

Network

Client / Server communication study

Tom Moulard (16920041)

2017 June 15

Contents

1	Introduction	3
2	Code	3
2.1	Actual code	3
2.2	Explanations	6
3	Interpretation	9
3.1	The Authentication Part	9
3.1.1	The (famous) Three Way Handshake	9
3.1.2	The Synchronize Message	9
3.1.3	The Synchronize - Acknowledgment Message	9
3.1.4	The Acknowledgment Message	10
3.2	The Data Transfer Part	10
3.3	The Close Connection Part	10
4	Conclusion	10

1 Introduction

When studying the operation of the exchanges between two computers, the best way to understand it is to see what they are actually saying to each others. I thus 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 how this program works in details and how the computers have managed to talk to each other in order to send this file.

2 Code

2.1 Actual code

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Mon Jun 5 13:00:00 2017
4
5  @author: tm
6
7  The goal is to do the same as "iperf" but by using large files
8
9  Usage:
10     - server : "python3 customIperf.py -s [-p <port>]"
11     - client : "python3 customIperf.py -c <server IP @> <file>"
12 """
13
14 usage = """
15 Usage:
16     - server : "python3 customIperf.py -s [-p <port>]"
17     - client : "python3 customIperf.py -c <server IP @> <file>"
18 """
19
20 # Connection Stuff
21 import socket
22 import sys
23
24 # For file management
25 import os
26
27 # Miscellaneous
28 import re
29
30 def get_constants(prefix):
31     """
32     Create a dictionary mapping socket module constants to their names.
33     """
```

```
34     return dict( (getattr(socket, n), n)
35                   for n in dir(socket)
36                   if n.startswith(prefix)
37                   )
38
39
40 def isServer(port=-1):
41     """
42     This manages the server part
43     will save by default the file in the current folder with the same name
44     as the one sent.
45     """
46     if port == -1:
47         # No port set so, getting on free now
48         s = socket.socket()
49         s.bind(('localhost', 0))
50         addr, port = s.getsockname()
51         s.close()
52     families = get_constants("AF_")
53     types = get_constants("SOCK_")
54     protocols = get_constants("IPPROTO_")
55     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
56     # Bind the socket to the port
57     server_address = ("localhost", port)
58     try:
59         sock.bind(server_address)
60     except Exception as e:
61         print(e)
62     sock.listen(1)
63     print("_____")
64     print("Server Listening on port {}".format(port))
65     print("Family {} Type: {} Protocol: {}".format(families[sock.family], types[sock.type], protocols[sock.proto]))
66     print("_____")
67     while True:
68         # Wait for connection
69         connection, client_address = sock.accept()
70         try:
71             print("Client connected: {}".format(client_address))
72             lenFileName = int(connection.recv(512).\
73                               decode(encoding="utf-8", errors="strict"))
74             fileName = str(connection.recv(lenFileName).\
75                            decode(encoding="utf-8", errors="strict"))
76             # Receive the data in small chunks and retransmit it
77             f = open(fileName, "w+b")
78             while True:
79                 data = connection.recv(64)
```

```
81         # print("received {}".format([data]))
82         f.write(data)
83         if not data:
84             f.close()
85             print("EOF from {} for the file {}".format(client_address, fileName))
86             break
87
88     finally:
89         # Clean up the connection
90         connection.close()
91         print("Client disconnected")
92
93 def isClient(serverIP, fileName):
94     """
95     This manages the client part
96     serverIP should be like : (ipAddress, port)
97     """
98     # Create a TCP/IP socket
99     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
100
101     # Connect the socket to the port where the server is listening
102     server_address = (serverIP[0], int(serverIP[1]))
103     print("connecting to {}:{}".format(serverIP[0], serverIP[1]))
104     while sock.connect_ex(server_address):
105         pass
106     print("Connection established")
107     try:
108         # sending fileName:
109         # first the len
110         sock.sendall("{}>512".format(str(len(fileName))).\
111             encode(encoding="utf-8", errors="strict"))
112         # and then the name
113         sock.sendall("{}".format(fileName).\
114             encode(encoding="utf-8", errors="strict"))
115         # crushing data
116         file = open(fileName, "r+b")
117         lines = file.read()
118         file.close()
119         # Send data
120         print("Sending file")
121         # sock.sendall(fileName)
122         sock.sendall(lines)
123         print("File sent")
124     finally:
125         print("Closing Connection")
126         sock.close()
127
```

```
128
129 def main():
130     if "-s" in sys.argv:
131         port = -1
132         if "-p" in sys.argv:
133             pos, ll, found, reg = 0, len(sys.argv), False, re.compile("\d+")
134             while pos < ll and not found:
135                 tmpRes = reg.findall(sys.argv[pos])
136                 if len(tmpRes) != 0:
137                     port = int(sys.argv[pos])
138                     found = True
139                 pos += 1
140             isServer(port=port)
141         elif "-c" in sys.argv:
142             args = sys.argv[2]
143             ip = ""
144             port = ""
145             pos = 0
146             ll = len(args)
147             while pos < ll and args[pos] != ";":
148                 ip += args[pos]
149                 pos += 1
150             pos += 1
151             while pos < ll:
152                 port += args[pos]
153                 pos += 1
154             isClient((ip, port), sys.argv[3])
155         else:
156             print(usage)
157             return 1
158
159 if __name__ == "__main__":
160     main()
```

customIperf.py

2.2 Explanations

This code initiates the connection between two computers and allows one to send a file to the other. Indeed, in order to proceed, we need to have two computers able to interpret a python code. We also need to indicate 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 following this method : *python3 customIperf.py -s* for the server.

The client computer requires some more datas, you need to know the IP address of the server. Wether it is Internal or external, that is up to you to choose, depending on how these two computers can

communicate together. Afterwards 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.

When running the code, the first thing is to need to indicate the role of the computer : `-s` for server, `-c` for client.

Server

Once the computer knows that it is going to be the server it will open the setted port to allow clients to connect to him. To set the port, you can either choose a random setting, or do it manually. Then the server will wait for a client signal to establish the connection.

When a client tries to connect to the server, informations will be gathered by the server 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 observe the fileName using it's length and will create a new file with this fileName.

After that 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. When the server sees a EOF signal, it closes the document and ends the connection with the client. Now another client can connect to the server in order to send 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 quite 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 nothing is founded, 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

Also, as you can notice, I made this code an only file to make it easier to use. It makes it easier to transmit between computers and easier to change from client to server or from server to client.

In case there is an issue with the connection, python will throw an exception to tell the user what happened to help him correct the mistakes. But the only weakness of my script is when transferring a file, if something is wrong, it won't be able to detect the issue and the output file located on the server might be corrupted.

Which lead us to possibles improvement that can be done for this script. In fact, this is the base of the process of advanced and stylish file transfer like emails, WWW(World Wide Web), FTP(File Transfer Protocol), or even streaming data for instance. So there is a lot of ways to improve this script, but it depends on the usage/main characteristics we want it to have like security, compression, effectiveness, ...

3 Interpretation

Now that the easy part has been completed, we can focus more deeply on what is really important here: the 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	0.000000345	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	0.000015895	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	76	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 the main display of Wireshark for this file sharing, where we can notice the different ports. Since i am running both the client and the server from my computer they all have the same ip, but only the port changes : 10000 for the server and 56158 for the client.

Just by looking at an overview of it, we can see three distinct part: The Authentication Part (or the Connection Establishment Part), The Data Transfer Part and The Close connection Part. Each following a natural flow, the authentication first, the data transfer in second and then 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 are synchronized and that they are 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 connection. 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 which 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 the 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 which 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 actual data transfer can begin. In my code, I specified that the length of the fileName should be less than 10512 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. The fileName of the file I sent 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 that it has received the packet and that he have a good checksum, so there is no data loss.

After this, my program sends the fileName encoded in UTF-8 in another packet. And the server replies that he received the packet and everything works right. Since the fileName has a length of 10, the server is waiting to 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 end 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

1	Server side output	7
2	Client side output	7
3	Wireshark output	9