Lab 1 Report

UDP Server and Client

I put the IP address, Port Number, Buffer Size and Transfer Rate as my program arguments.

I check program arguments and make sure that they are appropriate (Exp: Transfer rate is not below 0).

The connections and the packages were already made so I didn't play around with them much.

I created an if condition. It checked the Transfer Rate put in in the Program Argument. If it was 0 it would not loop just send the data. I approximated sending and receiving data as 0.5 milliseconds so I made the program sleep the rest of the time until I was 1 seconds.

```
socket.send(sendPacket);
socket.receive(receivePacket);
Thread.sleep((long) (1000-0.5));
```

If Transfer Rate was more than 0 It would loop that many times and the program would sleep ((1000 ms-(0.5*TR)/TR-1(Each loop))

```
int x=Integer.valueOf(args[2]);
long processTime= (long) (0.5*timeDevider);
long startTime=System.currentTimeMillis();
while (x!=0){
    try {
        socket.send(sendPacket);
        socket.receive(receivePacket);
        Thread.sleep((1000-processTime)/timeDevider);
```

TCP Server and Client

On the Client side I have done the same stuff with the program arguments and the same checks.

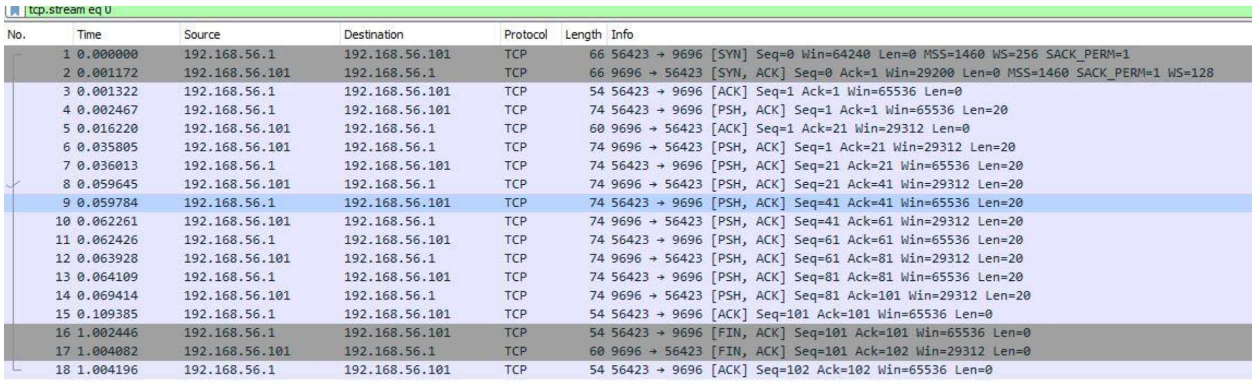
This time however in the Transfer Rate I have waited for the loop to end and took the record of the time it took and subtracted it from 1 second and waited the rest of the time.

```
long startT=System.currentTimeMillis();                          // If the Transfer rate is
more then 0 Iterate that many times
while (transferRate!=0){
    outStream.write(msg.getBytes(),0,msg.length());
    inputStream.read(buff);
    transferRate--;
}
long endt=System.currentTimeMillis()-startT;
Thread.sleep(1000-endt);
```

On the Servers side I have put a while loop which opens a new thread each time there is a new connection request.

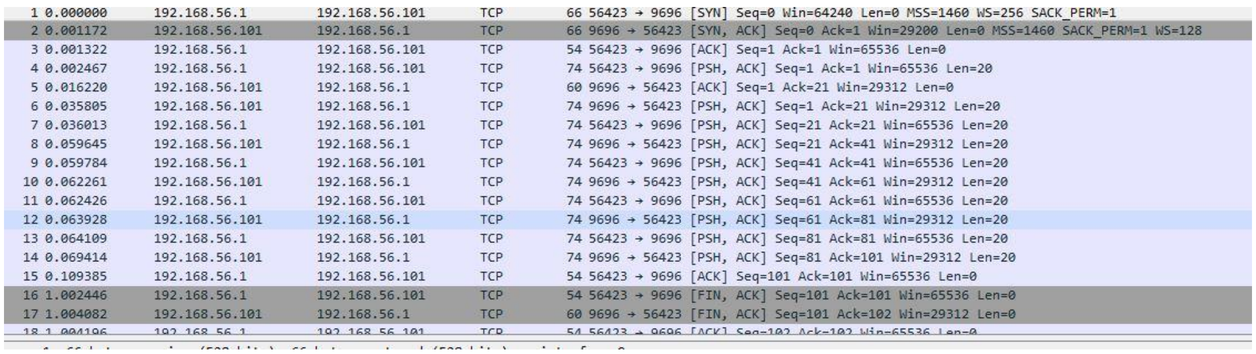
Each Thread has another while loop to handle multiple request from a client.

I have sent 5 messages back and forth:



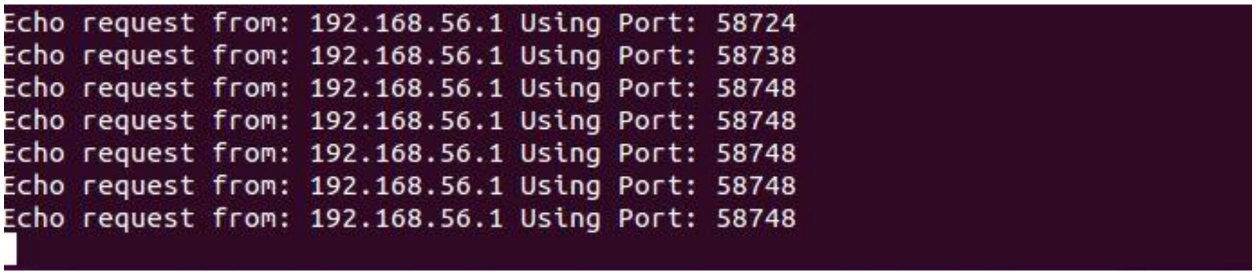
tcp.stream eq 0

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.56.1	192.168.56.101	TCP	66	56423 → 9696 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.001172	192.168.56.101	192.168.56.1	TCP	66	9696 → 56423 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
3	0.001322	192.168.56.1	192.168.56.101	TCP	54	56423 → 9696 [ACK] Seq=1 Ack=1 Win=65536 Len=0
4	0.002467	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20
5	0.016220	192.168.56.101	192.168.56.1	TCP	60	9696 → 56423 [ACK] Seq=1 Ack=21 Win=29312 Len=0
6	0.035805	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=1 Ack=21 Win=29312 Len=20
7	0.036013	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=21 Ack=21 Win=65536 Len=20
8	0.059645	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=21 Ack=41 Win=29312 Len=20
9	0.059784	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=41 Ack=41 Win=65536 Len=20
10	0.062261	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=41 Ack=61 Win=29312 Len=20
11	0.062426	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=61 Ack=61 Win=65536 Len=20
12	0.063928	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=61 Ack=81 Win=29312 Len=20
13	0.064109	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=81 Ack=81 Win=65536 Len=20
14	0.069414	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=81 Ack=101 Win=29312 Len=20
15	0.109385	192.168.56.1	192.168.56.101	TCP	54	56423 → 9696 [ACK] Seq=101 Ack=101 Win=65536 Len=0
16	1.002446	192.168.56.1	192.168.56.101	TCP	54	56423 → 9696 [FIN, ACK] Seq=101 Ack=101 Win=65536 Len=0
17	1.004082	192.168.56.101	192.168.56.1	TCP	60	9696 → 56423 [FIN, ACK] Seq=101 Ack=102 Win=29312 Len=0
18	1.004196	192.168.56.1	192.168.56.101	TCP	54	56423 → 9696 [ACK] Seq=102 Ack=102 Win=65536 Len=0



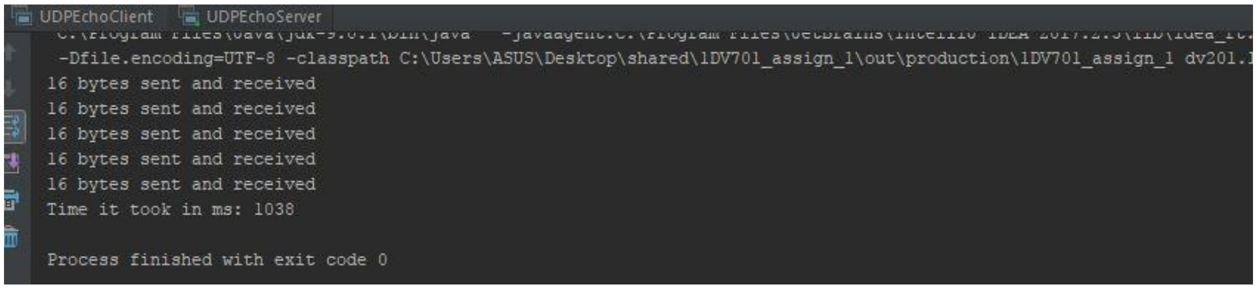
1	0.000000	192.168.56.1	192.168.56.101	TCP	66	56423 → 9696 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2	0.001172	192.168.56.101	192.168.56.1	TCP	66	9696 → 56423 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
3	0.001322	192.168.56.1	192.168.56.101	TCP	54	56423 → 9696 [ACK] Seq=1 Ack=1 Win=65536 Len=0
4	0.002467	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=20
5	0.016220	192.168.56.101	192.168.56.1	TCP	60	9696 → 56423 [ACK] Seq=1 Ack=21 Win=29312 Len=0
6	0.035805	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=1 Ack=21 Win=29312 Len=20
7	0.036013	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=21 Ack=21 Win=65536 Len=20
8	0.059645	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=21 Ack=41 Win=29312 Len=20
9	0.059784	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=41 Ack=41 Win=65536 Len=20
10	0.062261	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=41 Ack=61 Win=29312 Len=20
11	0.062426	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=61 Ack=61 Win=65536 Len=20
12	0.063928	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=61 Ack=81 Win=29312 Len=20
13	0.064109	192.168.56.1	192.168.56.101	TCP	74	56423 → 9696 [PSH, ACK] Seq=81 Ack=81 Win=65536 Len=20
14	0.069414	192.168.56.101	192.168.56.1	TCP	74	9696 → 56423 [PSH, ACK] Seq=81 Ack=101 Win=29312 Len=20
15	0.109385	192.168.56.1	192.168.56.101	TCP	54	56423 → 9696 [ACK] Seq=101 Ack=101 Win=65536 Len=0
16	1.002446	192.168.56.1	192.168.56.101	TCP	54	56423 → 9696 [FIN, ACK] Seq=101 Ack=101 Win=65536 Len=0
17	1.004082	192.168.56.101	192.168.56.1	TCP	60	9696 → 56423 [FIN, ACK] Seq=101 Ack=102 Win=29312 Len=0
18	1.004196	192.168.56.1	192.168.56.101	TCP	54	56423 → 9696 [ACK] Seq=102 Ack=102 Win=65536 Len=0

I have also sent request from 3 client to the server:



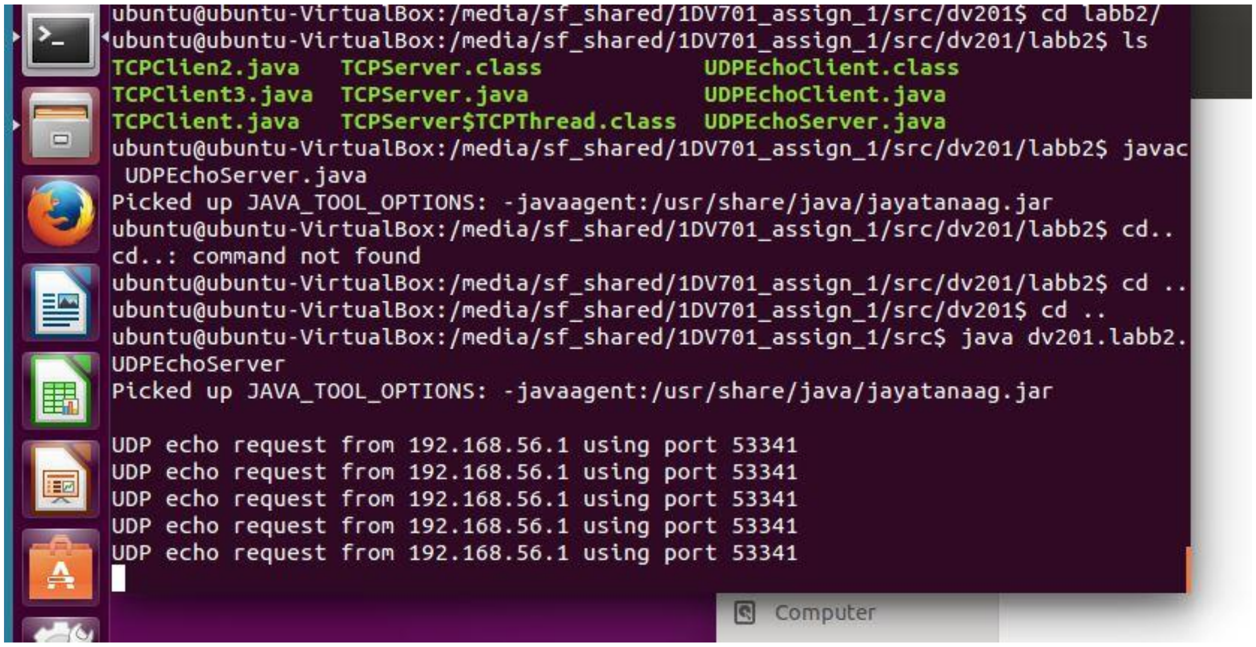
```
Echo request from: 192.168.56.1 Using Port: 58724
Echo request from: 192.168.56.1 Using Port: 58738
Echo request from: 192.168.56.1 Using Port: 58748
Echo request from: 192.168.56.1 Using Port: 58748
Echo request from: 192.168.56.1 Using Port: 58748
Echo request from: 192.168.56.1 Using Port: 58748
Echo request from: 192.168.56.1 Using Port: 58748
```

UDP Client 5 Requests:



```
 C:\Program Files\Java\jdk-9.0.1\bin\java  -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2017.2.5\lib\idea_rt.
 -Dfile.encoding=UTF-8 -classpath C:\Users\ASUS\Desktop\shared\1DV701_assign_1\out\production\1DV701_assign_1 dv201.l
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
16 bytes sent and received
Time it took in ms: 1038

Process finished with exit code 0
```

UDP Server 5 Requests:



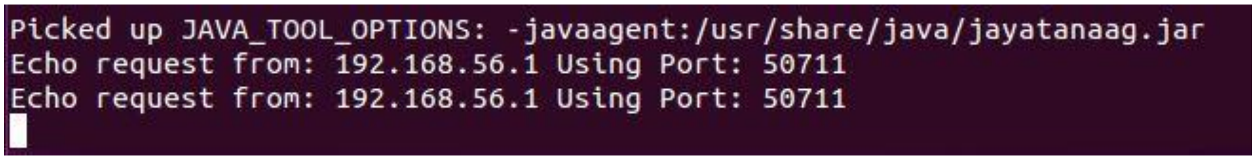
```
ubuntu@ubuntu-VirtualBox:/media/sf_shared/1DV701_assign_1/src/dv201$ cd labb2/
ubuntu@ubuntu-VirtualBox:/media/sf_shared/1DV701_assign_1/src/dv201/labb2$ ls
TCPClien2.java   TCPServer.class             UDPEchoClient.class
TCPClient3.java  TCPServer.java              UDPEchoClient.java
TCPClient.java   TCPServer$TCPThread.class   UDPEchoServer.java
ubuntu@ubuntu-VirtualBox:/media/sf_shared/1DV701_assign_1/src/dv201/labb2$ javac
 UDPEchoServer.java
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
ubuntu@ubuntu-VirtualBox:/media/sf_shared/1DV701_assign_1/src/dv201/labb2$ cd..
cd..: command not found
ubuntu@ubuntu-VirtualBox:/media/sf_shared/1DV701_assign_1/src/dv201/labb2$ cd ..
ubuntu@ubuntu-VirtualBox:/media/sf_shared/1DV701_assign_1/src/dv201$ cd ..
ubuntu@ubuntu-VirtualBox:/media/sf_shared/1DV701_assign_1/src$ java dv201.labb2.
UDPEchoServer
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar

UDP echo request from 192.168.56.1 using port 53341
UDP echo request from 192.168.56.1 using port 53341
UDP echo request from 192.168.56.1 using port 53341
UDP echo request from 192.168.56.1 using port 53341
UDP echo request from 192.168.56.1 using port 53341
```

What happens when the message is too big?

TCP:

When I decreased the size of the buffer less messages were sent and the amount of messages received were equal to the amount of messages sent.

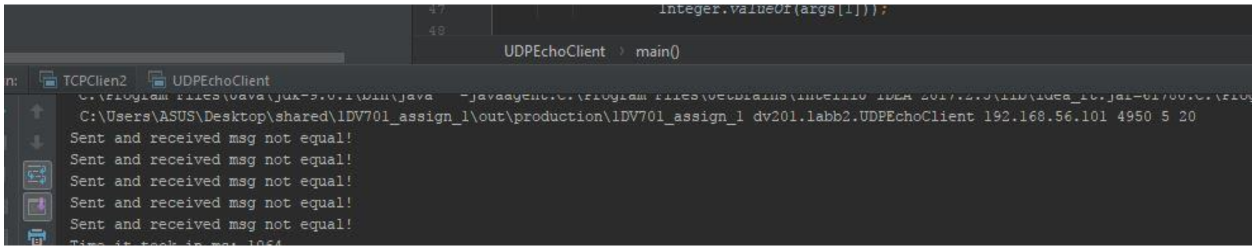
In the example below I have sent 47 byte message with the Client Buffer size of 10. There were only 2 messages sent and received.



```
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Echo request from: 192.168.56.1 Using Port: 50711
Echo request from: 192.168.56.1 Using Port: 50711
```

UDP:

When I sent a data which was bigger than the buffer size I got the same amount of packages as I have sent and the same amount of data in the packages however the full message isn't saved in the client. Also on the client side I get the following message:

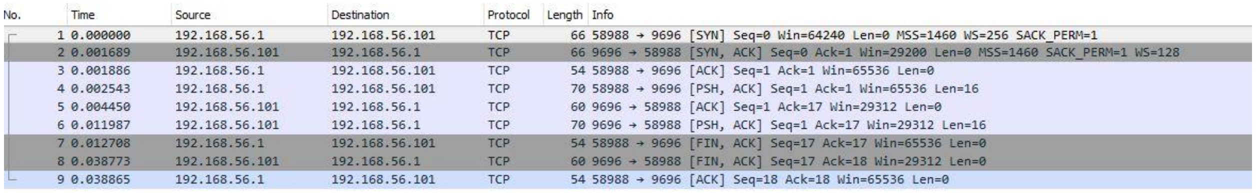


```
C:\Users\ASUS\Desktop\shared\1DV701_assign_1\out\production\1DV701_assign_1 dv201.labb2.UDPEchoClient 192.168.56.101 4950 5 20
Sent and received msg not equal!
Sent and received msg not equal!
Sent and received msg not equal!
Sent and received msg not equal!
Sent and received msg not equal!
```

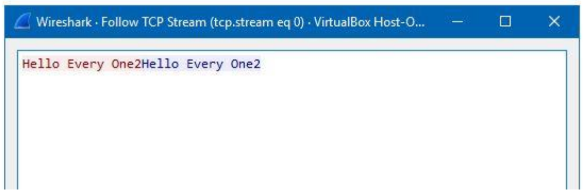
On the example above the message was 47 bytes and the Client Buffer size was 20.

Also when I sent a message bigger than the buffer size of the server the amount of packages I received was the same as the amount I had sent but the data inside was lost and I never received it back.

What is happening below?



```
No.  Time       Source          Destination     Protocol  Length Info
  1 0.000000   192.168.56.1    192.168.56.101  TCP          66 58988 → 9696 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
  2 0.001689   192.168.56.101  192.168.56.1    TCP          66 9696 → 58988 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
  3 0.001886   192.168.56.1    192.168.56.101  TCP          54 58988 → 9696 [ACK] Seq=1 Ack=1 Win=65536 Len=0
  4 0.002543   192.168.56.1    192.168.56.101  TCP          70 58988 → 9696 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=16
  5 0.004450   192.168.56.101  192.168.56.1    TCP          60 9696 → 58988 [ACK] Seq=1 Ack=17 Win=29312 Len=0
  6 0.011987   192.168.56.101  192.168.56.1    TCP          70 9696 → 58988 [PSH, ACK] Seq=1 Ack=17 Win=29312 Len=16
  7 0.012708   192.168.56.1    192.168.56.101  TCP          54 58988 → 9696 [FIN, ACK] Seq=17 Ack=17 Win=65536 Len=0
  8 0.038773   192.168.56.101  192.168.56.1    TCP          60 9696 → 58988 [FIN, ACK] Seq=17 Ack=18 Win=29312 Len=0
  9 0.038865   192.168.56.1    192.168.56.101  TCP          54 58988 → 9696 [ACK] Seq=18 Ack=18 Win=65536 Len=0
```



Wireshark · Follow TCP Stream (tcp.stream eq 0) · VirtualBox Host-O...

```
Hello Every One2Hello Every One2
```

(ACK contains the number of the next byte expected in the sequence)

1) Client has sent a Synchronization request through SYN Flag.
2) Server acknowledges that and sends back its own synchronization request through ACK and SYN flags
3) Client acknowledges the synchronization request by setting the ACK flag
4) Client send the data and also sets the ACK flag and PSH flag.
5) Server acknowledges that and sets the ACK flag.
6) Server sends back the data it has received and sets the ACK flag and PSH flag.
7) Client ends the Stream and sends the FIN flag with the ACK flag.
8) Server sends its own FIN flag and ACK flag.