(**V.0.6**, 30 Nov 2017)

Practical Exercises in Computer Networks and Distributed Systems

Stream Sockets and the Client/Server model (C language, WIP)

This practical illustrates basic concepts TCP protocol by creating a simple C/S pair and analyzing its behavior with the Wireshark protocol analyzer. Also, I introduce the Client/Server computing model and the relevant APIs such as Sockets and undertake the structured design of a simple, Socket-based C server program and the corresponding Client program.

A brief recollection of TCP/IP and Berkeley sockets

The TCP protocol module in a computer system of today offers some essential services to user programs such as reliable transmission, flow and congestion control. This variety of objectives makes TCP complex, thus, in order to isolate the application programmer from the complexity of TCP, programming languages incorporate APIs that allow programs to communicate via TCP. In the Java language, our language of choice for this course, offers Sockets, an API derived from the original C-language Berkeley socket API (Application Programming Interface).

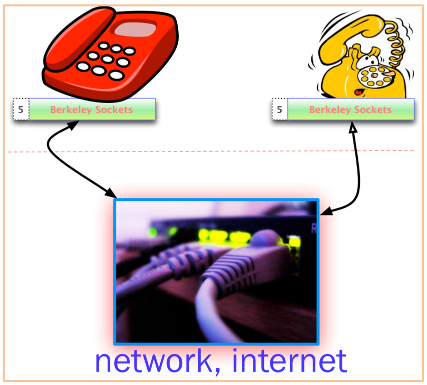

Fig. 1. TCP Connection analogy to telephone dialing

TCP is connection-oriented, that means that every time we want to transmit information reliably by using it, the protocol module will have to establish a connection with the remote computer, the result of this connection is that the communicating computers share state, a series of data structures that permit the execution of the sliding window, flow control and congestion control algorithms of TCP.

The Berkeley Socket API, originally programmed in C has been exported to other languages where the similarities are clear, this was so that interoperability could be guaranteed, for example, when a socket-based server program written in Java communicates with a client written in Python, we want both programs to interoperate correctly. The programmer's task in this case consists of implementing the application-level protocol correctly by sending the relevant protocol messages over the client and server sockets. The whole process, from connection setup through connection teardown is illustrated in Fig. 2.

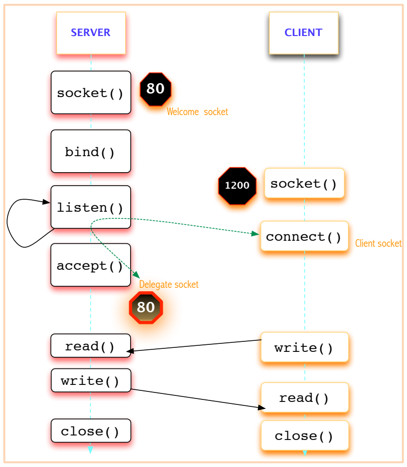

Fig. 2. Berkeley socket connection, communication and teardown

The TCP connection protocol, known as *three-way handshake*, is initiated when the client program creates a client socket and connects it with a listening server socket that is accepting connections. The three-way handshake implies the exchange of three messages between client and server and is normally started by the client in what is known as an *active open.* When this process has finished, the server socket and the client socket can begin to exchange app-protocol messages reliably.

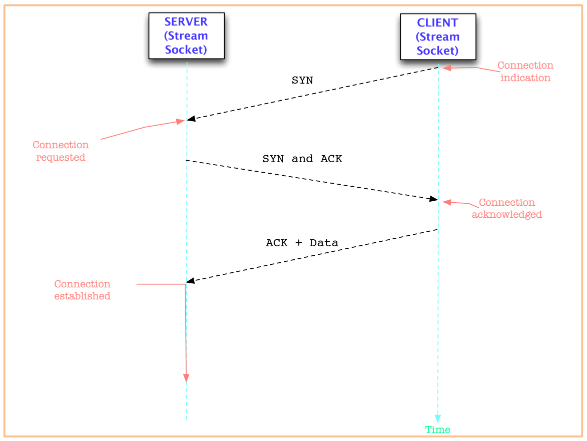

Fig. 3. Three-way handshake started by the client (Active open)

The socket created by the server program waits for connection requests coming from clients, when a new connection request is received and is successfully completed, this *welcome socket* generates a new socket (The *delegate socket*) which purpose consists of carrying out the bidirectional data transfers with the client socket. The welcome socket is forever waiting for new connection requests.

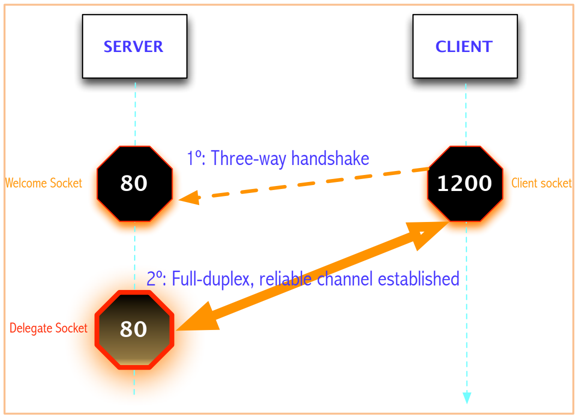

Fig. 3. After the three-way handshake, the delegate socket and the client socket begin the data transfer

Sockets as names of processes

A socket is created at the request of an application program executing as a process, such that, once a socket is created, it effectively identifies the program that created it, that is, if we know the socket we will be able to identify the unique process that created it. This process is the receiving end of the information conveyed through the socket. Since a socket gives a process a *name,* what is, then, the difference between any two sockets?

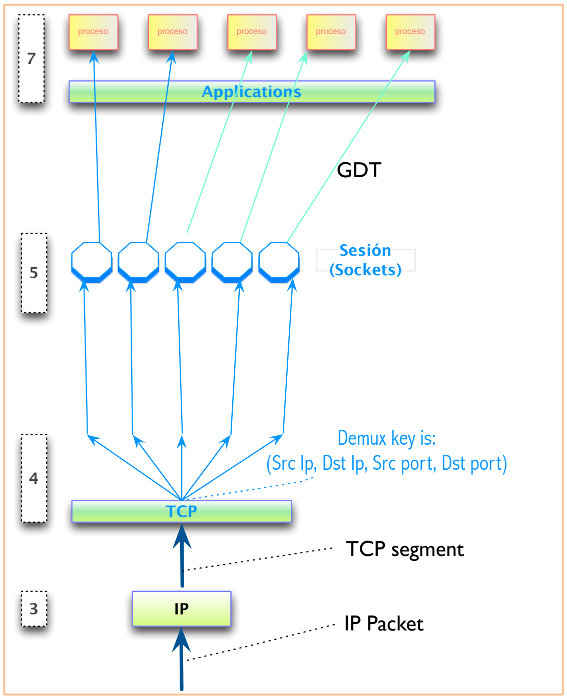

Fig. 5. Layer-4 demultiplexing keys in TCP

Two sockets within an operating system differentiate by their TCP port numbers, these are 16-bit integers constitute one of the components of the multiplexing keys at the transport layer, TCP. For example, when a web server performs a *passive open* to establish its Welcome Socket it specifies port number 80, the well-known (Standardized) port number for accessing the web server from the internet. Well-known ports and their corresponding services are related in file /etc/services in Unix and Linux and in a similar file in Windows. All in all, the port number applied to a socket is one of the four components of the TCP mux key (Src Ip, src, port, dst Ip, dst port) turning the socket interface into a *name space* for processes within a system, this namespace will allow client sockets to connect with server sockets and thereby with their creator processes, end-to-end. Figures 5 and 6 illustrate this concept of sockets as a namespace for processes.

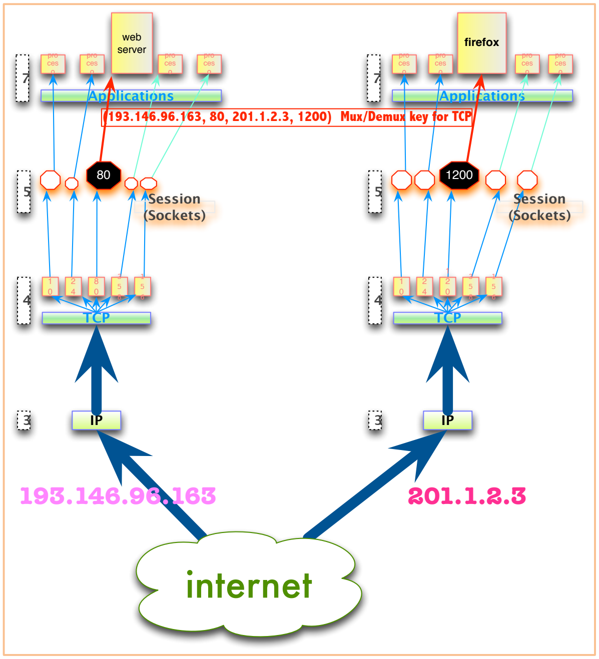

Fig. 4. Example of a demultiplexing key value that identifies a Web client-to-server TCP connection

A simple Stream-Socket Server

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

1. Request a terminal and download the server's source code:

```
$ wget http://paloalto.unileon.es/ds/Lab/TCP/TCP-Server.c
```

2. Compile server program:

```
$ gcc –o server TCP-Server.c
```

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

```
$ ./server
Server tight loop restarted
```

4. Have a class mate at a host other than yours run the $ nc command. This command allows you to connect with a server (TCP or UDP) and send it character strings; it results handy when checking simple clients and servers. It can also function as a server. Before running $ nc, you'll have to find out your IP address (The TCP port is number 50001 as included in the server program):

   ```
   $ nc <server IP address> 50001
   ```
   (You can type any string you want here; read the server's source code and test all the possibilities)

5. Now, send the "CERRAR" string to the server; it will close the server socket and end the program execution

6. Now, run Wireshark on the active network interface (Run $ ifconfig and choose an interface that has an IP)

7. When Wireshark is running, apply a display filter that suppresses all frames but those belonging to the flow between our Client and Server (Capture this filter string within the Filter textbox: `tcp.port == 50001`)

8. Run the server again:
   · Notice that when the server socket (Welcome socket) is created, no traffic is generated

9. With the server active, connect to it with $nc like above and observe the three-way handshake
   · Observe SYN segment from C to S
   · Does this segment contain any options (MSS, for example). MSS is the Maximum Segment Size.
   · Any other options contained in the the first segment sent by client?
   · Observe the response form the server; take note of any options. Observe whether the ACK number is consistent with the received Initial Sequence Number.
   · Finally, you'll see an ACK segment from C to S. When that segment arrives at S, the connection is in the ESTABLISHED state (C and S are in the ESTABLISHED state).

10. The Sliding Window (ESTBLISHED STATE) will govern all the transfers between C/S and you'll be able to watch them in your Wireshark

    · Check that the sequence numbers and acknowledgement numbers are the expected ones according to the TCP protocol

11. Finish the server by connecting to it with $ nc command and then sending the "CERRAR" string.
    · Carefully observe the segments exchanged by C and S upon connection close

12. Write a client program using the Sockets API provided above and the server program written in C
    · You may want to download a base client program which is commented but incomplete:

    ```
    $ wget http://paloalto.unileon.es/ds/Lab/TCP/TCP-Client.c
    ```

    · Once you have finished it, compile it and run it and check the application protocol with Wireshark like we did above. There should be no substantial difference between $ nc and your client program when connecting with the server.