<u>Networking Intro – Assignment 2</u>
<u>Ido Zeira – 322607177</u>

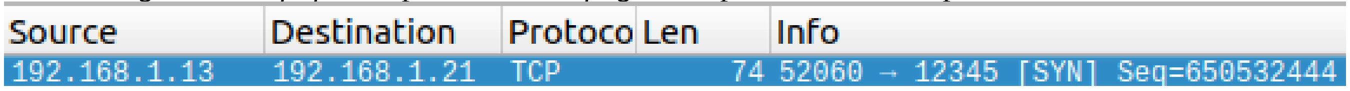
<u>Q.1</u>:

<u>Settings up the V0 communication analysis:</u>

I've started wireshark sniffing on *any* network interface, with the filter query: "tcp.port=12345", I set the destination (server) port to *12345* and the server IP to machine A's IP, started the **server.py** script on machine A (192.168.1.21) And started the **client.py** script on machine B (192.168.1.13).

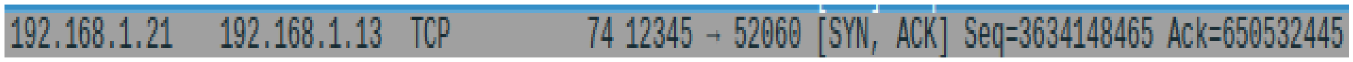
<u>Communication is as shown (See **traceV0.pcapng**):</u>

**The [Flag] in the info field represents a set flag in the packet. For example, [SYN] means SYNBit = 1.*



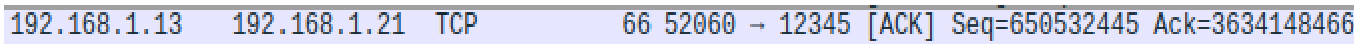
Source	Destination	Protoco	Len	Info
192.168.1.13	192.168.1.21	TCP	74	52060 → 12345 [SYN] Seq=650532444

B -> A: Syn sent to server, client requests a connection.
Seq = X (650532444).



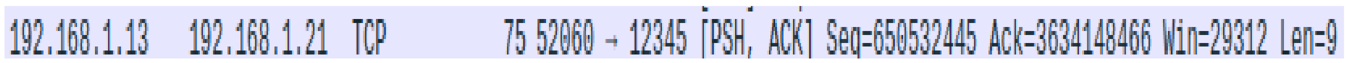
```
192.168.1.21    192.168.1.13   TCP          74 12345 → 52060 [SYN, ACK] Seq=3634148465 Ack=650532445
```

A -> B: Syn received by server, Sending SYNACK meaning the server accepts the connection request.
Seq = Y (3634148465), **Ack** = X+<u>1</u>.



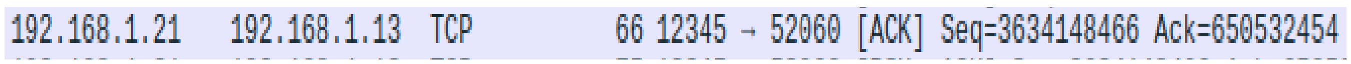
```
192.168.1.13    192.168.1.21   TCP          66 52060 → 12345 [ACK] Seq=650532445 Ack=3634148466
```

B -> A: Client acknowledges that the connection is established. (SYNRCV)
Seq = X+1, **Ack** = Y+<u>1</u>.



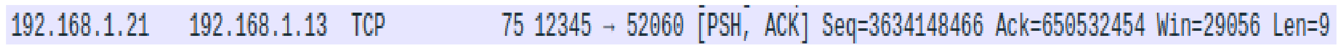
```
192.168.1.13    192.168.1.21   TCP          75 52060 → 12345 [PSH, ACK] Seq=650532445 Ack=3634148466 Win=29312 Len=9
```

B -> A: Client sends "Ido Zeira" (Len = 9).
Seq = X+1 (No change), **Ack** = Y+1 (No change).



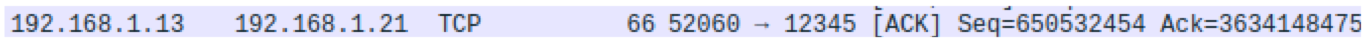
```
192.168.1.21    192.168.1.13   TCP          66 12345 → 52060 [ACK] Seq=3634148466 Ack=650532454
```

A -> B: Server acknowledges the received message.
Seq = Y+1, **Ack** = X+<u>10</u> (Added the length of "Ido Zeira" - 9).



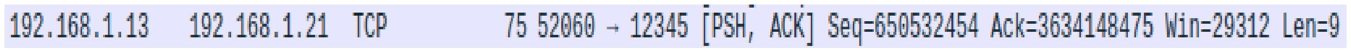
```
192.168.1.21    192.168.1.13   TCP          75 12345 → 52060 [PSH, ACK] Seq=3634148466 Ack=650532454 Win=29056 Len=9
```

A -> B: Server sends "IDO ZEIRA" (**Len** = 9).
Seq = Y+1 (No change), **Ack** = X+10 (No change).



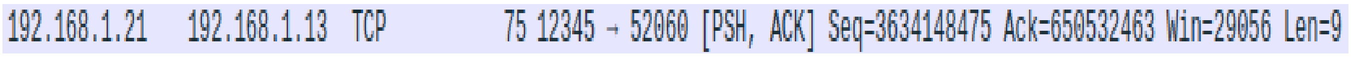
```
192.168.1.13    192.168.1.21   TCP          66 52060 → 12345 [ACK] Seq=650532454 Ack=3634148475
```

B -> A: Client acknowledges the received message.
Seq = X+10, **Ack** = Y+<u>10</u> (Added the length of "IDO ZEIRA" - 9).



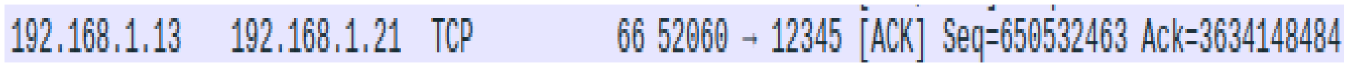
```
192.168.1.13    192.168.1.21   TCP          75 52060 → 12345 [PSH, ACK] Seq=650532454 Ack=3634148475 Win=29312 Len=9
```

B -> A: Client sends “322607177” (**Len** = 9).
Seq = X+10 (No change), **Ack** = Y+10 (No change).



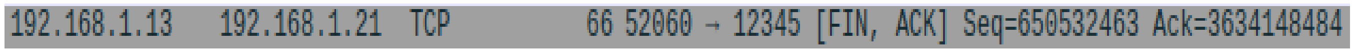
```
192.168.1.21    192.168.1.13   TCP          75 12345 → 52060 [PSH, ACK] Seq=3634148475 Ack=650532463 Win=29056 Len=9
```

A -> B: Server received the message and immediately sends “322607177” (**Len** = 9).
Seq = Y+10, **Ack** = X+19 (Added the length of “322607177” - 9).



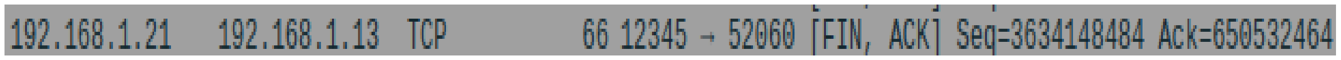
```
192.168.1.13    192.168.1.21   TCP          66 52060 → 12345 [ACK] Seq=650532463 Ack=3634148484
```

B -> A: Client acknowledges the received message.
Seq = X+19, **Ack** = Y+19 (Added the length of “322607177” - 9).



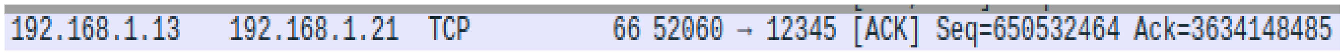
```
192.168.1.13    192.168.1.21   TCP          66 52060 → 12345 [FIN, ACK] Seq=650532463 Ack=3634148484
```

B -> A: Client sends a FIN signal to end the connection. (Ctrl + C)
Seq = X+19 (No change), **Ack** = Y+19 (No change).



```
192.168.1.21    192.168.1.13   TCP          66 12345 → 52060 [FIN, ACK] Seq=3634148484 Ack=650532464
```

A -> B: Server received signal and sends FIN back to confirm termination of session.
Seq = Y+19, **Ack** = X+20 (Added the length of FIN signal – 1).



```
192.168.1.13    192.168.1.21   TCP          66 52060 → 12345 [ACK] Seq=650532464 Ack=3634148485
```

B -> A: Client received signal and acknowledges end of connection.
Seq = X+20, **Ack** = Y+21 (Added the length of FIN signal – 1).

And the connection has been closed.

Q.2:

I’ve transported all server versions to A and all client versions to B.

V1:

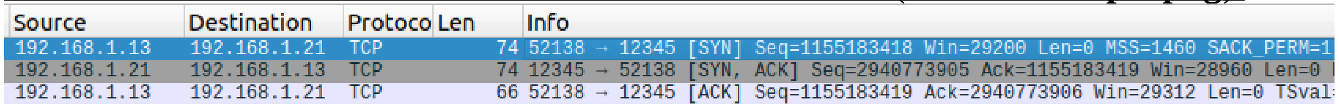
V1/server.py binds a socket to the port specified by cli argument and the 0.0.0.0 IP, similar to the s**erver.py** in Q1 (‘V0’). In line 9 we see “s.listen(1)” as opposed to “server.listen(5)” in v0 (line 7). The difference is that this time we allow up to 1 connection to be queued simultaneously instead of 5. and from there they both run the exact same way.

V1/client.py binds the socket the same as the V0 client, except that this time it is provided with server IP and port via cli argument. The main difference is that client V1 connects, sends “Hello World!”, receives the server response and disconnects, while the V0 client sends and receives data until the user sends ‘quit’.

Settings up the V1 communication analysis:

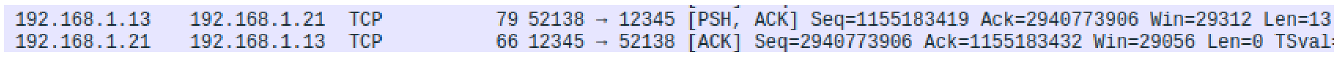
I’ve started wireshark sniffing on *any* network interface, with the filter query: “tcp.port=12345”, Started **V1/server.py** on machine A with port *12345* and then Started **V1/client.py** on machine B with server-IP *192.168.1.21* and port *12345*.

Communication between V1 Client and Server is as shown (See **traceV1.pcapng**):



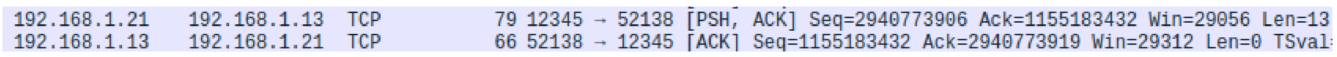
```
Source          Destination     Protoco Len  Info
192.168.1.13    192.168.1.21    TCP      74  52138 → 12345 [SYN] Seq=1155183418 Win=29200 Len=0 MSS=1460 SACK_PERM=1
192.168.1.21    192.168.1.13    TCP      74  12345 → 52138 [SYN, ACK] Seq=2940773905 Ack=1155183419 Win=28960 Len=0
192.168.1.13    192.168.1.21    TCP      66  52138 → 12345 [ACK] Seq=1155183419 Ack=2940773906 Win=29312 Len=0 TSval
```

In this part, just as before, the client sends SYN, the server ACKs the SYN and responds with a SYN which is then ACKed by the client – The connection was established.



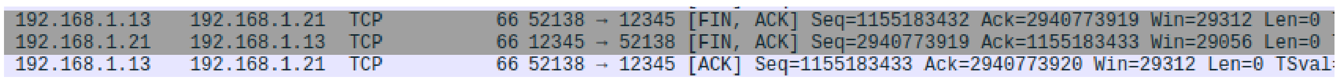
```
192.168.1.13    192.168.1.21    TCP      79  52138 → 12345 [PSH, ACK] Seq=1155183419 Ack=2940773906 Win=29312 Len=13
192.168.1.21    192.168.1.13    TCP      66  12345 → 52138 [ACK] Seq=2940773906 Ack=1155183432 Win=29056 Len=0 TSval
```

The client sends “Hello World!” (Len 13) to the server, and the server ACKs the message.



```
192.168.1.21    192.168.1.13    TCP      79  12345 → 52138 [PSH, ACK] Seq=2940773906 Ack=1155183432 Win=29056 Len=13
192.168.1.13    192.168.1.21    TCP      66  52138 → 12345 [ACK] Seq=1155183432 Ack=2940773919 Win=29312 Len=0 TSval
```

The server then sends his response “HELLO WORLD!” (Len 13) to the client, and the client ACKs it.



```
192.168.1.13    192.168.1.21    TCP      66  52138 → 12345 [FIN, ACK] Seq=1155183432 Ack=2940773919 Win=29312 Len=0
192.168.1.21    192.168.1.13    TCP      66  12345 → 52138 [FIN, ACK] Seq=2940773919 Ack=1155183433 Win=29056 Len=0
192.168.1.13    192.168.1.21    TCP      66  52138 → 12345 [ACK] Seq=1155183433 Ack=2940773920 Win=29312 Len=0 TSval
```

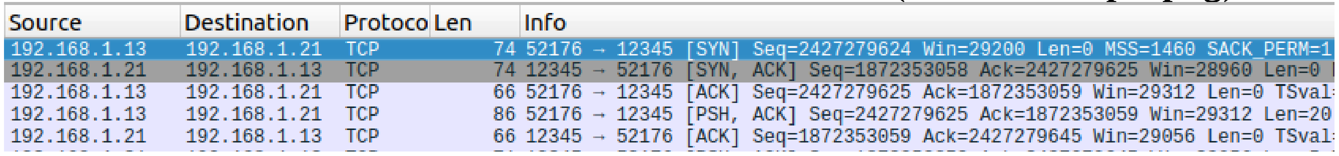
The client sends FIN, the server ACKs it and responds a FIN back, the client ACKs and the connection is closed.

<u>V2</u>:
V2/server.py is almost identical to v1, except that v2 only reads 5 characters of data each iteration, instead of 1024 in V1 and V0.

V2/client.py is almost identical to v1, except that v2 sends “World! Hello, World!” instead of “Hello World!” in V1.

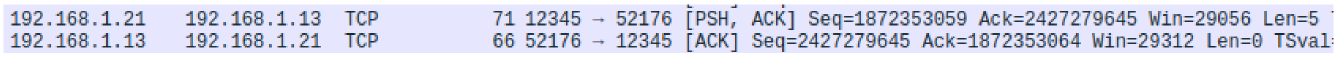
<u>Settings up the V2 communication analysis:</u>
I’ve started wireshark sniffing on *any* network interface, with the filter query: “tcp.port=12345”, Started V2/server.py on machine A with port *12345* and then Started V2/client.py on machine B with server-IP *192.168.1.21* and port *12345*.

<u>Communication between V2 Client and Server is as shown (See **traceV2.pcapng**):</u>



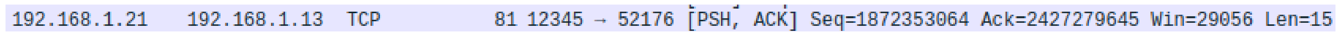
```
Source         Destination    Protoco Len  Info
192.168.1.13   192.168.1.21   TCP      74 52176 → 12345 [SYN] Seq=2427279624 Win=29200 Len=0 MSS=1460 SACK_PERM=1
192.168.1.21   192.168.1.13   TCP      74 12345 → 52176 [SYN, ACK] Seq=1872353058 Ack=2427279625 Win=28960 Len=0
192.168.1.13   192.168.1.21   TCP      66 52176 → 12345 [ACK] Seq=2427279625 Ack=1872353059 Win=29312 Len=0 TSval
192.168.1.13   192.168.1.21   TCP      86 52176 → 12345 [PSH, ACK] Seq=2427279625 Ack=1872353059 Win=29312 Len=20
192.168.1.21   192.168.1.13   TCP      66 12345 → 52176 [ACK] Seq=1872353059 Ack=2427279645 Win=29056 Len=0 TSval
```

Same as V1, a connection is established between server and client, client sends “World! Hello, World!” (Len 20) and server ACKs it.



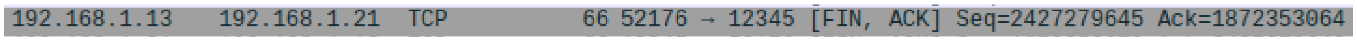
```
192.168.1.21   192.168.1.13   TCP      71 12345 → 52176 [PSH, ACK] Seq=1872353059 Ack=2427279645 Win=29056 Len=5
192.168.1.13   192.168.1.21   TCP      66 52176 → 12345 [ACK] Seq=2427279645 Ack=1872353064 Win=29312 Len=0 TSval
```

Server then captures the 5 letter prefix “World” and responds “WORLD” which is ACKed by client.



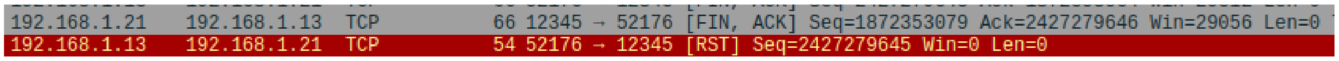
```
192.168.1.21   192.168.1.13   TCP      81 12345 → 52176 [PSH, ACK] Seq=1872353064 Ack=2427279645 Win=29056 Len=15
```

The server sends the remaining 3x5 letter responses together: “! HELLO, WORLD!”. Which were ultimately not expected by the client.



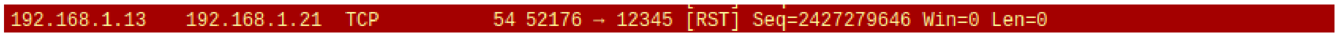
```
192.168.1.13   192.168.1.21   TCP      66 52176 → 12345 [FIN, ACK] Seq=2427279645 Ack=1872353064
```

The client sends a FIN signal to the server (and expects a FIN response with Sequence = Ack+1, because of the length of a FIN signal).



```
192.168.1.21   192.168.1.13   TCP      66 12345 → 52176 [FIN, ACK] Seq=1872353079 Ack=2427279646 Win=29056 Len=0
192.168.1.13   192.168.1.21   TCP      54 52176 → 12345 [RST] Seq=2427279645 Win=0 Len=0
```

The server then sends a FIN signal back, with a Sequence that was increased by the length of his last message’s data (+15), and the client responds that he is missing the FIN-ACK response since it didn’t mach the sequence. client expected a different sequence so he throws a RST signal to try and retrieve ‘the lost data’.



```
192.168.1.13   192.168.1.21   TCP      54 52176 → 12345 [RST] Seq=2427279646 Win=0 Len=0
```

Additionally, the client didn’t expect the FIN signal with Ack ending in 46 so he throws a RST to try and figure out the error.

V3:

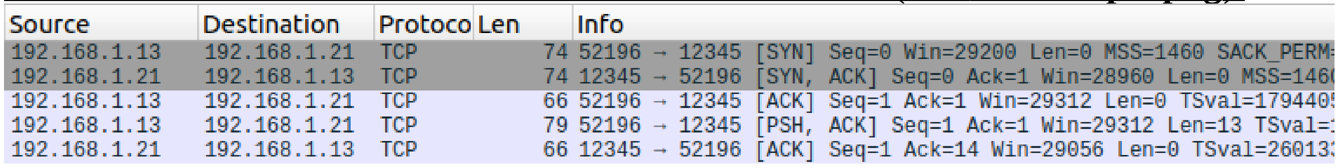
V3/server.py is almost identical to V1, except that it responds with data.upper()*1000.

V3/client.py is almost identical to V1, except that is receives the data twice.

Settings up the V3 communication analysis:

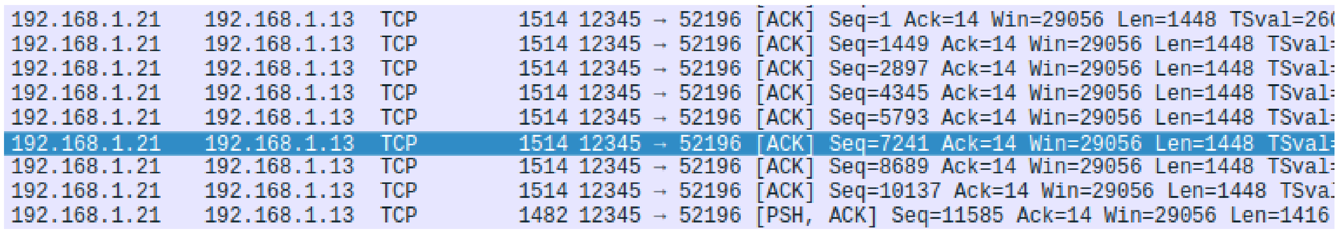
I've started wireshark sniffing on *any* network interface, with the filter query: "tcp.port=12345", Started V3/server.py on machine A with port *12345* and then Started V3/client.py on machine B with server-IP *192.168.1.21* and port *12345*.

Communication between V3 Client and Server is as shown (See **traceV3.pcapng**):



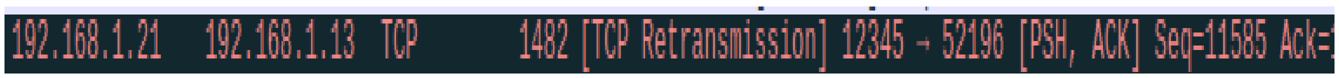
```
Source         Destination    Protoco Len    Info
192.168.1.13   192.168.1.21   TCP        74 52196 → 12345 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=
192.168.1.21   192.168.1.13   TCP        74 12345 → 52196 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460
192.168.1.13   192.168.1.21   TCP        66 52196 → 12345 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=179440
192.168.1.13   192.168.1.21   TCP        79 52196 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=13 TSval=
192.168.1.21   192.168.1.13   TCP        66 12345 → 52196 [ACK] Seq=1 Ack=14 Win=29056 Len=0 TSval=26013
```

Same as previous versions, Server and Client establish a connection, client sends "Hello World!" (Len 13), and server ACKs it.



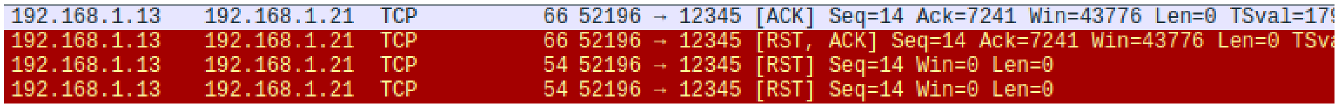
```
192.168.1.21   192.168.1.13   TCP      1514 12345 → 52196 [ACK] Seq=1 Ack=14 Win=29056 Len=1448 TSval=26
192.168.1.21   192.168.1.13   TCP      1514 12345 → 52196 [ACK] Seq=1449 Ack=14 Win=29056 Len=1448 TSval=
192.168.1.21   192.168.1.13   TCP      1514 12345 → 52196 [ACK] Seq=2897 Ack=14 Win=29056 Len=1448 TSval=
192.168.1.21   192.168.1.13   TCP      1514 12345 → 52196 [ACK] Seq=4345 Ack=14 Win=29056 Len=1448 TSval=
192.168.1.21   192.168.1.13   TCP      1514 12345 → 52196 [ACK] Seq=5793 Ack=14 Win=29056 Len=1448 TSval=
192.168.1.21   192.168.1.13   TCP      1514 12345 → 52196 [ACK] Seq=7241 Ack=14 Win=29056 Len=1448 TSval=
192.168.1.21   192.168.1.13   TCP      1514 12345 → 52196 [ACK] Seq=8689 Ack=14 Win=29056 Len=1448 TSval=
192.168.1.21   192.168.1.13   TCP      1514 12345 → 52196 [ACK] Seq=10137 Ack=14 Win=29056 Len=1448 TSva
192.168.1.21   192.168.1.13   TCP      1482 12345 → 52196 [PSH, ACK] Seq=11585 Ack=14 Win=29056 Len=1416
```

Server responds with "HELLO WORLD!"*1000, we see multiple ACKs probably because of MSS limiting the data to be 1448 each time.



```
192.168.1.21   192.168.1.13   TCP      1482 [TCP Retransmission] 12345 → 52196 [PSH, ACK] Seq=11585 Ack=
```

Because we preformed multiple ACKs with the same acknowledgment number, the server preformed a re-transmission of the packet with the smallest Seq number (from the last three tries).



```
192.168.1.13   192.168.1.21   TCP        66 52196 → 12345 [ACK] Seq=14 Ack=7241 Win=43776 Len=0 TSval=17
192.168.1.13   192.168.1.21   TCP        66 52196 → 12345 [RST, ACK] Seq=14 Ack=7241 Win=43776 Len=0 TSva
192.168.1.13   192.168.1.21   TCP        54 52196 → 12345 [RST] Seq=14 Win=0 Len=0
192.168.1.13   192.168.1.21   TCP        54 52196 → 12345 [RST] Seq=14 Win=0 Len=0
```

The Client acknowledges the packet with sequence 5793 and len 1448 (above the highlighted packet in screenshot 2), but the ACK could not be sent as another unexpected packet (Ack 7241) was received so it sent a RST.

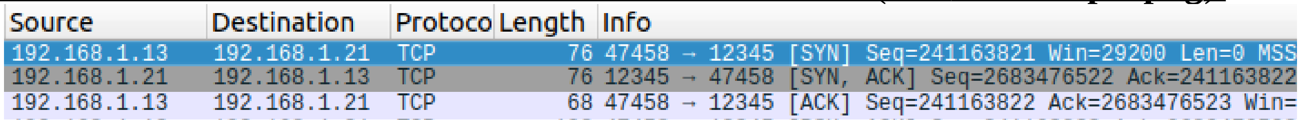
Afterwards, the client tries twice to receive a response for the RST without success and the connection is unexpectedly terminated (was reset by client).

<u>V4</u>:
V4/server.py is the almost the same as V1, except it waits 5ms between message interception tries.

V4/client.py is almost as V1, except it sends 4 messages each containing data*10 separately.

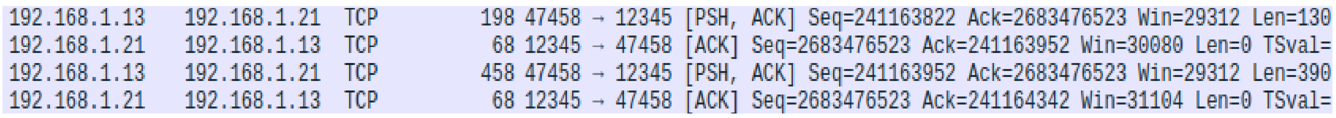
<u>Settings up the V4 communication analysis:</u>
I've started wireshark sniffing on *any* network interface, with the filter query: "tcp.port=12345", Started V4/server.py on machine A with port *12345* and then Started V4/client.py on machine B with server-IP *192.168.1.21* and port *12345*.

<u>Communication between V4 Client and Server is as shown (See **traceV4.pcapng**):</u>



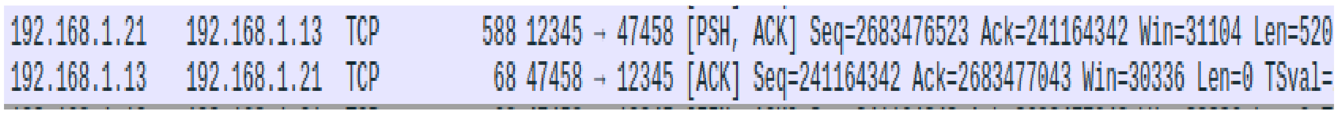
```
Source          Destination     Protoco Length  Info
192.168.1.13    192.168.1.21    TCP         76 47458 → 12345 [SYN] Seq=241163821 Win=29200 Len=0 MSS
192.168.1.21    192.168.1.13    TCP         76 12345 → 47458 [SYN, ACK] Seq=2683476522 Ack=241163822
192.168.1.13    192.168.1.21    TCP         68 47458 → 12345 [ACK] Seq=241163822 Ack=2683476523 Win=
```

The client and the server established a connection, similar to what we've seen already. (after sleeping for 5ms, Server.py is now at line 16, waiting for the first packet to be sent to it.)



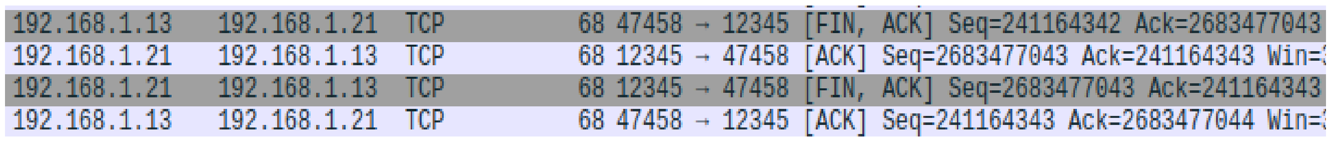
```
192.168.1.13    192.168.1.21    TCP        198 47458 → 12345 [PSH, ACK] Seq=241163822 Ack=2683476523 Win=29312 Len=130
192.168.1.21    192.168.1.13    TCP         68 12345 → 47458 [ACK] Seq=2683476523 Ack=241163952 Win=30080 Len=0 TSval=
192.168.1.13    192.168.1.21    TCP        458 47458 → 12345 [PSH, ACK] Seq=241163952 Ack=2683476523 Win=29312 Len=390
192.168.1.21    192.168.1.13    TCP         68 12345 → 47458 [ACK] Seq=2683476523 Ack=241164342 Win=31104 Len=0 TSval=
```

Client sends the first "Hello World!"*10, and the server immediately receives it and queues a response appropriately. At the same time, the client queues another 3 messages of similar content sequentially, which are afterwards, captured by the server together at the next iteration.



```
192.168.1.21    192.168.1.13    TCP        588 12345 → 47458 [PSH, ACK] Seq=2683476523 Ack=241164342 Win=31104 Len=520
192.168.1.13    192.168.1.21    TCP         68 47458 → 12345 [ACK] Seq=241164342 Ack=2683477043 Win=30336 Len=0 TSval=
```

The server then responds to all the 3 new messages as they were one and queues a message.toUpper() * 30 packet, which totals to 4 sequential packets each containing message.toUpper() *10, and since the client is yet to capture any packet sent previously, it captures them all together at client.py:14.



```
192.168.1.13    192.168.1.21    TCP         68 47458 → 12345 [FIN, ACK] Seq=241164342 Ack=2683477043
192.168.1.21    192.168.1.13    TCP         68 12345 → 47458 [ACK] Seq=2683477043 Ack=241164343 Win=
192.168.1.21    192.168.1.13    TCP         68 12345 → 47458 [FIN, ACK] Seq=2683477043 Ack=241164343
192.168.1.13    192.168.1.21    TCP         68 47458 → 12345 [ACK] Seq=241164343 Ack=2683477044 Win=
```

The client is then closing the connection, similar to previous versions, the server and the client both send FIN-ACK signals and the connection is terminated.