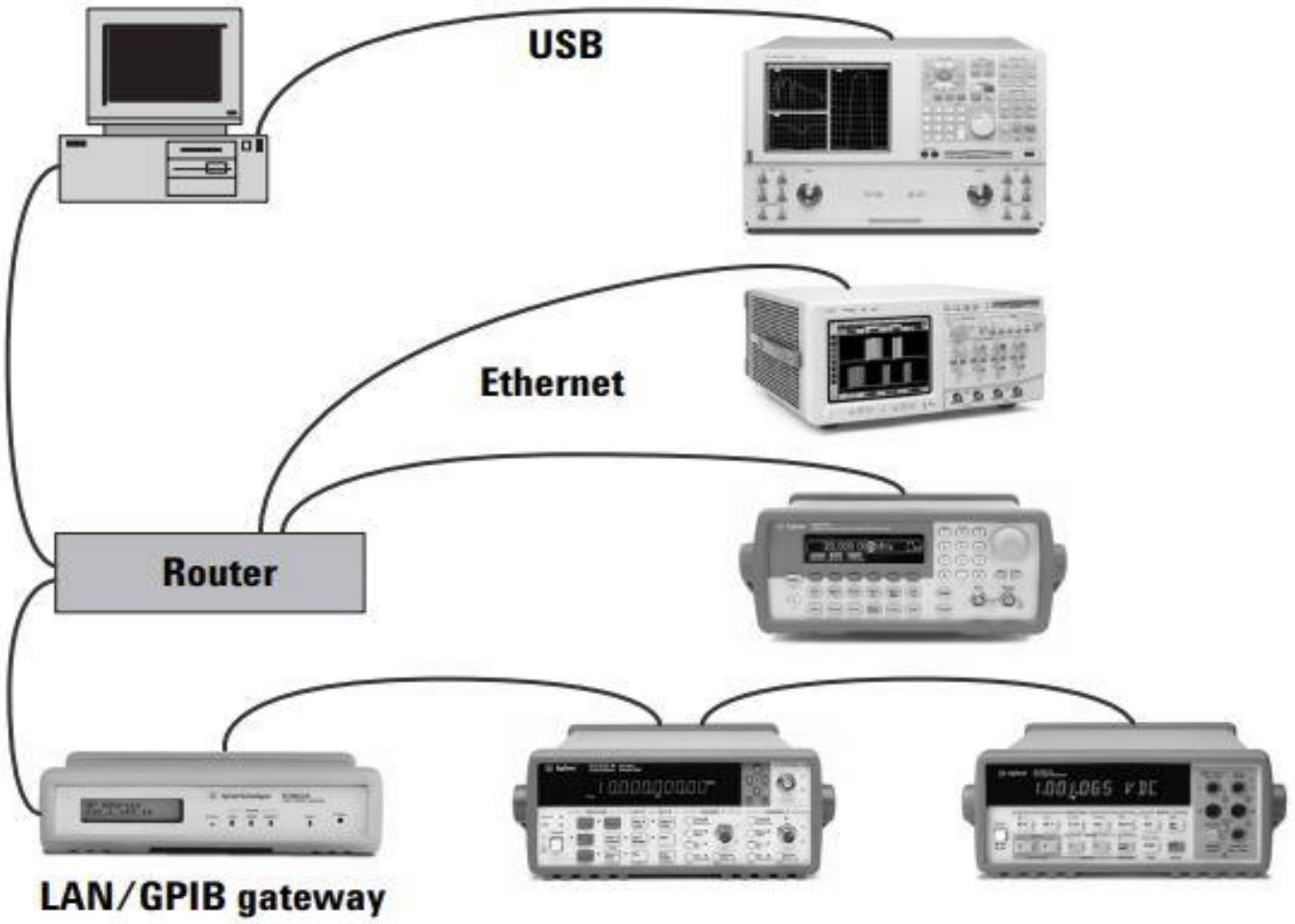
Primena TCP/IP u namenskim sistemima

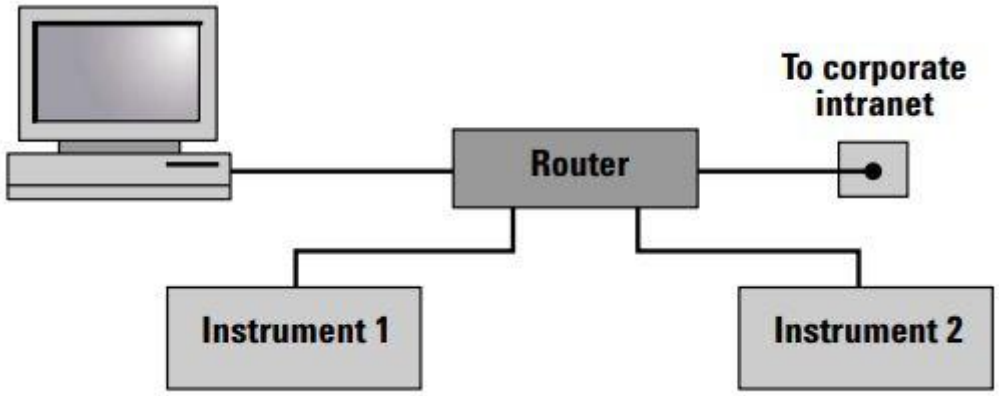
Mikrokontroleri i TCP-IP

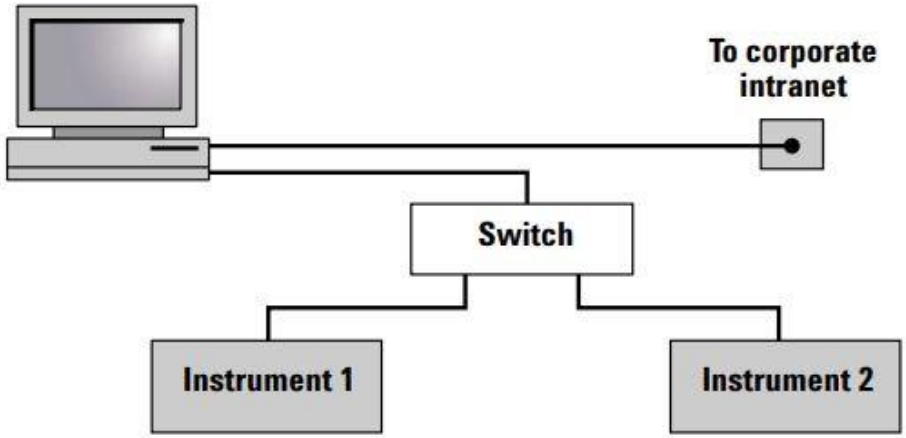
Ethernet or other multi drop protocols

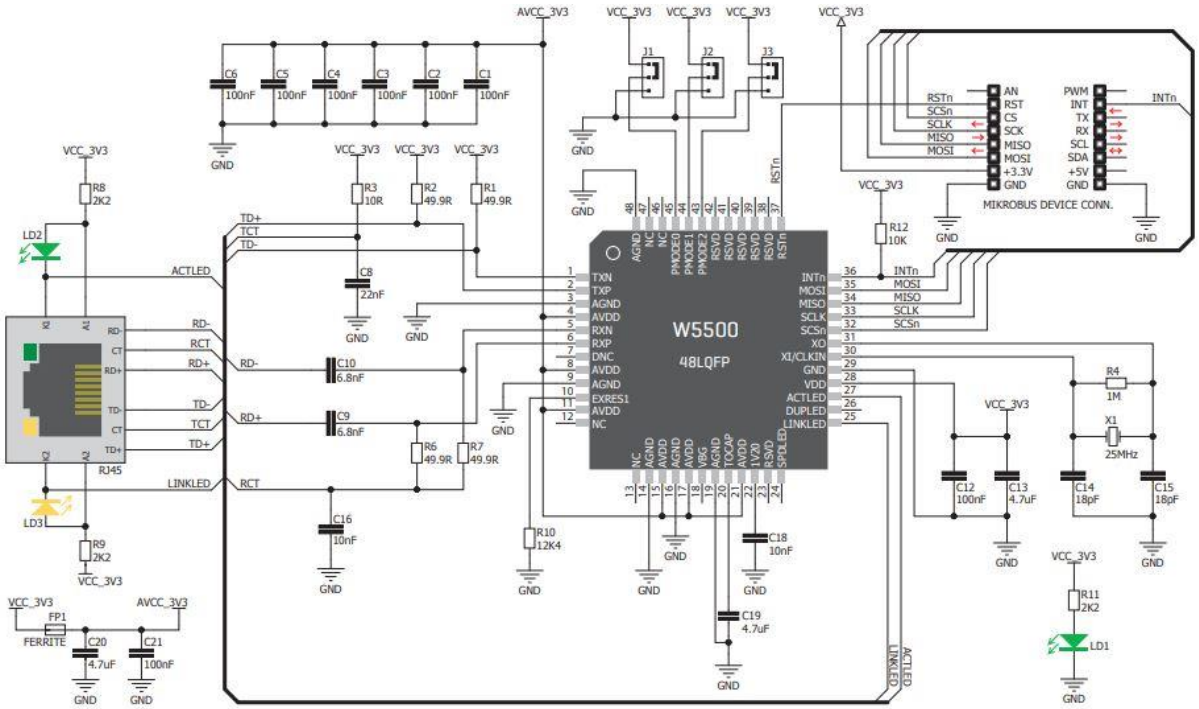
As Ethernet cards have become standard on PC's their price has fallen dramatically. The availability of Ethernet chips that are easily interfaced to both 8 and 16 bit micros, at less than $10 each, along with the ease and familiarity of using networked PC's will make this an increasingly popular communication media for industrial systems.

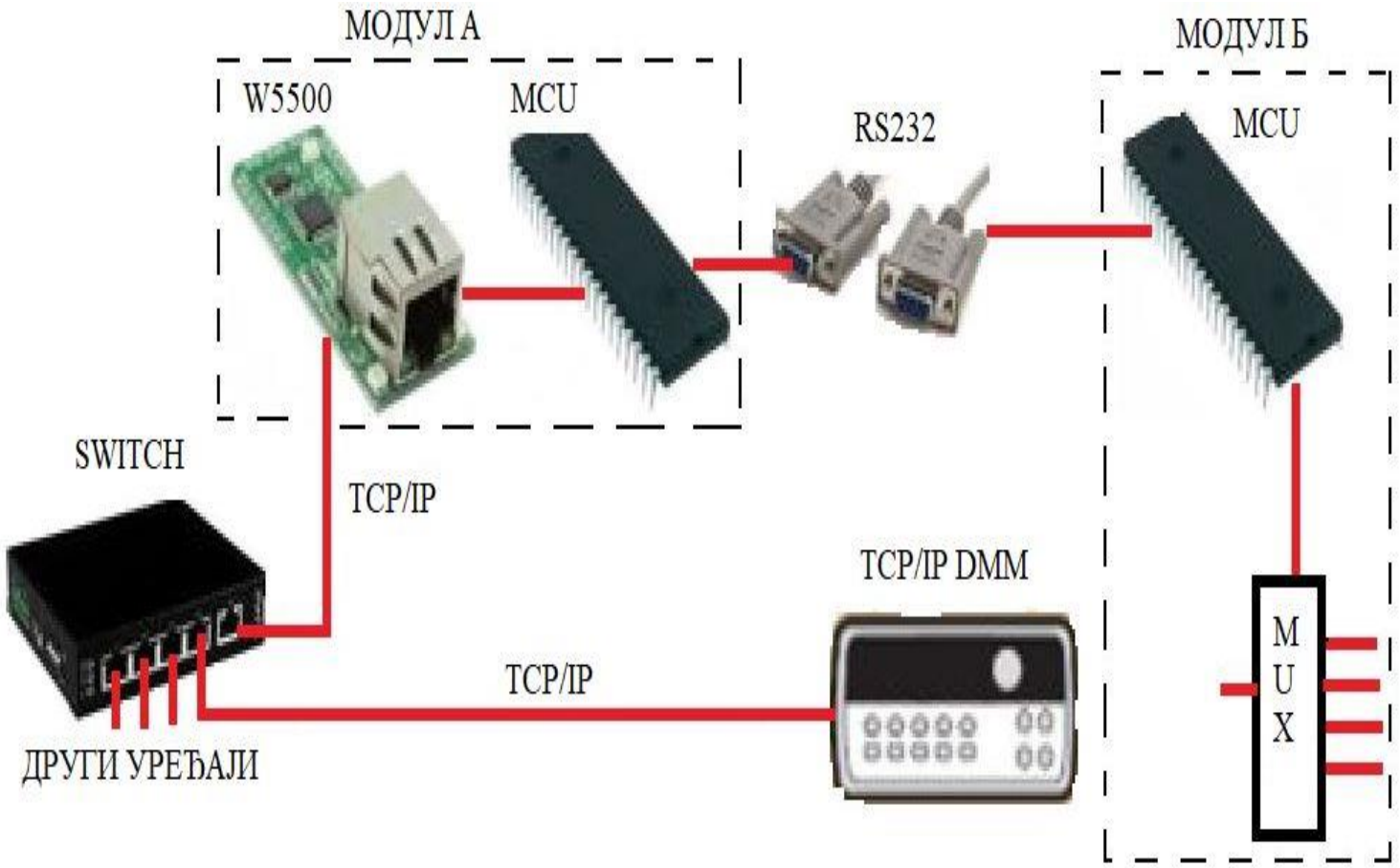
It is also clear that the software components required to control an Ethernet network could be easily adapted to operate multi station Transport Layers such as RS485, CAN and other proprietary multi drop links by the provision of appropriate hardware drivers.


USB
Ethernet
Router
LAN/GPIB gateway


To corporate intranet
Router
Instrument 1
Instrument 2


To corporate intranet
Switch
Instrument 1
Instrument 2


W5500
48LQFP
MIKROBUS DEVICE CONN.
RJ45
AVCC_3V3
VCC_3V3
GND
TXN
TXP
AGND
AVDD
RXN
RXP
DNC
EXRES1
NC
INTn
MOSI
MISO
SCLK
SCSn
XO
XI/CLKIN
VDD
ACTLED
DUPLED
LINKLED
RSTn
TD+
TCT
TD-
RD-
RCT
RD+
LINKLED
ACTLED
FERRITE
25MHz
1M
10K
12K4
2K2
10R
49.9R
100nF
6.8nF
2.2nF
10nF
4.7uF
18pF


МОДУЛ А
W5500
MCU
RS232
МОДУЛ Б
MCU
SWITCH
TCP/IP
TCP/IP DMM
TCP/IP
M
U
X
ДРУГИ УРЕЂАЛИ

Drivers and Stacks

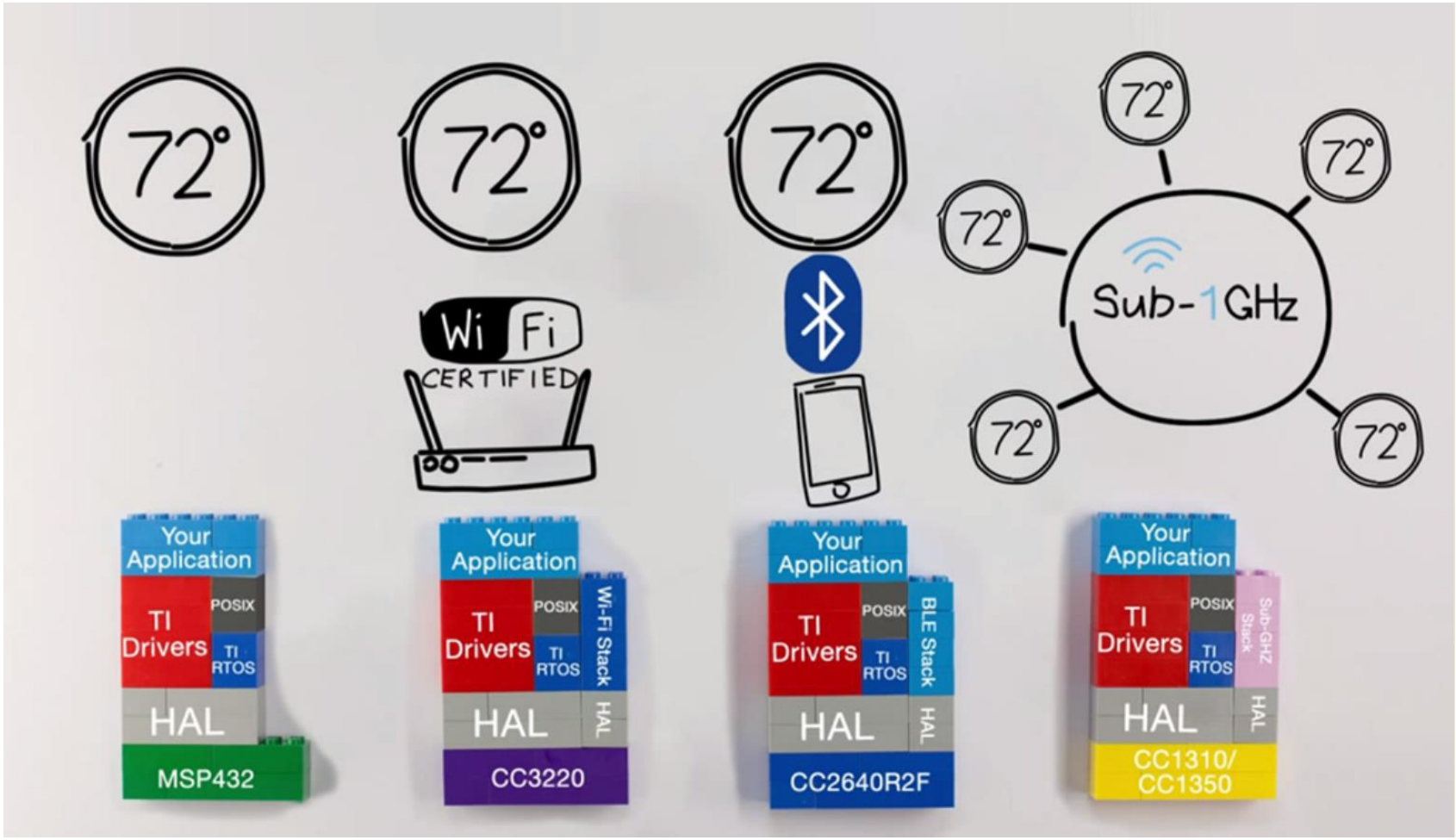
The need for something more complex than a conventional driver arises from the nature of communications networks - multiple interactions is going on at the same time.

Stack is extension of a device driver concept

A Stack is a set of co-operating programs written to work together in many different combinations (*~inheritance*).

Stack underlying structure is provided by a commonly agreed set of protocols (or message standards)

Each level of the stack hiding the messy detail of the level below as we become more and more application oriented.


72°
72°
72°
72°
72°
72°
72°
72°
Wi Fi
CERTIFIED
Sub-1GHz
Your Application
TI Drivers
POSIX
TI RTOS
HAL
MSP432
Your Application
TI Drivers
POSIX
TI RTOS
Wi-Fi Stack
HAL
HAL
CC3220
Your Application
TI Drivers
POSIX
TI RTOS
BLE Stack
HAL
HAL
CC2640R2F
Your Application
TI Drivers
POSIX
TI RTOS
Sub-GHZ Stack
HAL
HAL
CC1310/ CC1350

Layers

The TCP/IP stack is broken down into layers as shown in the table

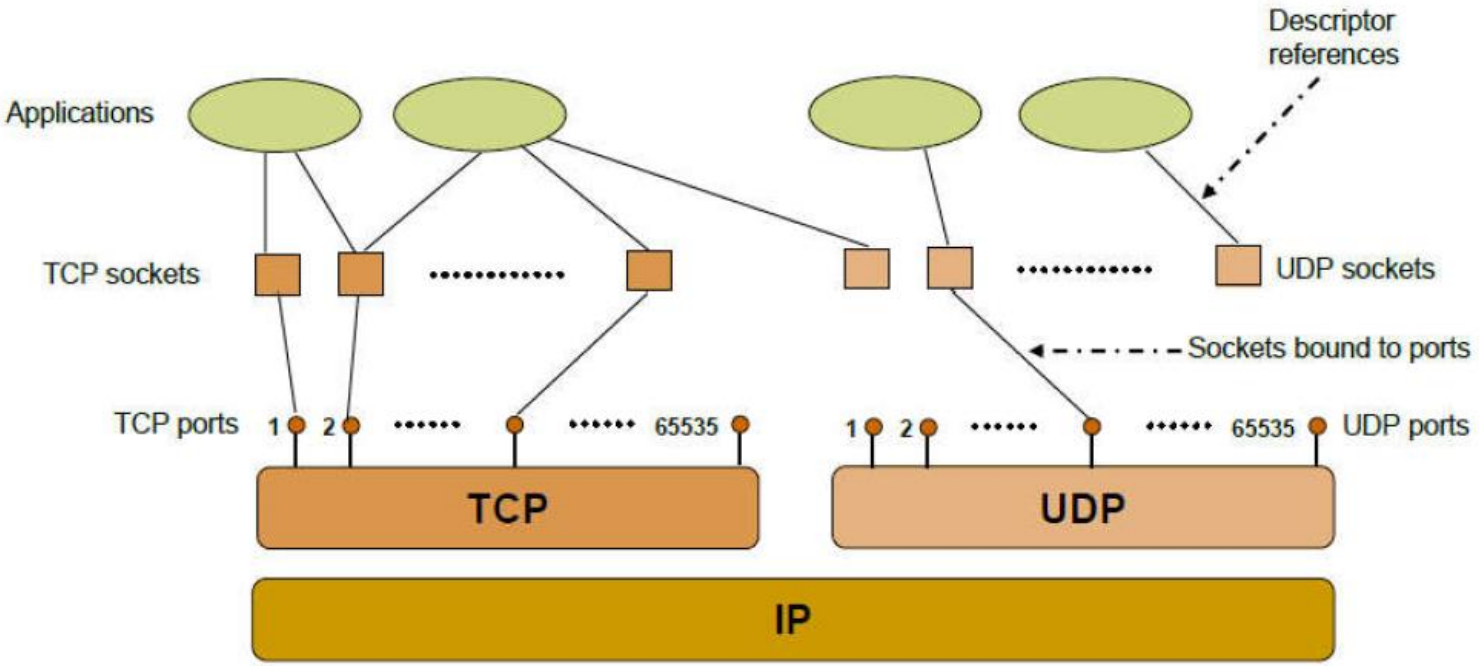
Layer	Examples	Function
Client Application	DHCP, FTP, WEB, YOUR APP	Do Something Useful
Transport	TCP, UDP	Send a Message
Internet	IP, PING, ARP	Component parts, Connect, Data, handshake
Link	PPP, SLIP	Customize for & talks to specific hardware
Physical	RS232, Ethernet	The hardware chip voltage, frequency, signaling techniques

Sockets **

• A ***socket*** is a communications end-point
• If you need to establish a connection with the other program, you need the *socket address* of the application that you want to connect to.
• Once a connection has been established to a socket in the addressee the applications programmer need only consider the data to be read/written.

- 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

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

Blocking and Non-Blocking Sockets

• A *blocking socket*: the program is "blocked" until the request for data has been satisfied. When the remote system does write some data on the socket, the read operation will complete and execution of the program will resume.
• *A non-blocking socket* requires that the application recognize the error condition and handles the situation appropriately.
• The default behavior for socket functions is to "block" and not return until the operation has completed

Client-Server Applications

Programs written to use TCP are developed using the *Client-Server model*.

•The Client application initiates what is called an *active open*. It creates a socket and actively attempts to connect to a Server program.
•The Server application creates a socket and passively listens for incoming connections from Clients, performing what is called a *passive open*.
•When the Client initiates a connection, the Server is notified that some process is attempting to connect with it.
•By *accepting* the connection, the Server completes what is called a *virtual circuit*, a logical communications pathway between the two programs.

- **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

The Server side

- Create a socket.
- Listen for incoming connections from Clients.
- Accept the Client connection.
- Send and receive information.
- Close the socket when the Client has finished or when the Server wishes to no longer be available.

The Client side:

- Create a socket.
- Specify the address and service port of the Server program.
- Establish the connection with the Server.
- Send and receive information.
- Close the socket when finished, terminating the conversation.

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

Using the socket interface

```
socket = mn_open(dest_ip, src_port, dest_port,
client, TCP, recv_buff, buff_len);

status = mn_send(socket, msg_ptr, msg_len);
status = mn_recv(socket, buff_ptr, buff_len);
status = mn_close (socket);
```

Multithreading

• Multithreading extends the idea of multitasking into applications, so you can subdivide specific operations within a single application into individual threads.
• Each of the threads can run in parallel.
•The OS divides processing time not only among different applications, but also among each thread within an application.

• Posix - pthreads

Thread Synchronization Mechanisms

- Mutual exclusion (mutex):
  - guard against multiple threads modifying the same shared data simultaneously
  - provides locking/unlocking critical code sections where shared data is modified
  - each thread waits for the mutex to be unlocked (by the thread who locked it) before performing the code section

Basic Mutex Functions

```
int pthread_mutex_init(pthread_mutex_t *mutex, const
pthread_mutexattr_t *mutexattr);

int pthread_mutex_lock(pthread_mutex_t *mutex);

int pthread_mutex_unlock(pthread_mutex_t *mutex);

int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Select Start,

Control Panel,

Programs and Features (or Programs)

- Select Turn Windows Features on or off
- Check the box for both Telnet Client and Telnet Server
- Select OK
- Verify that you can now Telnet the port

- **ipconfig/all,**
- **tlenet <Ipaddress> <17|13|…>**