Design Document

Data Communications - Assignment 2

Eric Tsang, A00841554, 40

Table of Contents

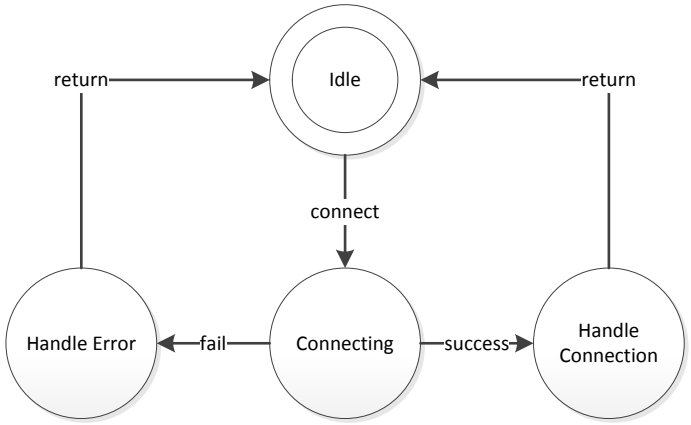
January 10, 2015

State Diagrams

This section contains a finite state machine for each of the main classes in the program. These classes include:

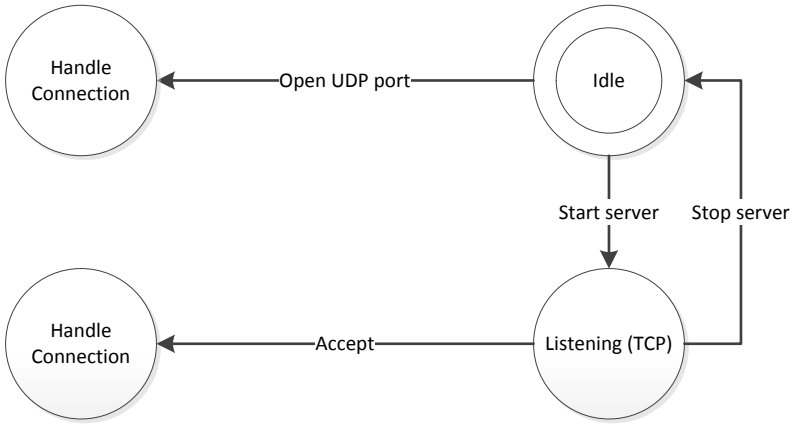
- **Client**; initiates connections with servers. both UDP and TCP.
- **Server**; listens for TCP connections and accepts them, or opens UDP ports.
- **Session**; maintains the application logic state of a connection. Used to wrap a socket, and give it application level commands, and callbacks.
- **Client Application**; wraps the client object, and give it application level commands in client mode. also keeps track of the client's control and test lines.
- **Server Application**; wraps the server object, and give it application level commands when in server mode.
- **Server Control Session**; maintains, and deals with a test session on the server. Every time a new client connects, a new one of these are created to deal with them, because this can deal with multiple

Client



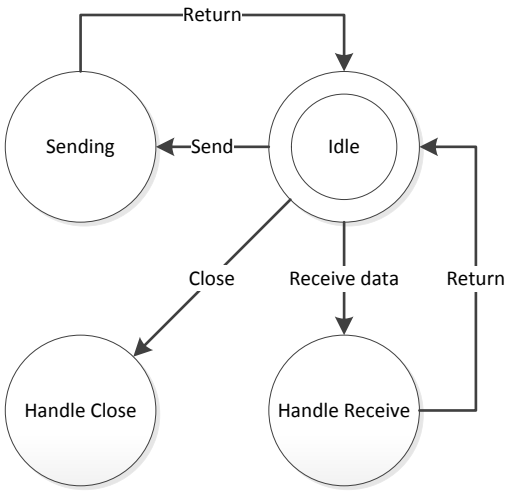
The client object is not directly coupled to any application logic. It is used to initiate connections, either by UDP, or TCP. Users must initiate this, and then assign their own error handling and connection handling function to take care of them when they occur.

Server



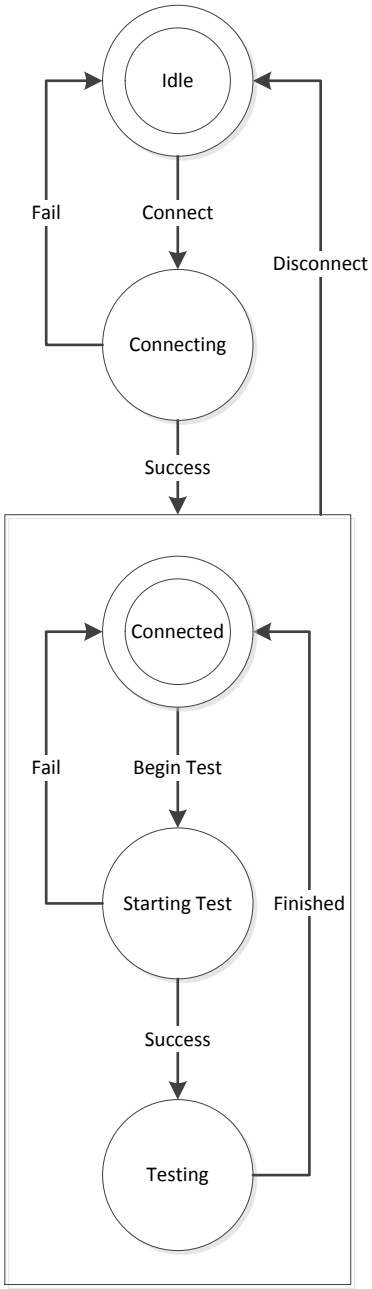
The server is a general object as well, and not coupled with any application level logic. Users must override the connection handling function, that is invoked whenever a new connection is accepted, or when it is invoked immediately when opening a UDP port.

Session



The session object wraps a socket and is used to interact with a connection. Its purpose is to make application level logic easier to integrate with the low level sockets. The session object provides a send function, and callbacks for receiving data, errors, and when the session ends.

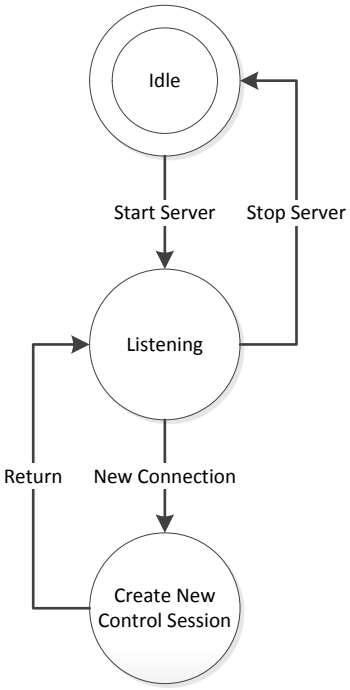
Client Application



This is the client application's STD:

- **Idle**; the application isn't doing anything yet. The user can press the connect button, and it will move on to the connecting state
- **Connecting**; the application verifies the entered remote IP address, and port number, then if that is valid, it will try to connect to the remote host. If inputs are insufficient, or connection fails, then the state goes back to the idle state. Upon successful connection, it moves on to the connected state.
- In any of the connected states (**Connected**, **Starting Test**, and **Testing**), the user can disconnect.
- **Connected**; the user can set the test parameters, and begin the test.
- **Starting Test**; test parameters are sent to the server, and validated. if valid, the server will send a go message to the client, and then the client will move on to the testing state. If the test parameters are not good, then the client fails, and goes back to the connected state.
- **Testing**; in this state, the client sends all the packets to the server, and then waits for the server's finish message before moving back to the connected state.

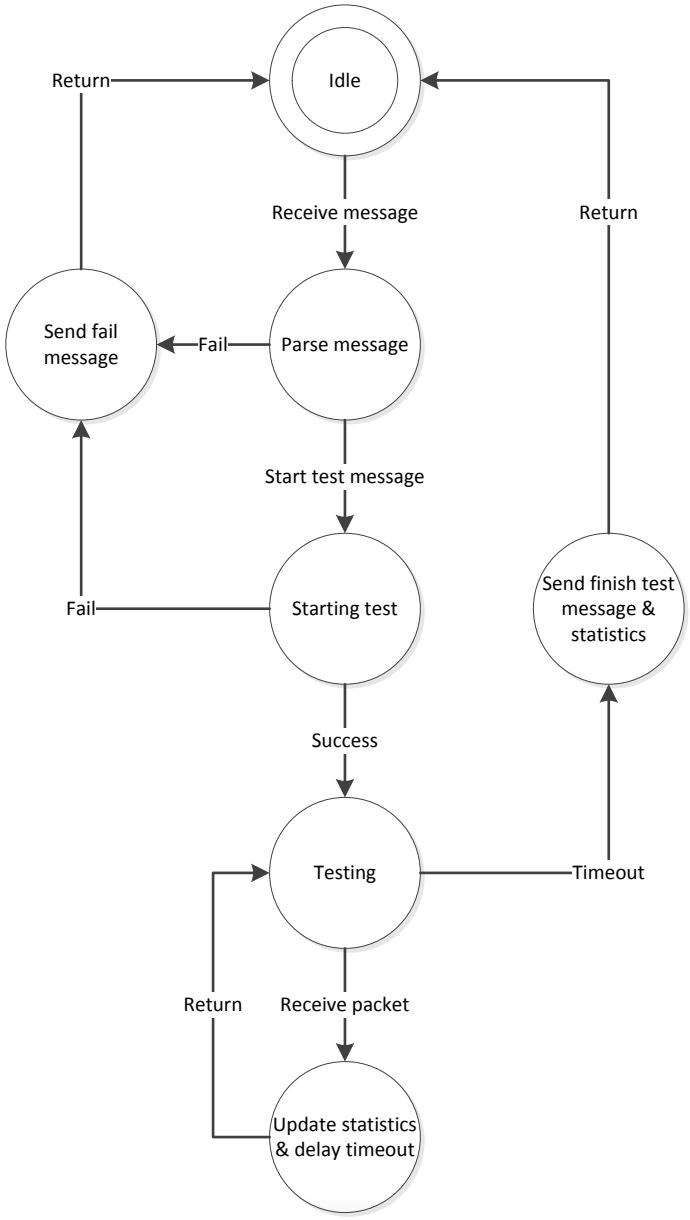
Server Application



The server application is used simply to listen for TCP connection requests. Things that connect here are considered to be control lines:

- **Idle**; the server is doing nothing. User may select start server, and that will make the server open the specified port for listening.
- **Listening**; server is listening for connection requests, and accepts them. When it accepts a new connection, it briefly goes to the Create New Control Session state.
- **Create New Control Session**; here, the server creates a new control session instance to deal with the connection, then it transitions back to the listening state, to take care of any other connections.

Server Control Session



The server control session is for taking care of both the client's control session, and test session:

- **Idle**; the session is not doing anything, and is waiting for commands from the client's control line.
- **Parse Message**; the session has received a message from the client messages here follow the control line protocol, so they need to be parsed.
- **Starting Test**; the received message was a start test message. Here, the server validates the test parameters, and sends a go ahead message to the client upon validation, and transitions to the testing state.
- **Testing**; here, the session is waiting for packets, and whenever it receives a packet, it will transition briefly to the Update statistics & delay timeout state. If the session has not received a packet for 5 seconds, it will time out, and declare the test ended.
- **Update Statistics & Delay Timeout**; here, the statistics are updates, and the test end timeout is delayed.
- **Send Fail Message**; in this state, the return code is mapped to a string, and then sent back to the client.
- **Send Finish Test Message & Statistics**; here, the session does just that, and then returns to the idle state.

Pseudocode

I'm sorry Aman! i don't have any. i learned my lesson...