

2DVA218 - LAB 3b

Group 3b:

- Fredrik Nygårds
- Oscar Einarsson
- Ragnar Winblad von Walter

Introduction

This report will describe how the program works in different scenarios. To trigger and explain unexpected behaviours we use `rand()` which can cause a packet to get lost or corrupted.

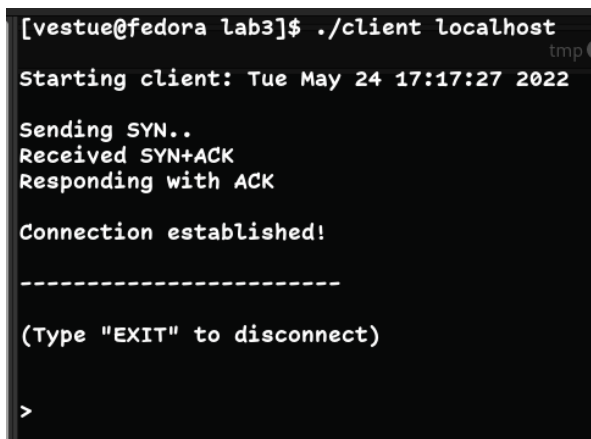
This will be shown in debug prints as “Oh nooo, ye packet is corrupt!” if the packet was turned into a corrupt packet or “Oh noo, .. is lost at seaa!” if the packet turned into a lost packet.

This behaviour can happen both client-side and server-side at all points during the program: connection, sliding window, and disconnection.

Connection setup

Client-side

Expected behaviour

A terminal window with a dark background and light-colored text. The prompt is [vestue@fedora lab3]\$ and the command is ./client localhost. The output shows the client starting, sending a SYN, receiving a SYN+ACK, and responding with an ACK. It then shows 'Connection established!' followed by a dashed line and a prompt '(Type "EXIT" to disconnect)'. A greater-than sign is at the bottom.

```
[vestue@fedora lab3]$ ./client localhost
Starting client: Tue May 24 17:17:27 2022
Sending SYN..
Received SYN+ACK
Responding with ACK
Connection established!
-----
(Type "EXIT" to disconnect)
>
```

If the client-side connection works as expected it will send a SYN then receive a SYN+ACK, reply by sending an ACK and then move to a state where the connection is established. It does not need any reply from the ACK to get to this state.

Unexpected behaviour

```
[vestue@fedora lab3]$ ./client localhost
Starting client: Tue May 24 17:23:19 2022

Sending SYN..
----- Oh nooo, SYN is lost at seaa! -----
Sending SYN..
----- Oh nooo, SYN is lost at seaa! -----
Sending SYN..
Received SYN+ACK
Responding with ACK

Connection established!

-----

(Type "EXIT" to disconnect)

>|
```

If the sent SYN gets lost it will eventually reach a timeout and get sent again. This process will repeat until the client receives a SYN+ACK.

Server-side

Expected behaviour

```
[vestue@fedora lab3]$ ./server
Starting server: Tue May 24 17:50:51 2022

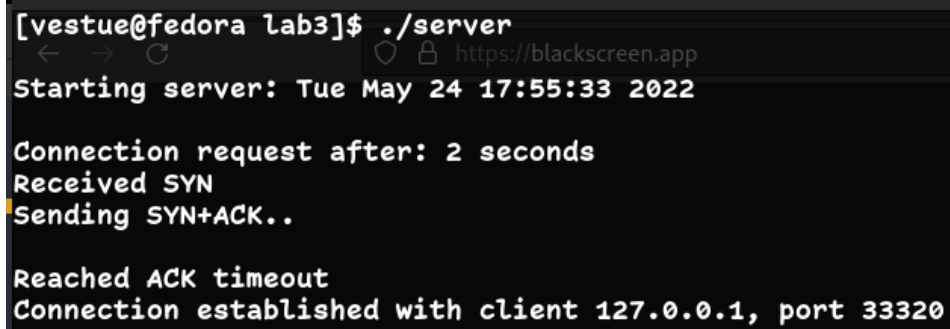
Connection request after: 2 seconds
Received SYN
Sending SYN+ACK..

Received ACK
Connection established with client 127.0.0.1, port 36380

|
```

The server responds to the SYN with a SYN+ACK. When it receives an ACK it will add the client to a dynamic array of connected clients and connection will be seen as established.

Unexpected behaviour

A terminal window with a dark background. At the top, there is a browser-like address bar with navigation icons and the URL 'https://blackscreen.app'. The terminal text shows a user running './server' in a directory named 'lab3' on a 'fedora' machine. The output indicates the server started on May 24, 2022, at 17:55:33. It then shows a 2-second delay, receipt of a SYN packet, and sending of a SYN+ACK. After reaching an ACK timeout, the connection is established with client 127.0.0.1 on port 33320.

```
[vestue@fedora lab3]$ ./server
Starting server: Tue May 24 17:55:33 2022

Connection request after: 2 seconds
Received SYN
Sending SYN+ACK..

Reached ACK timeout
Connection established with client 127.0.0.1, port 33320
```

If the SYN+ACK gets lost or gets no ACK as response it will eventually reach a timeout where it establishes connection with the client.

Selective Repeat

Client side

Expected behavior

```
Starting client: Sat May 28 11:43:05 2022
Sending SYN..
Received SYN+ACK
Responding with ACK
Connection established!
-----
(Type "EXIT" to disconnect)

>hello
Message sent at: Sat May 28 11:43:08 2022
-with SEQ(0)
-----
(Type "EXIT" to disconnect)

>Received Datagram at: Sat May 28 11:43:08 2022
-with ACK(0)
ackNum 0| baseSeq 0
-----
(Type "EXIT" to disconnect)

>
```

Client sends a message and receives an ACK, when ACK is received, base-sequence number will be moved and the timer will reset.

```

>
Message sent at: Sat May 28 11:59:12 2022
-with SEQ(5)
-----

(Type "EXIT" to disconnect)

>
Window is full!
-----

(Type "EXIT" to disconnect)

>
Window is full!
-----

(Type "EXIT" to disconnect)

>
Window is full!
-----

(Type "EXIT" to disconnect)

```

If the window gets full the client must wait for ACKs until it can send more packets.

Unexpected behavior

```

>first test
----- Oh nooo, ye packet is corrupt! -----
Message sent at: Sat May 28 12:18:40 2022
-with SEQ(0)
-----

```

A packet can get corrupted when it is sent, in this case the server will not be sending back an ACK on this package.

```

Sending timed out packet
Message sent at: Sat May 28 12:18:47 2022
-with SEQ(0)
Received Datagram at: Sat May 28 12:18:47 2022
-with ACK(0)

```

If no ACK is received then the packet will time out and it will be resent with its intended sequence number. If the re-sent packet also gets corrupted it will simply repeat this process again.

If the packet ACK-number is not the ACK-number of the base sequence we will receive the ACK but the base-sequence will not be moved

Server side

Expected behavior

```
Package corrupted!
-----
Reading from socket 5
Receiving data..
Received message:
hej'

Responding with ACK(1)
Sat May 28 10:21:30 2022
-----

Connection request after: 4987 seconds
Received SYN
Sending SYN+ACK..

Received ACK
Connection established with client 127.0.0.1, port 50183

-----
Reading from socket 6
Receiving data..
Received message:
hello

Responding with ACK(0)
Sat May 28 11:43:08 2022
-----
```

The server will respond with an ACK with the same sequence number of the received packet. Then it will wait for the next packet.

Unexpected behavior

```
-----
Reading from socket 4
Package corrupted!
-----
```

If the server receives a corrupt packet it will simply discard it and ignore sending an ACK back to the sender.

Go-back-N

Client side

Expected behaviour

```
Connection established!

-----

(Type "EXIT" to disconnect)

>Hi
Message sent at: Fri May 27 12:57:30 2022
-with SEQ(0)
-----

(Type "EXIT" to disconnect)

>
Received ACK(0)
-----
```

After the client sends a message it will get an ACK on that message and move the base sequence number up by one and reset the timer.

```
Message sent at: Fri May 27 13:01:08 2022
-with SEQ(1)
-----

(Type "EXIT" to disconnect)

>
Message sent at: Fri May 27 13:01:08 2022
-with SEQ(2)
-----

(Type "EXIT" to disconnect)

>
Window is full!
-----
```

If at any point the window gets full the client must then wait for an ACK before sending more packets.

Unexpected behaviour

If by chance a packet is corrupt when it is received by the server the client will not be getting an ACK on that packet.

```
Message sent at: Fri May 27 13:05:16 2022
-with SEQ(0)
-----

(Type "EXIT" to disconnect)

>
Received ACK(0)
-----

(Type "EXIT" to disconnect)

>
----- Oh nooo, ye packet is corrupt! -----
Message sent at: Fri May 27 13:05:17 2022
-with SEQ(1)
-----

(Type "EXIT" to disconnect)
```

As a timer is set on the base packet it will get a timeout after not receiving an ACK on the corrupt packet. It will then resend all packets that have been sent since the timer started. The intervals of re-sent packets are from base to next sequence number minus one because the next sequence is the next to be sent and has not yet been sent.

```
Sending timed out packet
Message sent at: Fri May 27 13:05:30 2022
-with SEQ(1)

Sending timed out packet
Message sent at: Fri May 27 13:05:30 2022
-with SEQ(2)

Sending timed out packet
Message sent at: Fri May 27 13:05:30 2022
-with SEQ(3)
```

After the packets have been re-sent it will hopefully get ACKs on the packets and be able to move the window. If a packet is lost on the way to the server the client will act the same as it will not be getting an ACK on the lost packet the timer will run out and all packets will be re-sent.

Server side

Expected behaviour

The server receives a non-corrupt packet, reads the packet and sends an ACK with the packet's sequence number and moves the expected sequence number up by one. It then waits for another packet.


```
Connection established with client 127.0.0.1, port 57492
```

```
-----  
Reading from socket 4  
Receiving data..  
seq: 0 | base: 0  
Received message:
```

```
Responding with ACK(0)  
Fri May 27 13:18:39 2022
```

```
-----  
Reading from socket 4  
Receiving data..  
seq: 1 | base: 1  
Received message:
```

```
Responding with ACK(1)  
Fri May 27 13:18:40 2022
```

Unexpected behaviour

If at any time the server receives a corrupt packet it will discard it and continue waiting for packets. Without sending an ACK.

```
Reading from socket 4  
Receiving data..  
seq: 4 | base: 4  
Received message:
```

```
Responding with ACK(4)  
Fri May 27 13:28:51 2022
```

```
-----  
Reading from socket 4  
Package corrupted!  
-----
```

If packets get received in the wrong order the server will discard the packet and continue waiting for the next expected packet. Without sending an ACK.

```
-----  
Reading from socket 4  
Receiving data..  
seq: 0 | base: 0  
Received message:
```

```
Responding with ACK(0)  
Fri May 27 13:26:23 2022
```

```
-----  
Reading from socket 4  
Receiving data..  
seq: 4 | base: 1  
-----
```

```
Reading from socket 4  
Receiving data..  
seq: 5 | base: 1  
-----
```

When it finally gets the expected packet it will continue as explained in the rubric Expected behaviour.

Disconnect

Sender-side

Expected behaviour

```
-----  
  
(Type "EXIT" to disconnect)  
  
>EXIT  
Sending FIN..  
Tue May 24 18:12:08 2022  
  
Received ACK  
Received FIN  
Sending ACK  
Disconnecting...  
  
Timeout reached  
Disconnected!
```

The client-side disconnect process starts by typing "EXIT". This sends a FIN to the server and then causes the client to wait for either an ACK or a FIN. If neither FIN or ACK gets received as a response the client will send another FIN. This process repeats until it gets an ACK or FIN.

Once it gets a FIN from the client it will ACK the FIN and start a timer. Once the timer runs out it will fully disconnect the client. In case the client receives another FIN it will restart this timer and send the ACK once more.

Unexpected behaviour

```
-----  
(Type "EXIT" to disconnect)  
  
>EXIT  
----- Oh nooo, FIN is lost at seaa! -----  
Sending FIN..  
Tue May 24 18:31:19 2022  
  
Sending FIN..  
Tue May 24 18:31:25 2022  
  
Received ACK  
Received FIN  
Sending ACK  
Disconnecting...  
  
Timeout reached  
Disconnected!
```

If the FIN gets lost it will keep resending FINs until it receives either an ACK or FIN.

Server-side

Expected behaviour

```
-----  
Reading from socket 4  
Receiving data..  
Received FIN!  
Responding with ACK  
  
Sending FIN..  
Received ACK  
  
Disconnected client 127.0.0.1, port 38066  
Tue May 24 18:04:10 2022  
-----
```

When the server receives a FIN request from a valid client it will first respond with an ACK and then a FIN shortly after. If the sent FIN gets responded with an ACK it will count the client as fully disconnected which removes the client from the array of clients and closes the individual client socket. If the ACK expected from the client gets lost the server will get a timeout for the sent FIN and send another FIN.

One thing that differs from the original plan in the lab 3a report is that there will be another timeout to handle the case if all ACKs get lost. If that happens it will first get a timeout to send another FIN and then get another timeout which causes the server to assume that the client has disconnected.

Unexpected behaviour

```
-----  
Reading from socket 4  
Receiving data..  
Received FIN!  
Responding with ACK  
  
Sending FIN..  
  
Sending FIN..  
Reached timeout while waiting for ACK  
  
Disconnected client 127.0.0.1, port 36631  
Tue May 24 18:44:15 2022  
-----  
^C
```

If the sent FIN gets lost it will timeout and attempt to send it again. Once it has reached a timeout that is big enough it will assume that the client has disconnected and finish the disconnection process by removing the client from the dynamic array of clients and then close the socket.