PROJECT DESIGN

Revision 2

18 April 2014

Team C

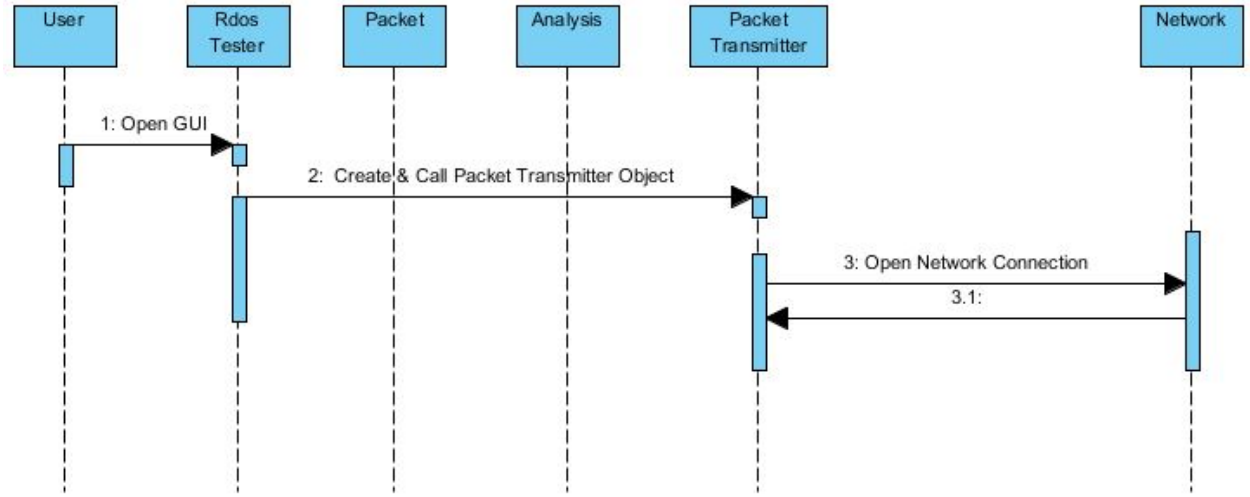
Jamie Lane, Bradley Norman, Daniel Ross

1. Introduction

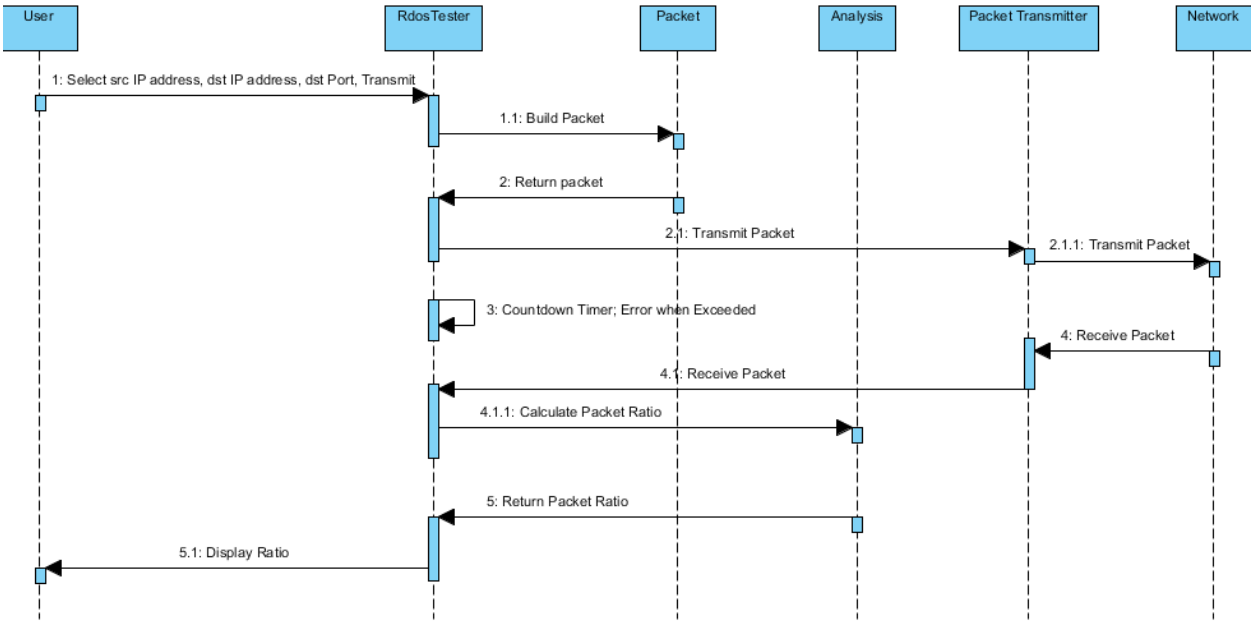
The RDoS Tester design document explains the static and dynamic sides of the design. The dynamic side of the design is shown in event-trace diagrams that demonstrate the following scenarios expected in the system: Start-up, Normal Operation, Error-handling, and Shut-down. The static side of the design is shown in the classes which have functionalities described using pseudocode.

2. Event-Trace Diagrams

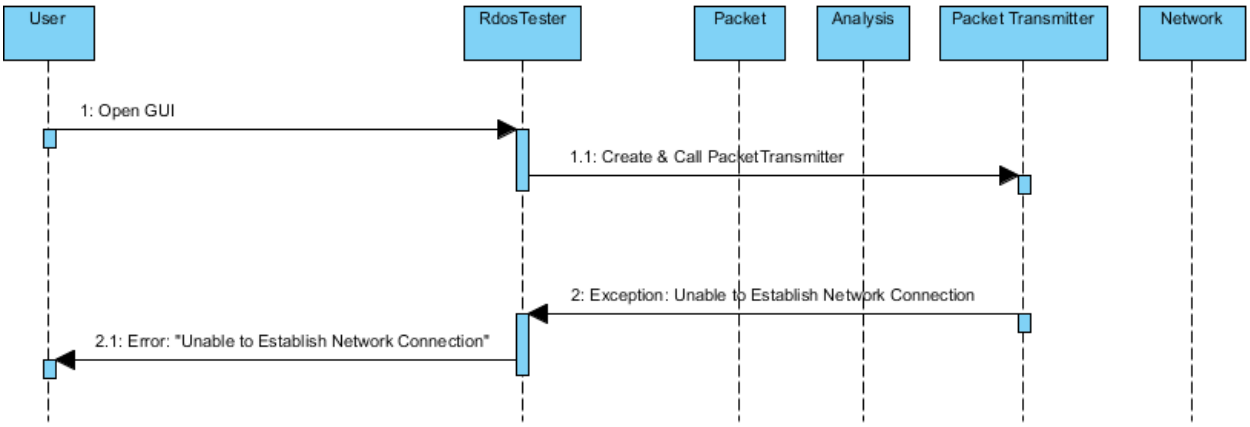
2.1 Scenario 1: Start-up



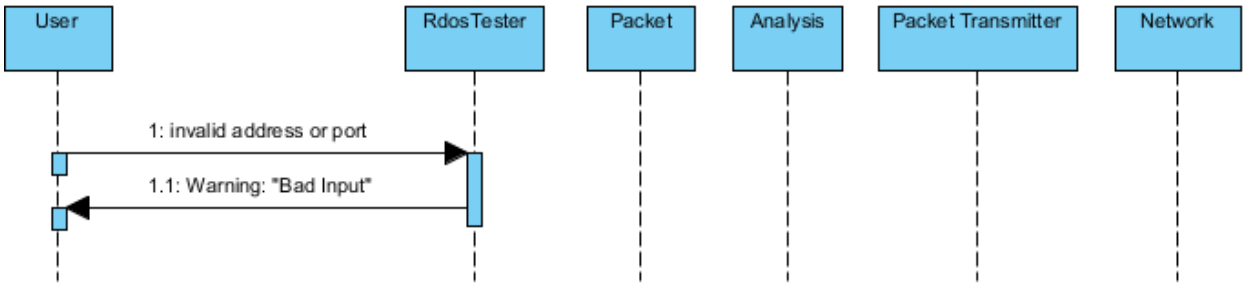
2.2 Scenario 2: Normal Operation



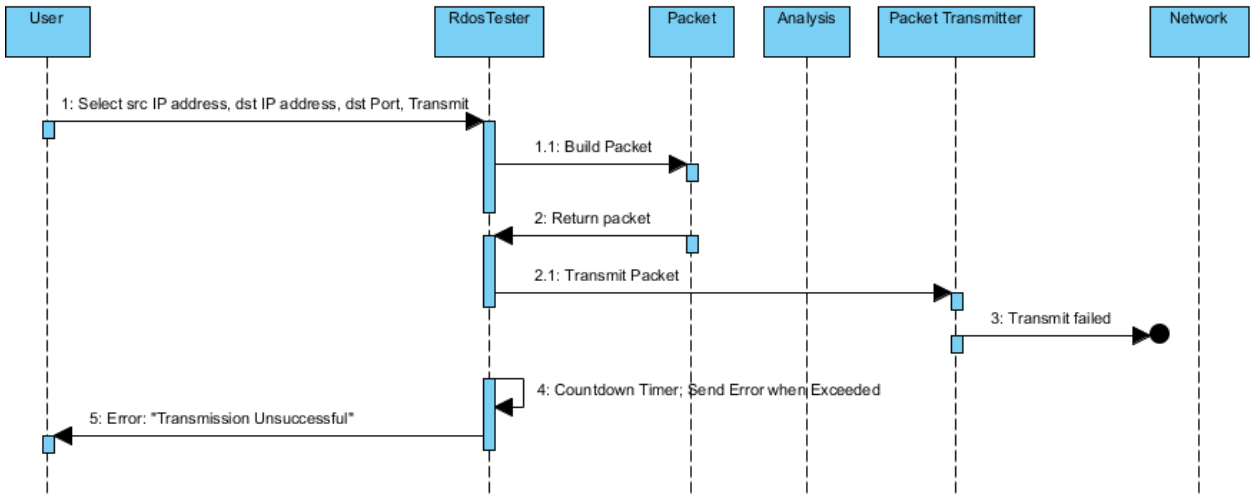
2.3 Scenario 3: Network unavailable on Start-up



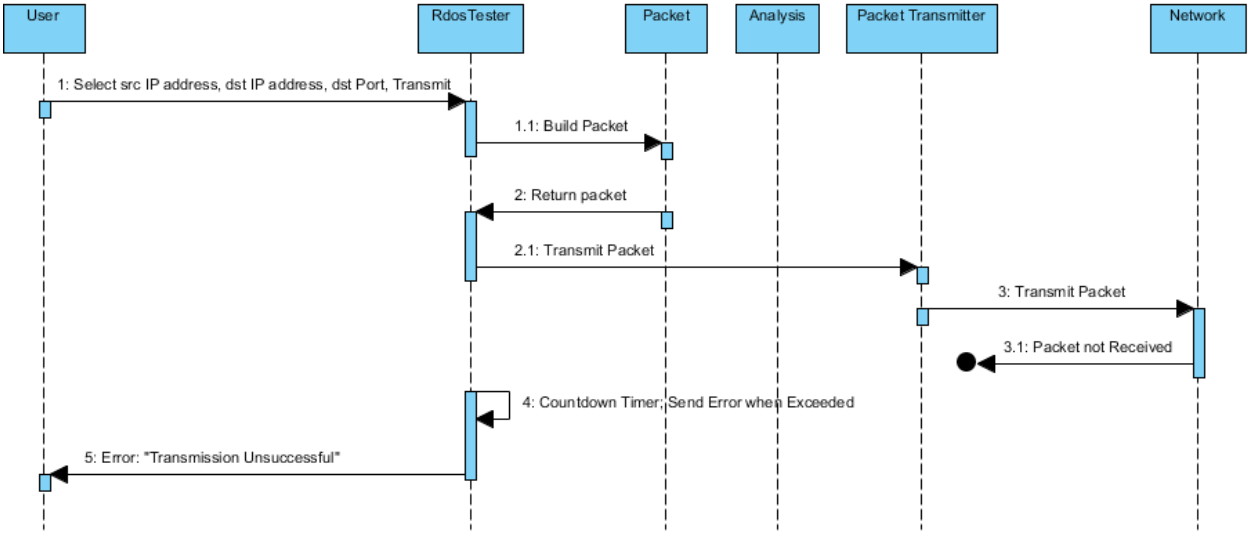
2.4 Scenario 4: Invalid user input



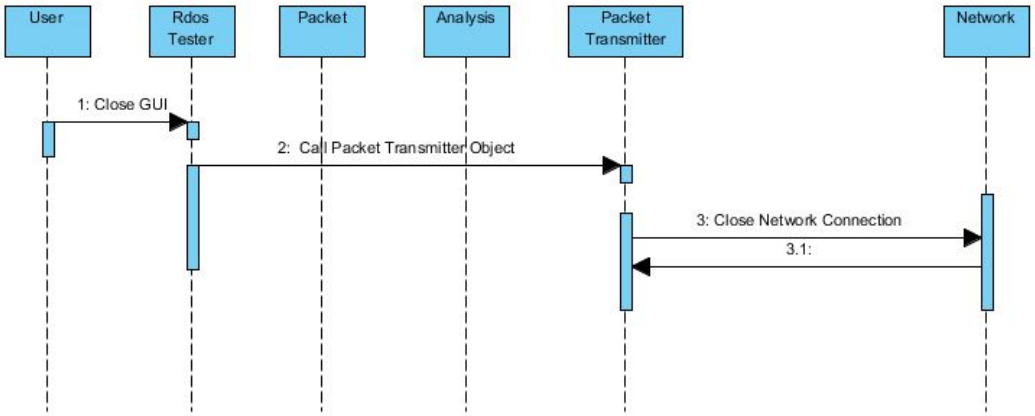
2.5 Scenario 5: Packet not transmitted



2.6 Scenario 6: Packet not received



2.7 Scenario 7: Shut-down



3. Class Design

3.1 Input/Output Subsystem

```
Class RdosTester
{
        // Initialize the text fields
        TextField srcIP1 = new TextField(size);
        set srcIP1 label “Source IP Address”;
        TextField srcIP2 = new TextField(size);
        set srcIP2 label “.“;
```

```
TextField srcIP3 = new TextField(size);
set srcIP3 label ".";
TextField srcIP4 = new TextField(size);
set srcIP4 label ".";

TextField dstIP1 = new TextField(size);
set dstIP1 label "Destination IP Address";
TextField dstIP2 = new TextField(size);
set dstIP2 label ".";
TextField dstIP3 = new TextField(size);
set dstIP3 label ".";
TextField dstIP4 = new TextField(size);
set dstIP4 label ".";

TextField port = new TextField(size);
set port label "Port";

//Create button
Button transmit = new Button;
set transmit label "Transmit";
listen to button;

//Create a status bar to display messages
Panel statusBar = new panel();

//Put the components in a panel
Panel panel = new panel();
set the layout of the panel;
add srcIP1, srcIP2, srcIP3, srcIP4 to panel;
add dstIP1, dstIP2, dstIP3, dstIP4 to panel;
add port to panel;
add button to panel;
add statusBar to panel;

void actionPerformed(action)
{
        if button was pushed
        {
                start 5-second timer;
                if fields contain valid data
                {


                        Packet originalPacket = new Packet(srcIP1 int, srcIP2 int, srcIP3 int,
                srcIP4 int, dstIP1 int, dstIP2 int, dstIP3 int, dstIP4 int
                        , port int);

                        int originalSize = get originalPacket size();
```

```
                    // try to transmit/receive packets
                    PacketTransmitter packetTransmitter = new PacketTransmitter();
                    Try {
                            packetTransmitter.open();
                            packetTransmitter.send(originalPacket);

                            Packet returnedPacket = packetTransmitter.receive();
                            int returnedSize = get returnedPacket size();
                            Analysis analysis = new Analysis( returnedSize, originalSize);
                            String ratio = analysis.getRatio();


                            message = "Returned Packet/Original Packet Ratio is: " ratio;
                    }
                    catch(network exception) {
                            message = "Network Unavailable";
                    }

                    stop timer;

                    if unsuccessful, message = "Transmission Error. Please try again.";
                    set statusBar to display message;
            }
            else
            {
                    message = "All field must contain digits only.";
                    set statusBar to display message;
            }

            if timer > 5 seconds
            {
                    stop timer;
                    message = "Packet Failed to transmit or be received. Please try again.";
                    set statusBar to display message;
            }

    }

void createAndShowGUI()
{
    //Create the window
    Window frame = new Window(window name);
    set frame to close when user hits X button;
    add the RdosTester main panel to frame;
    display the frame;
}
```

```
        void main()
        {
                run;
                createAndShowGUI();
        }

} // end class RdosTester
```

3.1.1 GUI Sample

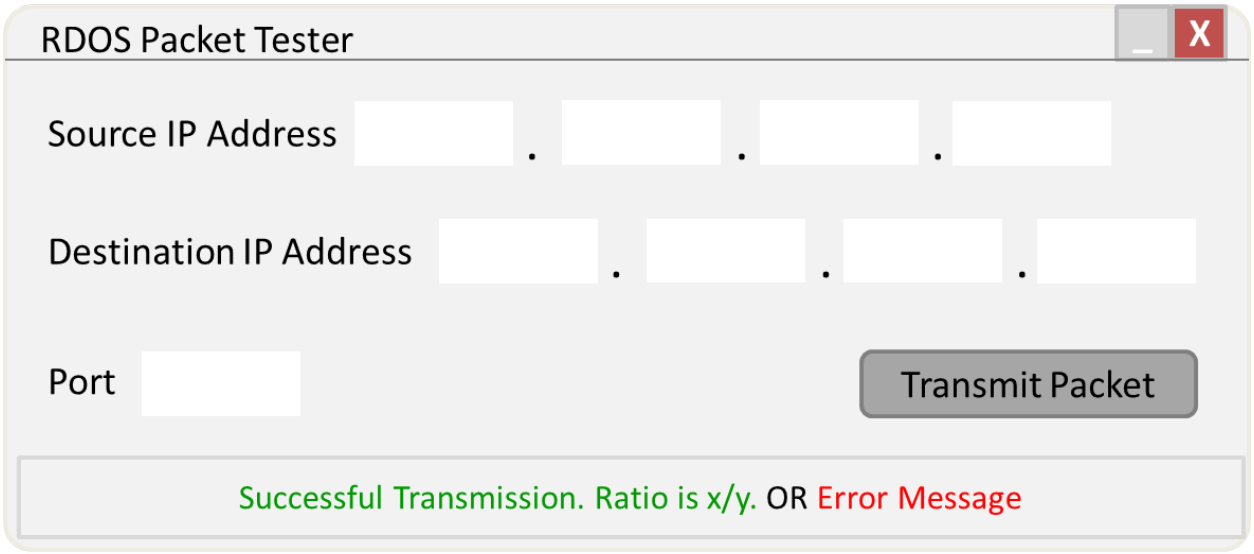


Figure 1. A sample of the GUI that will be provided by RdosTester.

3.2 Packet Subsystem

```
// required for raw socket access in Java
#include jNetPcap API;

// template for objects that represent IPv4 packets
Class Packet
{

        // variables
        String srcIp;
        String dstIp;
        String dstPort;
        String ipHeader;
        String udpPayload;
        String completePacket;
        Int packetSize;

        // constructor, for received packet
        Void packet(String completePacket)
```

```
{
        1. Set this.completePacket = completePacket;
        2. Set this.packetSize = packetSizeCalc(completePacket);
}

// constructor, for packet to transmit
Void packet(int srcIP1, int srcIP2, int srcIP3, int srcIP4, int dstIP1, int dstIP2, int dstIP3, int dstIP4,
int port)
{
        // save string representation of how address appears in hex
        1. Set this.srcIp = String((Hex)srcIP1 + (Hex)srcIP2 + (Hex)srcIP3 + (Hex)srcIP1));
        2. Set this.dstIp = String((Hex)dstIP1 + (Hex)dstIP2 + (Hex)dstIP3 + (Hex)dstIP1));
        // save string representation of how port appears in hex
        3. Set this.dstPort = String((Hex)port);
        4. Set this.ipHeader = ipHeaderMaker(srcIp, dstIp);
        5. Set this.udpPayload = udpPayloadMaker(dstPort);
        6. Set this.completePacket = combiner(ipHeader, udpPayload);
        7. Set this.packetSize = packetSizeCalc(completePacket);
}

// create IP header
String ipHeaderMaker(String srcIp, String dstIp)
{
        // call API to create header in IPv4 format
        1. Return jNetPcap.header(IP4, srcIp, dstIp);
}

// create UDP payload
String udpPayloadMaker(dstPort)
{
        1. String udpPayloadTemplate = "... status request ...";
        2. Int portOffset = 4;
        3. Overwrite contents of udpPayloadTemplate at portOffset with dstPort;
        4. Return udpPayloadTemplate;
}

// combine IP header and UDP payload to make a complete packet
String combiner(String ipHeader, String udpPayload)
{
        1. Return ipHeader concatenated with udpPayload;
}

// calculate size of packet
int packetSizeCalc(String completePacket)
{
        1. Return completePacket.size();
}
```

```
        String toString()
        {
                1. Return completePacket;
        }

} // end class Packet
```

3.3 PacketTransmitter Subsystem

```
// required for raw socket access in Java
#include jNetPcap API;

// establishes a network transmission path for complete IPv4 packets
Class PacketTransmitter
{

        // variables
        RawSocket outbound;
        RawSocket inbound;

        // constructor
        Void packetTransmitter()
        {

        }

        // open sockets
        Void open()
        {
                1. this.outbound = jNetPcap.openRawTransmit();
                2. this.inbound = jNetPcap.openRawReceive();
        }

        // send packet
        Void send(Packet transmitPacket)
        {
                1. outbound.setPacket(transmitPacket);
        }

        // receive packet
        Packet receive()
        {
                1. return inbound.getPacket();
        }

        // close sockets
        Void close()
        {
                1. Call jNetPcap.closeAll();
```

```
        }

} // end class PacketTransmitter
```

3.4 Analysis Subsystem

```
Class Analysis {

        // variables
        int originalSize;
        int receivedSize;
        float ratio;

        // constructor

        void Analysis (int receivedSize, int originalSize)
        {
                this. receivedSize = receivedSize;
                this. originalSize = originalSize;

                ratioCalculator();


        }

        // determine ratio
        void ratioCalculator()
        {
                ratio = receivedSize/originalSize;
        }

        // get ratio as a percentage
        int getRatio ()
        {
                return int(ratio * 100);

        )
}
```

4. Risk Analysis

One risk identified in the analysis of the RdosTester project was not mitigated in the design. The remaining risk is identified as the possibility of impacting the performance of an Open Arena server without the server administrator's permission. This risk will be mitigated by only testing with a server hosted by a member of the development team.