The Abdus Salam International Centre for Theoretical Physics

Advanced School on Programmable System-on-Chip for Scientific Instrumentation

FreeRTOS and TCP/IP communication: the lwIP library

Fernando Rincón

fernando.rincon@uclm.es

Contents

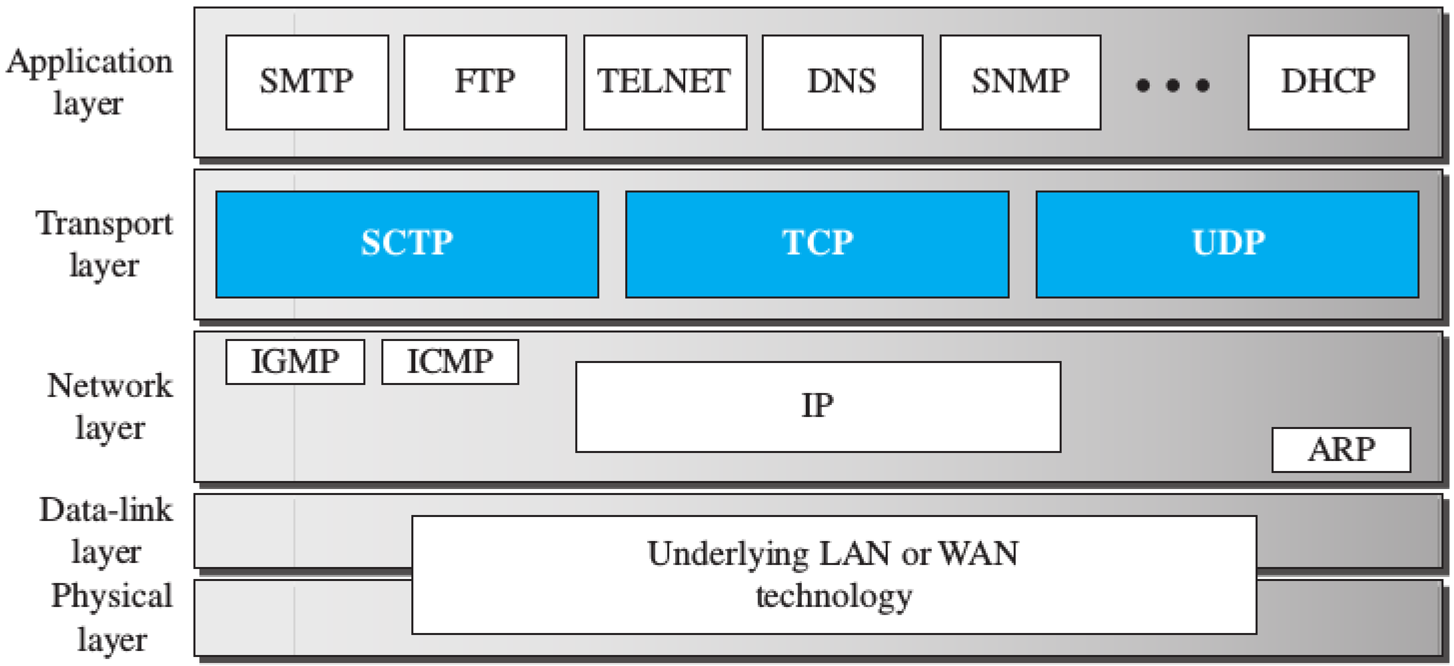
- The lwIP TCP/IP stack
  - The network stack
  - The socket concept
- Application Architectures
- lwIP and FreeRTOS
- https://www.xilinx.com/video/soc/networking-with-lwip-focused-free-rtos.html

lwIP TCP/IP stack

- lwIP stands for Lightweight IP:
  - Small footprint implementation
  - Specially well suited for embedded systems
- Supports a large number of protocols
  - UDP, TCP, ICMP, ARP, ...
- APIs:
  - Berkeley sockets:
    - requires an O.S.
  - Raw API
    - With or without OS
    - More control, but more complex to use
- Included in xilinx SDK
  - Also includes driver for Xilinx Ethernet driver
  - XAPP1026 is the reference application note

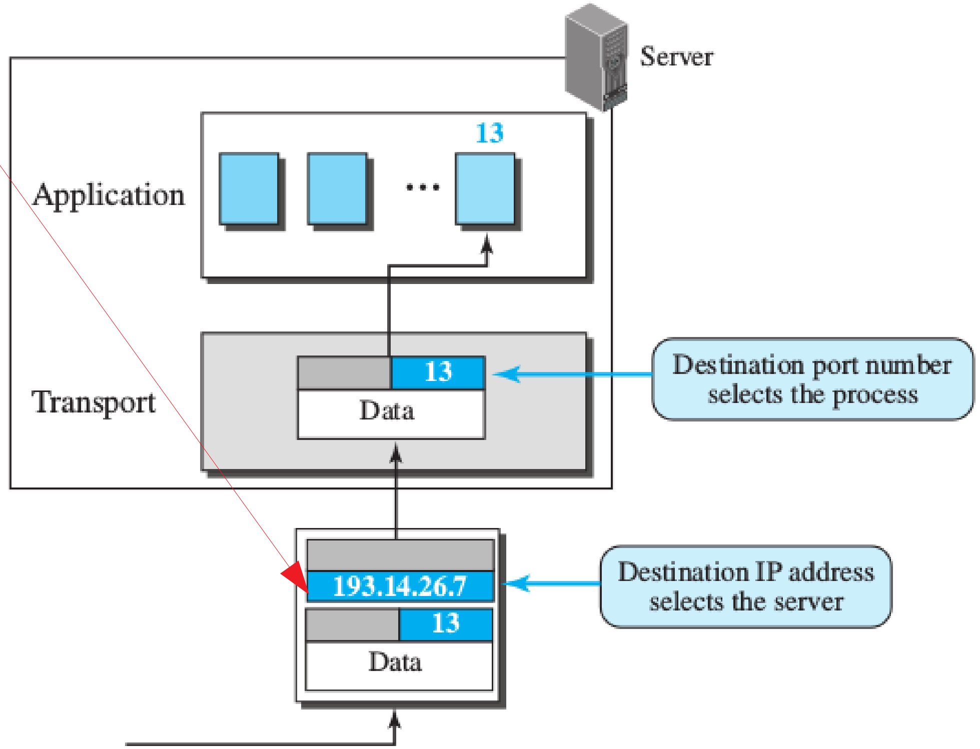
The network stack

- The network design is organized as a layer stack.
- Each layer provides a set of services to the upper layer and requires services from the lower layer.
- The layer 'n' of a node maintains a virtual conversation with the same layer the destination node. That conversation must meet a specific protocol.



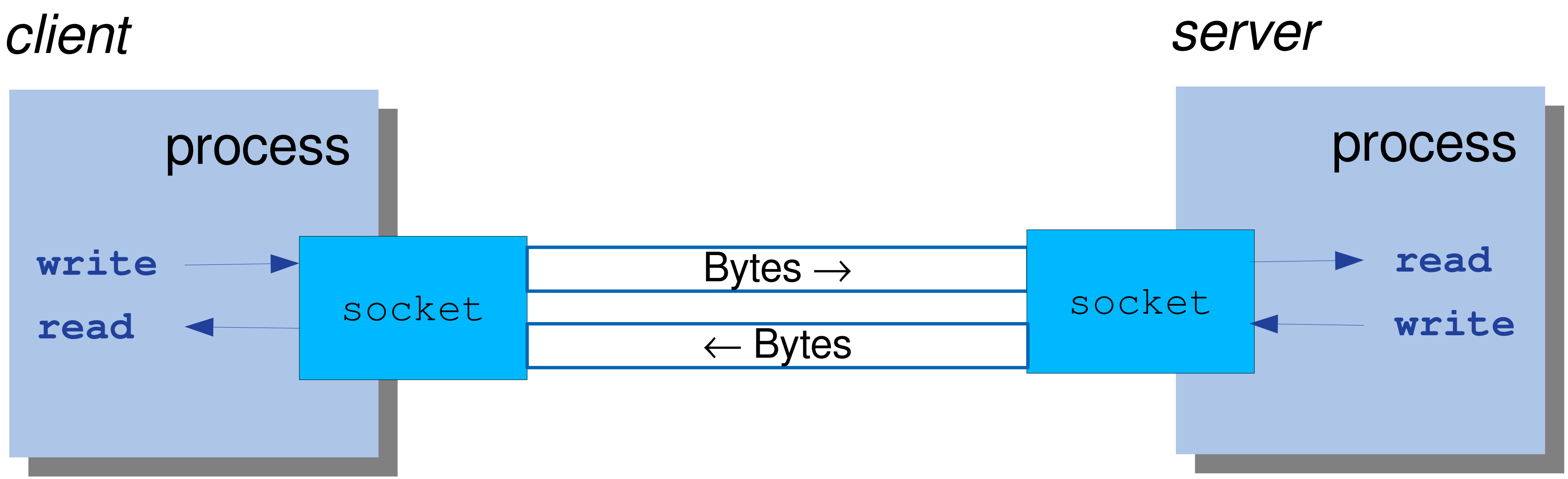
Network sockets

- Socket:
  - Basic abstraction for network programming
  - Combination of IP + port
  - Inter-process communication



Network sockets

- From the programming point of view
  - Channel abstraction
- Berkeley sockets (BSD sockets | POSIX sockets)
  - De facto standard API
- LwIP Socket API `lwip_socket(AF_INET, SOCK_STREAM, 0)`
  - ‘lite’ version of BSD socketss



Application Architectures: superloop

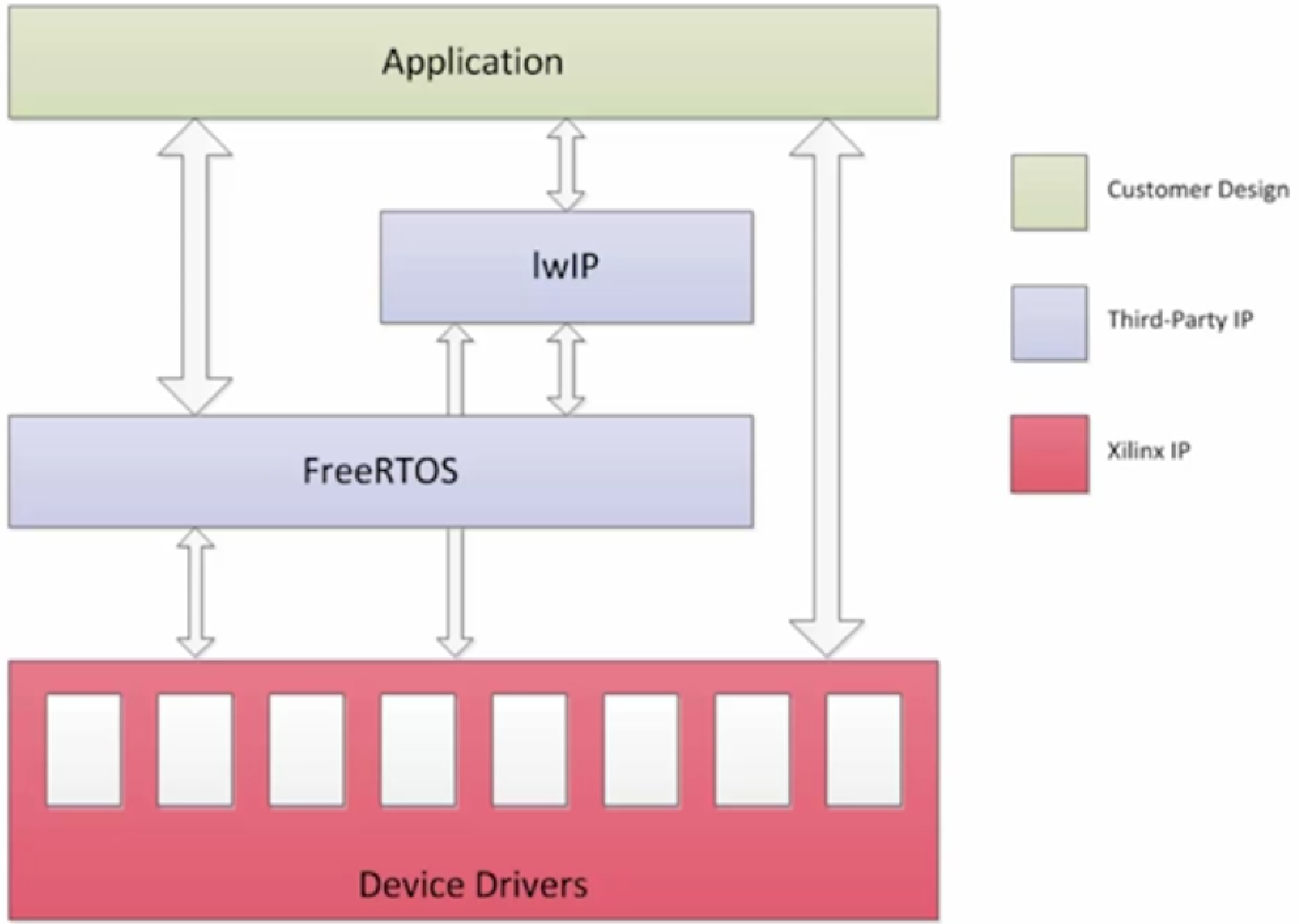
- Two main alternatives:
  - Superloop
  - multi-threaded
- Superloop:
  - forever loop that sequences the set of tasks
  - Typical in standalone implementations
  - Pros:
    - Simple
    - No OS overhead
  - Cons
    - Difficult to scale (low number of tasks)
    - Difficult to balance time and tasks priorities

```
int main() {
    init_system();
    …
    While(1) {
        do_a();
        do_b();
        do_c();
    }
    // You'l never get here
}
```

Application architectures: multi-threaded

- Multi-threaded:
  - multiple threads spawn to carry out multiple tasks concurrently
  - Each task has different priority and timing requirements
  - Requires an operating system
  - Pros:
    - More modular architecture
    - Tasks can be pre-empted. Avoid priority inversion
  - Cons:
    - More complex and extra overhead
    - Higher memory requirements
    - Thread execution is difficult to test

FreeRTOS Application Architecture



FreeRTOS network application

- Basic template for socket oriented programming:
  - First setup a network thread and start FreeRTOS task scheduler
  - Then the network thread:
    - Initializes lwip
    - Configures a network interface
    - Rises the interface
    - Starts another thread for the reception
    - Installs any other network tasks (new threads) required by the application
      - We'll suppose an echo server
    - Finally it deletes itself
  - Therefore after initialization several threads are active:
    - Reception
    - Echo server

FreeRTOS network application

- Initialization:

```
int main()
{
        xil_printf("UDP echo server\r\n");
        sys_thread_new("main_thrd", (void(*)(void*))setup_network_thread, 0,
                        THREAD_STACKSIZE,
                        DEFAULT_THREAD_PRIO);
        vTaskStartScheduler();
        while(1);
        return 0;
}
```

We need FreeRTOS to start task execution for the rest of the procedure

Equivalent to a FreeRTOS task

FreeRTOS network application

- Network configuration thread

```
void setup_network_thread()
{
    struct netif server_netif;
    struct ip_addr ipaddr, netmask, gw;
    unsigned char mac_ethernet_address[] = { 0x00, 0x0a, 0x35, 0x00, 0x01, 0x02 };

    /* initialize lwIP before calling sys_thread_new */
    lwip_init();

    IP4_ADDR(&ipaddr,  192, 168, 1, 22);
    IP4_ADDR(&netmask, 255, 255, 255,  0);
    IP4_ADDR(&gw, 192, 168, 1, 1);

    /* Add network interface to the netif_list, and set it as default */
    if (!xemac_add(&server_netif, &ipaddr, &netmask, &gw, mac_ethernet_address, PLATFORM_EMAC_BASEADDR)) {
        xil_printf("Error adding N/W interface\r\n");
        return;
    }
    netif_set_default(&server_netif);

    /* specify that the network if is up */
    netif_set_up(&server_netif);
```

Static IP configuration

… (continues in the next slide)

FreeRTOS network application

- Network configuration thread

```
    /* start packet receive thread - required for lwIP operation */
    sys_thread_new("xemacif_input_thread", (void(*)(void*))xemacif_input_thread, &server_netif,
            THREAD_STACKSIZE,
            DEFAULT_THREAD_PRIO);

    sys_thread_new("echod", echo_application_thread, 0,
                                    THREAD_STACKSIZE,
                                    DEFAULT_THREAD_PRIO);

    vTaskDelete(NULL);
    return;
}
```

Tasks that will remain active:

- Network data reception
- Echo server (that processes data received)

This task is just for configuration purposes

Network programming concepts: Error control

- Network communication at the IP level is unreliable
- If reliability is required at the transport layer. Error control is responsible for:
  - Detect and discard corrupt packets
  - Keep track of lost and discarded packets (resend)
  - Discard duplicates
  - Buffer out-of-order packets
- Implemented through:
  - Sequence numbers (in packets)
  - Acknowledgement and timers

Network programming concepts: Flow control

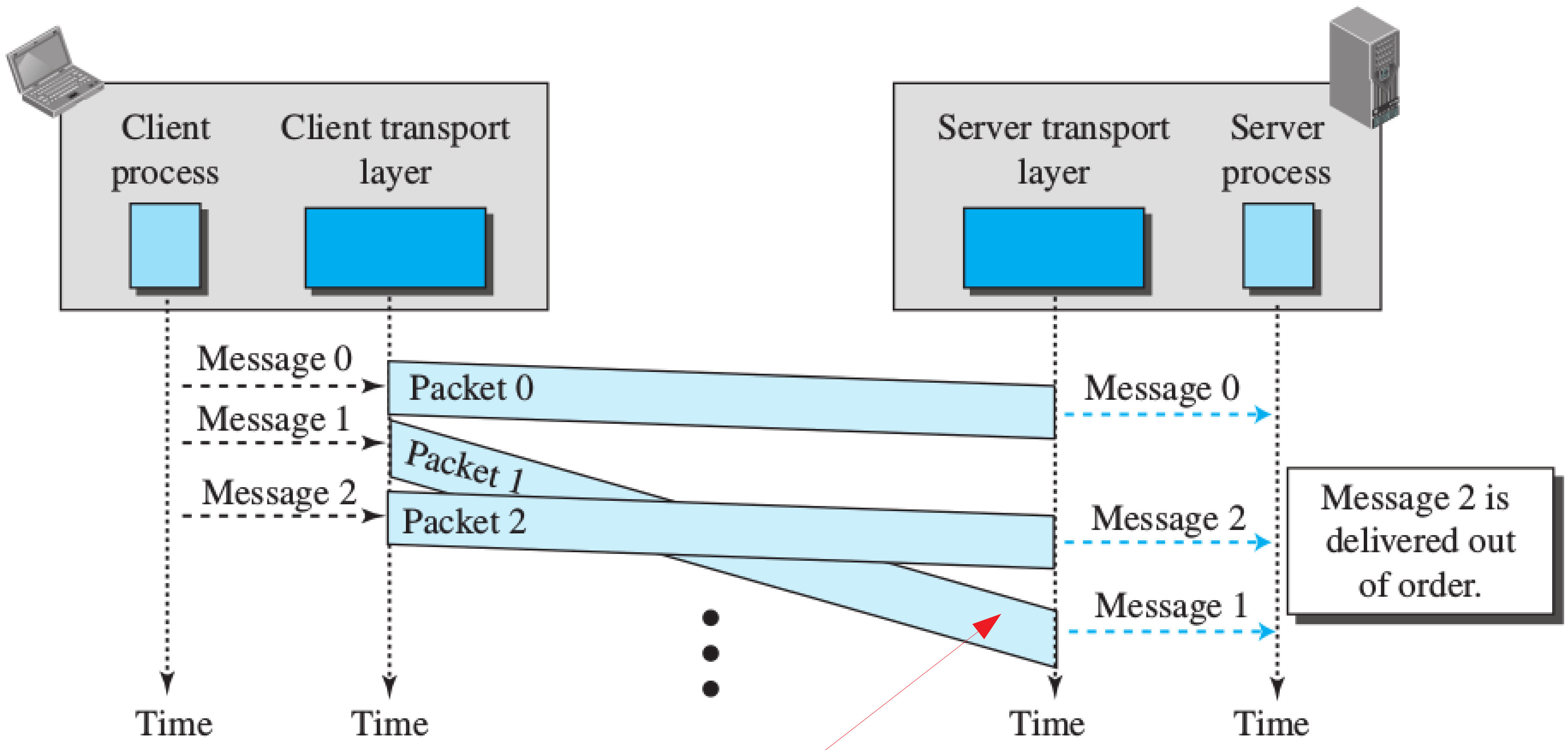
- Both the sender and receiver adjust the transmission speed
- sliding window:
  - Buffer used to make the transmission more efficient
    - See videos
  - as well as to control the flow of data so that the destination does not become overwhelmed with data
    - The destination can reduce the size of the window

UDP

- Unreliable protocol
  - No error control
    - The client ignores if the packet arrived correctly to the server
  - No flow control
    - No way to adjust the speed of both the sender and receiver
- But then why using UDP?
  - Extremely simple (minimum overhead) $\rightarrow$ The fastest way (lowest latency)
  - Control can be provided by de Aplication Layer
    - But that’s on you as a programmer
    - No help from the network stack
  - There are may applications where loosing part of the information can be tolerated:
    - Ex. Video conference

UDP

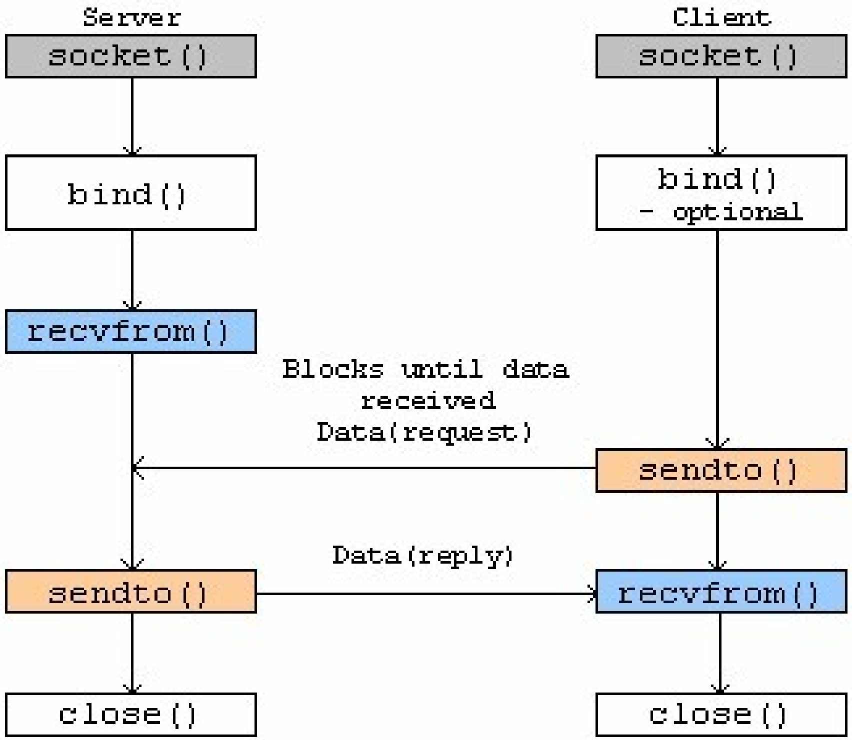
- UDP timing diagram



This packet will be discarded

UDP

- UDP socket Programming flow



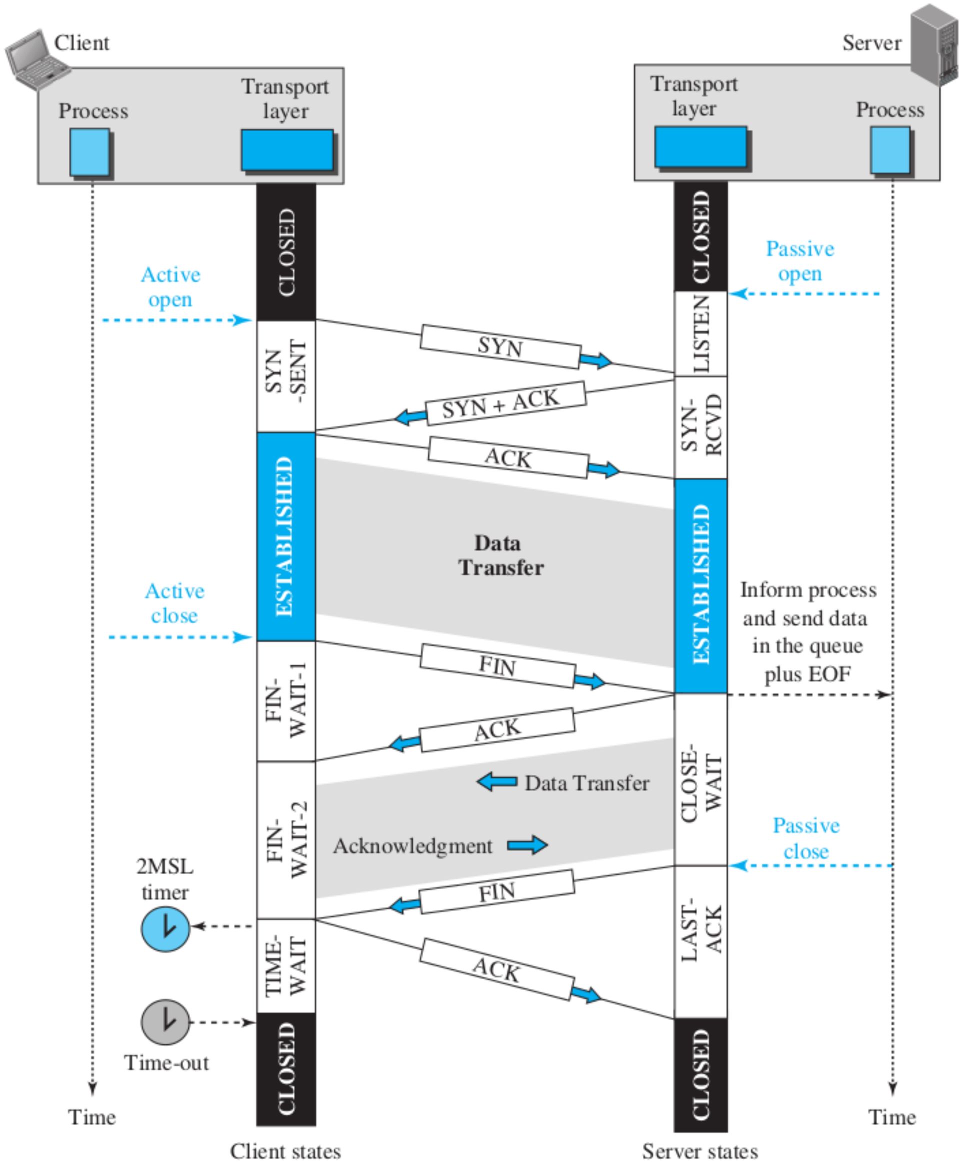

```
void echo_application_thread()
{
        int sock, new_sd;
        struct sockaddr_in address, remote;
        int size;

        int RECV_BUF_SIZE = 2048;
        char recv_buf[RECV_BUF_SIZE];
        int n, nwrote;

        if ((sock = lwip_socket(AF_INET, SOCK_DGRAM, 0)) < 0)
                return;

        address.sin_family = AF_INET;
        address.sin_port = htons(echo_port);
        address.sin_addr.s_addr = INADDR_ANY;

        if (lwip_bind(sock, (struct sockaddr *)&address, sizeof (address)) < 0)
                return;

        if ((n = read(sock, recv_buf, RECV_BUF_SIZE)) < 0) {
                xil_printf("%s: error reading from socket %d, closing socket\r\n",
                           __FUNCTION__, sock);
        }
```

TCP

- Connection-oriented protocol
- Reliable
  - Retransmission of lost or corrupted packets
  - Cumulative and selective ACKs
- Complex protocol with multiple phases (higher latency, lower throughput)
  - Connection establishment
  - Data transfer
  - Connection teardown
- Used when loosing information can’t be tolerated
  - HTTP / HTTPS
  - E-mail, text messaging

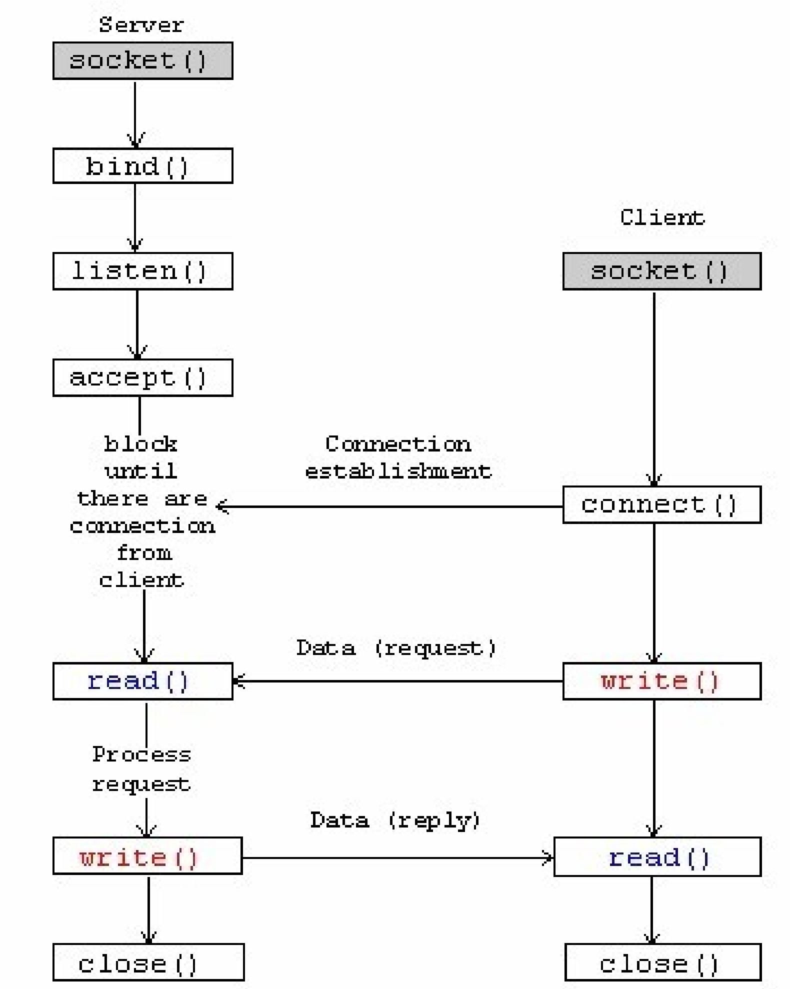
TCP

- TCP communication flow
  - Data transfers must be acknowledges
  - But acknowledges are packed. Not one per packet received



TCP

- TCP socket Programming flow



TCP

```c
void echo_application_thread()
{
        int sock, new_sd;
        struct sockaddr_in address, remote;
        int size;

        if ((sock = lwip_socket(AF_INET, SOCK_STREAM, 0)) < 0)
                return;

        address.sin_family = AF_INET;
        address.sin_port = htons(echo_port);
        address.sin_addr.s_addr = INADDR_ANY;

        if (lwip_bind(sock, (struct sockaddr *)&address, sizeof (address)) < 0)
                return;

        lwip_listen(sock, 0);

        size = sizeof(remote);

        while (1) {
                if ((new_sd = lwip_accept(sock, (struct sockaddr *)&remote, (socklen_t *)&size)) > 0) {
                        sys_thread_new("echos", process_echo_request,
                                (void*)new_sd,
                                THREAD_STACKSIZE,
                                DEFAULT_THREAD_PRIO);
                }
        }
}
```

TCP

```c
/* thread spawned for each connection */
void process_echo_request(void *p)
{
  int sd = (int)p;
  int RECV_BUF_SIZE = 2048;
  char recv_buf[RECV_BUF_SIZE];
  int n, nwrote;

  while (1) {
    /* read a max of RECV_BUF_SIZE bytes from socket */
    if ((n = read(sd, recv_buf, RECV_BUF_SIZE)) < 0) {
      xil_printf("%s: error reading from socket %d, closing socket\r\n", __FUNCTION__, sd);
      break;
    }

    /* break if client closed connection */
    if (n <= 0)
      break;

    /* handle request */
    if ((nwrote = write(sd, recv_buf, n)) < 0) {
      xil_printf("%s: ERROR responding to client echo request. received = %d, written = %d\r\n",
                 __FUNCTION__, n, nwrote);
      xil_printf("Closing socket %d\r\n", sd);
      break;
    }
  }

  /* close connection */
  close(sd);
  vTaskDelete(NULL);
}
```

lwIP performance

- Depends on
  - the concrete hardware (also the CPU not just the network interface)
  - and API used (RAW or socket)

Hardware Design Name	RAW Mode		Socket Mode	
	RX (Mb/s)	TX (Mb/s)	RX (Mb/s)	TX (Mb/s)
AC701_AxiEth_32kb_Cache	205	125	37	41
AC701_AxiEth_64kb_Cache	270	175	40	46.8
KC705_AxiEth_32kb_Cache	290	190	52.9	56.2
KC705_AxiEth_64kb_Cache	380	250	58.4	69.5
KC705_AxiEthernetlite_64kb_Cache	46	67	29	44
ZC702_GigE	943	949	521	542

Network tools: netcat

- The swiss army knife for network operations
- Can be configured to be a client or a server:
  - UDP example:
    - Server: nc -lu -p <port> [<hostname>]
    - Client: nc -u <hostname> <port>
  - TCP example:
    - Server: nc -l -p <port> [<hostname>]
    - Client: nc <hostname> <port>

Network tools: Wireshark

- Most popular network traffic sniffer
- Captures and analyzes any kind of network traffic
  - Will help you to understand the network stack
  - Protocol dissector
- Multi-platform (even mobile phones)
  - Free software

Network tools: Wireshark