Practices on Computer Networks and Distributed Systems

C/S echo application written in the C language (WIP)

Datagram Sockets

The Internet as a whole can be considered a packet-switching network where any information we want to send must be broken down into a series of small chunks before transmission. The resulting pieces are conventionally known as packets and each is transmitted and transported independently of the others. Packets are similar to letters in many respects, for example, suppose you need to send fifty A4-sized written sheets to a friend in New York City, you will have to break down the **50-page block** into smaller blocks such that each fits within the envelope size you own at the moment. Each of the 5 10-page blocks will have to be addressed to your friend. Potentially, some of them might be transported over different routes than the others and they might arrive in an order other than the original order. We must not forget an important fact now: any of the outgoing letters could be lost by the mail system or get damaged. The example just developed briefly represents the nature of Internet which is technically known as the **Internet Model of Service**.

The terms used in Internet parlance are not the same used to describe the physical world: the 50-page block would be named the message to send, each of the 5 10-page blocks would be called a packet and the network (The Internet) could deliver those packets out-of-order, duplicated, with errors or, even not deliver some of them at all. The Internet Model of Service is a ***best effort model,*** one which guarantees are very loose. There is an Internet model of service property that has no counterpart in the real world example: the network can, unexpectedly duplicate packets erroneously; this does not happen in the mail example - that figures. The key word here is *packet*, a sort of envelope that contains the Internet address of the host to which it is meant to be delivered. This is the basic nature of Internet: lack of deterministic guarantees, but this seems to leave us in a disadvantageous position as beginners, for, what is the rationale of a network that offers us no delivery guarantee? We will delve into the Internet Service Model in upcoming lectures, therefore, for the time being we will want to recover the lost trust, somehow: what the network can not guarantee us, we will compensate for those guarantees by increasing the responsibilities of the sending and the receiving hosts (An end-to-end responsibility, not the network's).

The key word is packet. The network switches packets, pretty much like the switchboard operator of the old circuit-switched telephone networks switched circuits, though you already know there are a lot of differences between both approaches. We build a packet and submit it to the network for transfer to its destination. The packet is the envelope in the mail analogy, then, where is the letter? The letter we name it *datagram* in the context of this practical exercise, that suffices for the time being.

All in all: we have some message that we want to deliver to some application running on a destination host, the *message gets* broken up into separate datagrams, each of which subsequently gets encapsulated into an IP packet, each of these is eventually submitted to the network for transfer to its destination host. How can we illustrate this in a practical manner? The best we can do is make a program that uses the protocols installed in our machine for transmitting information, specifically, we are programming in C against an API that provides us access to the UDP protocol's Service Interface.

A brief introduction to UDP: the simple *process* multiplexer

The example we are building in this practice aims to imitate the **ping program** that we studied in the Computer Networks course. Recall that the real ping command is based on an Internet control-plane protocol named **ICMP**; here, we set out to emulate ping by doing our programming against the UDP protocol (An Internet layer-3 transport protocol) instead of against ICMP (Also a layer-3 protocol, though not quite a transport protocol). How come UDP, what protocol is that?

Let's assume that the host where we are programming these examples has a single NIC which IP address is 193.146.101.46 whose DNS name is paloalto.unileon.es. The correctly assigned IP identifies a single host in Internet and each IP packet directed toward it is ultimately delivered to it (If all is functioning correctly). IP addresses help solve the problem of delivering packets to their intended destination host. Once a packet is delivered to its destination host, a decision should be made about how to deliver the packet's topmost payload to a specific process running in the host. We are assuming that the host is running a multitasking operating system like Linux where a multitude of processes will be running concurrently. UDP (User Datagram Protocol) provides a means of identifying a single process on the destination host that is to receive the topmost payload contained in the packet. UDP does so by providing a *name space for processes*, one that is standardized in the Internet: the port number. UDP allows exposing to the Internet a selected process in a host.

The User Datagram Protocol (UDP) data unit is named Datagram, every datagram has a source port field and a destination port field. The destination port contained in a given UDP datagram ultimately marks the destination process that is to receive the payload in the destination host -in the destination host's stack. Your C program runs in a process space in the host operating system and it may acquire a UDP port by creating and configuring what is known as a UDP socket. Once your program has taken hold of the UDP socket it can use it to transmit and receive datagrams over it, observe the following sketch of the steps required:

1. Your program acquires a UDP socket on port 60007, for example.

2. Now, your program is bound to that port: the UDP socket knows your program and your program knows the UDP socket

3. When your host receives an IP packet that contains a UDP datagram whose destination port is 60007, the datagram's payload (Typically an application-level block of information) will be delivered to your C program.

All in all: An IP address identifies a host in the Internet, whereas a UDP destination port identifies a single process running in that host. Ports are the multiplexing keys employed at the UDP protocol.

A C/S ping application based on UDP

We are starting with two base programs written in C, one for the client and the other one for the server. Follow the steps below to have the C/S application running:

1. Download client: `$ wget http://paloalto.unileon.es/ds/lab/echoClient.c`
2. Download server: `$ wget http://paloalto.unileon.es/ds/lab/echoServer.c`
3. Compile both programs:


```
$ gcc -o echoClient echoClient.c

$ gcc -o echoServer echoServer.c
```

4. Request an additional terminal (Shell) from your system, where you will run the server

5. Run server in the new terminal window
   `$ ./echoServer`

6. Run client in the old terminal window
   `$ ./echoClient`

If all has been successful so far, you will have to see the client sending 5 messages to the server over the Datagram socket. Observe the server reaction to each received message and how it responds to the client. Finally, confirm that the client receives the response produced by the server.

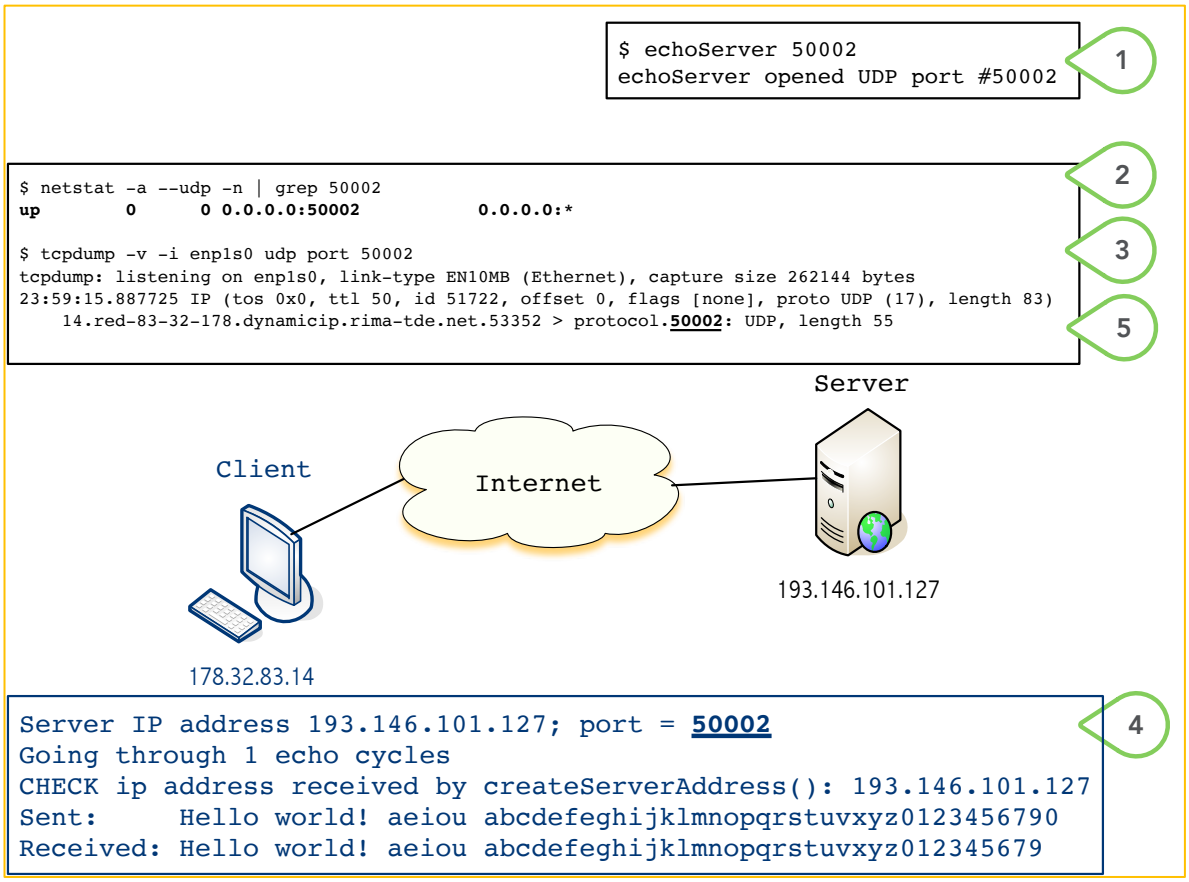

Figure 1. Overall practice setup and checks using UDP port 50002

Exercises

1. Observe the **C/S protocol** implemented by the client and server programs from scratch (No modification is to be done here). Start an appropriate tcpdump trace, one that includes a filter like "udp port 60007". As usual, deduce the protocol stack resulting from the trace. Depict that **protocol stack and annotate** it with the multiplexing key that applies to each of the protocols that appear on it.

2. Modify the client program so that it accepts the server IP address and port as **command line parameters**. The IP address that is to be supplied to your program must be supposed to be formatted in DDN (Dot Decimal Notation; for example, 192.168.1.123). The binary form to a DDN IP address is obtained by calling function inet_aton() – check its man page for details. Likewise, the port is provided as an ASCII string which has to be translated into its expected unsigned integer representation by calling function atoi(). Finally, the bytes from the integer returned by atoi() have to be ordered in the Network Byte Order by calling htons().

3. Modify the server program such that it indeed **echoes back** the same block of characters that it received from the client

4. Can **multiple clients** obtain echo service from your server? Devise a way to check this practically.

5. Can your client and server programs be executed at the **same IP host** and still function correctly? Devise a way to check this practically. Also, provide an explanation that is based on the concepts about the Internet Architecture.

6. [Preparation for upcoming homework] A UDP server is running in host 193.146.101.46 at port 60007. This server sends back a single short integer as the response to a legitimate client request. The only legitimate client request honored by the server is composed of the following string:

   `"Distributed_Systems_23-24"`

   Whenever the server receives that string from a client, it will react by sending back a 16-bit unsigned integer (unsigned short int). If the server receives but the above string, it will send back nothing.

   Write a client that complies with the provided specifications. Use tcpdump to observe the sent and the received messages.

