V 4.5

Lab Practicals on Computer Networks and Distributed Systems

Stream Sockets programming in C and TCP/Socket states

This practical aims to illustrate basic concepts about the TCP protocol by writing in C a simple, echo C/S application. In addition, the Linux ss command and other are used to investigate socket states.

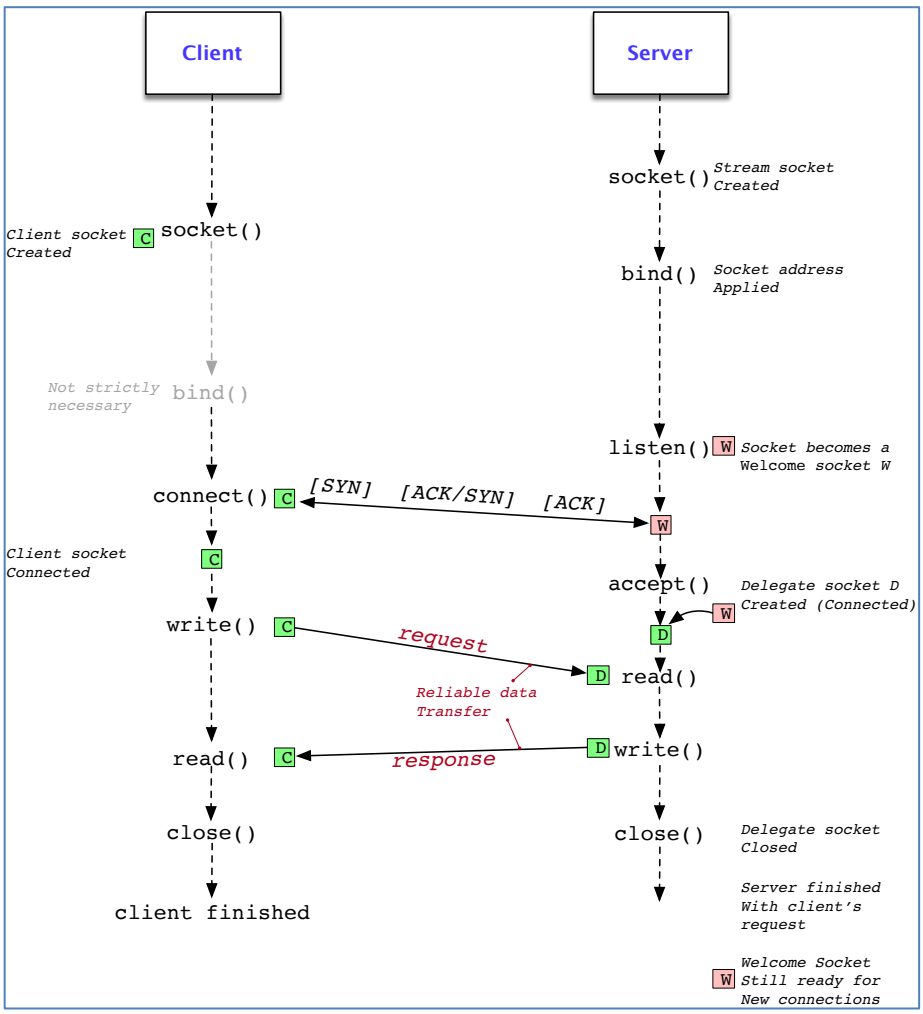


Fig. 1. TCP connection lifecycle: connection setup, data transfer and connection teardown.

Exercise 1. Programming a simple C/S protocol based on stream sockets. We provide you the C source code to a basic client and server for a simple example of a C/S application. Start by downloading, compiling and running the server in the Lab B6 host that you are using today.

a. Request a terminal and download the server's source code (Make sure that you replace the host name for **192.168.1.89** in case you are connected to the network in Lab B6):

```
[From Lab B6 net in Terminal A] $ wget http://192.168.1.89/ds/lab/TCP-Server-Base.c

(If you are out of Lab B6 net):
[From Internet]$ wget http://paloalto.unileon.es/ds/lab/TCP-Server-Base.c
```

b. Compile the server program:

```
$ gcc -o server TCP-Server-Base.c
```

c. Run the server (No need to sudo since this practical uses Stream sockets which require no particular permissions to be created)

```
[Terminal A] $ ./server 60000
Server loop restarted
```

d. Skim the server code, notice where the welcome and delegate sockets are created.

e. As yet, no TCP connection should have been created to your server, consequently when you run netstat, the only line containing TCP port number 60000 should be that that corresponds to the server socket. That line should manifest a LISTEN socket state and have a socket address of **<Your IP address; 60000>.** This line refers to your server program's Welcome Socket. Check this by executing each of the two Linux utilities in turn. Compare the results:

```
$ netstat -an --tcp

$ ss -an --tcp
```

f. Soon afterwards, we'll have a client program (nc) connect with your server from a host (H_c) other than the one in which you are doing this practical today (H_s). For the moment, run an instance of **tcpdump** that will allow you to watch the relevant TCP traffic that will cross your protocol stack, upwards or downwards, when it receives the TCP connection request from client H_c. The command line included below is fine but, you'll have to replace the interface for an interface appropriate to server H_s, in your case. As well, do su if necessary:

```
[Terminal B] $ su
...
[Terminal B] # /usr/sbin/tcpdump -i eno1 -nvvv -X tcp port 60000
```
A tcpdump trace will be printed out here that captures the TCP C/S connection lifecycle

g. In a new Linux terminal (Terminal C) at the same host you chose for your practical work, today, create an ssh connection to host H_c. Assume that the IP address of host H_c is 192.168.1.151, which is a possible IP address in Lab B6:

```
[Terminal C] $ ssh administrator@192.168.1.151
```

h. In Terminal C, at client host H_c, use the familiar nc command to connect to your server, and then type a few messages which will be sent to server H_s:

[Terminal C to Host Hc] `$ nc <IP address to host Hs> 60000`
You type this: HELLO!

i. **[Terminal B]** Interpret each of the TCP segments captured by tcpdump so far. We are particularly interested in understanding the frames comprising the two initial stages to a typical TCP connection lifecycle:

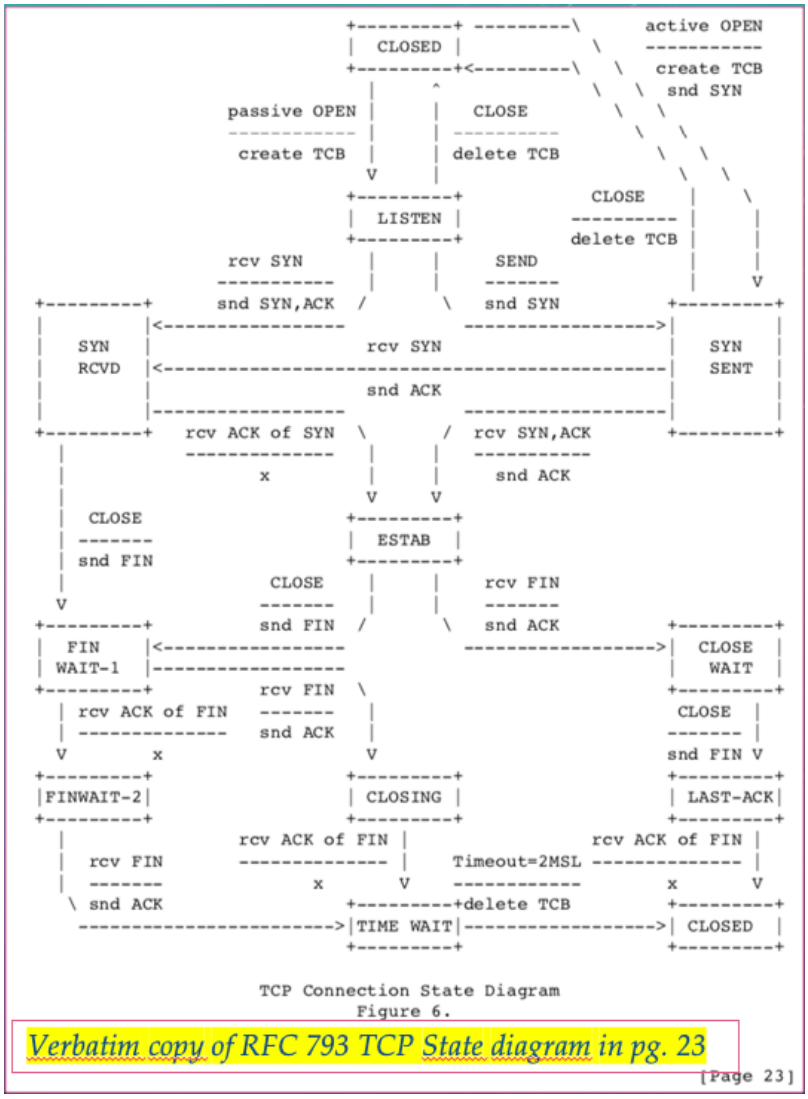
1. Connection setup
2. Bidirectional data transfer (Sliding window)

j. **[Terminal C to Host Hc]** *You type this, literally!:* `Shutdown server`

k. The message in red, above, when received by the server, will cause the connection to be closed and the **Welcome socket** to be shutdown and have the program finish, altogether. Keep the sniffer in Terminal B capturing the relevant traffic crossing the protocol stack.

l. **[Terminal B]** Interpret each of the TCP segments captured by tcpdump that belong to the connection teardown stage. Some of those segments will have the F (Finalize) flag set and, in some of them the ACK SN will be significant. We'll study the connection teardown phase with more deteail in an exercise below. For the moment, simply observe the TCP State diagram from RFC 793, in particular the variety of paths involved in the teardown phase:

3. Connection close *-this comes later.*



TCP Connection State Diagram
Figure 6.

Verbatim copy of RFC 793 TCP State diagram in pg. 23

[Page 23]

Figure 2. TCP state diagram from RFC 793

Exercise 2. Restart the server program and notice that when the server instantiates the Welcome socket, no traffic is generated, in other words, no TCP segments are sent; neither is it necessary for any TCP segments to be received at this moment. Check this to be true by observing a new tcpdump trace as you restart the server program; also, check that the Welcome Socket is successfully created after the server starts execution by issuing one of the commands below. Make sure you understand the full output produced by any of the two commands. If necessary, see the relevant man page:

```
$ netstat -an --tcp

$ ss -an --tcp
```

Exercise 3. The Sliding Window algorithm is in operation in the data transfer phase of a TCP connection. The server and client sockets are both in the ESTABLISHED STATE; the TCP state machine in Fig. 2 governs all of the data transfers performed by the client and server. You'll be able to watch those data transfers in Terminal B, where your tcpdump sniffer must be tracing the segments sent and received by the client and the server.

Now, we set out to write a client program that is compatible with the server written above. This new c/s pair will allow us to better watch the messages that they are going to exchange. We provide you the base client code in

the file below. Download that file at the client host (H_c, in Terminal C):

a. **[Terminal C to Host Hc]** `$ wget http://192.168.1.89/ds/lab/TCP-Client-Base.c`

b. **[Terminal A]** Restart the server program.

c. **[Terminal C to Host Hc]** Compile and run the program by passing it the correct IP address and port to H_s. This client program sends a message to the server that is *unknown* to it. Check this is true.

d. Observe the full TCP connection lifecycle with Wireshark. Of particular importance now is the data transfer phase. Recall, this phase is governed by the Sliding Window algorithm.

i. The client program sends the string "Hello world :-)". Check that in terminal B (tcpdump).

ii. Observe the SN field of the only segment that the string bytes will be encapsulated into. What's the length of thus payload?

iii. What is the response produced by the server to the message just received by it?

iv. Identify the segment or segments prouced by the server in response to "Hello world :-)". Particularly, we are interested in understanding the ACK SN field contents of the response segments.

v. Are the Acknowledgement numbers (ASN) produced by the receiver the expected ones according to the TCP protocol.

vi. If you notice some other interesting aspect of this c/s interaction, explain it here.

