V 1.0 11th -June-2021

Universidad de León
Ingeniería Informática
Course on Computer Networks

Homework On UDP Datagram Sockets

Details about this homework submission are published in the agora

Introduction. UDP Datagram sockets is the service interface to the UDP protocol. UDP is the Internet protocol used for the end-to-end communication among applications that don't require delivery guarantees. The protocol that provides those guarantees is TCP which we'll introduce in the Distributed Systems course which comes next September.

UDP adds a new *multiplexing layer* on top of IP, one that permits the identification of applications interested in communicating over Internet. The multiplexing keys provided by UDP are known as *UDP Ports,* 16-bit unsigned integers. For example, your personal computer or your cellular phone use reserved UDP port 123 for exchanging time-of-day information with their time server and they do this quite often over any work session. This homework aims to help you to discover UDP and program a UDP client that communicates with a provided UDP server.

Skimming the basics about UDP and Sockets. UDP is named *the simple multiplexer* in the Textbook by Peterson and Davie (P&D), and it certainly is a simple protocol. Along with the textbook by P&D, you can skim the following presentation to acquaint yourselves with the protocol:

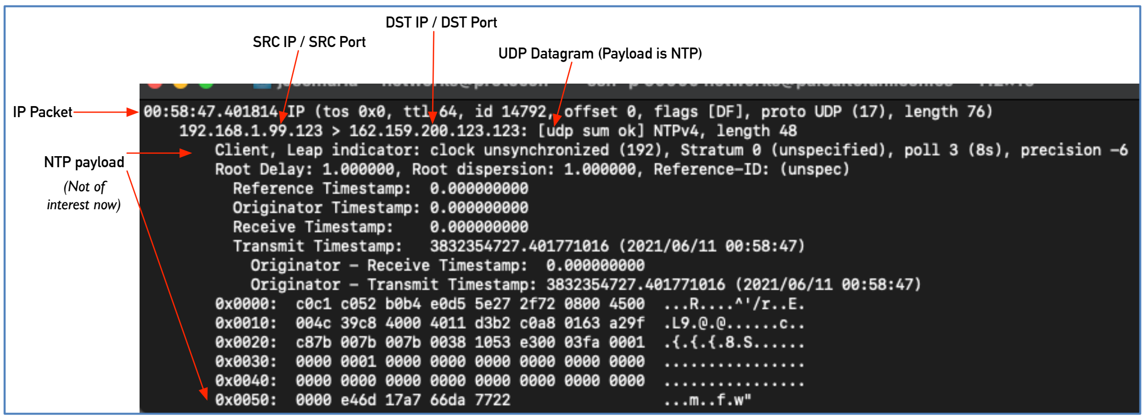
http://paloalto.unileon.es/ds/Lec/BasicTCP-1.pdf

You should start at slide no.5 and skim through slide no. 12 (Little's Law is no import at this time).

Exercise 1: NTP (Network Time Protocol) is a good example of an application protocol that uses UDP. The NTP application protocol runs on top of UDP and uses UDP port 123. When an NTP client sends a *clock synchronization request* to an NTP server, the request (An NTP request message) is encapsulated into an UDP datagram which destination port is 123. Is the source port also port 123?

We can run a tcpdump trace that captures an NTP request generated by your host and then respond to the preceding question. Execute the following command on your home computer (You should replace enp1s0 by whatever NIC name is appropriate) and let it run for a few minutes until your NTP client decides to send the NTP clock sync request. When the request is sent, you'll be able to identify the field structure of the UDP datagram that encapsulates the request, including the source and destination UDP ports used by the protocol. Observe what happened in paloalto.unileon.es in Lab B6, today:

```
$ tcpdump -XX -vvv -n udp port 123
```



```
00:58:47.401814 IP (tos 0x0, ttl 64, id 14792, offset 0, flags [DF], proto UDP (17), length 76)
    192.168.1.99.123 > 162.159.200.123.123: [udp sum ok] NTPv4, length 48
        Client, Leap indicator: clock unsynchronized (192), Stratum 0 (unspecified), poll 3 (8s), precision -6
        Root Delay: 1.000000, Root dispersion: 1.000000, Reference-ID: (unspec)
          Reference Timestamp:  0.000000000
          Originator Timestamp: 0.000000000
          Receive Timestamp:    0.000000000
          Transmit Timestamp:   3832354727.401771016 (2021/06/11 00:58:47)
            Originator - Receive Timestamp:  0.000000000
            Originator - Transmit Timestamp: 3832354727.401771016 (2021/06/11 00:58:47)
        0x0000:  c0c1 c052 b0b4 e0d5 5e27 2f72 0800 4500  ...R....^'/r..E.
        0x0010:  004c 39c8 4000 4011 d3b2 c0a8 0163 a29f  .L9.@.@......c..
        0x0020:  c87b 007b 007b 0038 1053 e300 03fa 0001  .{.{.{.8.S......
        0x0030:  0000 0001 0000 0000 0000 0000 0000 0000  ................
        0x0040:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0050:  0000 e46d 17a7 66da 7722                 ...m..f.w"
```

Figure 1. tcpdump trace of an NTP request

The NTP payload encapsulated into the UDP datagram is not of our interest at this time (It has been printed out in dark red). Observe the source and destination IP addresses and the respective ports printed out *in blue*. Now you can tell for sure that the NTP client sent an NTP request to the server using source port and destination port 123.

If your operating system is taking a long time to send the NTP clock sync request, install the ntpdate command (# apt install ntpdate) and execute it while you have the tcpdump trace running in a different window:

```
# ntpdate -s es.pool.ntp.org
```

Do this exercise at your own computer, and include the UDP datagram containing the NTP request and identify its *five* fields: Src port, dst port, checksum, length and payload. If you need more information in the trace, search for convenient options of tcpdump that might be of help.

Exercise 2. Checking that the server is running in paloalto.unileon.es. In this exercise we provide you with the source code to a UDP server program that expects a string containing a message sent by the client. This message is echoed back to the client along with a 4-byte integer that represents the number of requests received by the server so far. We wish to observe that the bytes comprising it must be ordered in a specific ordering known as Network Byte order. The client should translate the received integer into the format employed by the host architecture for representing integers, which may be one of Little Endian (Intel Architecture) or Big Endian (ARM and many others). These translations usually are enforced by using the right call from the ntohs() group of calls. The man page of ntohs() will lead you to the other three relevant calls and the provided server source code uses one of them.

The server program must be running in paloalto.unileon.es UDP port 60001. For the time being, let's check that we can communicate with it without any programming. We'll use the nc (netcat) program that we used in practice 1:

```
$ nc -u paloalto.unileon.es 60001
Hello! This is a CN practice!
Hello! This is a CN practice!
```

I typed the first message "`Hello! This is a CN practice!`". The second one is the result of the server sending me the echo back. In conclusion, the server is actually running. Now, check that you can reproduce the same result in your on PC and include your results below. Run the following tcpdump command so that you capture the UDP traffic, and remember that you may have to switch to super-user or use sudo directly. The datagram per-se is printed in blue:

```
$ tcpdump -i enp1s0 -XX -vvv -n udp port 60001

tcpdump: listening on enp1s0, link-type EN10MB (Ethernet), capture size 262144 bytes

19:16:17.059405 IP (tos 0x0, ttl 64, id 16353, offset 0, flags [DF], proto UDP (17),
length 58)
     192.168.1.99.34495 > 193.146.101.46.60001: [udp sum ok] UDP, length 30
      0x0000:  c0c1 c052 b0b4 e0d5 5e27 2f72 0800 4500  ...R....^'/r..E.
      0x0010:  003a 3fe1 4000 4011 1206 c0a8 0163 c192  .:?.@.@......c..
      0x0020:  652e 86bf ea61 0026 c2fc 4865 6c6c 6f21  e....a.&..Hello!
      0x0030:  2054 6869 7320 6973 2061 2043 4e20 7072  .This.is.a.CN.pr
      0x0040:  6163 7469 6365 210a                       actice!.

19:16:17.060023 IP (tos 0x0, ttl 64, id 21237, offset 0, flags [DF], proto UDP (17),
length 62)
     192.168.1.99.60001 > 192.168.1.1.34495: [udp sum ok] UDP, length 34
      0x0000:  c0c1 c052 b0b4 e0d5 5e27 2f72 0800 4500  ...R....^'/r..E.
      0x0010:  003e 52f5 4000 4011 6405 c0a8 0163 c0a8  .>R.@.@.d....c..
      0x0020:  0101 ea61 86bf 002a 280a 4865 6c6c 6f21  ...a...*(.Hello!
      0x0030:  2054 6869 7320 6973 2061 2043 4e20 7072  .This.is.a.CN.pr
      0x0040:  6163 7469 6365 210a 0000 0002            actice!.....
```

The first datagram that appears on the tcpdump trace corresponds to the sending of the message by nc command. The second datagram is the response provided by the server program. Observe that the server program sends back exactly the same message that was received along with the four bytes that comprise the value of a server counter that keeps a count of received requests at the moment. The sent-back message appears in red and the value of the counter provided by the server appears in melon (`0000 0002`).

Respond to these questions about the response datagram:

a. What's the source IP?

b. What's the source port?

c. What's the destination IP?

d. What's the source port?

e. What's the numeric value of the counter sent back by the server in decimal base? Explain how you calculate its value.

f. Download the source code for the server at the following link:

`http://paloalto.unileon.es/cn/Q/echoServer.c`

and explain the purpose of the following sentence in function appendCounter():

```
unsigned int c = htonl((unsigned) counter);
```

g. Assume that the server is running in your computer, what Linux IP stack management command will tell you if the UDP socket is open?

Exercise 3. **Programming a client.** After having checked that the server is running and understanding the logical structure of the response provided by it, your job consists of writing a client program that is compatible with the server. Your program should send a message like "Hello world!" and it should print out the full received response: the echoed back "Hello world!" alongside the received value of the communication counter.

I recommend that you start by studying the server source code which should be familiar to you because we have studied a few similar programs in the course on Computer Networks. Essentially, the same sockets calls used in the server may be used for the client. As is usual in CN, the Linux man should be very, very helpful. Write your client program in C and include the source code and a few screen dumps or output text explaining the tests that you have run.