PROJECT DESIGN

Revision 2.2

08 May 2014

Team C

Jamie Lane, Bradley Norman, Daniel Ross

Revision History

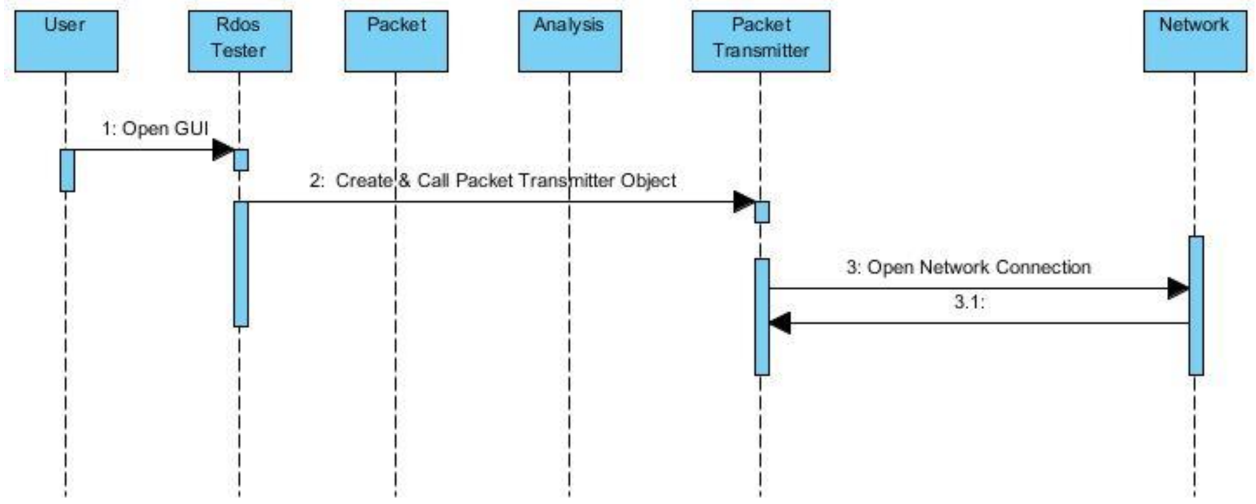
Date	Revision	Description
04/10/2014	0.1	Initial draft
04/11/2014	0.2	Updated normal operations event trace
04/12/2014	0.5	Added event handling event trace Added packet pseudocode Added packet transmitter pseudocode
04/13/2014	0.8	Updated rdosTester pseudocode Updated normal operations event trace
04/13/2014	1.0	Added start-up event trace Added shut-down event trace Added analysis pseudocode
04/13/2014	1.2	Updated document formatting
04/16/2014	1.6	Updated packet pseudocode Updated packet transmitter pseudocode Updated analysis pseudocode Updated rdosTester pseudocode
04/18/2014	2.0	Updated packet pseudocode Updated packet transmitter pseudocode Updated analysis pseudocode Updated rdosTester pseudocode
05/07/2014	2.1	Replace normal, packet not transmitted, and packet not received event trace diagrams. Updated pseudocode for RdosTester class.
05/08/2014	2.2	Updated PacketTransmitter pseudocode.

1. Introduction

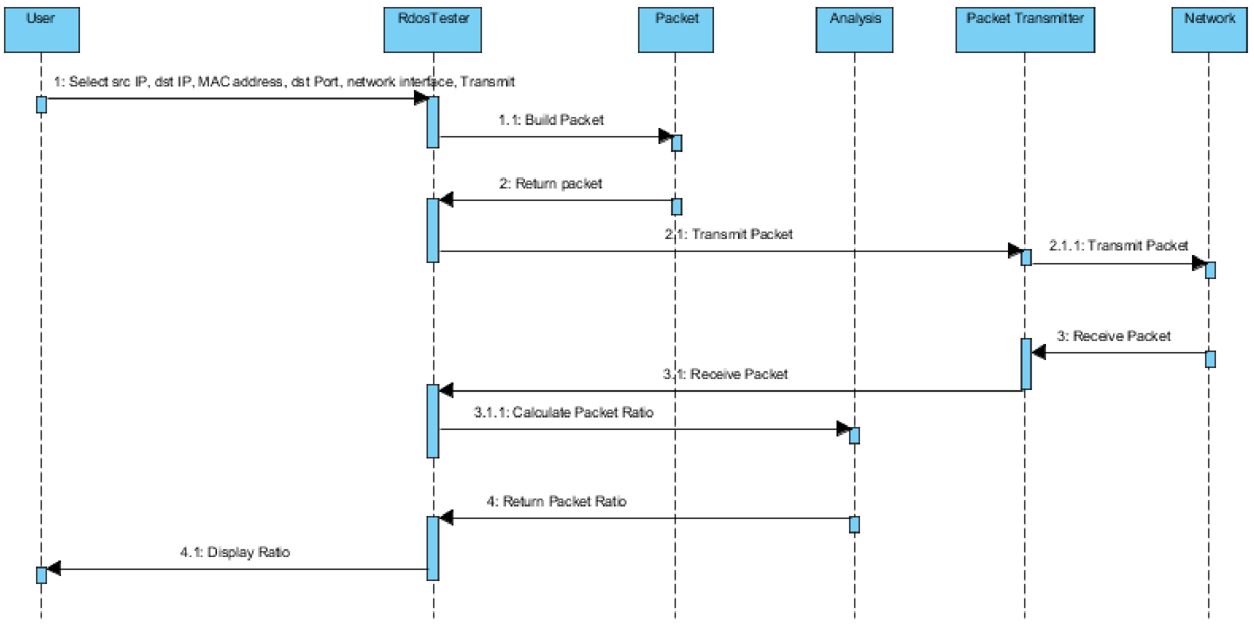
The RDoS Tester design document explains the static and dynamic sides of the design. The dynamic side of the design is shown in event-trace diagrams that demonstrate the following scenarios expected in the system: Start-up, Normal Operation, Error-handling, and Shut-down. The static side of the design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

2. Event-Trace Diagrams

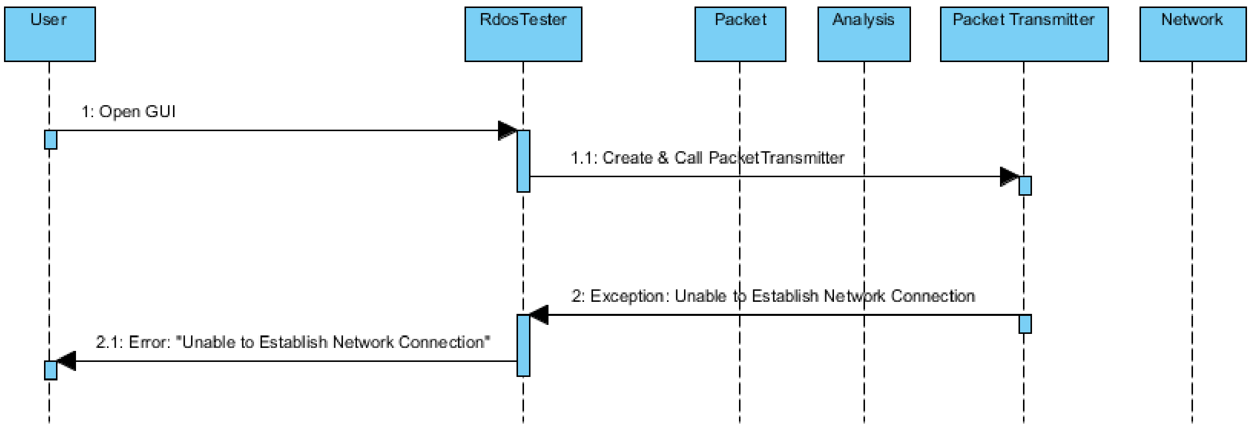
2.1 Scenario 1: Start-up



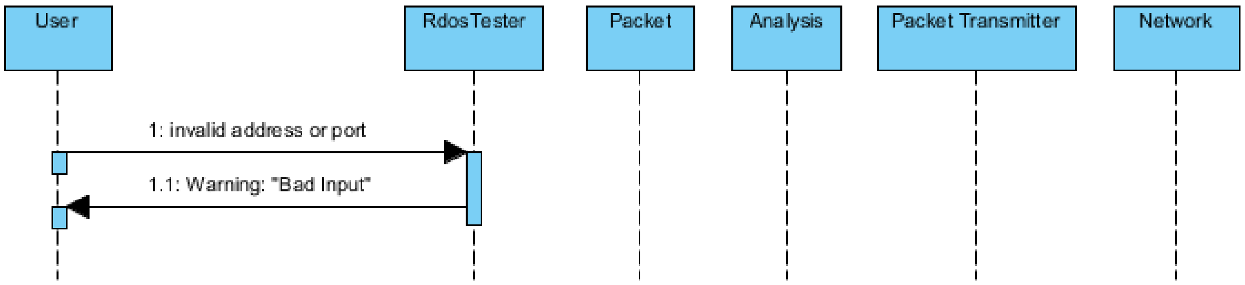
2.2 Scenario 2: Normal Operation



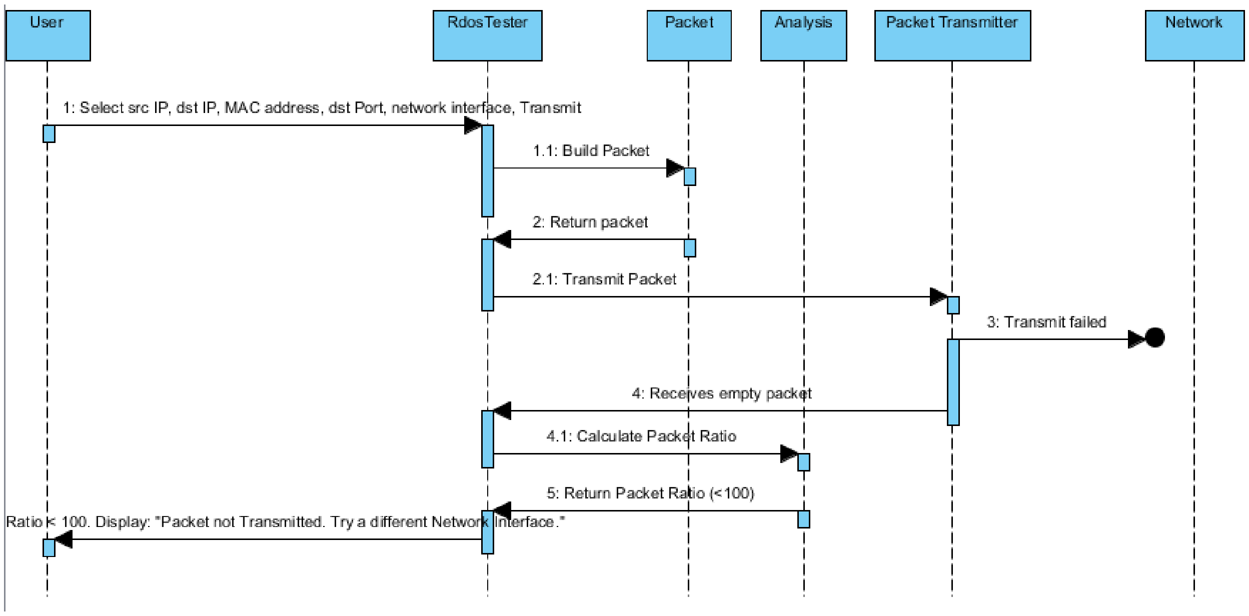
2.3 Scenario 3: Network unavailable on Start-up


User
RdosTester
Packet
Analysis
Packet Transmitter
Network
1: Open GUI
1.1: Create & Call PacketTransmitter
2: Exception: Unable to Establish Network Connection
2.1: Error: "Unable to Establish Network Connection"

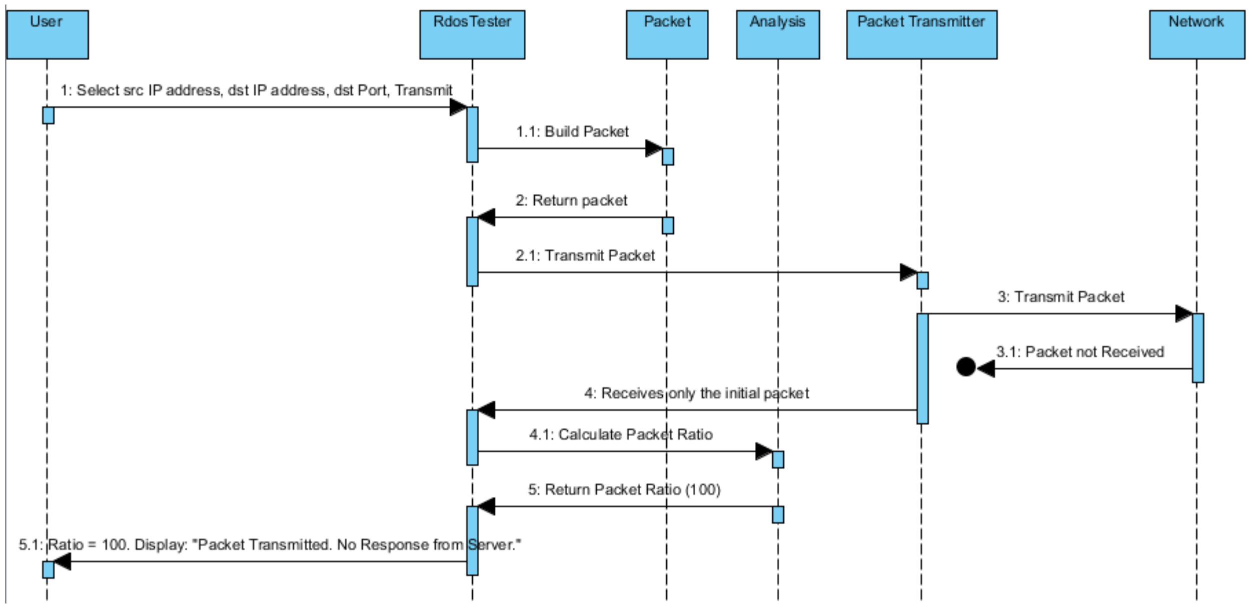
2.4 Scenario 4: Invalid user input



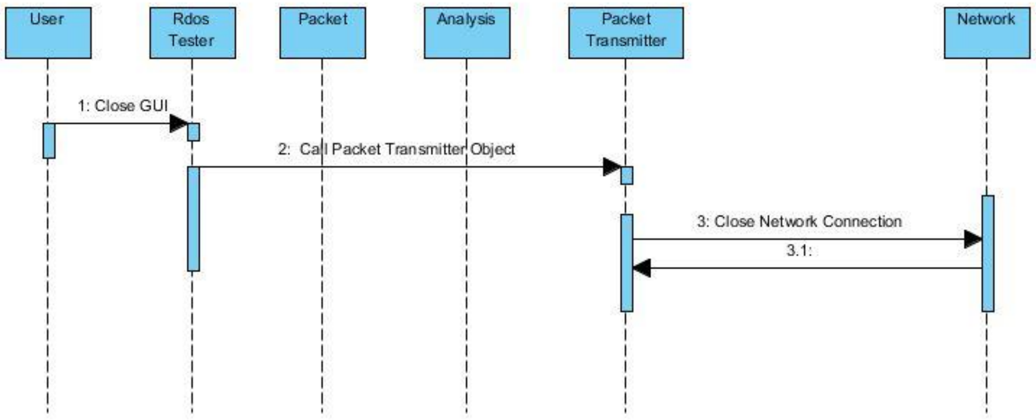
2.5 Scenario 5: Packet not transmitted


User
RdosTester
Packet
Analysis
Packet Transmitter
Network
1: Select src IP, dst IP, MAC address, dst Port, network interface, Transmit
1.1: Build Packet
2: Return packet
2.1: Transmit Packet
3: Transmit failed
4: Receives empty packet
4.1: Calculate Packet Ratio
5: Return Packet Ratio (<100)
Ratio < 100. Display: "Packet not Transmitted. Try a different Network Interface."

2.6 Scenario 6: Packet not received


User
RdosTester
Packet
Analysis
Packet Transmitter
Network
1: Select src IP address, dst IP address, dst Port, Transmit
1.1: Build Packet
2: Return packet
2.1: Transmit Packet
3: Transmit Packet
3.1: Packet not Received
4: Receives only the initial packet
4.1: Calculate Packet Ratio
5: Return Packet Ratio (100)
5.1: Ratio = 100. Display: "Packet Transmitted. No Response from Server."

2.7 Scenario 7: Shut-down



3. Class Design

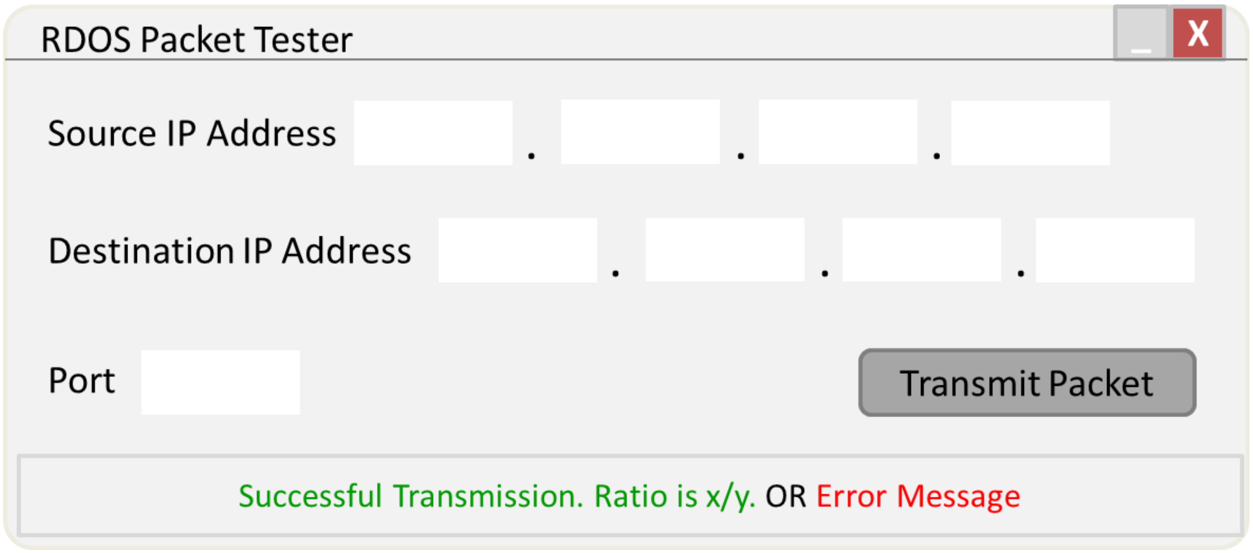
3.1 Input/Output Subsystem

```
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 mac1 = new TextField(size);
        set mac1 label “Gateway MAC  Address“;
        TextField mac2 = new TextField(size);
        set mac2 label “.“;
        TextField mac3 = new TextField(size);
        set mac3 label “.“;
        TextField mac4 = new TextField(size);
        set mac4 label “.“;
        TextField mac5 = new TextField(size);
        set mac5 label “.“;
        TextField mac6 = new TextField(size);
        set mac6 label “.“;



        TextField port = new TextField(size);
        set port label “Port”;

        ComboBox network = new ComboBox(size);
        set network label “Network Interface”;
        get network interface from PacketTransmitter;
        populate combobox with network interfaces;
```

```
//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 mac1, mac2, mac3, mac4, mac5, mac6 to panel;
add port to panel;
add combobox to panel;
add button to panel;
add statusBar to panel;

void actionPerformed(action)
{
        if button was pushed
        {
                clear the previous status bar

                validate data


                        Packet originalPacket = new Packet(srcIP1 int, srcIP2 int, srcIP3 int,
                srcIP4 int, dstIP1 int, dstIP2 int, dstIP3 int, dstIP4 int, mac1, mac2, mac3, mac4,
                mac5, mac6
                        , port int, networkInterface);

                        int originalSize = get originalPacket size();

                        // try to transmit/receive packets
                        PacketTransmitter packetTransmitter = new PacketTransmitter();
                                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”;
                    }

                    if(ratio < 100)
                        tell user the packet was not transmitted;
                    elseif(ratio == 100)
                        tell user the packet was not received;
                    else
                        tell user the ratio of the received/sent packet;




            }

        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



Successful Transmission. Ratio is x/y. OR Error Message

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;

        // hold list of network interfaces
        Array[] networkInterfaces;

        // hold source MAC address
        byte[] sourceMacAddress;

        // receive packet contents, temporary
        receivePacket;

        // constructor
        Void packetTransmitter()
        {
                networkInterfaces = populate with list of network interfaces, from jNetPcap;

        }

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

        // send packet
        Void send(Packet transmitPacket)
        {
                1. update Ethernet header with source and destination MAC
                2. update Ethernet checksum
                3. update IP header checksum
                4. update UDP header checksum
                5. outbound.setPacket(transmitPacket);
                6. create filter for capturing packets, based on port and protocol
                7. capture packets
                8. store last received packet
        }

        // receive packet
        Packet receive()
        {
                1. format receivePacket;
                2. return receivePacket;
        }

        // 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.