V 4.3

Lab Practicals on Computer Networks and Distributed Systems

Stream Sockets programming in C and the Client/Server model

All rights reserved © 2014-24, José María Foces Morán & José María Foces Vivancos

This practical aims to illustrate basic concepts about the TCP protocol by writing in C a simple, echo C/S application. The C/S traffic is captured by using the tcpdump sniffer such that the TCP protocol essentials can be understood. We begin by recollecting the Client/Server computing model and the relevant service interfaces to TCP (Stream Sockets).

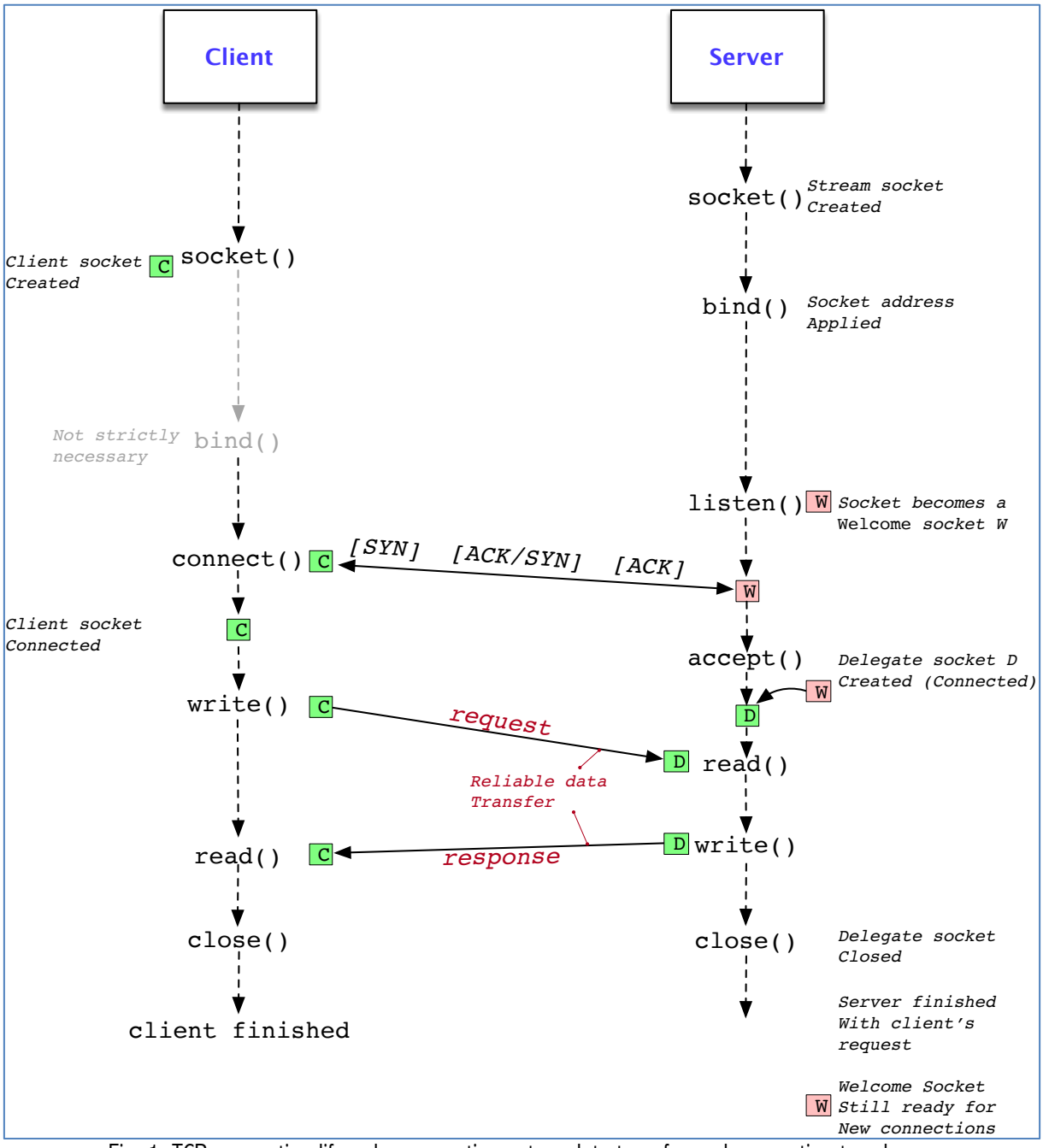

Fig. 1. TCP connection lifecycle: connection setup, data transfer and connection teardown.

All rights reserved © 2014-24, José María Foces Morán & José María Foces Vivancos

Sockets as Internet-ready names for host processes

A socket S_0 is created at the request of an application program executing in a host as a process. After a socket is created, it effectively identifies and locates the process that created it, not only within the host stack, but from throughout Internet – the socket's full address comprises as much the host's IP address as the TCP port number. Briefly, if we know the socket, then we will be able to identify the unique process that created it alongside the host where it was created. Suppose another socket S_1 is created in another host[1]. Suppose further that socket S_0 successfully creates a connection with S_1. In other words, sas yet S_0 and S_1 do have a fully functioning TCP connection. In the end, this means that the information sent between the two sockets will be *reliably* delivered at the other end, in both directions, as it were, whilst the connection is kept alive.

Socket addresses are of the essence if we are to understand the sockets API and TCP. Consequently, we wonder what a socket address is. Since a socket must identify one process (Exactly one client, or one server) within the realm of a single protocol stack, one that belongs to a specific Linux kernel, in our context. Since the host housing the stack is identified by at least one IP address throughout the Internet, a socket address must contain, at least, one of the host's IP addresses[2]. Furthermore, since that host (The client, or the server) may be running a number of processes, only one process should be pointed to by the socket at a time. Having a process be pointed to by a socket is achieved by making sure that the process obtains a single *port number* from the stack, one that will ultimately be associated with the socket. As we said above, a socket address is comprised of these two addresses:

- The host's IP address (Or any number of IP addresses out of all of the IP addresses allocated to the host stack)
- A single TCP port number from the stack's TCP

In summary, TCP socket address identifies a single process running under a given host kernel located anywhere throughout Internet.

[1] Conceptually, socket S_1 might be created within the same stack socket S_0 was created in.
[2] Actually, a socket can house all of the IP addresses that would have been applied to the stack. This semantics, *all of the IP addresses applied to this stack,* is represented by the 0.0.0.0 IP address.

All rights reserved © 2014-24, José María Foces Morán & José María Foces Vivancos

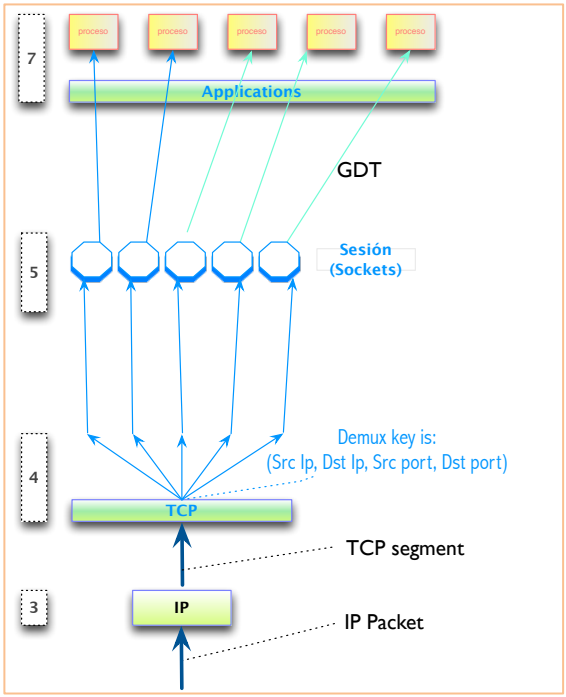

Fig. 2. Layer-3 (Transport) multiplexing keys in TCP

Following the given definition, two sockets that are created under a given kernel stack must differentiate by their TCP port numbers. These are 16-bit integers which must be a part of the multiplexing keys that operate in TCP, such that a receiving TCP is able to multiplex a received TCP segment's payload to the intended -destination- socket, and process thereby. Since TCP is a connection-oriented protocol, and since each connection involves a single client that connects to a single server, each connection must be properly identified lest one client receive the information that might have been sent to another.

Correct client-server communication entails that each client-server connection be uniquely identified. Since a TCP connection is comprised of two socket addresses (One for the server and the other for the client), a TCP connection is identified by those **two socket addresses,** which must be comprised of the following: Client IP address, client port number, Server IP address and Server port number. These four numbers are used by the client stack and by the server stack for identifying the received TCP segments at each side, which make up the c/s connection lifecycle. So as to profit by the vocabulary that we developed in the course on Computer Networks, we state that those four numbers constitute the TCP's multiplexing key.

An example of a TCP multiplexing key, one that identifies a C/S connection, operates when a web client makes a connection to a web server. The web server process must create a Server socket, or Welcome socket by following the naming convention that professors Kurose and Ross introduced in their celebrated Computer Networking textbook. A TCP Welcome Socket is a socket that is responsible exclusively for responding to the connection requests coming from clients. In the example of fig. 3, the TCP connection's multiplexing key is:

(193.146.96.163; 80; 201.1.2.3; 1200)

The web server, usually will associate the Welcome socket to port number 80, the well-known port number for accessing the web server from the Internet. Well-known ports and their corresponding service name strings are related in file /etc/services, as much in Unix as in Linux and in a similar file in Windows, located elsewhere, though. All in all, the port number applied to a socket becomes one of the four components of the TCP mux key (Src Ip, src, port, dst Ip, dst port). This turns the socket interface into a *name space* for processes within a

All rights reserved © 2014-24, José María Foces Morán & José María Foces Vivancos

system's stack; this namespace will allow client sockets to connect with Welcome sockets and thereby with their creator processes, end-to-end. Figures 5 and 6 illustrate this concept of sockets as a namespace for processes.

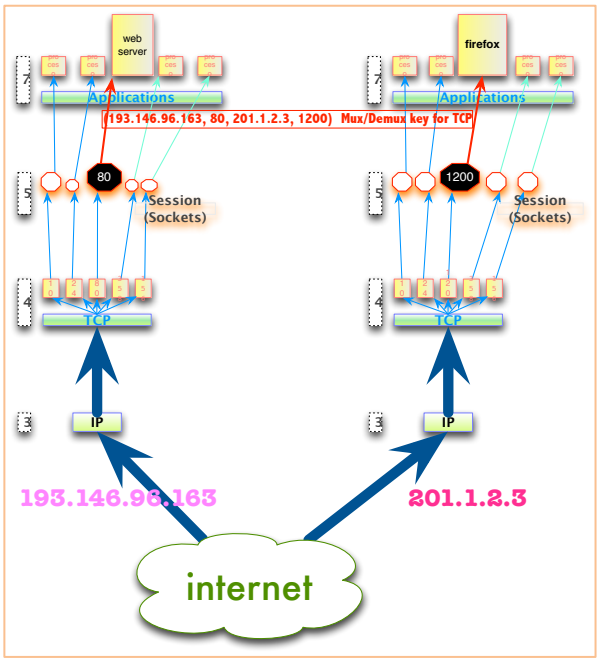

Fig. 3. Example of a multiplexing key value that identifies a Web client-to-server TCP connection

Exercise 1. The socket listing included below represents the stack state to host H from Lab B6. The snapshot listing was obtained shortly after H received a TCP connection request to port 80, which was successfully created and also used for exchanging a few C/S application messages. In this case those messages encapsulated http packets, which in the end resulted in a web page being downloaded from the server (Host H) to the client. In this exercise we aim to understand the overall interaction between the client and the server by interpreting the snapshot printed out by command netstat (No need for you to carry out the procedure explained below). Respond to each of the questions included below:

$ netstat -anc –tcp

```
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 192.168.1.210:48766     192.168.1.89:80         TIME_WAIT
tcp        0      0 192.168.1.210:22        192.168.1.89:39600      ESTABLISHED
tcp        0    172 192.168.1.210:22        192.168.1.89:51558      ESTABLISHED
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
```

The results that were received appear in the preceding listing. The line in blue and bold characters represents the c/s connection between the wget client and the web server at host 192.168.1.210 (Host H). The relevant TCP multiplexing key is underlined and, following the multiplexing key definition given above, it is comprised of these numbers: (192.168.1.210; 48766; 192.168.1.89; 80).

a. What is the state of the welcome socket that actually received the connection request from the client?

b. Check that the port number used by the welcome socket is 80.

All rights reserved © 2014-24, José María Foces Morán & José María Foces Vivancos

c. Check as well that any IP address available at the server host could have been used for accessing the server process that opened port 80 at host H.

d. Observe the TCP connection spawned by the welcome socket as a result of its receiving the remote connection request from the client. What is its state, at the precise moment when the listing was printed out?

e. Notice that the socket that at H that carries out the message interchanges from H to the client and back is technically known as *delegate socket*. What is the socket address of the delegate socket, that is, on H's side? Write down here its IP address plus its port number.

f. Notice also that the full multiplexing key will be comprised of two socket addresses, one from the client side and the other from the server (Host H) side. Write down the full multiplexing key by remarking the fact that it is comprised of two socket addresses, each on one end of the connection.

Exercise 2. It is time that we redo the preceding exercise at our *specific* host H_c (Client). Make a connection to web server H_s (192.168.1.89) at Lab B6 local net and observe its multiplexing key. In terminal A, Run the netstat Linux program to obtain a continuous listing of your stack's TCP sockets and their states. In another terminal (B), create a web connection to Lab B6 host 192.168.1.89 by using the wget program (This same host, internal to Lab B6, can be accessed from Internet via host paloalto.unileon.es).

```
[Terminal A] $ wget 192.168.1.89
paloalto's index.html file is downloaded and it is stored in your current directory

[Terminal B] $ netstat -anc --tcp
Capture the stack listing resulting from the preceding command which you will
interpret in the outline below
```

a. Write down the full multiplexing key that you have observed as your connection to the web server progressed.

b. What is the state of the welcome socket relevant to this exercise.

c. Check that the port number used by welcome socket is 80.

d. What is socket address to the delegate socket resulting from the TCP connection made from the client (H_s) to the server (H_c).

e. What is the state to the delegate socket at the moment when the netstat snapshot was taken?

All rights reserved © 2014-24, José María Foces Morán & José María Foces Vivancos