**1.INTRODUCTION**

One of the most vexing cyber security threats of today is use of very large coordinated groups of hosts for brute force attacks. These large groups of hosts are assembled by turning vulnerable hosts into so-called zombies, or bots, after which they can be controlled from afar. A collection of bots, when controlled by a single command and control (C&C) infrastructure, form what is called a botnet. Most important component of a botnet is its C&C channel because botmaster uses the C&C channel to issue commands to their bots and receive information from the compromised machines. Botnets may structure their C&C channels in different ways. In a centralized architecture, all bots in a botnet contact one (or a few) C&C server(s) *owned* by the botmaster. However, fundamental disadvantage of centralized C&C servers is that they represent a *single point of failure*. In order to overcome this problem, botmaster have recently started to build botnets with a more resilient C&C architecture, using a peer-to-peer (P2P) structure or hybrid P2P/centralized C&C structures. Bots belonging to a P2P botnet form an overlay network in which any of the nodes can be used by the botmaster to distribute commands to the other peers or collect information from them.

Although P2P botnets are more complex, and more costly to manage compared to centralized botnets, they offer higher resiliency against take-down since even if a significant portion of bots in a P2P botnet are disrupted the remaining bots may still be able to communicate with each other and with the botmaster.

The potential for orders of magnitude more damage exists in the future so detecting botnet is of great importance. A number of methods exist to detect and stop botnets, and these methods continue to mature. But detecting P2P botnet in particular is very difficult manly due to three factors first, P2P botnet traffic can easily blend with legitimate P2P traffic like that of utorrent, skype, Freecast, bit torrent etc. second, P2P botnets nowadays work in stealthy manner which are very hard to detect specially due to encryption and third as the volume of traffic to be analyzed increases efficiently its becomes difficult for existing botnet detection systems to work efficiently in a given time.

In our project we present a botnet detection system which is capable of detecting stealthy P2P botnet. Our system instead on relying on the malicious activity performed by bots in response to the command sent by the botmaster but focuses on the communication pattern between botmaster and its bots specifically by deriving a fingerprint. Our system then uses this fingerprint to identify P2P bots.

We first look out for P2P traffic, build statistical fingerprint to profile P2P traffic and then calculate their active time, use those fingerprint and active time to compare with fingerprint of an actual botnet and with the a threshold active time, form fingerprint clusters using k-mean algorithm and identify the botnet

**1.1.Motivation**

Sharing our common interest of Network Security and Ethical hacking, we decided to make our final year project on network security domain. We started researching for most dreaded security breaches and attack out of all the attacks Distributed denial of service (DDOS) attack came to top of our list. Every big company or websites have at one or other time are attacked by DDOS and are still vulnerable. So we decided to work on DDOS attacks and tried to come with a novel idea on how to prevent such attacks.

While researching more about DDOS attacks we came to the root cause of the modern DDOS attacks i.e Botnets. Studying more about botnet revealed as that botnets are behind almost every big attacks like that of DDOS attack, brute force cracking, spamming, phishing etc. Although there were methods available to detect botnet but with time and intelligence of hackers these methods started to become ineffective.

Along with the interest in field of network security and challenges provided by the expanding kingdom of illegal botnets were the prime motivation to develop a P2P botnet detection system.

**1.2.Objective of the work**

Objective of our project is to

* Successfully detect nodes which are bots or compromised host in a network.
* Successfully detect the botmaster sending commands to the botmaster.
* Detection system must be simple without much overhead and must be efficient.
* Build a java application to demonstrate the effectiveness of our botnet detection system

**1.3.Problem Statement**

Recently, botnets have become the root cause of many Internet attacks including much dreaded DDOS attacks. Traditionally botnet were using a centralized server for communicating but now a days Botnet have started using P2P technology as P2P botnets are harder to track and takedown as they have no single point of failure. We try to find proper countermeasures and build novel botnet detection system for P2P botnet.

**1.4.Methodology Adopted**

Since the P2P botnet uses P2P protocol to establish a C&C channel and communicate with the botmaster. Therefore P2P bots exhibit some network traffic patterns that are common to other P2P client applications (either legitimate or malicious).So we divide our systems into two phases. In the first phase, we aim at detecting all hosts within the monitored network that engage in P2P communications that might be botnet.. We analyze P2P traffic and extract a number of statistical features to identify flows generated by P2P clients like packet sent, packet received, bytes sent, byte received and protocol used. In the second phase, our system analyzes the traffic generated by the P2P clients and classifies them into either *legitimate* P2P clients or P2P *bots*. Specifically, we investigate the active time of a P2P client and identify it as a candidateP2P bot if it is persistently active on the underlying host. We further analyze the overlap of peers contacted by two *candidate* P2P bots to finalize detection. To illustrate the statistical features and motivate the related thresholds used by our system, we ran some popular P2P applications, including Bittorrent, µTorrent, Skype, and Ares, for 24 hours to collect their traffic traces. In this section we report a number of measurements on the obtained traffic traces to better motivate the design of statistical features.

**1.5.Organization of Report**

In the next Chapter (2) we present a survey of the related work. The Chapter that follows (2) consists of system requirement and specification. Chapter (4) consists of system analysis where we talk about existing system and the proposed system, Chapter (5) deals with Source code. Chapter (6) deals with Snapshots and Chapter (7) deals with testing, verification and validation. Chapter (8) has our results, Chapter (9) gives conclusion and future enhancement, Chapter (10) has all the referenced use.

[The other supporting information and the source code are gathered in the Appendix.]

**2.Literature Survey.**

We have surveyed various papers related to botnets and detection of botnets. The various papers are as follows: -

[1] Junjie Zhang  Perdisci, R. ; Wenke Lee ; Xiapu Luo; Sarfraz, U.”**Building a Scalable System for Stealthy P2P-Botnet Detection**” **in:**Information Forensics and Security, IEEE Transactions on  (Volume:9 ,  Issue: 1 ) JAN 2014 pp 27-38

Abstract: Peer-to-peer (P2P) botnets have recently been adopted by botmasters for their resiliency against take-down efforts. Besides being harder to take down, modern botnets tend to be stealthier in the way they perform malicious activities, making current detection approaches ineffective. In addition, the rapidly growing volume of network traffic calls for high scalability of detection systems. In this paper, we propose a novel scalable botnet detection system capable of detecting stealthy P2P botnets. Our system first identifies all hosts that are likely engaged in P2P communications. It then derives statistical fingerprints to profile P2P traffic and further distinguish between P2P botnet traffic and legitimate P2P traffic. The parallelized computation with bounded complexity makes scalability a built-in feature of our system. Extensive evaluation has demonstrated both high detection accuracy and great scalability of the proposed system.

[2] Han Zhang ; Papadopoulos. C. ; Massey. D. .“Detecting encrypted botnet traffic” in Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference

Abstract: Bot detection methods that rely on deep packet inspection (DPI) can be foiled by encryption. Encryption, however, increases entropy. This paper investigates whether adding highentropy detectors to an existing bot detection tool that uses DPI can restore some of the bot visibility. We present two high-entropy classifiers, and use one of them to enhance BotHunter. Our results show that while BotHunter misses about 50% of the bots when they employ encryption, our high-entropy classifier restores most of its ability to detect bots, even when they use encryption.

[3] Stefan R, Pierfrancesco U, Andreas B, Alessandro :”Botnet detection revisited: theory and practice of finding malicious P2P networks via Internet connection graphs” in 2013 IEEE conference on Digital Object Identifier

Abstract: n this paper we review state-of-the-art botnet detection algorithms that reveal the control traffic of malicious peer-to-peer (P2P) networks by targeting topological properties of their interconnectivity graph. This class of detection methods does not rely on the exchanged content and therefore is also applicable to encrypted control traffic. However, in practice, an ISP monitoring customer traffic over an edge router will usually see only a fraction of the overall botnet, thus restricting the available bot connectivity information and limiting the applicability of general community detection approaches. In this paper we critically review graph based detection methods suitable for edge router monitoring using two types of real network traces. We show experimentally that using meta-graphs of mutual contacts proposed by Coskun et al. 2010 has the highest potential on result quality. We improve this approach by presenting a computationally less complex algorithm with similar result quality. Furthermore we explain ways to alleviate the cost of dealing with false positives in the result set.

[4] Garant, D. ; Wei Lu : “Mining Botnet Behaviors on the Large-scale Web Application Community” in 2013 27th International Conference on Digital Object Identifier.

Abstract: Botnets are networks of compromised computers controlled under a common command and control channel. Recognized as one of the most serious security threats on current Internet infrastructure, botnets are often hidden in existing applications, e.g. IRC, HTTP, or peer-to-peer, which makes botnet detection a challenging problem. In this paper we propose a new, centralized, fully-encrypted, botnet system called Weasel. A set of signatures are examined and formalized to differentiate the behaviors of Weasel and normal web applications. Through these signatures, we apply a set of data mining techniques to detect the web based botnet behaviors on a web application community formed on a campus backbone network. The proposed approach was evaluated with over 400 thousand flows collected over seven consecutive days on a large scale network and results show the proposed approach successfully detects the botnet flows with a high detection rate and an acceptably low false alarm rate.

[5] Mohammed Jamil Elhalabi, Selvakumar Manickam,Loai Bani Melhim, Mohammed Anbar and Huda Alhalabi **“A REVIEW OF PEER-TO-PEER BOTNET DETECTION TECHNIQUES”** in Journal of Computer Science Jan 2014

Abstract: In recent years, Peer-to-Peer technology has an extensive use. Botnets have exploited this technology efficiently and introduced the P2P botnet, which uses P2P network for remote control of its bots and become one of the most significant threats to computer networks. They are used to make DDOS attacks, generate spam, click fraud and steal sensitive information. Compared with traditional botnets, P2P botnets are harder to be defended and hijacked. In this study we discuss various P2P botnet detection approaches and evaluate their effectiveness. We identify the advantages and shortcomings of each of the discussed techniques. This can guide the researchers to a better understanding of P2P botnets and easier for them developing more sufficient detection techniques. Our evaluation shows that each technique has its own advantages and limitations. Two or more detection techniques might be used together, in order to have a robust P2P botent detection.

**3.Botnet**

A botnet is a group of security-compromised computers infected with malicious computer applications called bots. Once infected with a bot and configured according to the attackers specifications, an attacker can gain complete control of the infected machine. The attacker or the propagator of a botnet is also known as botnet herder or botmaster.

While botnets are often named after the malware that created them, multiple botnets typically use the same malware, but are operated by different entities. The term "botnet" can be used to refer to any group of computers, such as IRC bots, but the term is generally used to refer to a collection of computers (called “bot” zombie computers) that have been recruited by running malicious software. A botnet's originator (known as "bot master") can control the group remotely, usually through an IRC, and often for criminal purposes. This server is known as the command-and-control (C&C) server. Though rare, more experienced botnet operators program command protocols from scratch. These protocols include a server program, a client program for operation, and the program that embeds the client on the victim's machine.

These communicate over a network, using a unique encryption scheme for stealth andprotection against detection or intrusion into the botnet. A bot typically runs hidden and uses a covert channel (e.g. Twitter, or IM) to communicate with its C&C server. Generally, the perpetrator has compromised multiple systems using various tools (exploits, buffer overflows, as well as others). Newer bots can automatically scan their environment and propagate themselves using vulnerabilities and weak passwords. Generally, the more vulnerability a bot can scan and propagate through, the more valuable it becomes to a botnet controller community. The process of stealing computing resources as a result of a system being joined to a "botnet" is sometimes referred to as "scrumping." Botnet servers are typically redundant, linked for greater redundancy so as to reduce the threat of a takedown.Actual botnet communities usually consist of one or several controllers that rarely have highly developed command hierarchies; they rely on individual peer-to-peer relationships.

Botnet architecture evolved over time, and not all botnets exhibit the same topology for command and control. Advanced topology is more resilient to shutdown, enumeration or discovery. However, some topologies limit the marketability of the botnet to third parties. Typical botnet topologies are Star, Multi-server, Hierarchical and Random. To thwart detection, some botnets are scaling back in size. As of 2013, the average size of a network was estimated at 70,000 computers.

**3.1 Formation**

This example illustrates how a botnet is created and used to send email spam.



* A botnet operator sends out viruses or worms, infecting ordinary users' computers, whose

payload is a malicious application—the *bot*.

* The *bot* on the infected PC logs into a particular C&C server.
* A spammer purchases the services of the botnet from the operator.
* The spammer provides the spam messages to the operator, who instructs the compromised machines via the control panel on the web server, causing them to send out spam messages.

Botnets are exploited for various purposes, including denial-of-service attacks, creation or misuse of SMTP mail relays for spam (see Spambot), click fraud, mining bitcoins, spamdexing, and the theft of application serial numbers, login IDs, and financial information such as credit card numbers. The botnet controller community features a constant and continuous struggle over who has the most bots, the highest overall bandwidth, and the most "high-quality" infected machines, like university, corporate, and even government machines.

**3.2 Types of attacks**

* In distributed denial-of-service attacks, multiple systems submit as many requests as possible to a single Internet computer or service, overloading it and preventing it from servicing legitimate requests. An example is an attack on a victim's phone number. The victim is bombarded with phone calls by the bots, attempting to connect to the Internet.
* Adware advertises a commercial offering actively and without the user's permission or awareness, for example by replacing banner ads on web pages with those of another advertiser.
* Spyware is software which sends information to its creators about a user's activities – typically passwords, credit card numbers and other information that can be sold on the black market. Compromised machines that are located within a corporate network can be worth more to the bot herder, as they can often gain access to confidential corporate information. Several targeted attacks on large corporations aimed to steal sensitive information, such as the Aurora botnet.
* E-mail spams are e-mail messages disguised as messages from people, but are advertising, annoying, or malicious.
* Click fraud occurs when the user's computer visits websites without the user's awareness to create false web traffic for personal or commercial gain.
* Fast flux is a DNS technique used by botnets to hide phishing and malware delivery sites behind an ever-changing network of compromised hosts acting as proxies.
* Brute-forcing remote machines services such as FTP, SMTP and SSH.
* Worms. The botnet focuses on recruiting other hosts.
* Scareware is software that is marketed by creating fear in users. Once installed, it can install malware and recruit the host into a botnet. For example users can be induced to buy a rogue anti-virus to regain access to their computer.
* Exploiting systems by observing users playing online games such as poker and see the players' cards.

**3.3 Lifecycle of a Botnet**

The life cycle of a botnet is fairly standard, irrespective of the control and communication architecture used or the initial method of spreading the malicious code to the target machines infection. Different variants of botnets may have a few additional steps to take care of extra functionalities provided by them.

The following are the stages in a botnets life

1. Exploiting one of the many known vulnerabilities of a target system, the malicious code

gains entry into the system and converts it into a bot

1. The newly infected machine sends a message to the botmaster to let him or her know that it has joined the team. This process is known as rallying.

3. The next step for a bot is to secure itself on this newly infected machine, using the command and control server the bot tries to download and install an anti-antivirus software which will render any antivirus software on the machine ineffective.

4. Amazingly one of the next things that a bot does is to fix vulnerabilities on the host system by applying patches, this will prevent other malicious programs from entering the system and taking over the control of the system.

5. One of the main features of the botnet is that its functionality can change from time to time. The modular design of the botnets allows the botmaster to install payload modules which will implement the functionality which is currently required.

6. The bot then, depending on its architecture listens to either command and control servers or its peers for commands. Most of the botnets use Internet Relay Chat or IRC as the preferred communication medium between the servers and the bots.

7. The bot then executes the command and reports the status back to the command and control servers at appropriate time.

8. At this step the bot again goes back to stage 5, whereby it waits listening on the specified ports for new instructions or new payloads to be delivered.

9. On receiving a predetermined command from the botmaster the bot can completely erase all its traces on the computer and abandon the client. The bot often leaves the system in an unsecured state to ensure that future infections can occur.

**3.4 Botnet Control & Communication Architecture**

The botmaster choreographs the issuance of commands to the entire botnet using the control and communication infrastructure. The control and communication architecture of botnets is a major area of interest to both botnet creators and security researchers trying to detect and disable botnets.

**Peer to Peer Botnets**

As the limitations of the centralized command and control architecture became apparent, the attackers began experimenting with a new kind of architecture which would allow them to avoid detection and continue their attacks from distributed locations. These new architectures utilized Peer to Peer communication protocols and are now the most common among the newer kind of botnets.

The problem of single point as I case of centralized botnet of failure is solved by creating redundancy among the command and control servers. These servers can work completely independent of each other, thus even if one of the servers is detected and brought down there is very little effect on the overall functionality of the botnet. In this design not even the servers are aware of all the computers in the botnet, each server is aware of only a subset of the total number of infected computers, this adds to the stealth of the botnet.

Though the botnets using these newer approaches have succeeded in removing single point of failure to a large extent they are plagued with newer problems like scalability and large amount of network traffic. The peer to peer nature of these newer kinds of botnets results in a large amount of network traffic, making it susceptible to monitoring via network flow analysis.

**4.System Analysis**

**4.1 Existing System**

**4.1.1 Botminer**

Proposed a general botnet detection framework, named BotMiner. This framework is proposed for both centralized IRC and P2P botnets. BotMiner suppose that bots are coordinated malware and shows the same communication patterns and malicious activities.

The first stage in the proposed framework is clustering hosts with similar malicious activities and communication patterns from a network traffic and the resulted clusters are named A-Plane and C-plane for activity traffic and C&C communication traffic, respectively. The second stage is applying a cross correlation between A-plane and C-plane clusters. As a result from the correlation process, hosts that show both kinds of behaviors are detected as bots. Real network traffic was used to evaluate the proposed framework. The results show relatively high detection efficiency, with low numbers of false positives and false negatives. Furthermore, reasonable time and resources have been employed. BotMiner has two main limitations, the first one that it targets a group of infected computers within a monitored network, but in fact in a monitored network there is only a single compromised host and this single host may belong to a larger botnet. Therefore, BotMiner is not effective in detecting compromised hosts. The second limitation of BotMiner is its assumption of the systematic classification of any infected hosts. In case of P2P botnet, the bot may have malicious behaviors but still exchange normal C&C messages, so that bot will not be considered as a bot for the BotMiner. Under such scenario, BotMiner may not detect bots that exchange covert C&C messages.

**4.1.2 Network Streams Analysis**

They present a general P2P botnet detection framework, which includes three main algorithms.P2P nodes detection algorithm: Filtering can be applied on the P2P botnet, according to its features of paroxysm and distribution of the network streams. P2P nodes clustering algorithm: Clustering is proposed based on the connection characteristics of the nodes. The research uses K-mean clustering algorithm which based on the connection degree between the pair of nodes. Botnet behaviors detection algorithm: By extracting the similarities of the malicious behaviors of the bots, which may occur several times a day, the algorithm can detect if the P2P network is infected by bots.Unlike other detection models, the testing characteristics of this model taken from net stream macroscopical statistic, so it can be used to detect unknown protocol P2P botnets effectively. They ran a simulation of the three model algorithms in LAN circumstances and have goocd results of extracting the P2P stream, clustering and detecting botnets from normal network (Liu *et al*., 2010).

**4.1.3 Multi-Phased Flow Model**

P2P bots generate phased flows to connect with outside peers in order to construct the botnet. Based on this, the researcher proposed a multi-phased flow model to detect malicious traffic. The proposed model identifies P2P botnet by observing similar flows between network hosts. The proposed system consists of three stages

Flow grouping: Where the system group huge volume of traffic generated by P2P botnets and make clustering of TCP/UDP connections. Flow Compression: Extract information from each flow group value.

Flow Modeling: Modeling the P2P flows using a constructed matrix based on the transition information.Finally, the likelihood ratio is computed based on the probability-based models and used in detecting bots. The experimental evaluation was carried on Storm, Nugache and Spam Thru botnets. The detection rates ware 100, 95, 96% respectively

**4.1.4 Node Behavior Detection**

This research proposed a new method to detect the P2P bots inside the LAN. It use correlation between the Process name and both ports and network traffic (the protocols). To evaluate the system on real network, a storm bot infected dataset has been used. The research was conducted in University Technology Malaysia (UTM), which has a UTM-AntiBot to monitor the input and output flows and the network communication. In this research UTM Antibot has been used to observe thenetwork traffic between the internet and the internal host. After filtering out all the processes of the network traffic, PPNT correlates each process with its associated port and the connected IP. A behavior of a normal user under a controlled LAN has been examined. The research resulted that it is impossible to send thousands of SMTP packets in less than 10 minutes and considering UDP packets with fixed port, SMTP packets confirm that this user is a part of Storm Botnet. Acceptable but not high rate of detection has been shown in their experimental results.

**4.1.5**.**Network Behavior Analysis and Machine Learning.**

This research proposes a new method for detecting botnets through identifying the network behavior characteristics. This approach aimed to detect P2Pbotnet Command and Control (C&C) phase, which allows detecting the bots before attacking their victims. In addition, this study discussed the requirements of online botnet detection framework and investigates the ability of five Machine Learning (ML) techniques to meet these requirements. The evaluation results show the promising performance of ML techniques, but none of them satisfy all the requirements of the online botnet detection framework.

**4.1.6. Association between Common Network**

Behaviors and Host Behaviors This research proposed a new P2P Botnet detection approach relying on the association between common host and network behaviors. The proposed framework consists of six stages as following:

Detected system: To distinguish between the single and communication program, since the main characteristic of the bots is communication with other bots on other computers

Filtering: To reduce traffic load, so the system can work more efficiently

Extract features from P2P data: Detect the more relevant features to make a subset of features that describe properly the P2P data

Botnet detection: Based on the data source this stage includes host data detection and network data detection. The objective is to detect the known botnet and the unknown malware

Report: If the detected behavior is known, the system report, if not the system will detect the bot behavior by correlating host and network behaviors

Solution: After finding out the botnet, the system can either fire it back or take it down

This method has some limitations such as, bots that using encryption algorithms cannot be detected.

**4.1.7.User Behavior Sociality and Traffic Entropy Function**

Based on the user behavior and the social action of Botnet nodes that differ from normal nodes, this research proposed a new structure to identify P2P Botnet and consider it as a key basis for P2P Botnet detection. The proposed structure of P2P Botnet includes:

Analyzing sociality characteristics as centrality from the original network data, by making too high entrality nodes as suspicious ones Finding out data packet size characteristic can be use the entropy concept to make model for the data packet of the suspicious node Make deep data packet detection, with improved entropy After doing experimental evaluations of the proposed structure, the results show that this structure can identify the P2P botnet with high accuracy. However, the identification accuracy reduces when the download rate of net traffic is very high, or the user video streaming is too big

**4.1.8.Data Mining**

This research proposed a P2P botnet detection approach which relies on monitoring gateway traffic and analyze network behavior using data mining techniques. To evaluate the proposed method, they used a freeware WEKA and three popular algorithms J48, Naïve Bayes and Bayesian networks for data mining. The resulted accuracy rates were 98, 89 and 87%, respectively for the three algorithms. Based on the results, the proposed method can used in distinguishing infected bots flows from other bots and the most appropriate algorithm among the three algorithms was J48.

**4.2. Proposed System**

NODE\_2

BOTMASTER

NODE\_1

ROUTER

4.2.a. Pictorial Representation Of Proposed System.

In this we have 2 nodes,1 router and a Botmaster node.Transfer of files can happen through all the 3 nodes via router.The Botmaster is used to send commands to either of the nodes which is known as attacking a node and making it a bot.The attacked node sends some garbage files or spam to the other nodes depending on the commands given by the botmaster.

We can set a threshold value for the no. of files to be sent in a time interval after we attack a particular node.After some time interval we notice the packets that have been transferred and form them into 2 clusters on the basis of Eucledian Distance.

Eucledian Distance=Square\_root[(pkts\_sent-pkts\_rcvd)­2 + (byts\_sent-byts\_rcvd)2]

All the packets having similar distance are grouped into one cluster and the remaining are grouped into another.After clustering we can find whether a particular cluster has more number of packets than the threshold value that we have set and if it more than the threshold value then we identify that particular node as a bot.After identifying the bot we

can trace back and find the node which has sent the commands to this particular node and find the ip address of the BotMaster.

**5.SOURCE CODE**

package com.nodes;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Image;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.BufferedReader;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.net.ServerSocket;

import java.net.Socket;

import java.net.URL;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.HashMap;

import java.util.Random;

import java.util.Timer;

import java.util.TimerTask;

import java.util.Vector;

import javax.imageio.ImageIO;

import javax.swing.ButtonGroup;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JRadioButton;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.JTextField;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.JTableHeader;

import javax.swing.table.TableColumn;

import com.botnet.dao.Botnet\_DAO;

import com.propertybag.Constants;

import com.propertybag.PropertyBag;

public class Node\_1 extends JFrame

{

//define all variables used.

private JPanel outer\_panel;

private JPanel send\_panel;

private JPanel view\_panel;

private JMenuBar menuBar;

private JMenu send\_menu,view\_menu;

private JMenuItem send\_menu\_item;

private JMenuItem view\_menu\_item;

private JLabel protocol\_lbl;

private JLabel destination\_lbl;

private JLabel data\_lbl;

private JRadioButton tcp\_jrb,udp\_jrb;

private JButton power\_btn;

private JLabel background\_lbl;

private JComboBox jc\_box;

private JRadioButton browse\_jrb;

private JRadioButton text\_jrb;

private JFileChooser file\_chooser;

private File curFile;

private File f;

private JLabel filename\_lbl;

private JButton send\_btn,power\_btn1;

private JTextField filename\_tfield;

Timer attack\_timer;

public byte filebyte[] = new byte[10000];

byte byteArray[] = new byte[1024];

private String filename;

private String protocol\_type="";

private String pattern="";

private ButtonGroup buttonGroup,buttonGroup1;

JTable table;

TableColumn col;

Vector columnNames;

Vector data;

Dimension dim = new Dimension(6,1);

JTableHeader header;

JScrollPane view\_scrollpane=new JScrollPane();

public Node\_1() throws IOException

{

super();

initialize();

}

private void initialize() throws IOException

{

this.setSize(new java.awt.Dimension(700,600));

this.setLocation(100, 100);

this.setContentPane(getOuterJPanel());

this.setTitle("Node\_1"); //set the title of dialog box

this.setDefaultCloseOperation(javax.swing.JFrame.EXIT\_ON\_CLOSE);

Thread t=new Thread(new PortListener());

t.start();

Thread t1=new Thread(new PortListener1());

t1.start();

}

private JPanel getOuterJPanel() throws IOException

{

if(outer\_panel==null)

{

//to set the background colour

background\_lbl = new JLabel();

URL background\_img\_url = Node\_1.class.getResource("back2.jpg");

Image i1 = ImageIO.read(background\_img\_url);

Image background\_img = i1.getScaledInstance(700,600, java.awt.Image.SCALE\_SMOOTH);

ImageIcon background\_img\_icon = new ImageIcon(background\_img);

background\_lbl.setBounds(0,0,700,600);

background\_lbl.setIcon(background\_img\_icon);

background\_lbl.setVisible(true);

//project title image

JLabel title\_lbl = new JLabel();

URL img\_url = Node\_1.class.getResource("Top.jpg");

Image i2= ImageIO.read(img\_url);

Image img = i2.getScaledInstance(700,100, java.awt.Image.SCALE\_SMOOTH);

ImageIcon img\_icon = new ImageIcon(img);

title\_lbl.setBounds(0,0,700,100);

title\_lbl.setIcon(img\_icon);

title\_lbl.setVisible(true);

menuBar = new JMenuBar();//to create a menu

menuBar.setBounds(1, 100, 700, 30);

send\_menu=new JMenu("Send");

send\_menu\_item=new JMenuItem("Send");

send\_menu\_item.setToolTipText("Send");

send\_menu\_item.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent arg0)

{

send\_panel.setVisible(true);

view\_panel.setVisible(false);

}

});

send\_menu.add(send\_menu\_item);

view\_menu=new JMenu("View");

view\_menu\_item=new JMenuItem("View");

view\_menu\_item.setToolTipText("View");

view\_menu\_item.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent arg0)

{

//to create the header

columnNames = new Vector();

data=new Vector();

columnNames.add("Sender");

columnNames.add("Receiver");

columnNames.add("Se\_Bytes");

columnNames.add("Rec\_Bytes");

columnNames.add("S\_Packets");

columnNames.add("Rec\_Packets");

columnNames.add("Date\_Time");

data=Botnet\_DAO.fetchtransaction();

DefaultTableModel model = new DefaultTableModel (data,columnNames);

table = new JTable(model)

{

public boolean isCellEditable(int rowIndex, int colIndex)

{

return false; //Disallow the editing of any cell

}

};

header = table.getTableHeader();

header.setBackground(Color.ORANGE);

table.setIntercellSpacing(new Dimension(dim));

SetRowHight(table);

for (int i = 0; i < table.getColumnCount(); i++)

{

col = table.getColumnModel().getColumn(i);

col.setMaxWidth(400);

}

table.getColumnModel().getColumn(0).setPreferredWidth(39);

table.getColumnModel().getColumn(1).setPreferredWidth(39);

table.getColumnModel().getColumn(2).setPreferredWidth(39);

table.getColumnModel().getColumn(3).setPreferredWidth(52);

table.getColumnModel().getColumn(4).setPreferredWidth(52);

table.getColumnModel().getColumn(5).setPreferredWidth(52);

table.getColumnModel().getColumn(6).setPreferredWidth(150);

System.out.println("data "+data);

Dimension size = table.getPreferredScrollableViewportSize();

table.setAutoResizeMode(JTable.AUTO\_RESIZE\_ALL\_COLUMNS);

view\_scrollpane.setViewportView(table);

view\_scrollpane.setBounds(40, 40, 530, 350);

view\_panel.add(view\_scrollpane);

view\_panel.setVisible(true);

send\_panel.setVisible(false);

}

});

view\_menu.add(view\_menu\_item);

menuBar.add(send\_menu);

menuBar.add(view\_menu);

menuBar.setVisible(false);

outer\_panel=new JPanel();

outer\_panel.setLayout(null);

outer\_panel.setBounds(new java.awt.Rectangle(0,0,700,600));

outer\_panel.add(getsendpanel());

outer\_panel.add(getviewpanel());

outer\_panel.add(getonbutton());

outer\_panel.add(getoffbutton());

outer\_panel.add(menuBar);

outer\_panel.add(title\_lbl);

outer\_panel.add(background\_lbl);

outer\_panel.setVisible(true);

}

return outer\_panel;

}

private JPanel getviewpanel()

{

if(view\_panel==null)

{

view\_panel=new JPanel();

view\_panel.setLayout(null);

view\_panel.setBounds(new java.awt.Rectangle(50,150,700,400));

view\_panel.setBorder(javax.swing.BorderFactory.createTitledBorder (null, "", javax.swing.border.TitledBorder.LEFT, javax.swing.border.TitledBorder.CENTER, new java.awt.Font ("Tahoma", java.awt.Font.BOLD, 18), new java.awt.Color (150,57,120)));

view\_panel.setVisible(false);

}

return view\_panel;

}

private JPanel getsendpanel()

{

if(send\_panel==null)

{

//to create protocol label and select tcp or udp

protocol\_lbl = new JLabel();

protocol\_lbl.setBounds(new java.awt.Rectangle(25,25,200,100));

protocol\_lbl.setFont(new java.awt.Font("Tahoma", java.awt.Font.BOLD, 18));

protocol\_lbl.setForeground(new Color(255, 0, 127));

protocol\_lbl.setText("Protocol");

buttonGroup = new ButtonGroup();

tcp\_jrb=new JRadioButton("TCP");

tcp\_jrb.setBounds(new java.awt.Rectangle(150,65,75,29));

tcp\_jrb.setForeground(new Color(111,143,237));

tcp\_jrb.setFont(new java.awt.Font("verdana", java.awt.Font.PLAIN,12));

buttonGroup.add(tcp\_jrb);

udp\_jrb = new JRadioButton("UDP");

udp\_jrb.setBounds(new java.awt.Rectangle(250,65,75,29));

udp\_jrb.setForeground(new Color(111,143,237));

udp\_jrb.setFont(new java.awt.Font("verdana", java.awt.Font.PLAIN,12));

buttonGroup.add(udp\_jrb);

//to select destination node

destination\_lbl=new JLabel();

destination\_lbl.setBounds(new java.awt.Rectangle(25,100,200,100));

destination\_lbl.setFont(new java.awt.Font("Tahoma", java.awt.Font.BOLD, 18));

destination\_lbl.setForeground(new Color(255, 0, 127));

destination\_lbl.setText("Destination");

jc\_box=new JComboBox();

jc\_box.addItem("Node\_2");

jc\_box.setBounds(150, 130, 200,30);

//to create data label

data\_lbl=new JLabel();

data\_lbl.setBounds(new java.awt.Rectangle(25,175,200,100));

data\_lbl.setFont(new java.awt.Font("Tahoma", java.awt.Font.BOLD, 18));

data\_lbl.setForeground(new Color(255, 0, 127));

data\_lbl.setText("Data");

//to create filename label

filename\_lbl=new JLabel();

filename\_lbl.setBounds(new java.awt.Rectangle(25,250,100,100));

filename\_lbl.setFont(new java.awt.Font("Tahoma", java.awt.Font.BOLD, 18));

filename\_lbl.setForeground(new Color(255, 0, 127));

filename\_lbl.setText("Filename");

filename\_tfield = new JTextField();

filename\_tfield.setBounds(new java.awt.Rectangle(125,285,200,30));

filename\_tfield.setFont(new java.awt.Font("verdana", java.awt.Font.BOLD, 14));

filename\_tfield.setForeground(new Color(191,163,159));

//to create browse radio button

buttonGroup1 = new ButtonGroup();

text\_jrb=new JRadioButton("Text");

text\_jrb.setBounds(new java.awt.Rectangle(25,250,75,29));

text\_jrb.setForeground(new Color(111,143,237));

text\_jrb.setFont(new java.awt.Font("verdana", java.awt.Font.PLAIN,12));

buttonGroup1.add(text\_jrb);

browse\_jrb = new JRadioButton("Browse");

browse\_jrb.setBounds(new java.awt.Rectangle(150,215,75,29));

browse\_jrb.setForeground(new Color(111,143,237));

browse\_jrb.setFont(new java.awt.Font("verdana", java.awt.Font.PLAIN,12));

//to browse a file from system

browse\_jrb.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e)

{

try

{

file\_chooser = new JFileChooser();

f = new File(new File("filename.txt").getCanonicalPath());

file\_chooser.setSelectedFile(f);

int retval = file\_chooser.showOpenDialog(browse\_jrb);

if (retval == JFileChooser.APPROVE\_OPTION)

{

File field = file\_chooser.getSelectedFile();

}

curFile = file\_chooser.getSelectedFile();

filename\_tfield.setText(curFile.getAbsolutePath().toString());

}

catch (IOException e1)

{

e1.printStackTrace();

}

}

});

buttonGroup1.add(browse\_jrb);

send\_panel=new JPanel();

send\_panel.setLayout(null);

send\_panel.setBounds(new java.awt.Rectangle(150,150,500,400));

send\_panel.setBorder(javax.swing.BorderFactory.createTitledBorder (null, "", javax.swing.border.TitledBorder.LEFT, javax.swing.border.TitledBorder.CENTER, new java.awt.Font ("Tahoma", java.awt.Font.BOLD, 18), new java.awt.Color (150,57,120)));

send\_panel.setVisible(false);

send\_panel.add(protocol\_lbl);

send\_panel.add(destination\_lbl);

send\_panel.add(udp\_jrb);

send\_panel.add(tcp\_jrb);

send\_panel.add(jc\_box);

//send\_panel.add(text\_jrb);

send\_panel.add(browse\_jrb);

send\_panel.add(data\_lbl);

send\_panel.add(filename\_lbl);

send\_panel.add(filename\_tfield);

send\_panel.add(getsend\_btn());

}

return send\_panel;

}

private JButton getonbutton()

{

if(power\_btn==null)

{

power\_btn=new JButton("ON");

power\_btn.setBounds(new java.awt.Rectangle(550,100,100,25));

power\_btn.setFont(new java.awt.Font("Tahoma", java.awt.Font.BOLD, 12));

power\_btn.setForeground(new Color(111,143,237));

power\_btn.addActionListener(new java.awt.event.ActionListener()

{

public void actionPerformed(java.awt.event.ActionEvent e)

{

menuBar.setVisible(true);

power\_btn.setVisible(false);

power\_btn1.setVisible(true);

DateFormat dateFormat = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");

Date date = new Date();

String datetime=dateFormat.format(date).toString();

String a[]=datetime.split(" ");

String node="Node\_1";

boolean insertflag=Botnet\_DAO.addlog(a[0].toString (),a[1].toString(),node,"0");

Botnet\_DAO.updatestatus("Node\_1","ON");

}

});

}

return power\_btn;

}

private JButton getoffbutton()

{

if(power\_btn1==null)

{

power\_btn1=new JButton("OFF");

power\_btn1.setBounds(new java.awt.Rectangle(550,100,100,25));

power\_btn1.setFont(new java.awt.Font("Tahoma", java.awt.Font.BOLD, 12));

power\_btn1.setForeground(new Color(111,143,237));

power\_btn1.addActionListener(new java.awt.event.ActionListener()

{

public void actionPerformed(java.awt.event.ActionEvent e)

{

menuBar.setVisible(false);

power\_btn.setVisible(true);

power\_btn1.setVisible(false);

view\_panel.setVisible(false);

send\_panel.setVisible(false);

DateFormat dateFormat = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");

Date date = new Date();

String datetime=dateFormat.format(date).toString();

String a[]=datetime.split(" ");

String node="Node\_1";

boolean insertflag=Botnet\_DAO.updatelog(a [0].toString(),node,a[1].toString());

Botnet\_DAO.updatestatus("Node\_1","OFF");

}

});

}

return power\_btn1;

}

public void SetRowHight(JTable table)

{

int height = table.getRowHeight();

table.setRowHeight(height+10);

table.setBackground(new Color(196,143,207));

}

public JButton getsend\_btn()

{

if(send\_btn==null)

{

send\_btn=new JButton("Send");

send\_btn.setBounds(new java.awt.Rectangle(150,350,100,25));

send\_btn.setFont(new java.awt.Font("Tahoma", java.awt.Font.BOLD, 12));

send\_btn.setForeground(new Color(111,143,237));

send\_btn.addActionListener(new java.awt.event.ActionListener()

{

public void actionPerformed(java.awt.event.ActionEvent e)

{

try

{

String status=Botnet\_DAO.getstatus("Router");

if(status.equals("ON"))

{

//to send file wen send is pressed

String from="Node\_1";

String to=jc\_box.getSelectedItem().toString();

String packets=PropertyBag.getProperty("pkt\_byte");

String protocol\_type = null;

if(tcp\_jrb.isSelected())

{

protocol\_type="TCP";//4

}

else if(udp\_jrb.isSelected())

{

protocol\_type="UDP";//4

}

else

{ JOptionPane.showMessageDialog(null, "Select Protocol Type");

}

int packet=Integer.parseInt(packets);//2

filename=curFile.getAbsolutePath();

String ip[]=Botnet\_DAO.getSenduserip("Router"); //Reading data of the file

File data = new File(curFile.getAbsolutePath());

FileInputStream fins=new FileInputStream(data);

DataInputStream ins = new DataInputStream(fins);

BufferedReader br = new BufferedReader(new InputStreamReader(ins));

StringBuffer buffer = new StringBuffer();

String src\_fcontent;

String content;

while ((src\_fcontent = br.readLine()) != null)

{

buffer.append(src\_fcontent.trim()+ "\n");

}

content=buffer.toString();

byte buffer1[] = content.getBytes();

int bytes=buffer1.length;

packet=bytes/packet;

if(packet==0)

{

packet=1;

}

System.out.println("========> File name=-==========================>" +curFile.getName());

pattern=from+"#"+to+"#"+packet+"#"+bytes+"#"+protocol\_type+"#"+content+"#"+curF ile.getName().trim();

Socket sock=new Socket(ip[1], Integer.parseInt(ip[0].trim()));

DataOutputStream dos=new DataOutputStream(sock.getOutputStream());

dos.writeUTF(pattern);

dos.close();

sock.close();

}

else

{

JOptionPane.showMessageDialog(null, "Router is OFF");

}

}

catch(Exception e1)

{

JOptionPane.showMessageDialog(null, "Something wrong went..!");

e1.printStackTrace();

}

}

});

}

return send\_btn;

}

class PortListener implements Runnable

{

@Override

public void run()

{

ServerSocket servsock;

try

{

servsock = new ServerSocket(Constants.Sending\_Node1\_Port);

while(true)

{

Socket sock=servsock.accept();

DataInputStream dis=new DataInputStream(sock.getInputStream());

String data=dis.readUTF();

String out[]=data.split("#");

String fname="Received\_Files/"+out[1].trim();

FileOutputStream fos=new FileOutputStream(new File(fname));

fos.write(out[0].getBytes());

System.out.println(" ====== File Received in Node 1 =====================");

//JOptionPane.showMessageDialog(null, "File Received from Node 2");

}

}

catch (IOException e)

{

dispose();

//e.printStackTrace();

}

}

}

class PortListener1 implements Runnable

{

@Override

public void run()

{

ServerSocket servsock;

try

{

servsock = new ServerSocket(Constants.Receiving\_Node1\_Port);

while(true)

{

Socket sock=servsock.accept();

DataInputStream dis=new DataInputStream (sock.getInputStream());

String data=dis.readUTF();

attack\_timer = new Timer();

attack\_timer .schedule(new AttackTransfer(),0,1\*1000); //subsequent rate

}

}

catch (IOException e)

{

//JOptionPane.showMessageDialog(null, "Node 2 is Already Opened");

dispose();

//e.printStackTrace();

}

}

}

class AttackTransfer extends TimerTask

{

public void run()

{

File data = new File("Node\_1/attack.txt");

FileInputStream fins;

try

{

fins = new FileInputStream(data);

DataInputStream ins = new DataInputStream(fins);

BufferedReader br = new BufferedReader(new InputStreamReader(ins));

StringBuffer buffer = new StringBuffer();

String src\_fcontent;

String content;

while ((src\_fcontent = br.readLine()) != null)

{

buffer.append(src\_fcontent.trim()+ "\n");

}

content=buffer.toString();

String packets=PropertyBag.getProperty("pkt\_byte");

int packet=Integer.parseInt(packets);//2

byte buffer1[] = content.getBytes();

int bytes=buffer1.length;

packet=bytes/packet;

if(packet==0)

{

packet=1;

}

Random rand=new Random();

int r=rand.nextInt(1);

if(r==0)

protocol\_type="TCP";

else

protocol\_type="UDP";

System.out.println("========> byts.len" +bytes+" ===== "+packet);

pattern="Node\_1"+"#"+"Node

2"+"#"+packet+"#"+bytes+"#"+protocol\_type+"#"+content+"#"+"attack.txt";

String status=Botnet\_DAO.getstatus("Node\_1");

String status1=Botnet\_DAO.getstatus("Router");

if(status.equals("ON") && status1.equals("ON"))

{

String ip[]=Botnet\_DAO.getSenduserip("Router");

Socket sock=new Socket(ip[1], Integer.parseInt(ip[0].trim()));

DataOutputStream dos=new DataOutputStream(sock.getOutputStream());

dos.writeUTF(pattern);

dos.close();

sock.close();

}

}

catch (FileNotFoundException e)

{

e.printStackTrace();

}

catch (IOException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

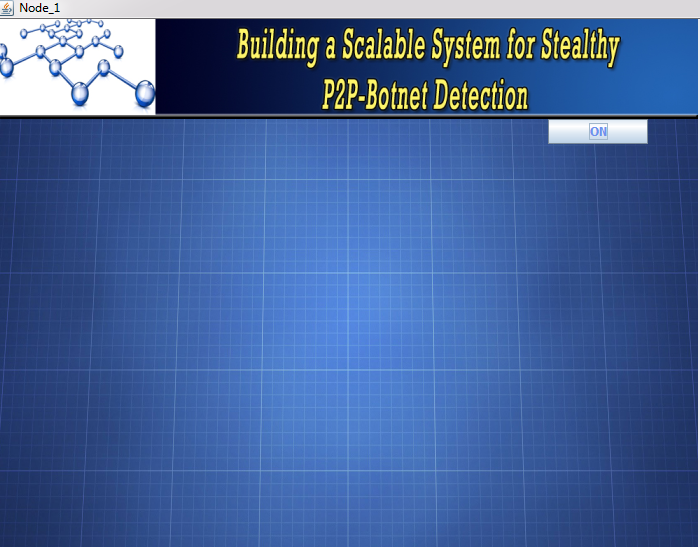
public static void main(String[] args) throws IOException

{

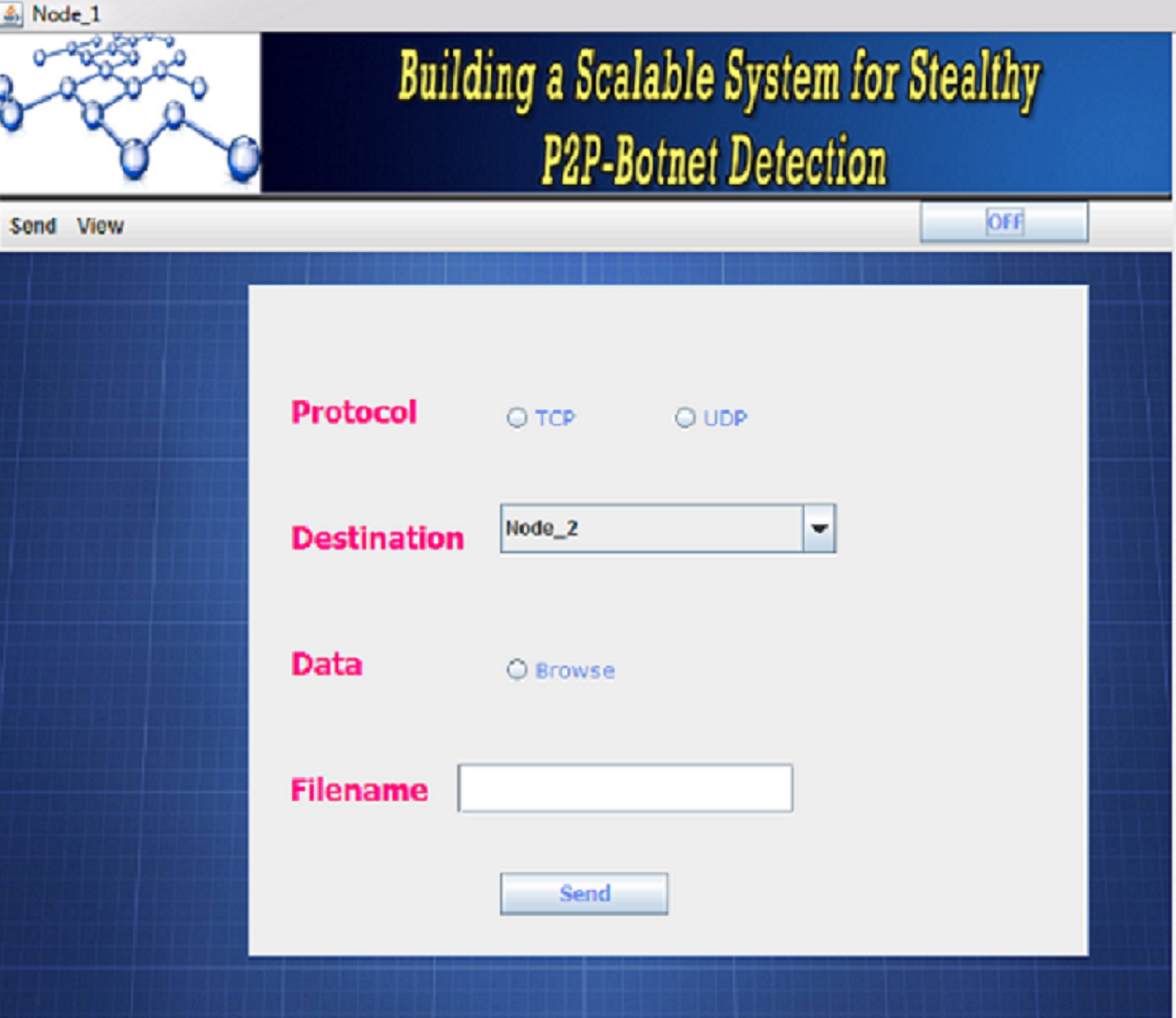
new Node\_1().setVisible(true); }

}

**6.SNAPSHOTS**



6.1.a.Screenshot of node1



6.1.b.Screenshot of SEND in node1



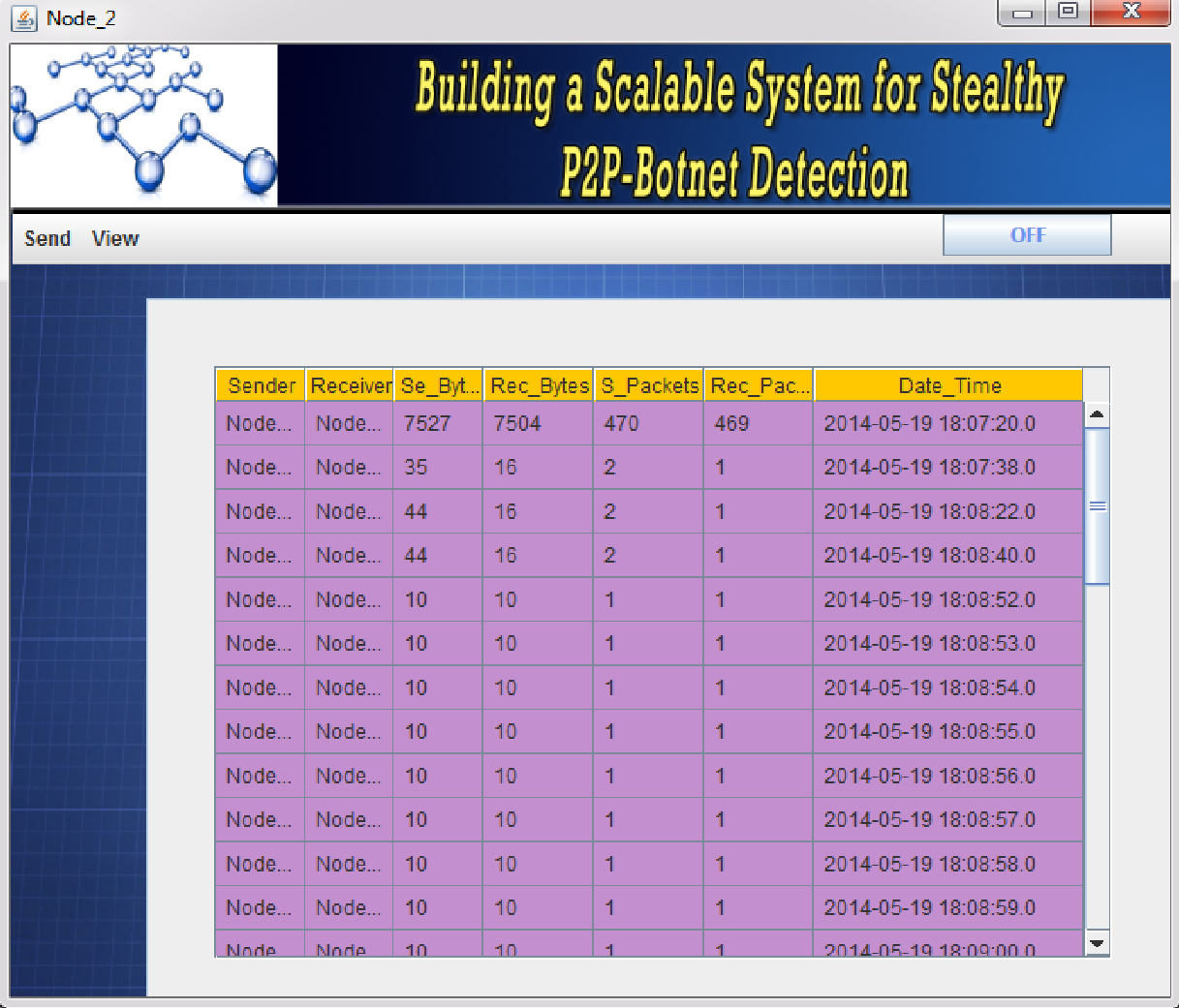
6.1.c.Screenshot of VIEW in node1



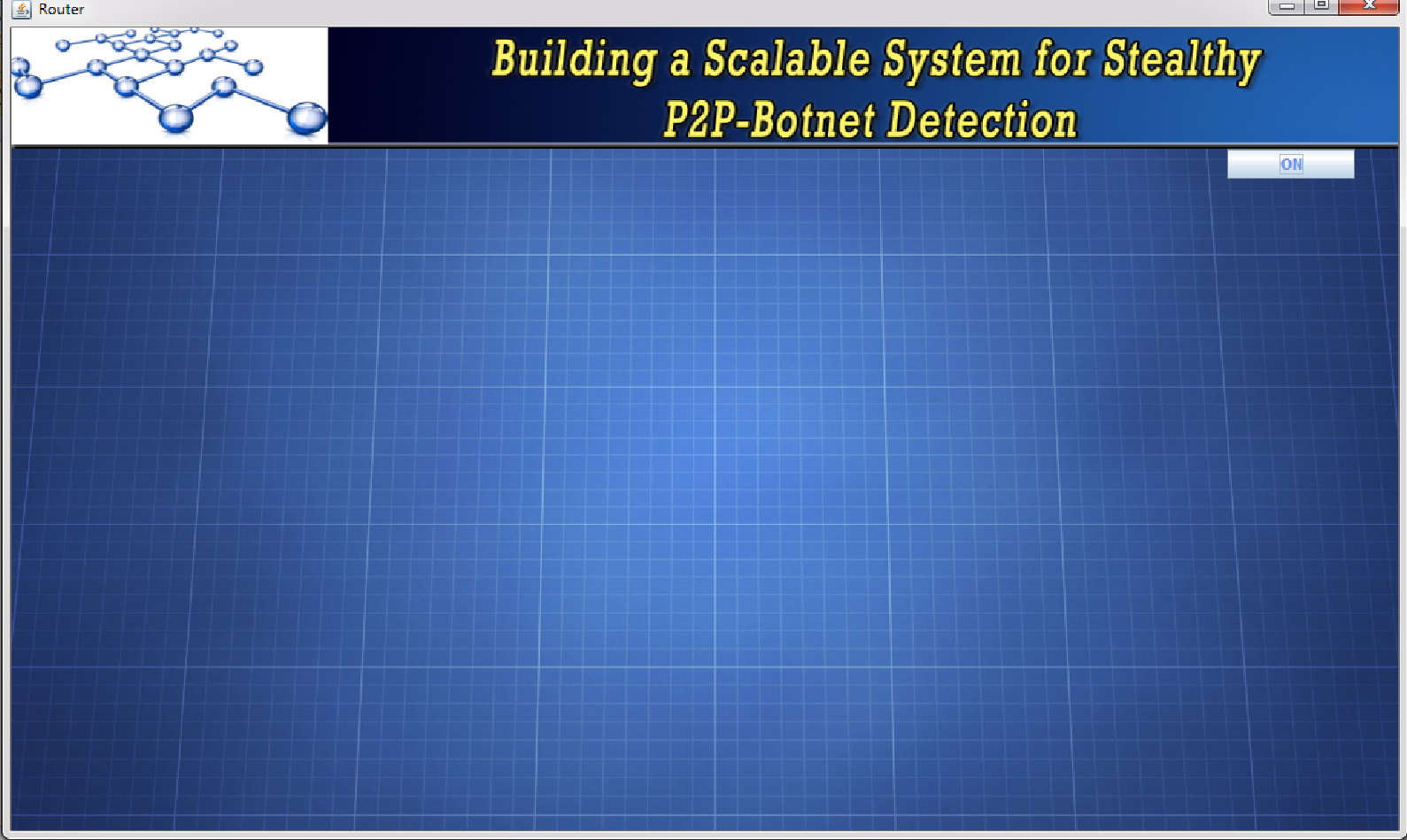
6.2.a.Screenshot of node2



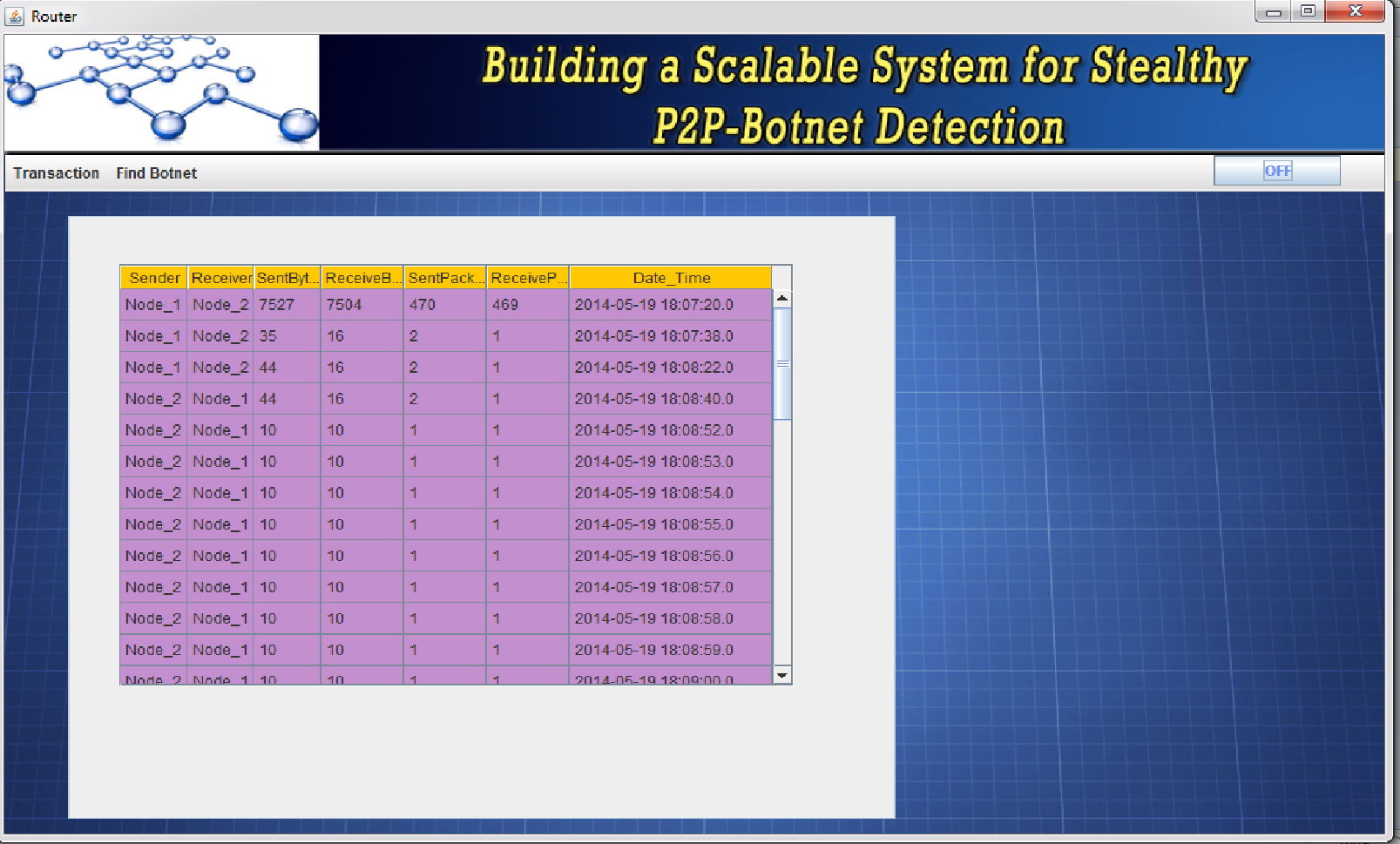
6.2.b.Screenshot of SEND in node2



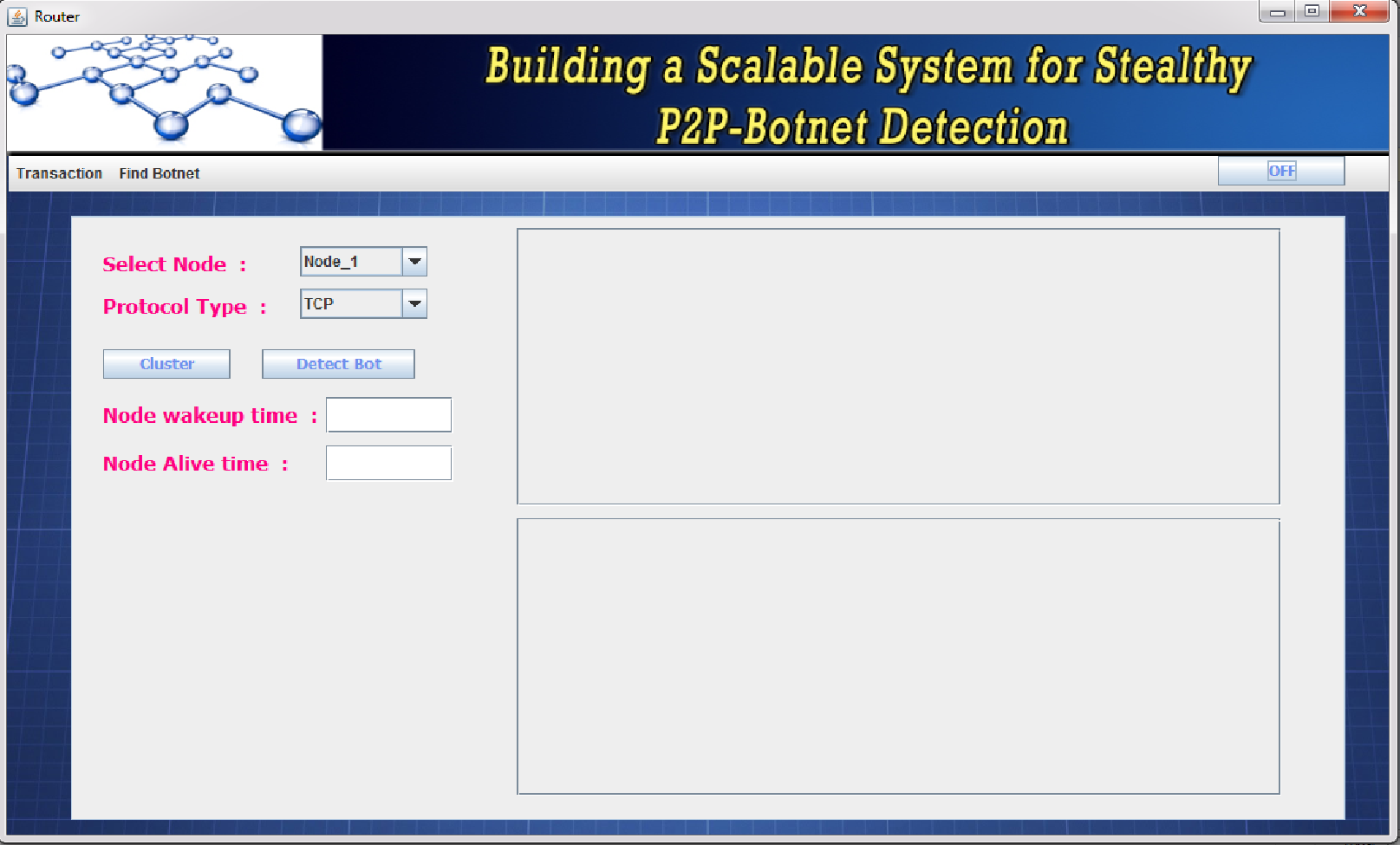
6.2.c.Screenshot of VIEW in node2



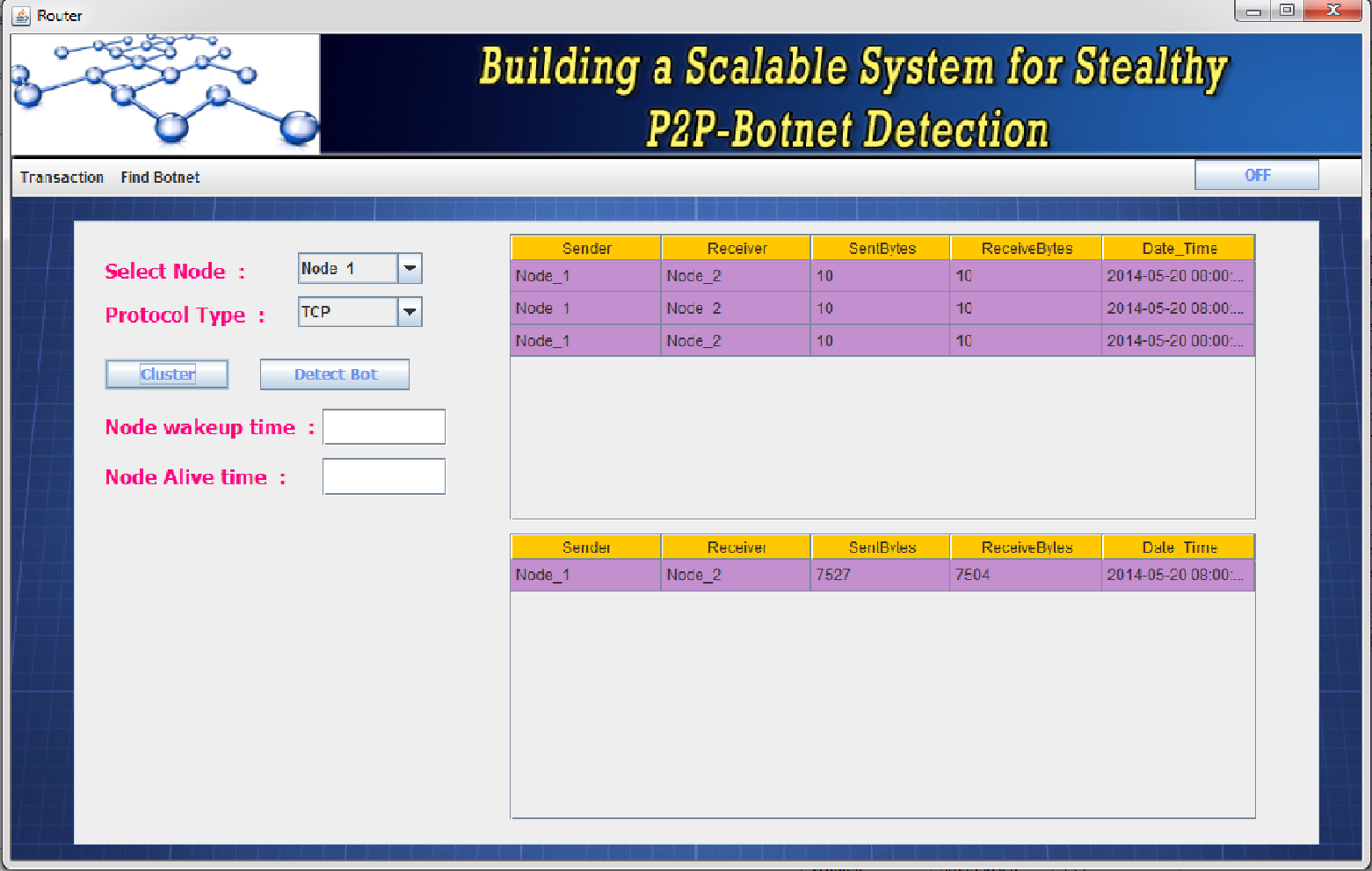
6.3.a.Screenshot of ROUTER



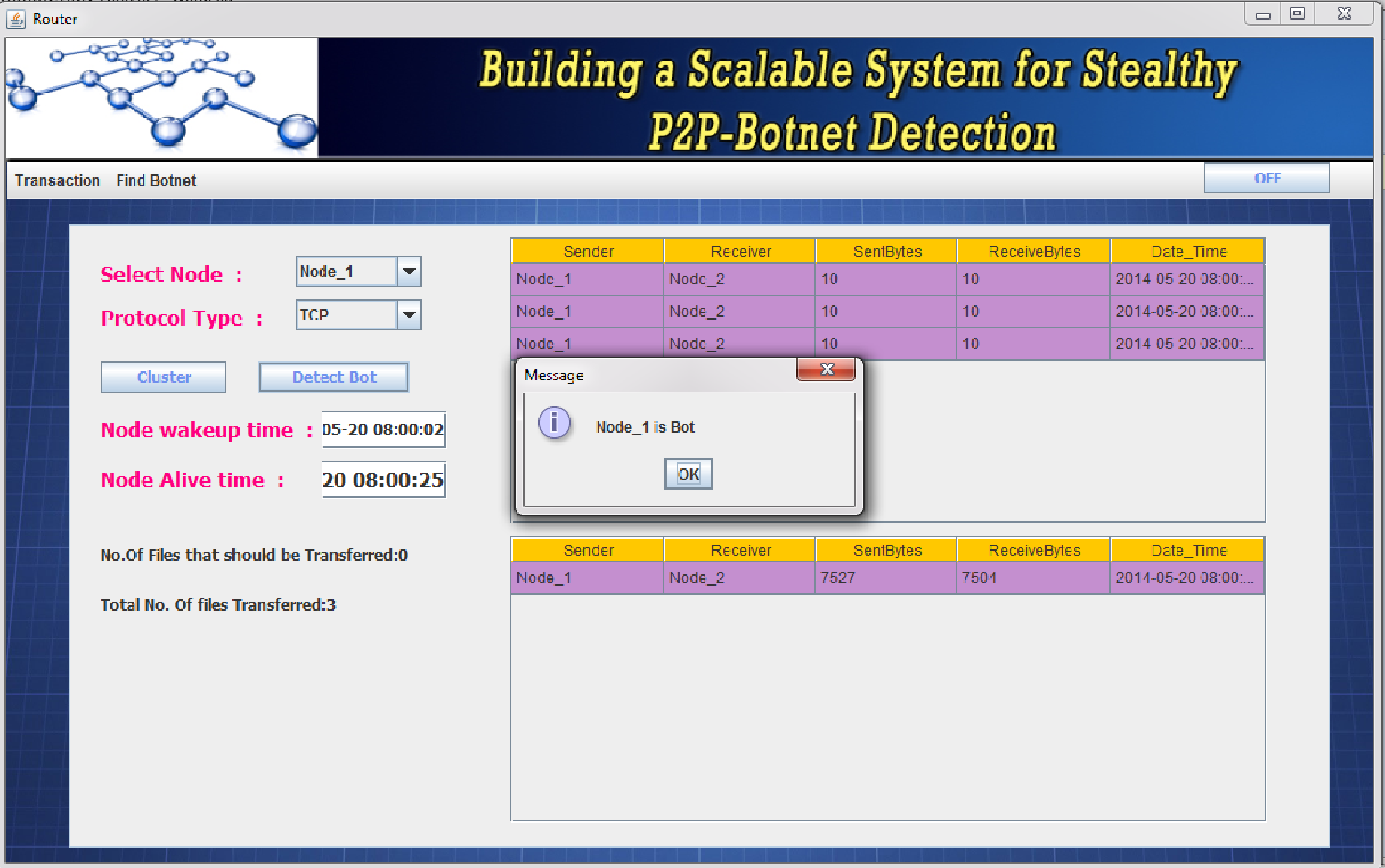
6.3.b.Screenshot of TRANSACTION in router



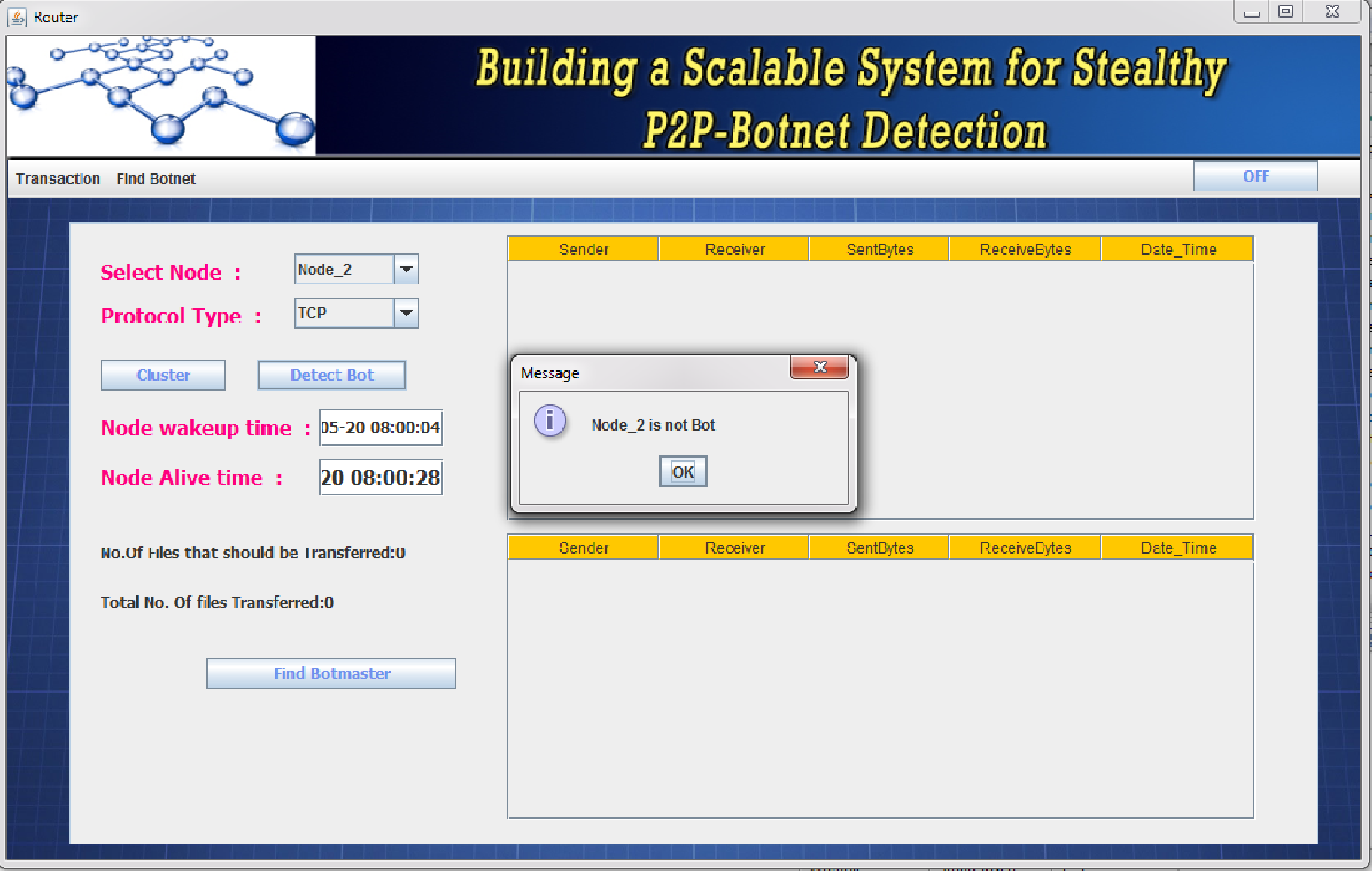
6.3.c.Screenshot of FIND\_BOTNET in router



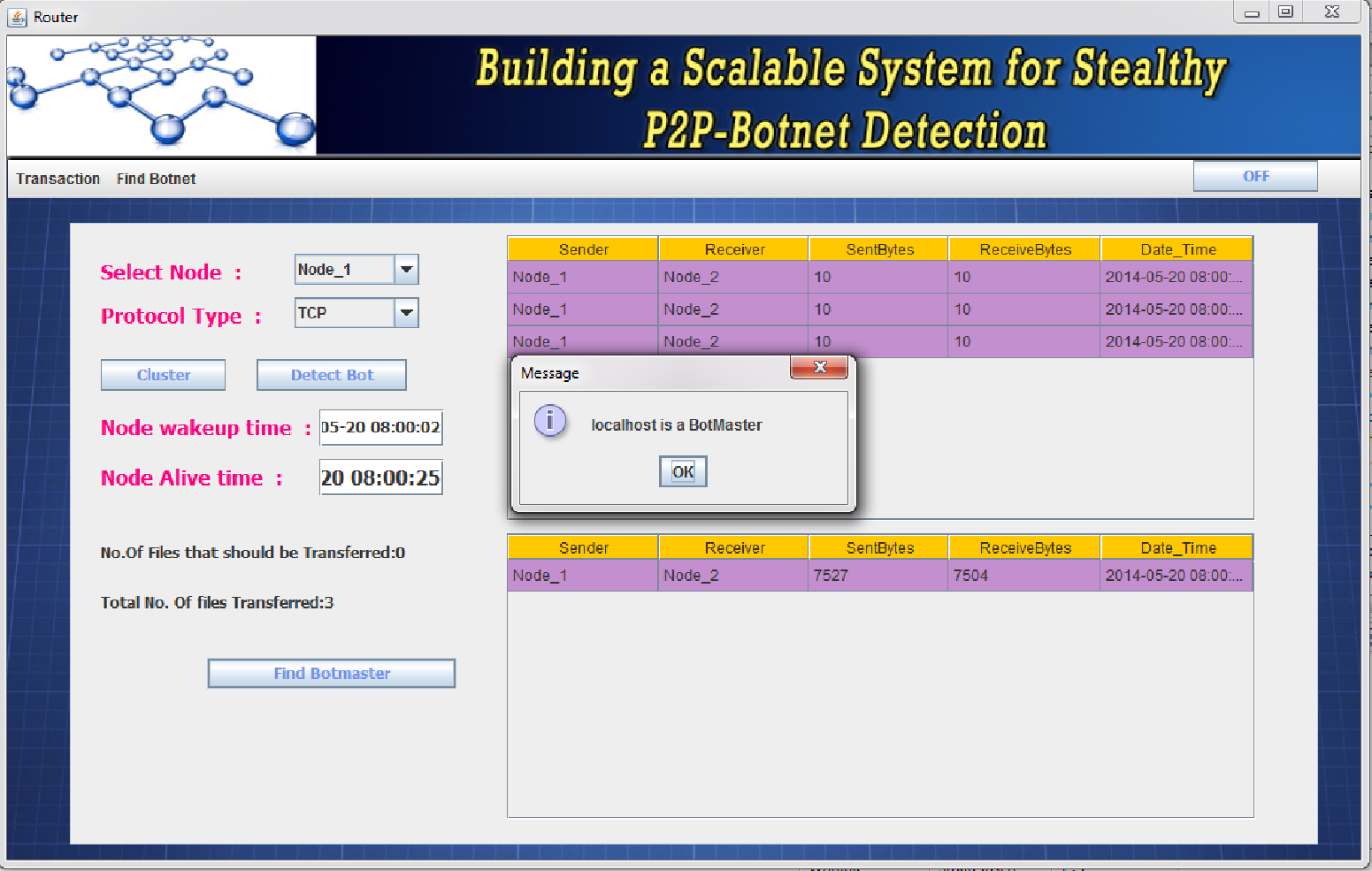
6.3.d.Screenshot of CLUSTER Formation



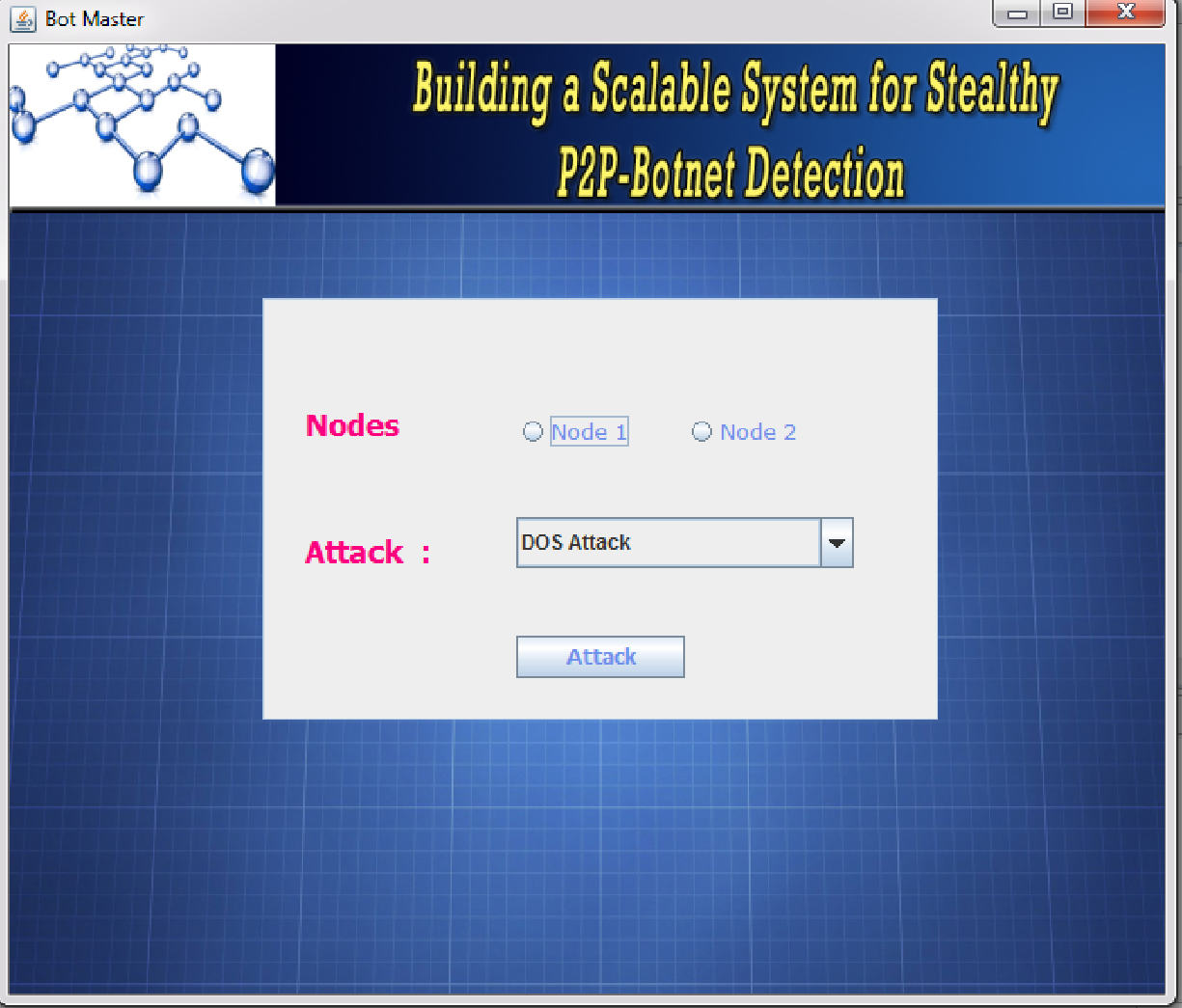
6.3.e.Screenshot of BOTNET detection



6.3.f.Screenshot of Botnet Detection



6.3.g.Screenshot of BOTMASTER detection



6.4.Screenshot of BOTMASTER

**7. VERFICATION AND VALIDATION**

**Verification and Validation** (**V&V**) is the process of checking that a software system meets specifications and that it fulfills its intended purpose. It may also be referred to as software quality control. It is normally the responsibility of software testers as part of the software development lifecycle.

Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements. This is done through dynamic testing and other forms of review.

Verification and validation are not the same thing, although they are often confused

* Validation: Are we building the right product?
* Verification: Are we building the product right?

**Testing**

Testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

**7.1 Unit testing**

All the code written for the system has been tested for errors and exceptions to ensure that no compiler error is present. The codes are syntactically correct and bug free. Multiple compilations and debugging were carried out during the Unit Testing which removed any errors in coding and has resulted in reliable code.

**7.2 White Box Testing**

All the methods used in the system have been tested for errors. During white box testing considering the system working is known, each method has been tested for proper return values, parameters and desired result. Calling sequence and data value of all methods have been tested to ensure proper working of the system as a whole.

**7.3 Black Box Testing**

Following tests have been carried out as black box testing

* Random file is sent from one node to other.
* Random node is used to deploy attack
* Results are compared for different nodes, packet sent and attack.

Any modification needed was incorporated to conform to the project statement.

**8. CONCLUSION**

In this project, we developed a l botnet detection system that is able to identify stealthy P2P botnets, whose malicious activities may not be observable. To accomplish this task, we derive statistical fingerprints of the P2P communications to first detect P2P clients and further distinguish between those that are part of legitimate P2P networks (e.g.,Skype) and P2P bots. The evaluation results demonstrated that the proposed system accomplishes high accuracy on detecting stealthy P2P bots.

**9.REFERENCES**

1. G. Gu, R. Perdisci, J. Zhang, and W. Lee, “Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection,” in *Proc. USENIX Security*, 2008, pp. 139–154.

2.S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov, “BotGrep: Finding P2P bots with structured graph analysis,” in *Proc. USENIX* *Security*, 2010, pp. 1–16.

3.J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, “Detecting stealthy P2P botnets using statistical traffic fingerprints,” in *Proc.IEEE/IFIP 41st Int. Conf. DSN*, Jun. 2011, pp. 121–132.

4.D. Liu, Y. Li, Y. Hu, and Z. Liang, “A P2P-botnet detection model and algorithms based on network streams analysis,” in *Proc. IEEE FITME*, Oct. 2010, pp. 55–58.

5. S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, *et al.*,“Detecting P2P botnets through network behavior analysis and machine learning,” in *Proc. 9th Annu. Int. Conf. PST*, Jul. 2011,pp. 174