Network

Client / Server communication study

Tom Moulard (16920041)

2017 June 12

Contents

1 Introduction

When studying how computers talk to each other. The best way to understand it is to see what they actually say to each others. For this, I created a small Python Program to make two computer communicate, one as a server and the other as a client sending a file to the server. We are going to see who this program works in details and how the two computers have managed to talk to each other in order to send this file.

2 Code

2.1 Actual code

```
# -*- coding: utf-8 -*-
"""
Created on Mon Jun 5 13:00:00 2017

@author: tm

The goal is to do the same as "iperf" but by using large files

Usage:
    - server : "python3 customIperf.py -s"
    - client : "python3 customIperf.py -c <server IP @> <file>"
"""

usage = """
Usage:
    - server : "python3 customIperf.py -s"
    - client : "python3 customIperf.py -c <server IP @> <file>"
"""

# Connection Stuff
import socket
import sys

# For file management

def get_constants(prefix):
    """
    Create a dictionary mapping socket module constants to their names.
    """
    return dict( (getattr(socket, n), n)
                 for n in dir(socket)
                 if n.startswith(prefix)
                 )
```

```
34
35
36  def isServer():
37      """
38      This manages the server part
39      will save by default the file in the current folder with the same name
40      as the one sent.
41      """
42      port      = 10000
43      families  = get_constants("AF_")
44      types     = get_constants("SOCK_")
45      protocols = get_constants("IPPROTO_")
46      sock      = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
47      # Bind the socket to the port
48      server_address = ("localhost", port)
49      try:
50          sock.bind(server_address)
51      except Exception as e:
52          print(e)
53      sock.listen(1)
54      print("-------------------------------------------------------------")
55      print("Server Listening on port {}".format(port))
56      print("Family {} Type: {} Protocol: {}".\
57          format(families[sock.family], types[sock.type], protocols[sock.proto]))
58      print("-------------------------------------------------------------")
59      while True:
60          # Wait for connection
61          connection, client_address = sock.accept()
62          try:
63              print("Client connected: {}".format(client_address))
64              lenFileName = int(connection.recv(512).\
65                  decode(encoding="utf-8", errors="strict"))
66              fileName    = str(connection.recv(lenFileName).\
67                  decode(encoding="utf-8", errors="strict"))
68              # Receive the data in small chunks and retransmit it
69              f = open(fileName, "w+b")
70              while True:
71                  data = connection.recv(64)
72                  # print("received \"{}\"".format([data]))
73                  f.write(data)
74                  if not data:
75                      f.close()
76                      print("EOF from {} for the file \"{}\"".\
77                          format(client_address, fileName))
78                      break
79          finally:
80              # Clean up the connection
```

```
81              connection.close()
82              print("Client disconnected")
83
84  def isClient(serverIP, fileName):
85      """
86      This manages the client part
87      serverIP should be like : (ipAdress, port)
88      """
89      # Create a TCP/IP socket
90      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
91
92      # Connect the socket to the port where the server is listening
93      server_address = (serverIP[0],  int(serverIP[1]))
94      print("connecting to {}:{}".format(serverIP[0], serverIP[1]))
95      while sock.connect_ex(server_address):
96          pass
97      print("Connection established")
98      try:
99          # sending fileName:
100         # first the len
101         sock.sendall("{:>512}".format(str(len(fileName))).\
102             encode(encoding="utf-8", errors="strict"))
103         # and then the name
104         sock.sendall("{}".format(fileName).\
105             encode(encoding="utf-8", errors="strict"))
106         # crushing data
107         file    = open(fileName, "r+b")
108         lines   = file.read()
109         file.close()
110         # Send data
111         print("Sending file")
112         # sock.sendall(fileName)
113         sock.sendall(lines)
114         print("File sent")
115     finally:
116         print("Closing Connection")
117         sock.close()
118
119
120 def main():
121     if "-s" in sys.argv:
122         isServer()
123     elif "-c" in sys.argv:
124         args = sys.argv[2]
125         ip   = ""
126         port = ""
127         pos  = 0
```

```
128         ll    = len(args)
129         while pos < ll and args[pos] != ":":
130             ip  += args[pos]
131             pos += 1
132         pos += 1
133         while pos < ll:
134             port += args[pos]
135             pos  += 1
136         isClient((ip, port), sys.argv[3])
137     else:
138         print(usage)
139         return 1
140 
141 if __name__ == "__main__":
142     main()
```

customIperf.py

2.2 Explanations

This code is really simple to understand, it initiate the connection between two computers and allow one to send a file to the other. Indeed, to make this works, we need to have two computers able to interpret python code. And we need to establish which one is going to be the server and with one the client. Once it is done, we have to run the interpreter in a terminal like this : *python3 customIperf.py -s* for the server.

For the client you should have a little more knowledge : you need to know the IP address of the server. Internal or external, it is up to you to choose depending on how these two computers can talk with each other. And then you can run this : *python3 customIperf.py -c <Server IP address> <file>*. The file can be every file on you computer, it will e stored on the server with the same name.

The code is really easy to follow, when running the code, the first thing is to determine what this computer is going to be : *-s* for server, *-c* for client.

Server

Once the computer knows that it is going to be the server it will open the pre-defined port to allow clients to connect to him and will wait for a client signal to establish the connection.

When a client tries to connect to the server the server will gather informations about the client to be able to communicate with him.

Once it is done, the server will wait to receive a number which correspond of the length of the fileName. then it will listen the fileName using it's length and will create a new file with this fileName.

After the document has been created, the server will gather all informations going through the network buffer to watch rights packets to add to the file. and when the server sees a EOF signal, it closes the document and closes the connection with the client. Now another client can connect to the server to send him another file.

```
$ python3 customIperf.py -s
------------------------------------------------------------
Server Listening on port 10000
Family AF_INET Type: SOCK_STREAM Protocol: IPPROTO_IP
------------------------------------------------------------
Client connected: ('127.0.0.1', 48120)
EOF from ('127.0.0.1', 48120) for the file "fileToTest"
Client disconnected
```

Figure 1: Server side output

client

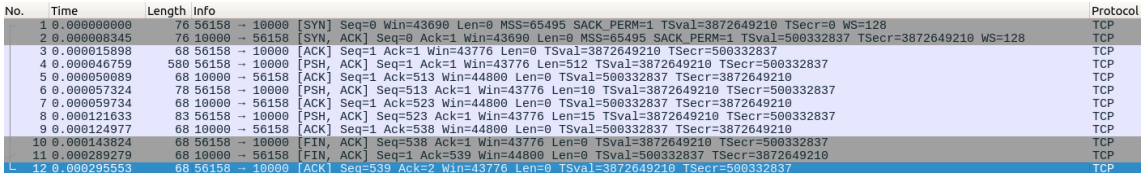
The client is pretty much the opposite of the server.
The client is going to try to establish the connection with the ip/port given by the user to the server. If there is nothing, the client will just wait for the server to be started.
when the connection is established, it will send the length of the fileName, the fileName and then the document.

```
$ python3 customIperf.py -c 127.0.0.1:10000 fileToTest
connecting to 127.0.0.1:10000
Connection established
Sending file
File sent
Closing Connection
```

Figure 2: Client side output

3 Interpretation

Now that the easy stuff is gone, we can look more in depth on what is really important here: network.



No.	Time	Length	Info	Protocol
1	0.000000000	76	56158 → 10000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=3872649210 TSecr=0 WS=128	TCP
2	0.000008345	76	10000 → 56158 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=500332837 TSecr=3872649210 WS=128	TCP
3	0.000015898	68	56158 → 10000 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=3872649210 TSecr=500332837	TCP
4	0.000046759	580	56158 → 10000 [PSH, ACK] Seq=1 Ack=1 Win=43776 Len=512 TSval=3872649210 TSecr=500332837	TCP
5	0.000050089	68	10000 → 56158 [ACK] Seq=1 Ack=513 Win=44800 Len=0 TSval=500332837 TSecr=3872649210	TCP
6	0.000057324	78	56158 → 10000 [PSH, ACK] Seq=513 Ack=1 Win=43776 Len=10 TSval=3872649210 TSecr=500332837	TCP
7	0.000059734	68	10000 → 56158 [ACK] Seq=1 Ack=523 Win=44800 Len=0 TSval=500332837 TSecr=3872649210	TCP
8	0.000121633	83	56158 → 10000 [PSH, ACK] Seq=523 Ack=1 Win=43776 Len=15 TSval=3872649210 TSecr=500332837	TCP
9	0.000124977	68	10000 → 56158 [ACK] Seq=1 Ack=538 Win=44800 Len=0 TSval=500332837 TSecr=3872649210	TCP
10	0.000143824	68	56158 → 10000 [FIN, ACK] Seq=538 Ack=1 Win=43776 Len=0 TSval=3872649210 TSecr=500332837	TCP
11	0.000289279	68	10000 → 56158 [FIN, ACK] Seq=1 Ack=539 Win=44800 Len=0 TSval=500332837 TSecr=3872649210	TCP
12	0.000295553	68	56158 → 10000 [ACK] Seq=539 Ack=2 Win=43776 Len=0 TSval=3872649210 TSecr=500332837	TCP

Figure 3: Wireshark output

This is showing to main display of Wireshark for this file sharing. We can see the different ports. Since i am running both the client and the server from my computer they all have the same ip, just the port changes : 10000 for the server and 56158 for the client.
Just by looking at it without any detail we can see three distinct part: The Authentication Part (or the Connection Establishment Part), The Data Transfer Part and The Close connection Part. Each follow a natural flow, the authentication first, the data transfer second and then closing the connection.

3.1 The Authentication Part

As we can see in the column Protocol, this connection was established in TCP which means that the authentication part was managed by a three way handshake protocol.

3.1.1 The (famous) Three Way Handshake

This protocol is implemented in three phases : the Synchronize(SYN) message, synchronize-Acknowledgment(SYN-ACK) message and the Acknowledgment(ACK) message. This protocol ensure that both client and server is synchronized and ready to communicate.

3.1.2 The Synchronize Message

This message is send by the client to ask the server if he is available for a new client to connect to him. In our case, it is the first packet on the list with the info beginning by "[SYN]" and we can see the port source is the client and the port destination is the server. We can also notice the the Seq=0 witch means that it is the first packet of this discussion.

3.1.3 The Synchronize - Acknowledgment Message

This is the response from the server to the client, as we can see on the source and destination port and on flags : "[SYN - ACK]". We can observe that the flag Seq is also null showing that it is

the server's own sequence but the ACK number is 1 witch means that the server received the first packet.

3.1.4 The Acknowledgment Message

This is the final packet send / received for the Handshake. This packet (n°3 here) have an ACK number at 1 (SEQ from the packet n°2 + 1) and the server does not need to reply for this one knowing that both client and server are ready to discuss.

3.2 The Data Transfer Part

After the communication has been established and an authentication performed, the real data transfer can begin. In my code, I specified that the length of the fileName should be less than 10$\hat{5}$12 and therefore the first packet the program send is a packet with a length of 580 : 512 of data ("len=512") and 68 of header. For the file i sent, the fileName is equal to "fileToTest" and has a length of 10 characters. Also, we can note that the ACK number follows the SEQ number of the previous packet.
Then the server replies with another packet, with a SEQ number following the last ACK one, to tell the client he received the packet and he have a god checksum, so there is no data loss.
After this, my program send the fileName encoded in UTF-8 in another packet. And the server replies that he received the packet and everything is right. Since the fileName has a length of 10, the server know he will receive a packet with a data of this length and the "len=10" characteristic can attest this.
To finish the data sending, the program sends the content of the file. the server will just receive all packet from this client and append them to the file.

3.3 The Close Connection Part

To properly finish the connection the client sends a "FIN" packet to tell the server to close the connection and the file.
The server simply respond that he got all packet and that he will close the connection.

4 Conclusion

Now we can understand how theses communication occurred to transmit a file between two computers using a TCP connection.

List of Figures