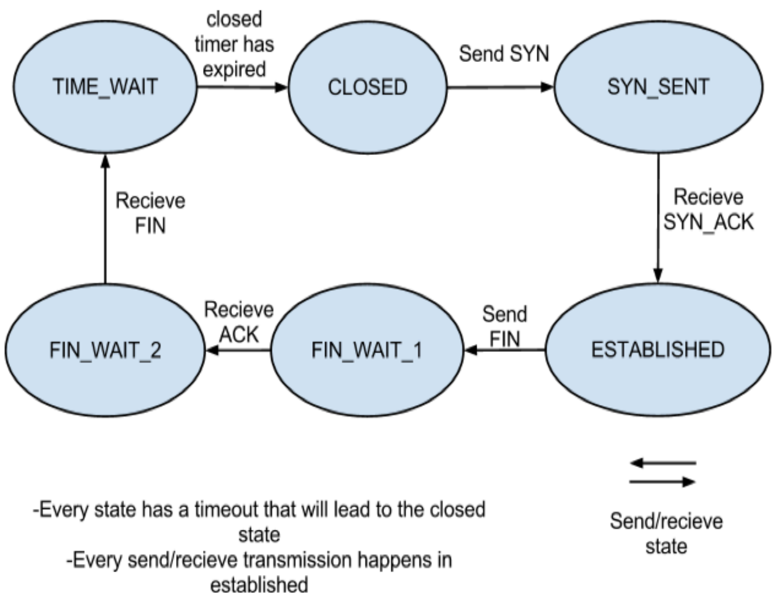
KTN1

GROUP 38

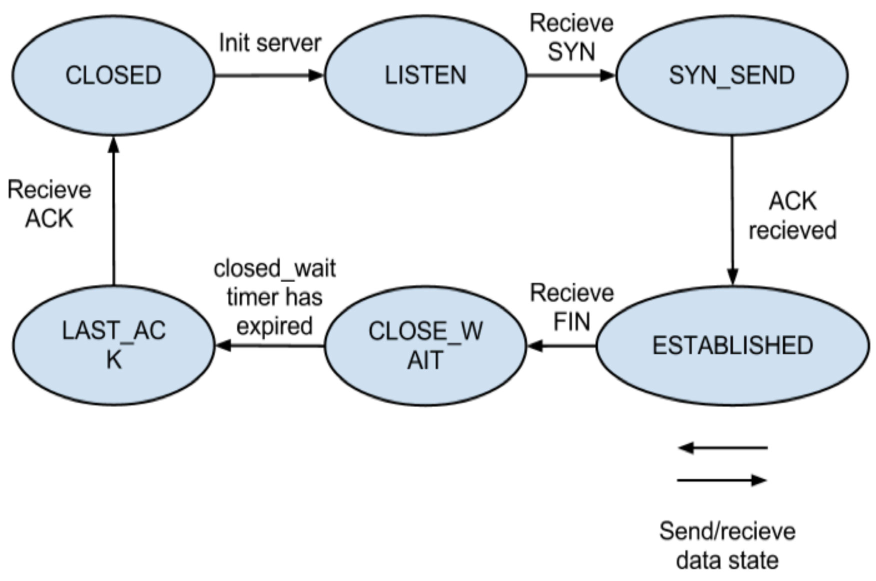
The following document contains detailed information on how we are going to implement the KTN specific part of the project.

State diagram of A1

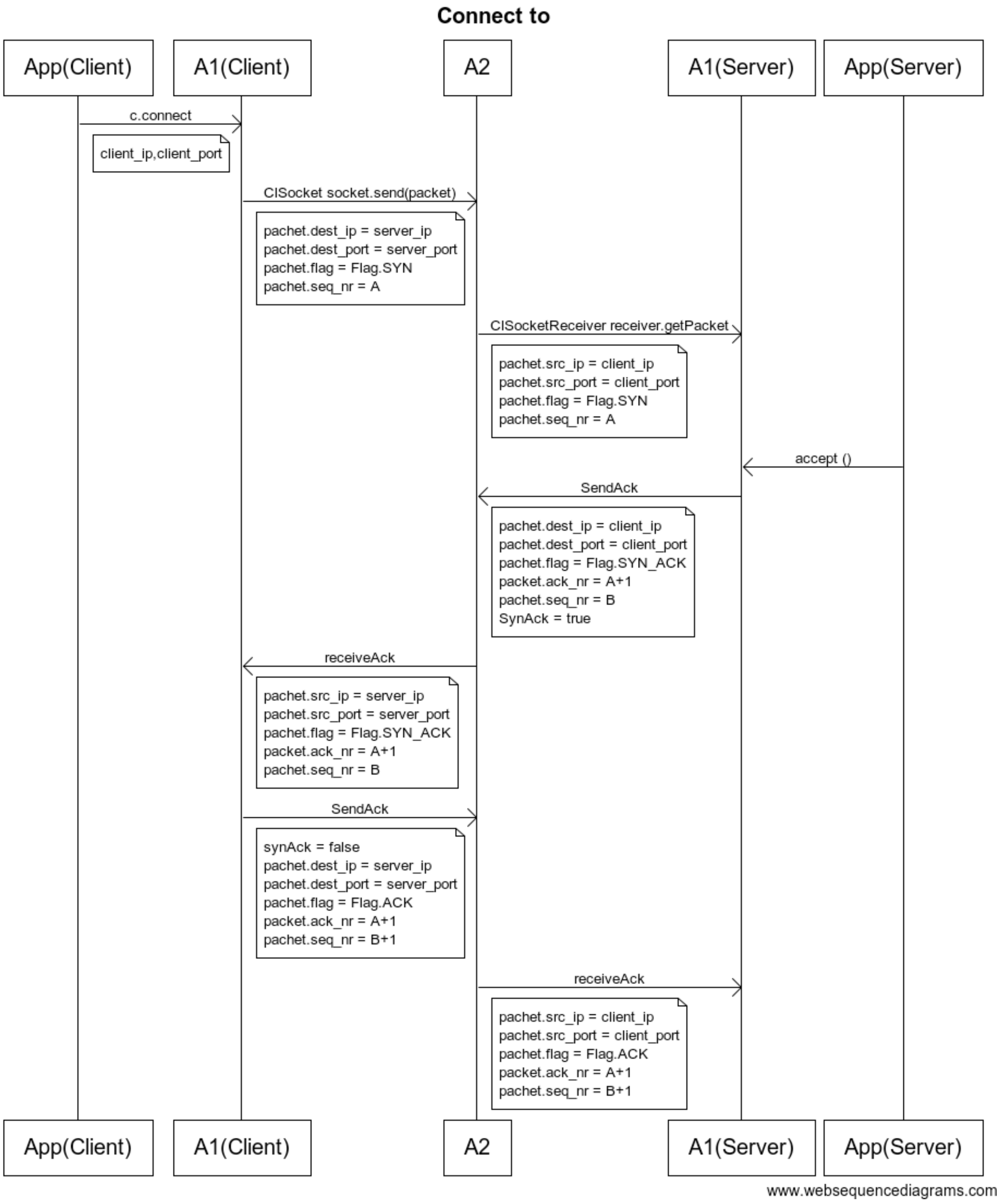
Client state diagram



Server state diagram

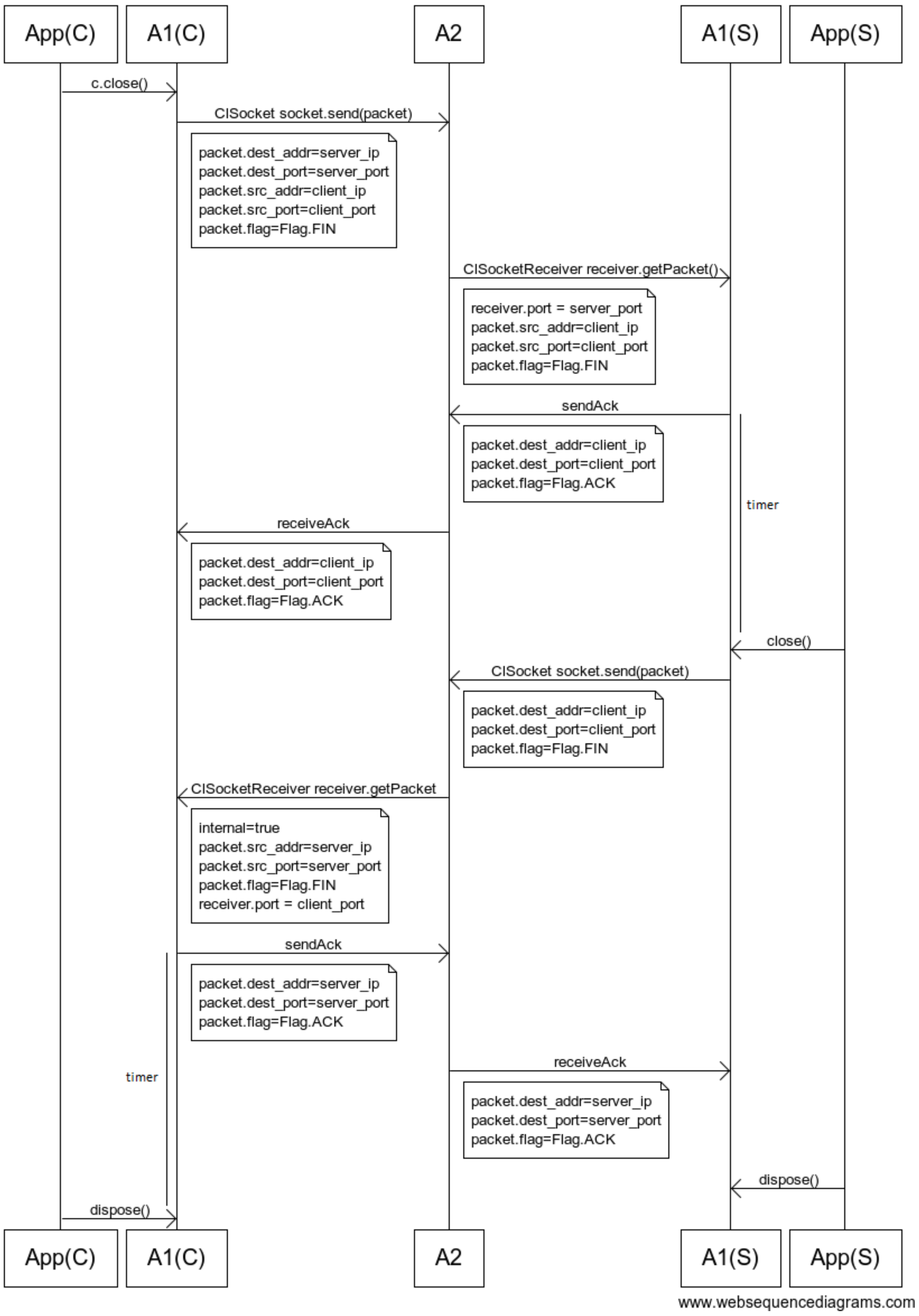


Sequence diagrams



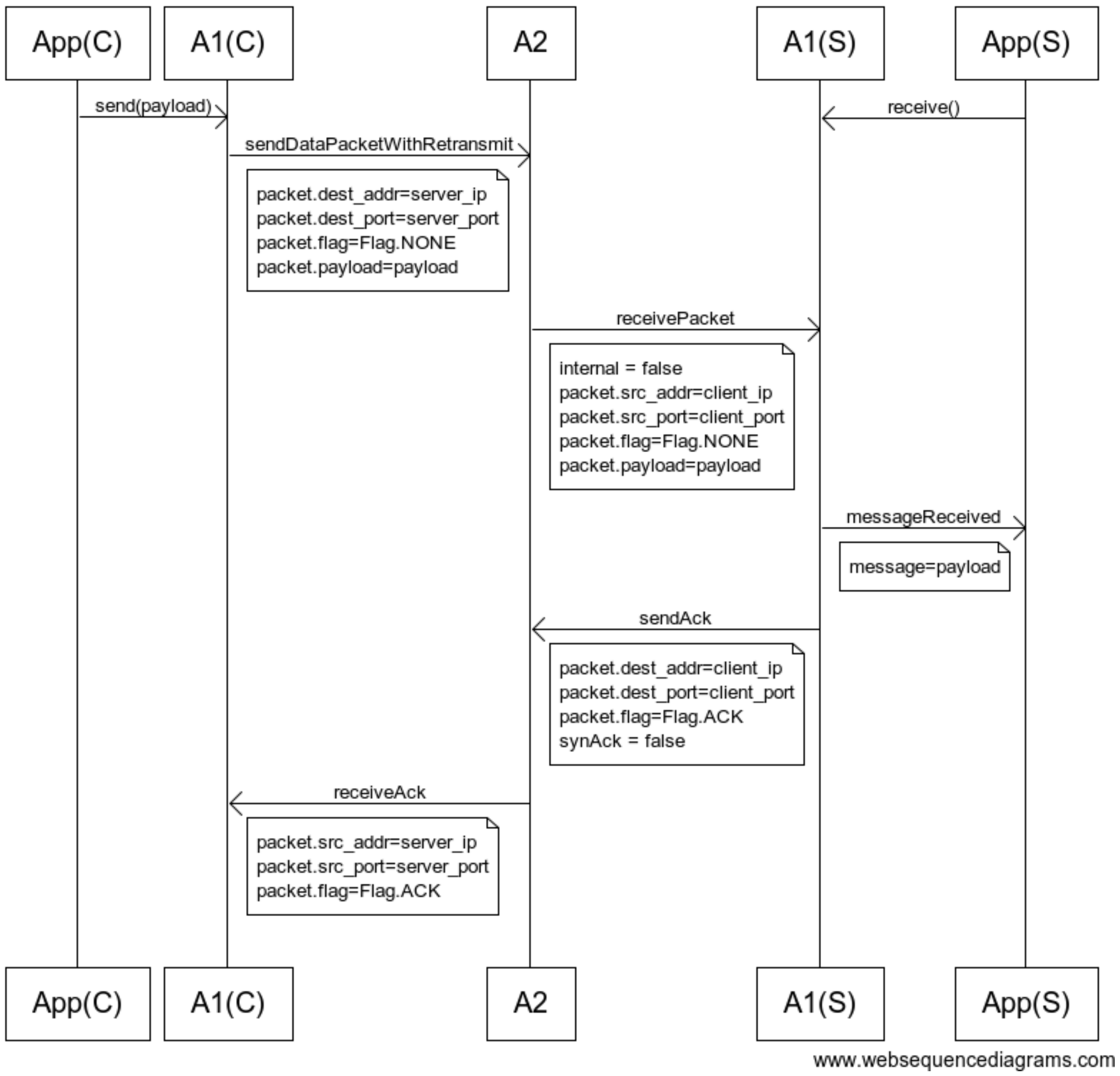
Connect to
App(Client)
A1(Client)
A2
A1(Server)
App(Server)
c.connect
client_ip,client_port
ClSocket socket.send(packet)
pachet.dest_ip = server_ip
pachet.dest_port = server_port
pachet.flag = Flag.SYN
pachet.seq_nr = A
ClSocketReceiver receiver.getPacket
pachet.src_ip = client_ip
pachet.src_port = client_port
pachet.flag = Flag.SYN
pachet.seq_nr = A
accept ()
SendAck
pachet.dest_ip = client_ip
pachet.dest_port = client_port
pachet.flag = Flag.SYN_ACK
packet.ack_nr = A+1
pachet.seq_nr = B
SynAck = true
receiveAck
pachet.src_ip = server_ip
pachet.src_port = server_port
pachet.flag = Flag.SYN_ACK
packet.ack_nr = A+1
pachet.seq_nr = B
SendAck
synAck = false
pachet.dest_ip = server_ip
pachet.dest_port = server_port
pachet.flag = Flag.ACK
packet.ack_nr = A+1
pachet.seq_nr = B+1
receiveAck
pachet.src_ip = client_ip
pachet.src_port = client_port
pachet.flag = Flag.ACK
packet.ack_nr = A+1
pachet.seq_nr = B+1
www.websequencediagrams.com

Disconnect


App(C)
A1(C)
A2
A1(S)
App(S)
c.close()
ClSocket socket.send(packet)
packet.dest_addr=server_ip
packet.dest_port=server_port
packet.src_addr=client_ip
packet.src_port=client_port
packet.flag=Flag.FIN
ClSocketReceiver receiver.getPacket()
receiver.port = server_port
packet.src_addr=client_ip
packet.src_port=client_port
packet.flag=Flag.FIN
sendAck
packet.dest_addr=client_ip
packet.dest_port=client_port
packet.flag=Flag.ACK
timer
receiveAck
packet.dest_addr=client_ip
packet.dest_port=client_port
packet.flag=Flag.ACK
close()
ClSocket socket.send(packet)
packet.dest_addr=client_ip
packet.dest_port=client_port
packet.flag=Flag.FIN
ClSocketReceiver receiver.getPacket
internal=true
packet.src_addr=server_ip
packet.src_port=server_port
packet.flag=Flag.FIN
receiver.port = client_port
sendAck
packet.dest_addr=server_ip
packet.dest_port=server_port
packet.flag=Flag.ACK
timer
receiveAck
packet.dest_addr=server_ip
packet.dest_port=server_port
packet.flag=Flag.ACK
dispose()
dispose()
App(C)
A1(C)
A2
A1(S)
App(S)

www.websequencediagrams.com

Send/Receive



High-level description of A1

We will be using the given ConnectionImpl class to implement the connection part of A1. ConnectionImpl is a subclass of AbstractConnection and it implements the Connection interface. We will also be using most of the utillity functions provided in AbstractConnection.

Use of admin

We will be using the Admin framework for invaluable logging and debugging information. Using the log system, we can conveniently check how the packet is affected by A2, which will in turn help us in developing A1. We will also use Admin to create isolated errors, which will help us in developing A1 to be reliable.

Implementation of the class ConnectionImpl

NOTE:
The following method implementations will use the isValid() method to check for packet contamination whenever it receives a new packet. Similarily, before sending, they will use packet.setChecksum() to set the checksum equal to the value of packet.calculateChecksum(). This will ensure that when the packet arrives, we can use isValid() to check if anything went wrong along the way.

Implementation of connect(InetAdress remoteAdress, int remotePort)
Condition: State.CLOSED

This sets the Connection's member variables remoteAdress and remotePort to the corresponding input parameters. It then uses the a ClSocket to send off the packet with Flag.SYN into A2. After sending, the Connection changes state to State.SYN_SENT.

The thread will then wait for a SYN_ACK from the server. When it receives it, it will send an ACK and change the state to State.ESTABLISHED.

The method returns.

Implementation of accept()
Condition: State.LISTEN

When the server application is set up, it will create a listening Connection that calls accept() continously to accept incoming connections.

accept() uses CLSocketReceiver to help listening for incoming packets from A2 with a SYN flag from source addresses which it doesn't already have an active Connection with.

If it receives a SYN packet from a new source, it will use the src_addr and src_port parameters of the packet to create a new connection. It then sets the new connections state to State.SYN_RCVD.

The thread wil then wait for an ACK from the source. When this is received, it will change the newly created Connection's state to State.ESTABLISHED

The method returns the newly created Connection. The APP layer can then save it appropriately by adding it to a list of active Connections.

Implementation of send(string payload)
Condition: State.ESTABLISHED

The method uses the string received as a parameter to create a new packet with the string as the payload. The packet will use the src_addr, src_port, dest_addr and dest_port information that is found in the active Connection's member variables.

The packet is then sent into A2 by using sendDataPacketWithReTransmit to the server. After it is sent, it will wait for an ACK from the destination host. If it doesn't receive anything until RETRANSMIT time, it will try to retransmit the packet. If this persist until TIMEOUT time, the method will return null.

If an ACK is received in time the method will return the received ACK.

Implementation of receive()
Condition: State.ESTABLISHED

This method is used to receive APP relevant payloads (AKA. externalPackets in the framework). It will use the inherited receivePacket method to get the payload from A2.

After receiving the packet, it will send an ACK back using sendAck.

The method returns the received packet's payload, which is in the form of a string.

Implementation of close()
Condition: State.ESTABLISHED

This will create a packet with a FIN flag and send it into A2 using CLSocket socket.send(packet).

After sending it, the Connection will change its state to State.FIN_WAIT_1. In this state it will wait for an ACK.

After receiving and ACK, it will change its state to FIN_WAIT_2, and will wait for a FIN from the destination host.

After receiving the FIN it will again change the state to State.TIME_WAIT and send another ACK.

It will then wait to see it if receives another FIN. If another FIN is received we know that the ACK just sent never reached the destination, so it will send the ACK again.

If the Connection does not receive another FIN in CLOSE_TIMER time, it will assume that the ACK reached it's destination and change the state of the connection to State.CLOSED.

The method returns and informs the APP that the Connection is closed, so the APP can appropriately dispose of it.

Implementation of isValid()
Condition: State.ESTABLISHED

This method is used to validate an incoming packet. This will eliminate common sending errors like ghost packets and packet contamination.

We use this by calling getChecksum and calculateChecksum from the incoming packet. If these match the packet is not contaminated and the method returns true. If they are not equal, the method returns false.

Tests

Testene skal gjennomføres på den ferdig implementerte A1. Hver test skal gjennomføres ved at to instanser av et testprogram skal kommunisere over det upålitelige nettverket A2, med settings i settings.xml som angitt i settings under. På hver test skal det forsøkes å koble til, sende data, motta data og koble fra. Tilkobling eller frakobling feiler, data som sendes eller mottas avviker fra de forventede dataene eller om det kastes en feilmelding, skal dette loggføres og koden skal feilsøkes.

ID	Settings	Formål
1	<loss>0.0</loss> <delay>0.0</delay> <ghost>0.0</ghost> <payload>0.0</payload> <header>0.0</header>	Å avdekke om A1 har noen problemer som ikke avhenger av feil fra det upålitelige nettverket.
2	<loss>0.1</loss> <delay>0.0</delay> <ghost>0.0</ghost> <payload>0.0</payload> <header>0.0</header>	Å avdekke om A1 har problemer med å håndtere at enkelte pakker går tapt.
3	<loss>0.0</loss> <delay>0.1</delay> <ghost>0.0</ghost> <payload>0.0</payload> <header>0.0</header>	Å avdekke om A1 har problemer med å håndtere at enkelte pakker kommer fram senere enn antatt.

4	<loss>0.0</loss> <delay>0.0</delay> <ghost>0.1</ghost> <payload>0.0</payload> <header>0.0</header>	Å avdekke om A1 har problemer med å håndtere at enkelte vilkårlige pakker dukker opp fra ingensteds.
5	<loss>0.0</loss> <delay>0.0</delay> <ghost>0.0</ghost> <payload>0.1</payload> <header>0.0</header>	Å avdekke om A1 har problemer med å håndtere at enkelte pakker kommer fram med skadet payload.
6	<loss>0.0</loss> <delay>0.0</delay> <ghost>0.0</ghost> <payload>0.0</payload> <header>0.1</header>	Å avdekke om A1 har problemer med å håndtere at enkelte pakker kommer fram med ødelagt header.
7	<loss>0.5</loss> <delay>0.0</delay> <ghost>0.0</ghost> <payload>0.0</payload> <header>0.0</header>	Å avdekke om A1 har problemer med å håndtere at mange pakker går tapt.
8	<loss>0.0</loss> <delay>0.5</delay> <ghost>0.0</ghost> <payload>0.0</payload> <header>0.0</header>	Å avdekke om A1 har problemer med å håndtere at mange pakker kommer fram senere enn antatt.
9	<loss>0.0</loss> <delay>0.0</delay> <ghost>0.5</ghost> <payload>0.0</payload> <header>0.0</header>	Å avdekke om A1 har problemer med å håndtere at mange vilkårlige pakker dukker opp fra ingensteds.
10	<loss>0.0</loss> <delay>0.0</delay> <ghost>0.0</ghost> <payload>0.5</payload> <header>0.0</header>	Å avdekke om A1 har problemer med å håndtere at mange pakker kommer fram med skadet payload.
11	<loss>0.0</loss> <delay>0.0</delay> <ghost>0.0</ghost> <payload>0.0</payload> <header>0.5</header>	Å avdekke om A1 har problemer med å håndtere at mange pakker kommer fram med ødelagt header.

12	<loss>0.1</loss> <delay>0.0</delay> <ghost>0.1</ghost> <payload>0.1</payload> <header>0.0</header>	Ødelagte headere og delays er blandt de tingene som er vanskelig å avdekke, spessielt når de forekommer sammen. Derfor kjører vi først en test med disse feilene på 0%. Formålet med testen er å avdekke om A1 håndterer pakketap, spøkelsespakker og ødelagte payloads samtidig. 10% feil på hver burde holde lang vei til å teste dette, og økes den risikerer vi at det totale tapet stiger raskt.
13	<loss>0.05</loss> <delay>0.05</delay> <ghost>0.05</ghost> <payload>0.05</payload> <header>0.05</header>	Formålet med testen er å avdekke om A1 kan håndtere alle typer tap samtidig. 5% burde her være hardt nok til å avdekke alle feil, samtidig som det ikke blir alt for høyt.

Error handling

When sending packets we always expect an ACK in return. If an ACK is not received, we will resend the packet. If a single cumulative ACK is received, this indicates a lost packet, and the packet should be retransmitted. If a double ACK is received, it indicates a gap, and the transmitting of packets should return back to the point indicated by the seq. number in the double packet.

Package lost

We can check for this error by looking at the incoming packets sequence number, and check if it equals the Connections cumulative sequence number. If it does send a single cumulative ACK, ACKing both in order sequence.
If a gap in the sequencenumbers are detected, a double ACK, with the sequencenumber of the packet expected on the lower end of the gap should be transmitted.

Package delayed

If the received packet has a sequence number that the host already have returned an ACK for, it means that a packet got delayed. The receiver then has to send an ACK with an ACK number that is includes the delayed packet, but also any packets received after.

Package has errors

We check this by using the isValid() method in the Connection class. If the checksum does not match, it means that the packet has errors, which will be identified when isValid() is run. If the packet is invalid, we simple do not send an ACK back to the source. This will cause the source to resend the packet, and when it arrives without errors, then we can finally send the ACK.

Ghost package

This error will occur if the header of the packet gets contaminated while being sent. This will cause the packet to potentially be sent to a completely different host and/or socket. We can handle this error by checking if we have an established connection with the source. If not, the packets flag has to be Flag.SYN (this would mean we have a new incoming connection), else we disregard the packet.