Collaboration Project
KTN1 Phase 1
Gruppe 34

Bjørn Åge Tungesvik Tina Syversen André Philipp Odd Magnus Trondrud
Eivind Kvissel Håvard Høiby

March 12, 2012

Part I
The Design

1 Connection

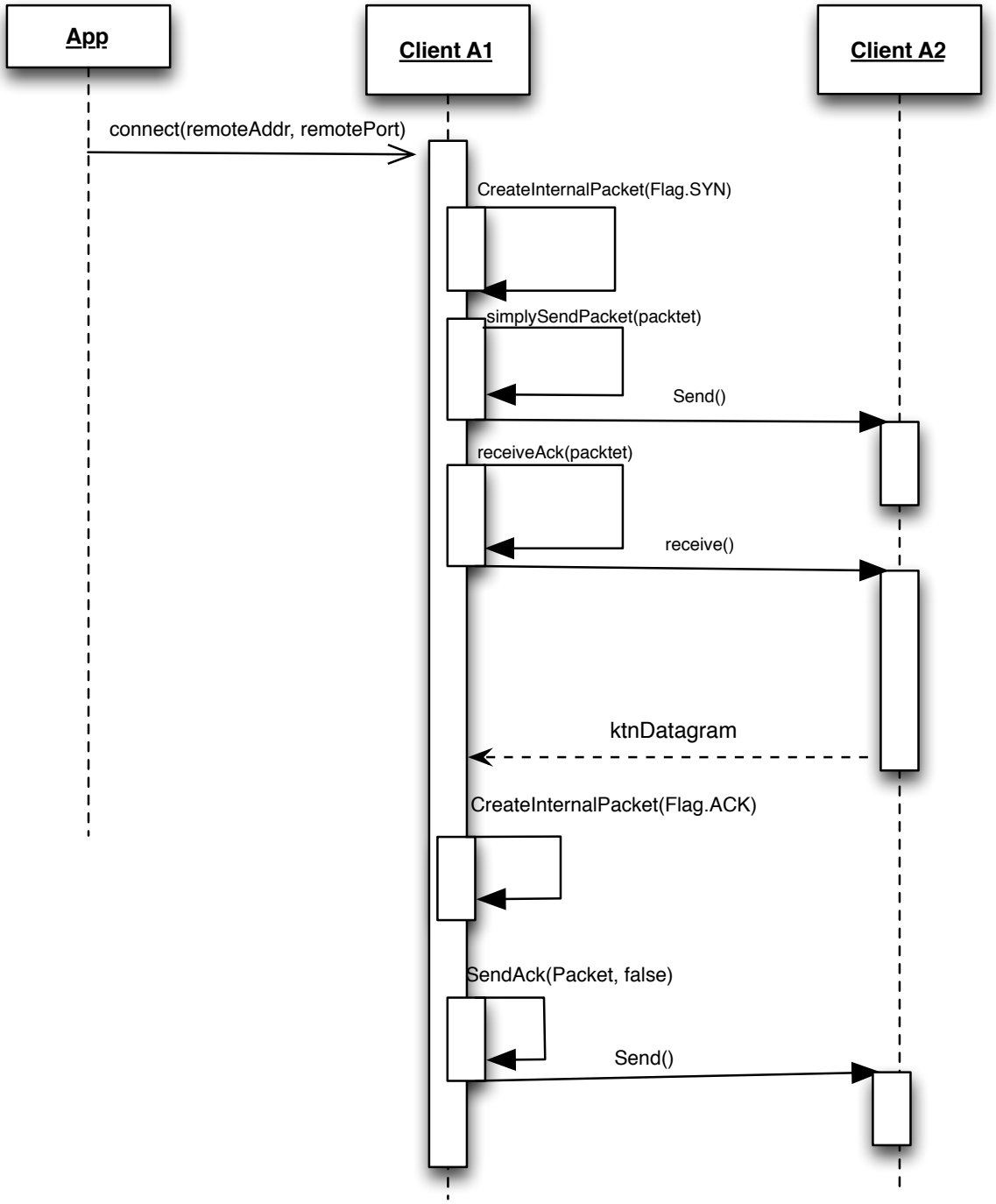
TCP employs a three way handshake to create a connection between two hosts. This functionality can be realized by implementing the `connect()` and `accept()` methods mentioned in the connection interface. The `connect()` method will handle the client side of the handshake, while `accept()` will handle the server side.

The TCP client starts in the closed state. A new TCP is initiated by sending a packet with the `SYN` bit set to 1, and its randomly chosen internal sequence number. After the `SYN` segement is sent, the client enters the `SYN_SENT` state, waiting for the corresponding `SYN_ACK` from the server.

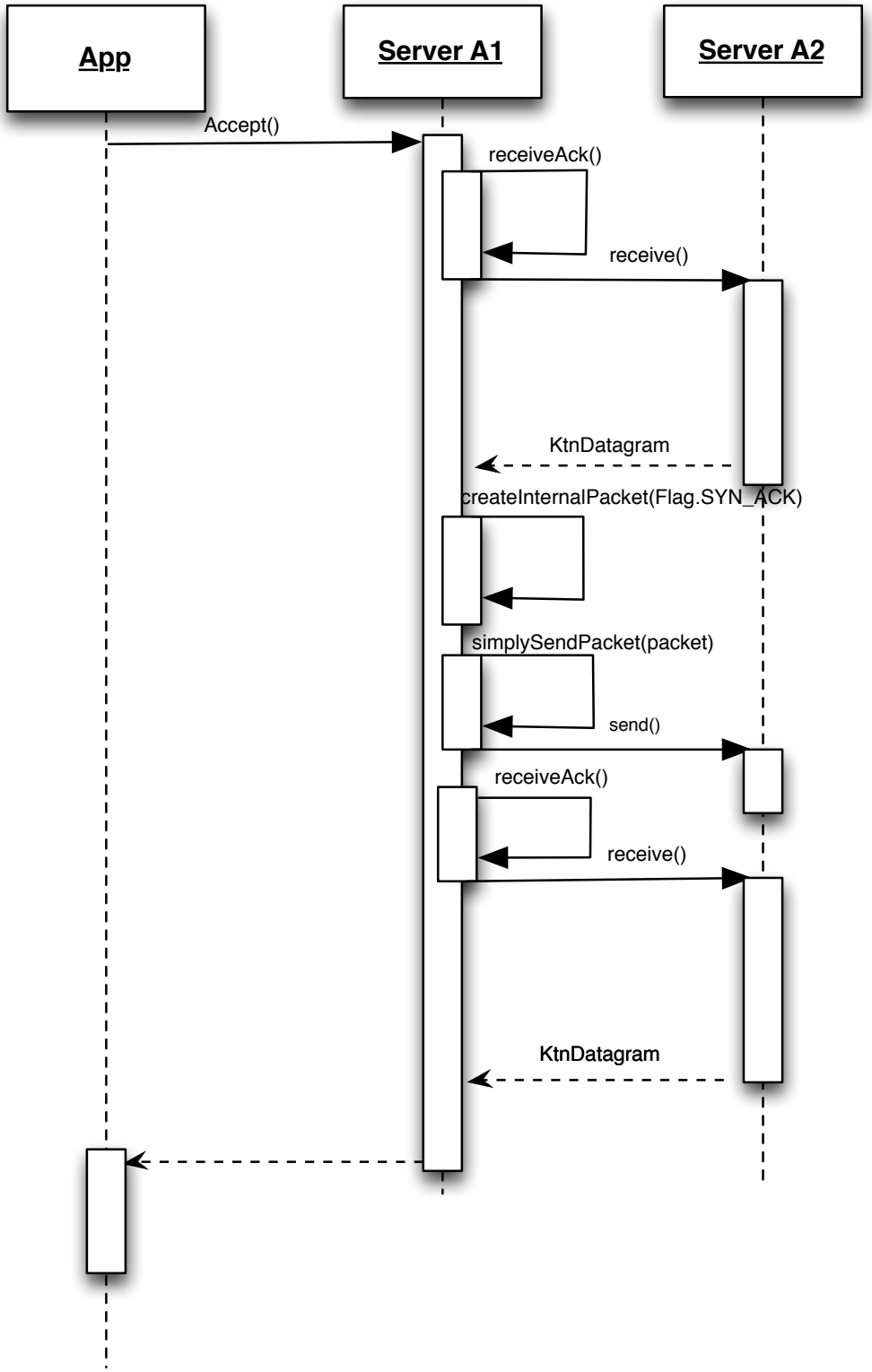
The server will respond to the `SYN` packet by sending a `SYN_ACK` packet, with the `SYN` bit set to 1, its internal sequence number, and the `ACK` field set to the next expected sequence number.

When the client has received the `SYN_ACK` packet, it enters the `ESTABLISHED` state, and sends an `ACK` packet, informing the server that the `SYN_ACK` packet was received. When the server receives the `ACK` packet, the server enters the `ESTABLISHED` state.

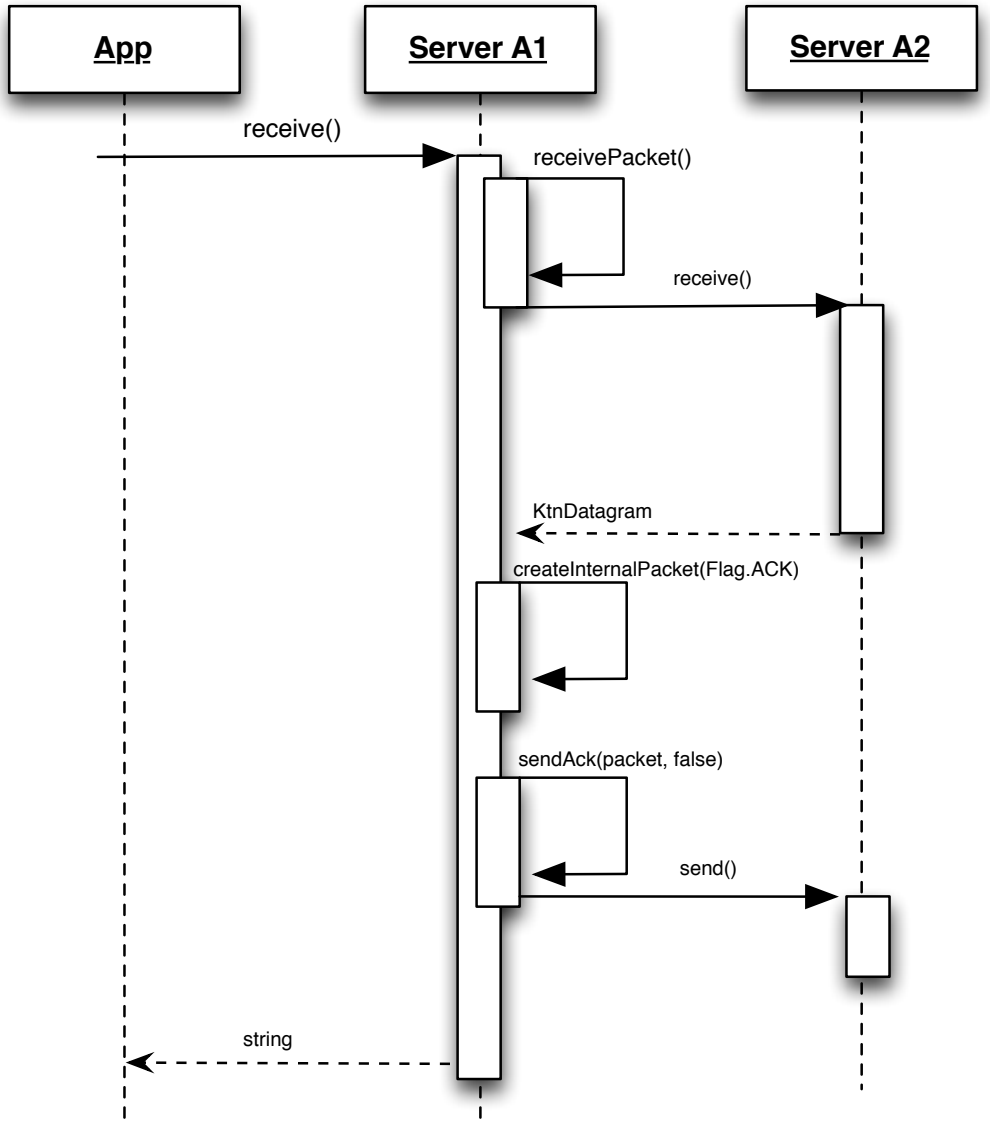
Sequence Diagram - connect()



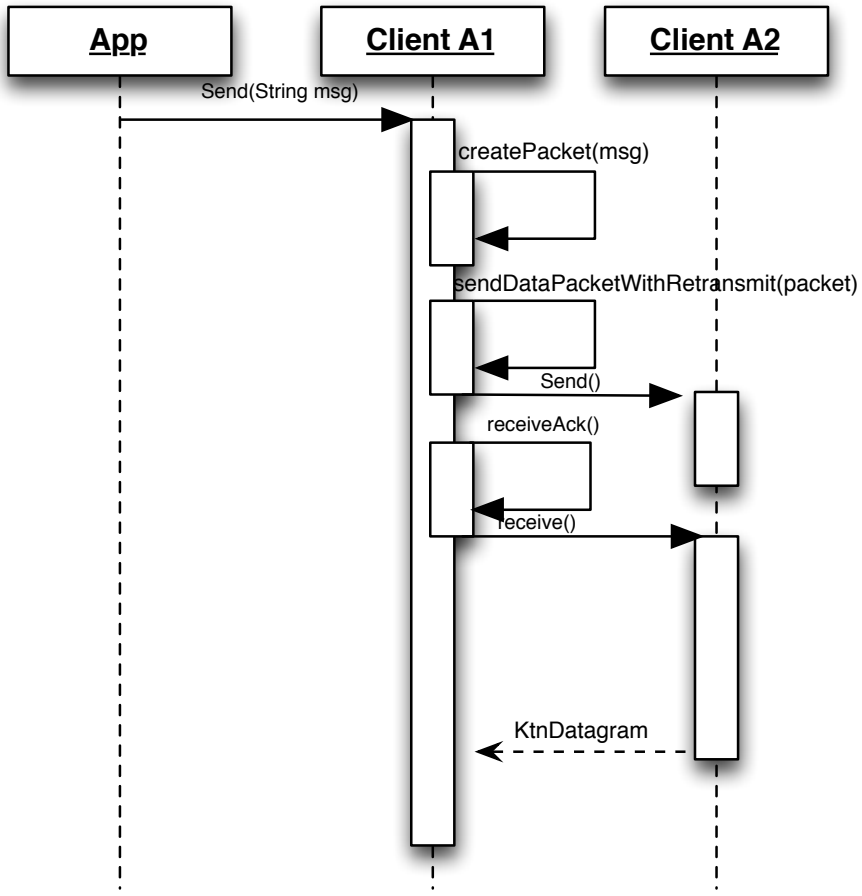
Sequence Diagram - accept()



Sequence Diagram - receive()



Sequence Diagram - send()



2 Disconnection

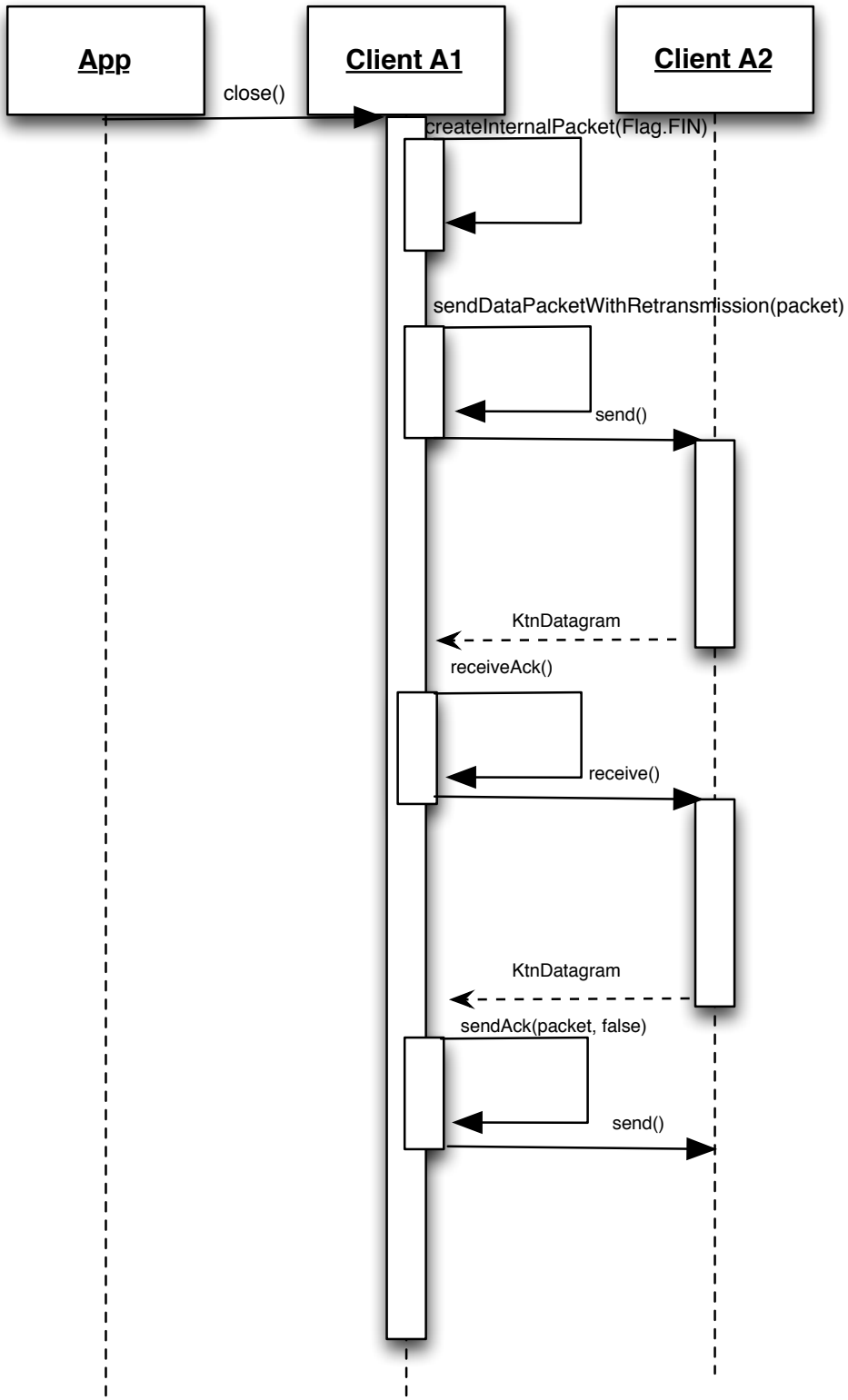
For this discussion, lets presume that the clients initiate the close operation using the `close()` method (Note: the server can also use the close operation). This causes the client to send a `SYN` packet with the `FIN` bit set to 1, and enter the `FIN_WAIT_1`. While in the `FIN_WAIT_1` state the client waits for an `ACK` packet. Upon receiving the `ACK` packet the client switches to `FIN_WAIT_2` state.

In `FIN_WAIT_2` state the client is expecting a `FIN` packet from the server. As mentioned above, it's possible to receive the `FIN` packet directly. Anyway, when the `FIN` packet is received, the client will respond with an `ACK` packet, and enter the `TIME_WAIT` state. This state allows the client to re-send the last final `ACK` packet in case its lost. After the waiting period, the client formally closes the connection.

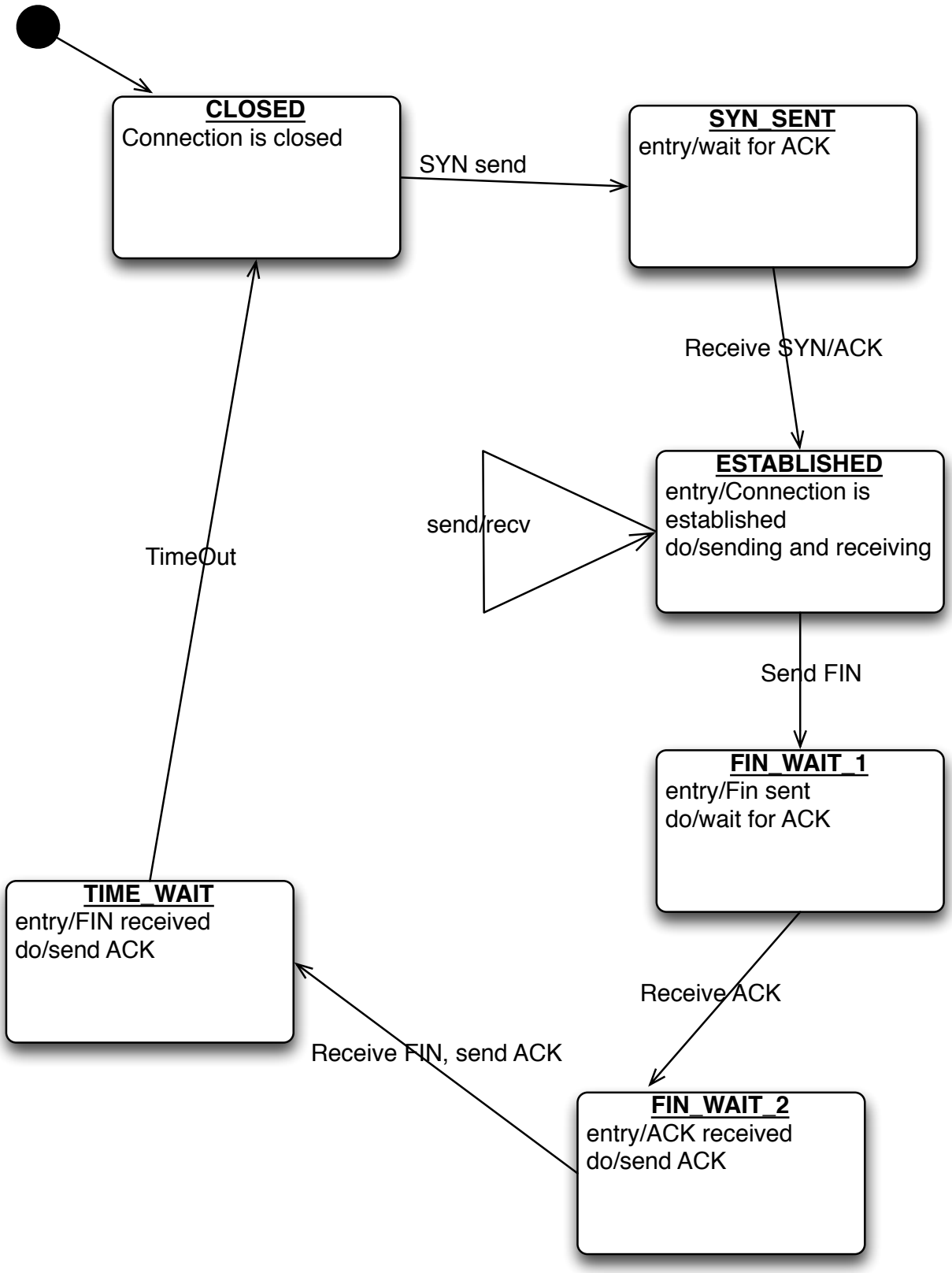
The server will receive the `FIN` packet, send an `ACK` packet as a response, and enter the `CLOSE_WAIT` state. While in the close wait state, the server sends a `FIN` packet to the client, and enters the `LAST_ACK` state. In this state the server wait for a `ACK` packet from the client. When the packet is finally received, the connection is finally closed.

Note: The server can also terminate a connection, thereby switching roles with the client.

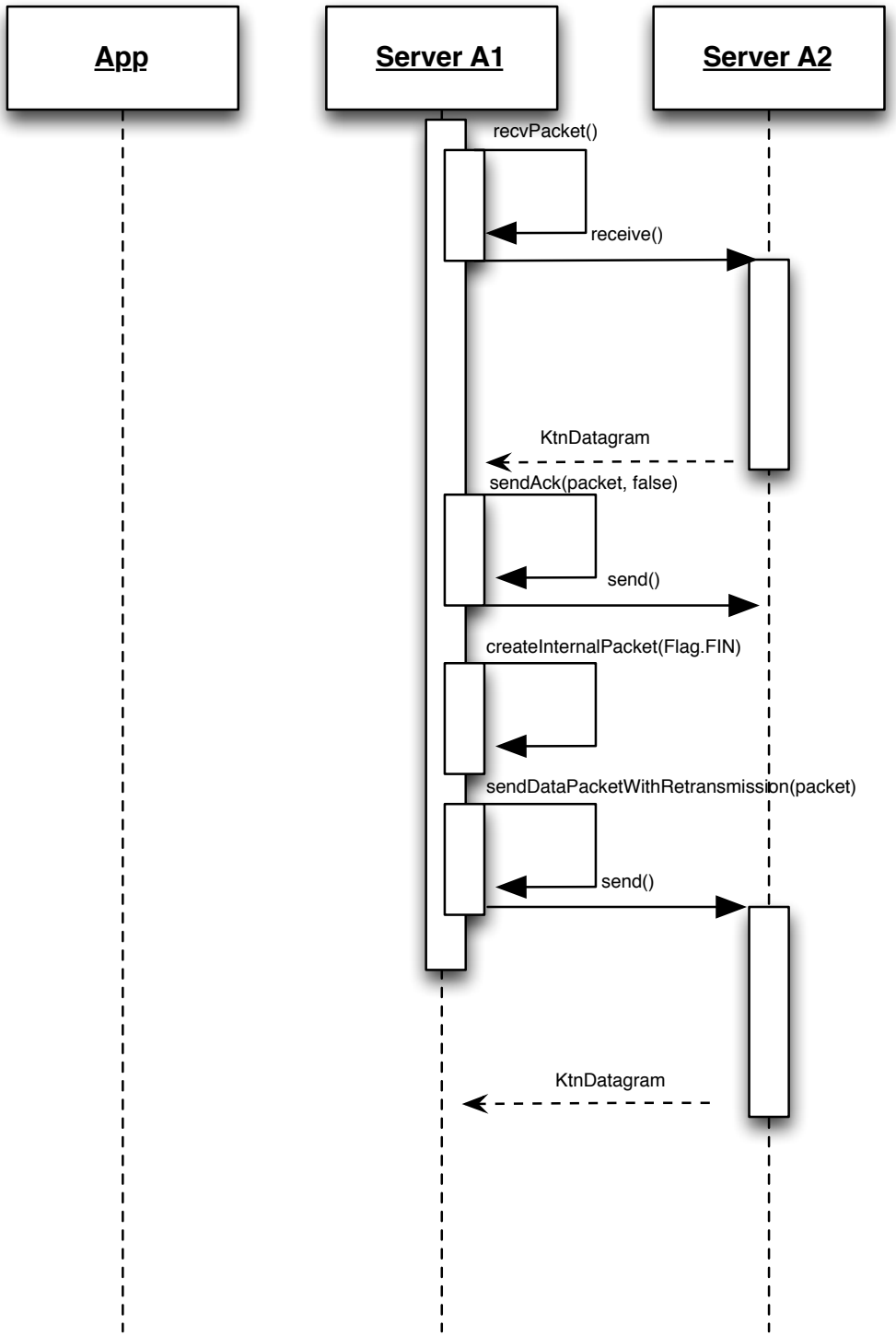
Sequence Diagram - `close()`



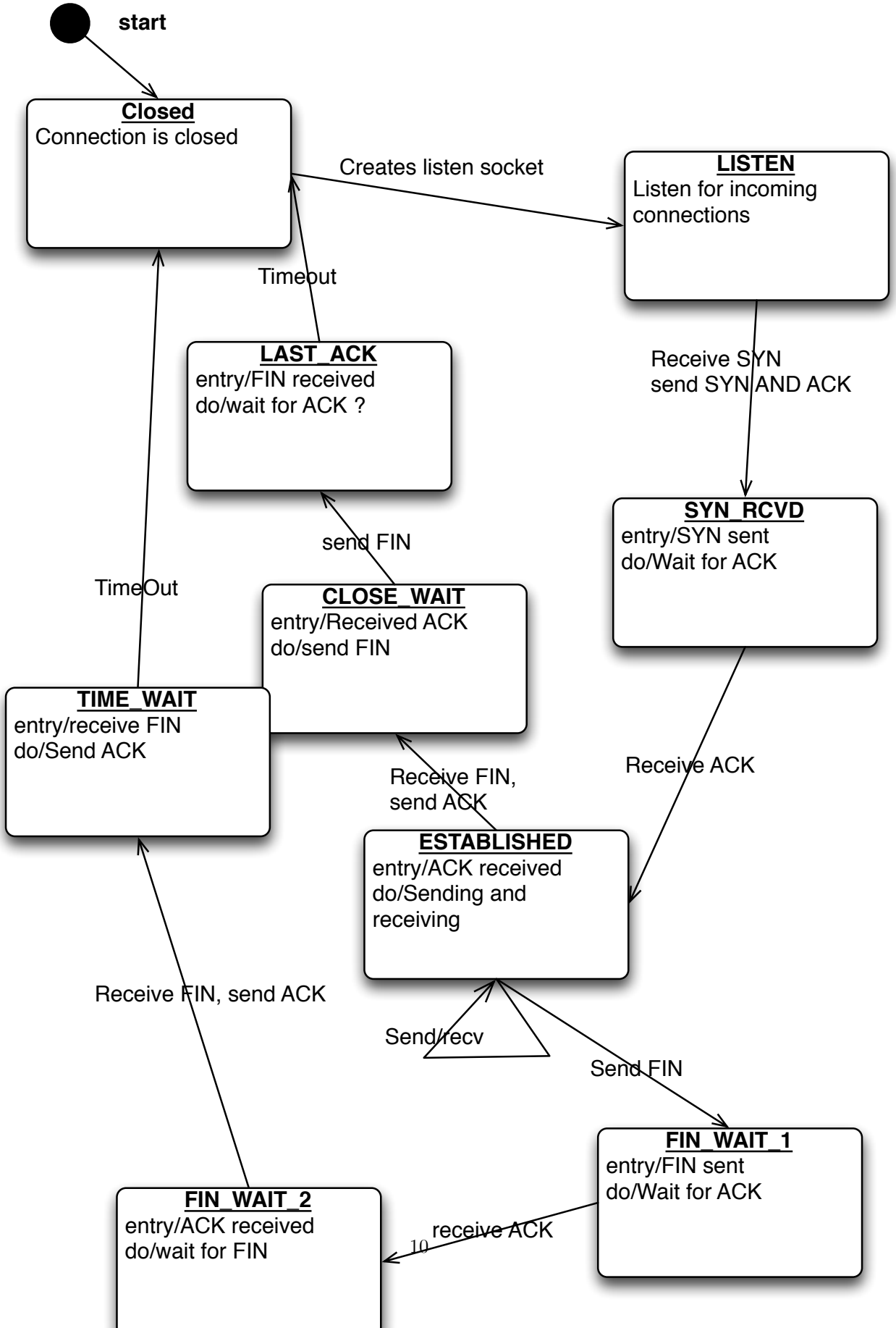
State Machine Diagram - Close Connection



Sequence Diagram - ServerClose



State Machine Diagram - Close Connection, Server



ServerClose

3 Error Handling

Packet Loss

When sending a packet the receiver of the packet will ask about it until the packet is received. The packet is timed so that if the sender does not receive an `ACK` during a given time(timeout), the sender will send the packet again.

Packet Delay

A1 saves all the packets in a buffer, but it does not store duplicates. So when the sender does not receive an `ACK`, it believes that the packet is lost and sends it again. This means that the receiver gets two of the same packet. This problem can be solved by checking the sequence number. If a packet with the same sequence number has already been received, the packet will be dropped, but the receiver will still send an `ACK` for the duplicated packet.

Packet Error/Corruption

Right before a message is received it will be checked for errors using a checksum to see if the packet contains any errors. If the packet does contain errors an `ACK` will be sent, saying that the packet contains errors.

Ghost Packets

The checksum will be checked and if it does not chek out be discarded. If it is approved, the sequence number will be checked. If it does not have the right sequence number it will be thrown away, if not it will be approved as an ordinary packet.

Packet Checks Out

If the packet checks out it will be passed on upwards.

Part II
The Tests

4 Testing Plan

The tests in this plan are laid out in an incremental fashion. First tests will be performed for all features (connection, tear down, send and receive) without errors enabled to ensure that everything works in a trivial environment. Then each error will be added one by one. After two errors have been tested individually those two are tested in combination, when three errors have been tested all three errors will be tested together. And so on until all errors are accounted for.

Error Probability

Each test case, containing errors, will be run twice at 10% and 50% error rate in conjunction with the specifications (ref: Kompendium for Fellesprosjektet chap. 6.3.1).

Test repetition

All test containing the sending and reception of an actual payload will dispatch 10 distinct payloads, in order to evaluate the received payloads precisely.

5 Initial Connection (no errors enabled) - T-KTN01

Preconditions

Methods `connect()` and `accept()` must be implemented in A1 (`ConnectionImpl`). The `errors` variable in `settings.xml` should be false.

Dependencies

NA

Objective

Setup a connection between two hosts running the same implementation of A1.

Instructions

This is done by creating a `ConnectionImpl` object on both the server and the client side. First the server executes the `accept()` method with port X, then the client executes the method `connect()` with the server IP and port X.

Expected Result

A three-way handshake should be observed. This can be verified on the client side by the `connect()` method returning without throwing a `SocketTimeoutException`. On the server side the test is passed if the accept method returns a `Connection` as opposed to throwing a `SocketTimeoutException`.

6 Connection and Tear Down

Preconditions

The `close()` method must be implemented in A1 (`ConnectionImpl`). The `errors` variable in `settings.xml` should be false.

Dependencies

T-KTN01

Objective

Close a connection between two hosts running the same implementation of A1.

Instructions

This is done by first creating a connection as described in **T-KTN01**. After the connection is established either the client or the server calls the `close()`. This should be tested in two separate runs.

Expected Result

A four-way handshake should be observed. The caller of the `close()` method can verify a successful test if method returns without an exception.

7 Send and Receive (no errors enabled) - T-KTN03

Preconditions

Methods `send()` and `receive()` must be implemented in A1 (`ConnectionImpl`) The errors variable in `settings.xml` should be false.

Dependencies

T-KTN02

Objective

Send a packet between two hosts running the same implementation of A1.

Instructions

A connection must be established as in T-KTN01. The server calls `receive()` in a loop in order to handle multiple messages. The client calls `send()` 10 times with distinct payloads containing a sequence number. When the test is executed the client calls `close()` as described in **T-KTN02** to end the session.

Expected Result

The packages sent by the client host should be received by the server host. This test should be considered a success if the sending client can call `send()`, and the receiving client can call `receive()` without either client throwing a `ConnectionException` or an `IOException`. And all the 10 Strings returned by the `receive()` function should be equal to the String arguments to the `send()` function.

8 With Loss Error - T-KTN04

Preconditions

Methods `send()` and `receive()` must be implemented and working, and the `loss` variable in the `settings.xml` should be set to first 10% errors and then 50%. The errors variable should be true and all other numeric variables must be 0%

Dependencies

T-KTN03

Objective

Test if the system can handle packet loss.

Instructions

A connection must be established as in **T-KTN01**. The method `send()` should be called on the client side and `receive()` in a loop on the server side. The send method should be applied 10 times with distinct String arguments. Once the packages has been received by the second host, the connection should close by the client, as described in **T-KTN02**.

Expected Result

The `connect()` and `accept()` must first return without exceptions. Regardless of packet loss error rate (unless it's 100%, in which case there should be thrown a `SocketTimeoutException` on the clients `connect()` call), the receiving client should receive all packets sent by the sender client. I.E. all the 10 distinct String should be equal. Additionally, all `send()` and `receive()` calls must return without throwing either a `ConnectionException` or an `IOException`.

9 Test With Delay - T-KTN05

Preconditions

Methods `send()` and `receive()` must be implemented and working, and the `delay` variable in the `settings.xml` should be set to first 10% errors and then 50%. The `errors` variable should be true and all other numeric variables must be 0%

Dependencies

T-KTN03

Objective

Test if the system can handle packet delay.

Instructions

A connection must be established as in **T-KTN01**. The server calls `receive()` in a loop in order to handle multiple messages. The client calls `send()` 10 times with distinct payloads containing a sequence number. When the test is executed the client calls `close()` as described in **T-KTN02** to end the session.

Expected Result

The `connect()` and `accept()` must first return without exceptions. The test should be considered a success if all 10 messages are received by the server in the order the client sends them. Additionally, all `send()` and `receive()` calls must return without throwing either a `ConnectionException` or an `IOException`.

10 With Loss and Delay - T-KTN06

Preconditions

Methods `send()` and `receive()` must be implemented and working, and the `loss` and `delay` variables in the `settings.xml` should be set to first 10% errors, and then 50%. The `errors` variable should be true and all other numeric variables must be 0%

Dependencies

T-KTN03

Objective

Test if the system can handle packet loss and delay at the same time.

Instructions

A connection must be established as in **T-KTN01**. The method `send()` should be called on the client side and `receive()` in a loop on the server side. The send method should be applied 10 times with distinct String arguments with sequence numbers. Once the packages has been received by the second host, the connection should close by the client, as described in **T-KTN02**.

Expected Result

The `connect()` and `accept()` must first return without exceptions. The test should be considered a success if 10 messages are received by the server in the order the client sends them. Additionally, all `send()` and `receive()` calls must return without throwing either a `ConnectionException` or an `IOException`.

11 Ghost Test - T-KTN07

Preconditions

Methods `send()` and `receive()` must be implemented and working, and the `ghost` variable in the `settings.xml` should be set to first 10% errors, and then 50%. The `errors` variable should be true and all other numeric variables must be 0%

Dependencies

T-KTN03

Objective

Test if the system can handle ghost packages.

Instructions

A connection must be established as in **T-KTN01**. The server calls `receive()` in a loop in order to handle multiple messages. The client calls `send()` 10 times with distinct payloads containing a sequence number. When the test is executed the client calls `close()` as described in **T-KTN02** to end the session.

Expected Result

The `connect()` and `accept()` must first return without exceptions. The test should be considered a success if only the 10 messages sent by the client is received by the server. Additionally, all `send()` and `receive()` calls must return without throwing either a `ConnectionException` or an `IOException`.

12 With Loss, Delay and Ghosts - T-KTN08

Preconditions

Methods `send()`and `receive()` must be implemented and working, and the `loss`, `delay` and `ghost` variables in the `settings.xml` should be set to first 10% errors and then 50%. The `errors` variable should be true and all other numeric variables must be 0%

Dependencies

T-KTN03

Objective

Test if the system can handle packet loss, delay and ghost packages at the same time.

Instructions

A connection must be established as in T-KTN01. The method `send()` should be called on the client side and `receive()` in a loop on the server side. The send method should be applied 10 times with distinct String arguments with sequence numbers. Once the packages has been received by the second host, the connection should close by the client, as described in **T-KTN02**.

Expected Result

The `connect()` and `accept()` must first return without exceptions. The test should be considered a success if only 10 messages are received by the server in the order the client sends them. Additionally, all `send()` and `receive()` calls must return without throwing either a `ConnectionException` or an `IOException`.

13 Test Payload Bit Errors - T-KTN09

Preconditions

The `send()` and `receive()` methods should be implemented and working, the program should first be tested for 10%, and then 50% payload bit errors. The `errors` variable should be true and all other numeric variables must be 0%

Dependencies

T-KTN03

Objective

Test if the system can handle payload bit errors.

Instructions

A connection must be established as in **T-KTN01**. The server calls `receive()` in a loop in order to handle multiple messages. The client calls `send()` 10 times with distinct payloads containing a sequence number. When the test is executed the client calls `close()` as described in **T-KTN02** to end the session.

Expected Result

The `connect()` and `accept()` must first return without exceptions. The test should be considered a success if all the 10 messages sent by the client is received by the server and content is equal. Additionally, all `send()` and `receive()` calls must return without throwing either a `ConnectionException` or an `IOException`.

14 Lost Ghosts and Delayed Payload Errors - T-KTN10

Preconditions

Methods `send()` and `receive()` must be implemented and working, and the loss, delay, ghost and payload variables in the `settings.xml` should be set to first 10% errors and then 50%. The `errors` variable should be true and header variables must be 0%.

Dependencies

T-KTN07, **T-KTN09**

Objective

Test if the system can handle packet loss, delay, payload bit errors and ghost packages at the same time.

Instructions

A connection must be established as in **T-KTN01**. The method `send()` should be called on the client side and `receive()` in a loop on the server side. The send method should be applied 10 times with distinct String arguments. Once the packages has been received by the second host, the connection should close by the client, as described in **T-KTN02**.

Expected Result

The `connect()` and `accept()` must first return without exceptions. The test should be considered a success if only 10 messages are received by the server in the order the client sends them and the content of the messages is equal. Additionally, all `send()` and `receive()` calls must return without throwing either a `ConnectionException` or an `IOException`.

15 Erroneous Heads - T-KTN11

Preconditions

Methods `send()` and `receive()` must be implemented and working, and the header variable in the `settings.xml` should be set to first 10% errors and then 50%. The `errors` variable should be true and all other numeric variables must be 0%

Dependencies

T-KTN03

Objective

Test if the system can handle bit errors in the header.

Instructions

A connection must be established as in **T-KTN01**. The server calls `receive()` in a loop in order to handle multiple messages. The client calls `send()` 10 times with distinct payloads containing a sequence number. When the test is executed the client calls `close()` as described in **T-KTN02** to end the session.

Expected Result

The `connect()` and `accept()` must first return without exceptions. The test should be considered a success if all the 10 messages sent by the client is received by the server and content is equal. Additionally, all `send()` and `receive()` calls must return without throwing either a `ConnectionException` or an `IOException`.

16 The Full Monty - T-KTN12

Preconditions

Methods `send()` and `receive()` must be implemented and working, and all numeric variables in the `settings.xml` should be set to first 10% errors and then 50%.

Dependencies

T-KTN10, **T-KTN11**

Objective

Test if the system is robust enough to handle all errors enabled at the same time.

Instructions

A connection must be established as in **T-KTN01**. The server calls `receive()` in a loop in order to handle multiple messages. The client calls `send()` 10 times with distinct payloads containing a sequence number. When the test is executed the client calls `close()` as described in **T-KTN02** to end the session.

Expected Result

The `connect()` and `accept()` must first return without exceptions. This test should be considered successful if the receiving side receives only 10 messages with the exact content and order the sending side sends. Additionally, all `send()` and `receive()` calls must return without throwing either a `ConnectionException` or an `IOException`.