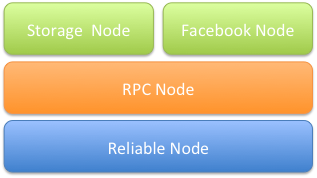
# First assignment

For building the Storage and Facebook layers, we created two abstractions to help the implementation:

1. Reliable Node: Implements in order delivery, at most once semantics, message retransmission and a connection oriented protocol similar to TCP (We will talk about the similarities below)
2. RPC Node: Implements a RPC layer that serializes automatically the method name and arguments. This layer also implements some command queuing to avoid sending a command to the server when there is an outstanding command still being executed (More explained below)

The diagram can be seen below:



## Reliable node

The two most important methods are:

1. sendReliableMessage(int targetNode, byte[] payload): Sends a message reliably using the underlying unreliable transport.
2. onReliableMessageReceived(int from, byte[] payload): It’s called when a packet is received (It makes sure to discard duplicates, and properly order the packets)

It’s worth first showing the package format:

1. From:
2. To:
3. Package Type:
4. Connection ID:
5. Sequence Number:

Each field contains 32 bits for simplicity. We also decided to include the from/to in the packet for debugging purposes (With the current framework we didn’t explicitly needed those fields). The Package Type can be of the following 4 types:

1. Connect: Sent initially to establish the connection. Notice this is a little different from the TCP 3 way handshake (syn, ack, syn/ack). We have the connection establishment for mainly handling crashes, 2 scenarios are explained below:
   * We can detect stale packets when a node crashes (For example, the client sent packet A and B, A was received, then the server crashed. Then when the server restarts it can detect that B was a packet from the previous session)
   * We can notify the client that a given connection doesn’t exist so it can reset
2. Ack: For each either Data or Connect packet, an ACK is sent. If an ACK is not received within 3 clocks, the packet is retransmitted.
3. Data: For each user packet, a Data packet is sent. We are not handling if a packet doesn’t fit in a MTU (Maximum transmit unit)
4. Reset: The connection reset is fired currently under 2 scenarios:
   * We receive a stale packet (For example, a packet from a different connection)
   * We give up retrying to send the packet 3 times

The method that is called when the connection gets reset is onConnectionAborted.

## Facebook Node

As mentioned earlier the facebook server is a RPCNode and implements the following rpc calls:

1. create\_user <login> <password>
2. login <login> <password>
3. logout <token>
4. add\_friend <token> <friend\_login>
5. accept\_friend <token> <friend\_login>
6. write\_message\_all <token> <message>
7. read\_message\_all <token>

Operation #2 returns the token of the user session and is used in the other operations (3,4,5,6,7).

One example of client server interaction is given below:

|  |
| --- |
| start 0  start 1  1 create\_user a apass  1 create\_user b bpass  1 login a apass  1 login b bpass  1 add\_friend 9561723318 b  1 accept\_friend 1583707579 a  1 write\_message\_all 9561723318 hello world  1 read\_message\_all 1583707579 |

Note1: Due to the framework, the server is hardcoded to be at address 0. The client can be any address that is not 0.

### Recoverability of Facebook node

On the following operations an operation will be appended to the log file:

1. create\_user
2. add\_friend
3. accept\_friend
4. write\_message\_all

For the previous example we will have the following logfile:

|  |
| --- |
| create\_user a apass  create\_user b bpass  add\_friend a b  accept\_friend b a  write\_message\_all a hello world |

Notice that we omit the token and just use the login for identification purposes. This makes the logfile easier to read and don’t compromise security since the operations were already validated before being logged to the disk.

When the server restarts it will execute the operations in the same order as in the log file.