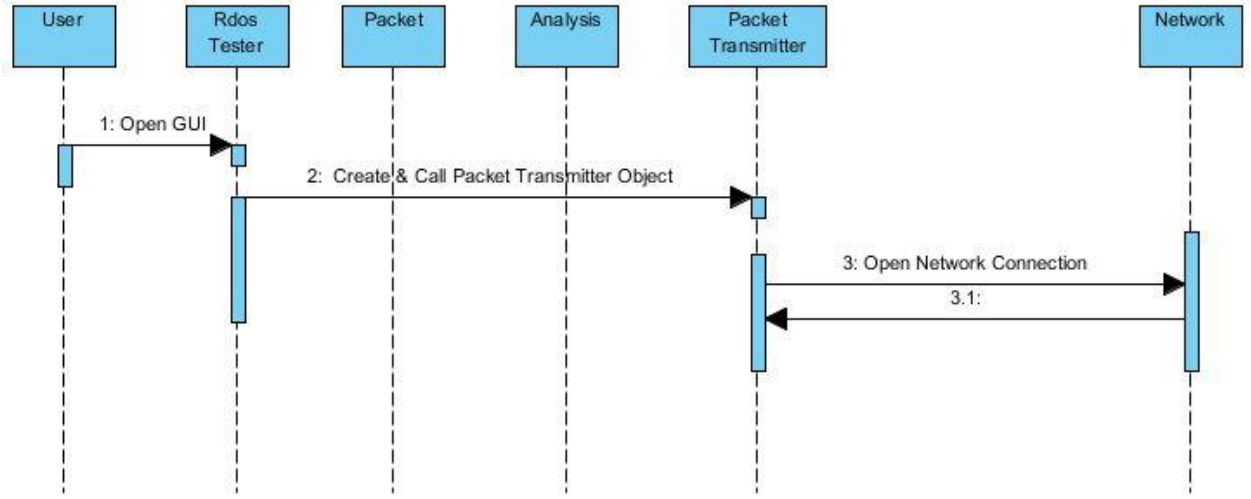
PROJECT DESIGN
13 April 2014
Team C
Jamie Lane, Bradley Norman, Daniel Ross

Introduction

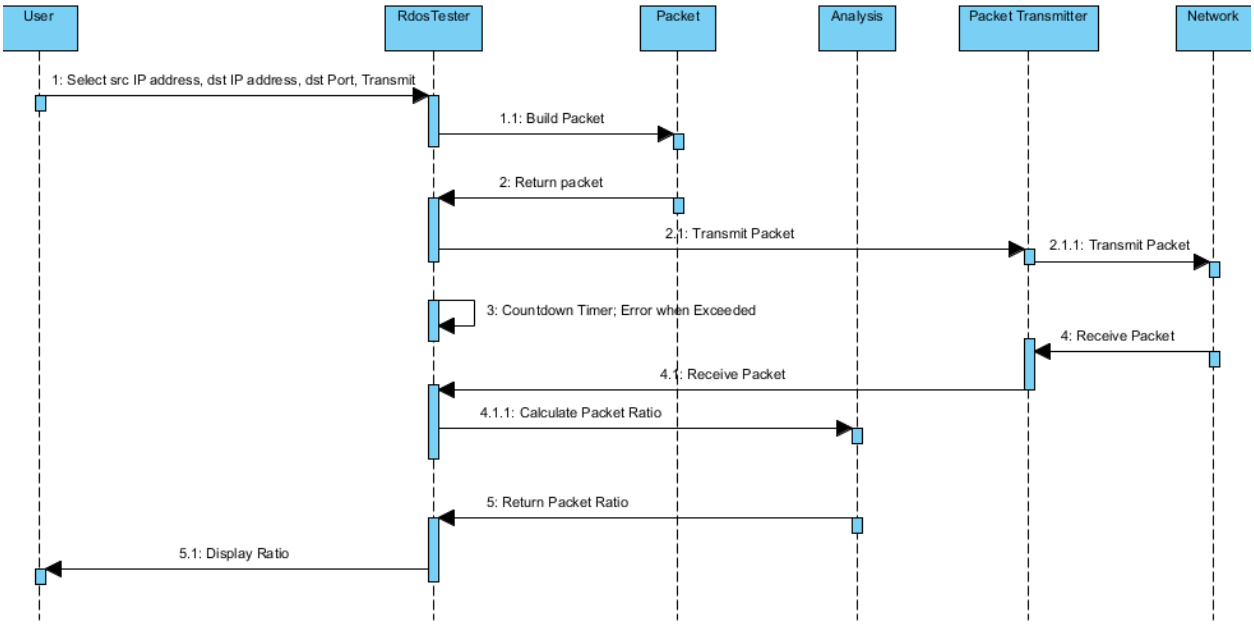
The RDoS Tester design document explains the static and dynamic sides of the design. The dynamic side of the design is shown in event-trace diagrams that demonstrate the following scenarios expected in the system: Start-up, Normal Operation, Error-handling, and Shut-down. The static side of the design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

Event-Trace Diagrams

Scenario 1: Start-up

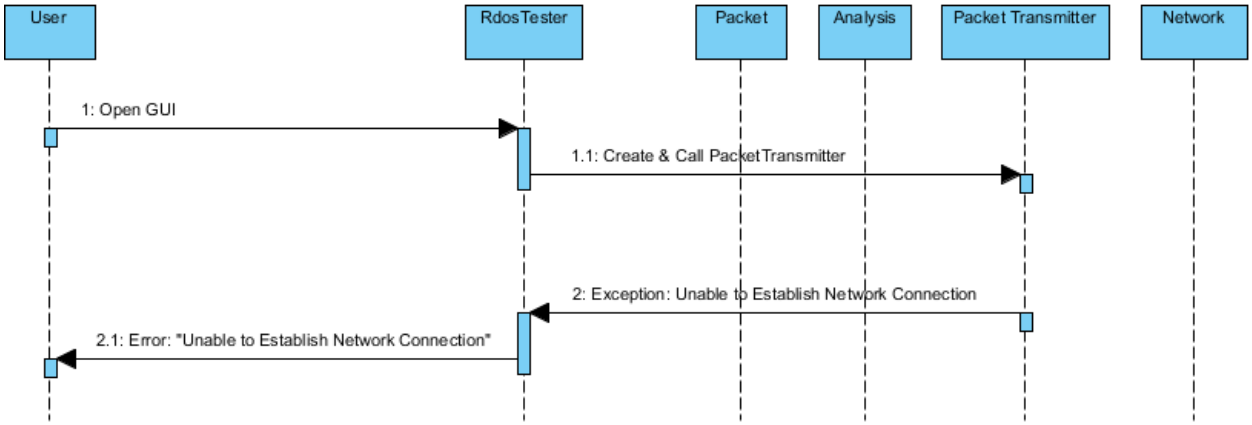


Scenario 2: Normal Operation

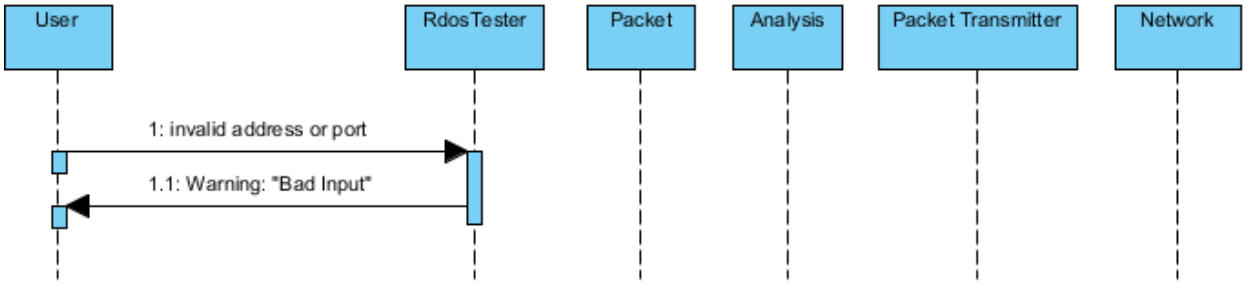


Scenario 3: Error-Handling

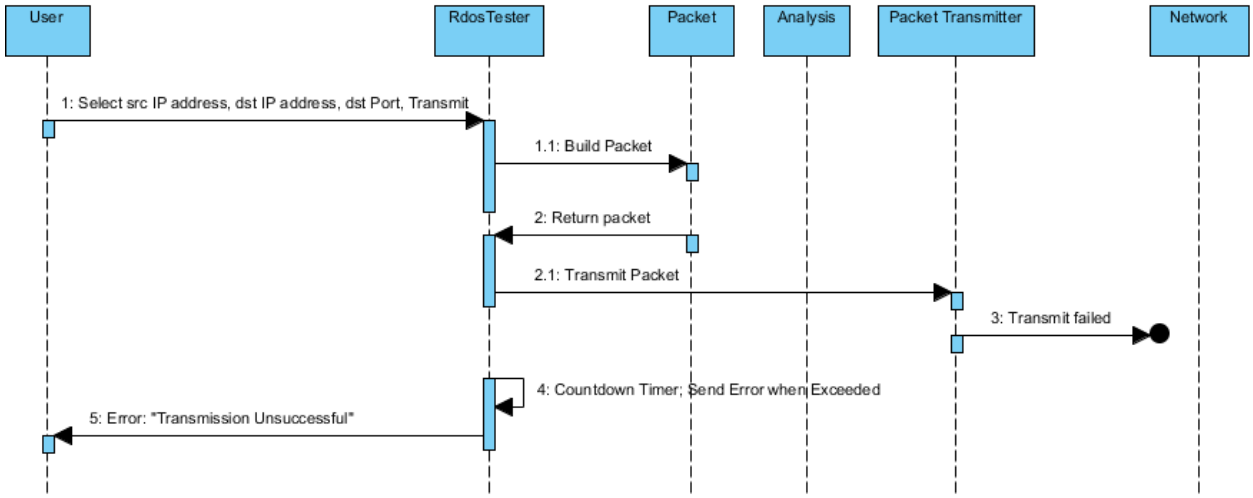
Network unavailable on Start-up



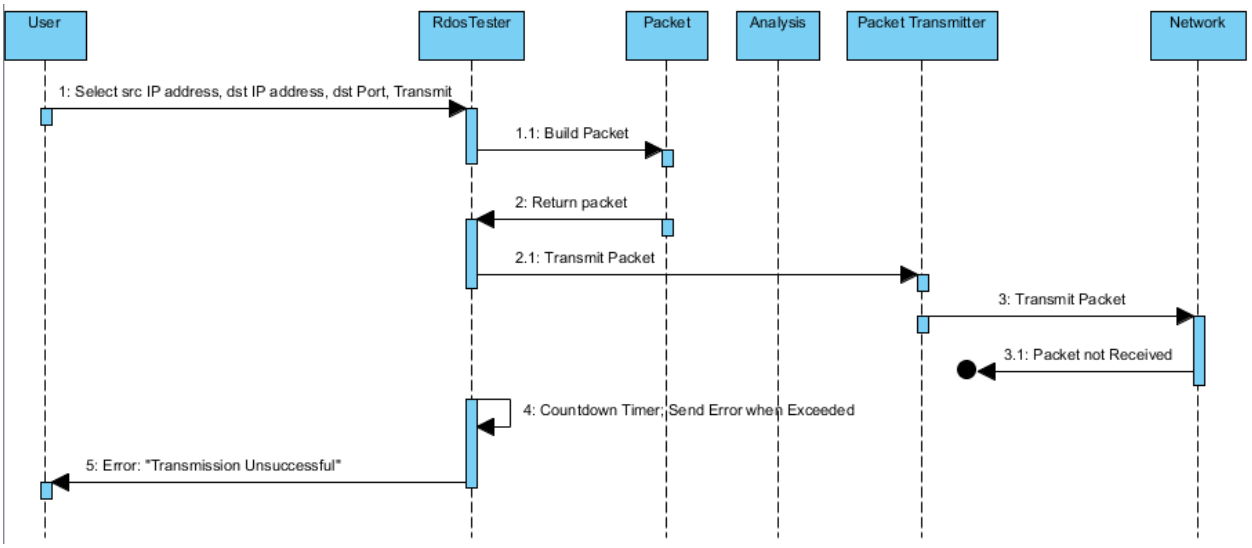
Invalid user input



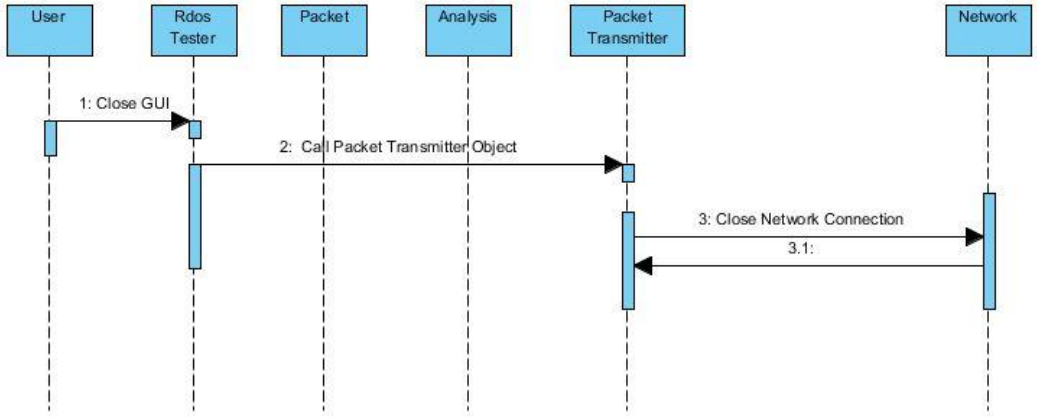
Packet not transmitted



Packet not received



Scenario 4: Shut-down



Class Design

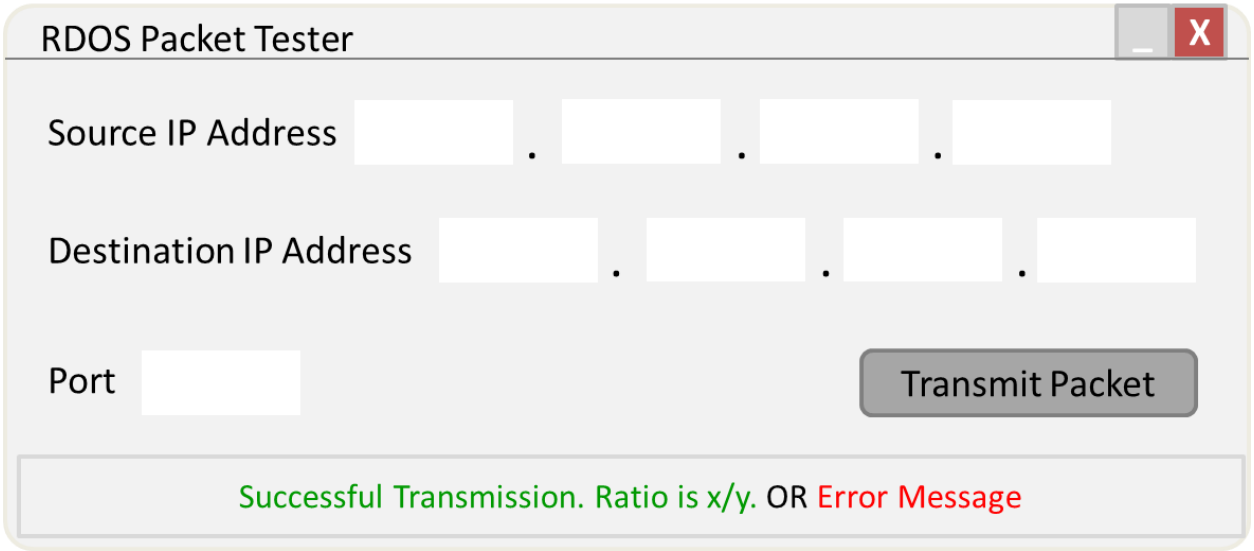
Input/Output Subsystem

Class RdosTester
{

```
// Initialize the text fields
TextField sourceIP1 = new TextField(size);
set sourceIP1 label “Source IP Address”;
TextField sourceIP2 = new TextField(size);
set sourceIP2 label “.“;
TextField sourceIP3 = new TextField(size);
set sourceIP3 label “.“;
TextField sourceIP4 = new TextField(size);
set sourceIP4 label “.“;

TextField destinationIP1 = new TextField(size);
set destinationIP1 label “Destination IP Address“;
TextField destinationIP2 = new TextField(size);
set destinationIP2 label “.“;
TextField destinationIP3 = new TextField(size);
set destinationIP3 label “.“;
TextField destinationIP4 = new TextField(size);
set destinationIP4 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 sourceIP1, sourceIP2, sourceIP3, sourceIP4 to panel;
add destinationIP1, destinationIP2, destinationIP3, destinationIP4 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
                {
                        concatenate sourceIP text fields;
                        concatenate destinationIP text fields;
                        Packet originalPacket = new Packet(concatenated sourceIP,
                                concatenated destinationIP, port);
                        int originalSize = get originalPacket size();

                        try to transmit/receive packets
                        PacketTransmitter packetTransmitter = new
                        PacketTransmitter(originalPacket);
                        Packet returnedPacket = get packetTransmitter returned packet;
                        int returnedSize = get returnedPacket size();
                        double ratio = Analysis(originalSize, returnedSize);
                        message = "Received Packet/Original Packet Ratio is: " ratio;
                        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();
    }
}
```



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(String srcIp, String dstIp, String dstPort)
{
        1. Set this.srcIp = srcIp;
        2. Set this.dstIp = dstIp;
        3. Set this.dstPort = dstPort;
        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)
{
        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
```

Transmission 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(transmitPacket);
        }

        // receive packet
        Void receive(Packet transmitPacket)
        {
                1. inbound(transmitPacket);
        }
```

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

} // end class PacketTransmitter
```

Analysis Subsystem

```
Class PacketCalculator {

        // variables
        int packetInSize;
        int packetOutSize;
        float ratio;

        // determine packet size and ratio
        void analysis (Packet packetIn, Packet packetOut)
        {
                packetSizeCalculator(packetIn, packetOut);
                ratioCalculator();

        }

        // determine ratio
        void ratioCalculator()
        {
                ratio = packetInSize/packetOutSize;
        }

        // determine packet size
        void packetSizeCalculator (Packet packetIn, Packet packetOut)
        {
                packetInSize = packetIn.size();
                packetOutSize = packetOut.size();
        }

        // get ratio
        float getRatio ()
        {
                return ratio;

        )
}
```

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.