CONTROLLING IP SPOOFING THROUGH INTER-DOMAIN PACKET FILTERS

By

A.BHARATHI

Reg. No: 0820108006

of

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna university, Coimbatore)

COIMBATORE – 641 006

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements
for the award of the degree
of

MASTER OF ENGINEERING
IN

COMPUTER SCIENCE AND ENGINEERING

MAY, 2010

BONAFIDE CERTIFICATE

Certified that this project report titled "**Controlling ip spoofing through IDPF (Inter Domain Packet Filters)**" is the bonafide work of **BHARATHI.A(0820108001)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report of dissertation on the basis of which a degree or ward was conferred on an earlier occasion on this or any other candidate.

GUIDE

(Mrs. V.VANITHA)

HEAD OF THE DEPARTMENT

(Dr.S.THANGASAMY)

The candidate with **University Register No. 0820108006** was examined by us in Project Viva-Voce examination held on _17/5/10_

Internal Examiner

External Examiner

VIVEKANANDHA

INSTITUTE OF ENGINEERING AND TECHNOLOGY FOR WOMEN

ELAYAMPALAYAM, TIRUCHENGODE.

Department of Computer Applications

NATIONAL CONFERENCE
ON
" EMERGING TECHNOLOGIES IN ADVANCED COMPUTING AND COMMUNICATION "

13th March, 2010

CERTIFICATE

This is to Certify that ~~Mr.~~ / Ms. / ~~Dr.~~ A. BHARATHI of ME, Kumaraguru College of Engineering has participated in the " National Conference on ETACC '10 ", organized by " VIVACIOUS " the professional association of Computer Applications on 13th March 2010 and presented a paper on Controlling In Spoofing Through Inter-Domain Packet Filters

A. Ahamed Kabeer
HOD & Organizing Secretary

V.
Principal

Chairman & Secretary

ACKNOWLEDGEMENT

I express my profound gratitude to our Chairman **Padmabhusan Arutselvar Dr.N.Mahalingam B.sc, F.I.E** and Correspondent **Shri.Balasubramanian M.Com, M.B.A (USA)** and joint Correspondent **Dr.A.Selvakumar Ph.D.,** for giving this great opportunity to pursue this course.

I thank **Dr.S.Rama Chandran,** Principal, Kumaraguru College of Technology, Coimbatore, for providing me with the necessary facilities and Infrastructure to work on this project.

I express my sincere thanks to **Dr.S.Thangasamy,** Dean, Department of Computer Science and Engineering, for being my greatest of inspiration and for embedding the quest for innovative ideas.

I express my thanks to Mrs. P.Devaki, H.O.D Incharge, CSE Dept who has helpful in doing this project.

I record my thanks to the My Guide and project coordinator **Mrs.V.Vanitha, M.E,** Assistant professor, Department of Computer Science and Engineering, who has extraordinarily helpful in preparing this project work successfully.

I would like to convey my honest thanks to all Teaching staff members and Non Teaching staff members of the department for their support. I would like to thank all my classmates who gave me proper light moments and study breaks apart from extending some technical support whenever I needed them most.

I dedicate this Project work to my **Parents** for no reasons but feeling from bottom of my heart, without their love this work wouldn't possible.

ABSTRACT

The Distributed Denial-of-Service (DDoS) attack is a serious threat to the legitimate use of the Internet. Prevention mechanisms are thwarted by the ability of attackers to forge or spoof the source addresses in IP packets.

By employing IP spoofing, attackers can evade detection and put a substantial burden on the destination network for policing attack packets. In this paper, I propose an interdomain packet filter (IDPF) architecture that can mitigate the level of IP spoofing on the Internet.

A key feature of this scheme is that it does not require global routing information. IDPFs are constructed from the information implicit in Border Gateway Protocol (BGP) route updates and are deployed in network border routers.

The conditions under which the IDPF framework correctly works in that it does not discard packets with valid source addresses. Based on extensive simulation studies, the partial employment on the Internet, IDPFs can proactively limit the spoofing capability of attackers. In addition, they can help localize the origin of an attack packet to a small number of candidate networks.

ஆய்வுச்சுருக்கம்

இன்றைய காலக்கட்டதில் ஒரு பிணையத்தில் தகவல் பரிமாறுதல் மிக கவனமாகவும் ரகசியமாகவும் இருக்க வேண்டும் இல்லையென்றால் அந்த தகவல்கள் மற்றவர்களுக்கு வெளிப்பட்டு தவறான மாறுதல்கள் எற்பட வாய்ப்புண்டு.

ஒரு பிணையத்தில் அங்கிகாரம் பெற்ற கணினிகள் பல உள்ளன அதில் அங்கிகாரம் பெறாத உறுபினர்கள் சில தேவையற்ற தரவுகளை வழங்கிக்கு அனுப்பிகொண்டே இருக்கும், அப்படி அனுப்பும் போது வழங்கி செயலிழந்து அங்கிகாரம் பெற்ற உறுப்பினர்களுக்கு தேவையான செய்திகளை தர முடியாமல் போய்விடுகிறது அது ஒரு பிணையத்தின் செயல்ப்பாட்டை குறைக்கும்.

அந்த தேவையற்ற தரவுகளால் ஒரு பிணையத்தின் வேகமான செயல்ப்பாட்டை மிகவும் குறைக்கும் அதனால் பணி இடங்களில் ஒழுங்கான பிணையத்தை உருவாக்க முடியாமல் போகிறது.

இந்த ஆய்வில் ஐடிபிஃப் வழங்கி என்ற உறுப்பை பயன்படுத்தி உள்வாங்கும் தரவுகளை சோதித்து தேவையற்ற தரவுகளை நீக்கி விட்டு தேவையான தரவை மட்டும் வழங்கிக்கு அனுப்பும்.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
DDoS	Distributed Denial-of-Service
WLANS	Wireless Local Area Networks
IP	Internet Protocol
AS	Autonomous System
BGP	Border Gateway Protocol
IDPF	Inter Domain Packet Filters
GUI	Graphic User Interface
AWT	Abstract Window Tool
JFC	Java Foundation Classes

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF NETWORK SECURITY

In the field of networking, the specialist area of network security consists of the provisions made in an underlying computer network infrastructure, policies adopted by the network administrator to protect the network and the network-accessible resources from unauthorized access, and consistent and continuous monitoring and measurement of its effectiveness (or lack) combined together.

Network security involves all activities that organizations, enterprises, and institutions undertake to protect the value and ongoing usability of assets and the integrity and continuity of operations.

1.1.1 Network Security Tools

Secure network infrastructure: Switches and routers have hardware and software features that support secure connectivity, perimeter security, intrusion protection, identity services, and security management. Dedicated network security hardware and software-Tools such as firewalls and intrusion detection systems provide protection for all areas of the network and enable secure connections.

1.2 NETWORKS

There are wired and wireless networks. The first is widely known as infrastructure networks since the gateways that connect them to other networks (like the internet) are fixed and wired. The bridges in these networks are also known as base stations. In an environment like this, a node is able to roam freely and establish a connection link with the nearest base station that is within its communication range. As the mobile node moves out of the base station that it was connected with, it falls into the range of another and hand off occurs between the old base station and the current one, enabling the mobile

unit to continue communication seamlessly through the network. These types of networks are most widely applied in office areas.

The second type of wireless networks is the infrastructure less mobile network that is also known as an ad hoc network.

1.3 IP Spoofing

The basic protocol for sending data over the Internet and many other computer networks is the Internet Protocol ("IP"). The header of each IP packet contains, among other things, the numerical source and destination address of the packet. The source address is normally the address that the packet was sent from. By forging the header so it contains a different address, an attacker can make it appear that the packet was sent by a different machine. The machine that receives spoofed packets will send response back to the forged source address, which means that this technique is mainly used when the attacker does not care about response or the attacker has some way of guessing the response.

In certain cases, it might be possible for the attacker to see or redirect the response to his own machine. The most usual case is when the attacker is spoofing an address on the same LAN or WAN.

IP spoofing is most frequently used in denial-of-service attacks. In such attacks, the goal is to flood the victim with overwhelming amounts of traffic, and the attacker does not care about receiving responses to his attack packets. Packets with spoofed addresses are thus suitable for such attacks. They have additional advantages for this purpose - they are more difficult to filter since each spoofed packet appears to come from a different address, and they hide the true source of the attack. Denial of service attacks that use spoofing typically randomly choose addresses from the entire IP address space, though more sophisticated spoofing mechanisms might avoid unroutable addresses or unused portions of the IP address space. The proliferation of large botnets makes spoofing less important in denial of service attacks, but attackers typically have spoofing available as a tool, if they want to use it, so defenses against denial-of-service attacks that

rely on the validity of the source IP address in attack packets might have trouble with spoofed packets. Backscatter, a technique used to observe denial-of-service attack activity in the Internet, relies on attackers' use of IP spoofing for its effectiveness.

IP spoofing can also be a method of attack used by network intruders to defeat network security measures, such as authentication based on IP addresses. This method of attack on a remote system can be extremely difficult, as it involves modifying thousands of packets at a time. This type of attack is most effective where trust relationships exist between machines. For example, it is common on some corporate networks to have internal systems trust each other, so that a user can log in without a username or password provided he is connecting from another machine on the internal network (and so must already be logged in). By spoofing a connection from a trusted machine, an attacker may be able to access the target machine without authenticating.

Packets sent using the IP protocol include the IP address of the sending host. The recipient directs replies to the sender using this source address. However, the correctness of this address is not verified by the protocol. The IP protocol specifies no method for validating the authenticity of the packet's source. This implies that an attacker can forge the source address to be any desired. This is almost exclusively done for malicious or at least inappropriate purposes. Given that attackers can exploit this weakness for many attacks, it would be beneficial to know if network traffic has spoofed source addresses.

This is a well-known problem and has been well described In all but a few rare cases, sending spoofed packets is done for illegitimate purposes Sending IP packets with forged source addresses is known as *packet spoofing* and is used by attackers for several purposes. These include obscuring the true source of the attack, implicating another site as the attack origin, pretending to be a trusted host, hijacking or intercepting network traffic, or causing replies to target another system.

Because none of these are desirable, it is useful to determine if a packet has a spoofed source address. In cases where an ongoing attack is occurring it is beneficial to determine if the attack is from a particular location. In many cases we are able to determine when packets are spoofed, and generally from where they originate. Spoofing of network traffic can occur at many layers. Examples include network layer spoofing as

well as session and application layer spoofing (e.g. email spoofing). All of these have significant security concerns. However, for the purposes of this paper we will focus only IP packet spoofing. A related issue is attacks that cause packets to be routed to a different host than the sender intends. These are attacks on routing] and the DNS system. Packet spoofing is restricted to false source addresses in the IP packet header.

IP spoofing will remain popular for a number of reasons. First, IP spoofing makes isolating attack traffic from legitimate traffic harder: packets with spoofed source addresses may appear to be from all around the Internet. Second, it presents the attacker with an easy way to insert a level of indirection. As a consequence, substantial effort is required to localize the source of the attack traffic. Finally flood attacks use IP spoofing and require the ability to forge source addresses.

CHAPTER 2

LITERATURE SURVEY

2. 1 DENIAL-OF-SERVICE

A "denial-of-service" attack is characterized by an explicit attempt by attackers to prevent legitimate users of a service from using that service. Attacks can be directed at any network device, including attacks on routing devices and web, electronic mail, or Domain Name System servers.

A DoS attack can be perpetrated in a number of ways. The five basic types of attack are:

1. Consumption of computational resources, such as bandwidth, disk space, or processor time
2. Disruption of configuration information, such as routing information.
3. Disruption of state information, such as unsolicited resetting of TCP sessions.
4. Disruption of physical network components.
5. Obstructing the communication media between the intended users and the victim so that they can no longer communicate adequately.

2.2 ROUTING

Routing is the process of selecting paths in a network along which to send network traffic. Routing is performed for many kinds of networks, including the telephone network, electronic data networks (such as the Internet), and transportation networks. This article is concerned primarily with routing in electronic data networks using packet switching technology.

In packet switching networks, routing directs packet forwarding, the transit of logically addressed packets from their source toward their ultimate destination through intermediate nodes; typically hardware devices called routers, bridges, gateways,

firewalls, or switches. General-purpose computers with multiple network cards can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the routers' memory, is very important for efficient routing. Most routing algorithms use only one network path at a time, but multipath routing techniques enable the use of multiple alternative paths.

Routing, in a more narrow sense of the term, is often contrasted with bridging in its assumption that network addresses are structured and that similar addresses imply proximity within the network. Because structured addresses allow a single routing table entry to represent the route to a group of devices, structured addressing (routing, in the narrow sense) outperforms unstructured addressing (bridging) in large networks, and has become the dominant form of addressing on the Internet, though bridging is still widely used within localized environments.

router is a networking device whose software and hardware are usually tailored to the tasks of routing and forwarding information. For example, on the Internet, information is directed to various paths by routers.

2.3 BGP (BORDER GATEWAY PROTOCAL):

Each node only selects and propagates to neighbors a single best route to the destination, if any. Both the selection and the propagation of best routes are governed by locally defined routing policies.

Import policies:

Neighbor-specific import policies are applied upon routes learned from neighbors.

Export policies:

Whereas neighbor-specific export policies are imposed on locally selected best routes before they are propagates to the neighbors.

Based on the policies we select the path from source to destinations.

Each node only selects and propagates to neighbors a single best route to the destination, if any. Both the selection and the propagation of best routes are governed by

locally defined routing policies. Two distinct sets of routing policies are typically employed by a node: import policies and export policies. Neighbor-specific import policies are applied upon routes learned from neighbors. whereas neighbor-specific export policies are imposed on locally selected best routes before they are propagated to the neighbors. In general, import policies can affect the "desirability" of routes by modifying route attributes. Let r be a route (to destination d) received at v from node u. We denote by import (v <- u) [{r}] the possibly modified route that has been transformed by the import policies. The transformed routes are stored in v's routing table. The set of all such routes is denoted as candidateR (v, u);

CandidateR (v, d) = {r: import (v <- u) [{r}]! = {} r.prefix d ɣu E (v)}

Here, N (v) is the set of v's neighbors.

Among the set of candidate routes candidateR (v, d); node v selects a single best route to reach the destination based on a well-defined procedure. To aid in description, we shall denote the outcome of the selection procedure at node v, that is, the best route, as bestR (v, d) which reads the best route to destination d at node v. Having selected bestR (v, d) from candidateR (v, d) v then exports the route to its neighbors after applying neighbor-specific export policies. The export policies determine if a route should be forwarded to the neighbor and if so, they modify the route attributes according to the policies. We denote by export (v <- u) [{r}] the route sent to neighbor u by node v after node v applies the export policies on route r.

BGP is an incremental protocol: updates are generated only in response to network events. In the absence of any event, no route updates are triggered or exchanged between neighbors, and we say that the routing system is in a stable state.

2.4 IDPF (INTER DOMAIN PACKET FILTER)

IDPFs can independently be deployed in each S. IDPFs are deployed at the border routers so that IP packets can be inspected before they enter the network.

IDPFs are used locally exchange BGP Updates to compare the paths.

If the Source address is not valid it will discard the packets.

IDPFs are completely oblivious to the specifics of the announced routes. Following a network failure, the set of feasible upstream neighbors will not admit more

members during the period of routing convergence, assuming that AS relationships are static, which is true in most cases. Hence, for the first type of routing dynamics (network failure), there is no possibility that the filters will block a valid packet. We illustrate this as follows: Consider an IDPF-enabled AS v that is on the best route from s to d. Let u = bestU(s; d; v) and U = feasibleU(s; d; v).A link or router failure between u and s can have three outcomes: 1) ASu can still reach ASs, and u is still chosen to be the best upstream neighbor for packet M(s; d), that is, u =bestU(s; d; v). In this situation, although u may explore and announce multiple routes to v during the path exploration process [30], the filtering function of v is unaffected. 2) ASuis no longer the best upstream neighbor for packet M(s; d), and another feasible upstream neighbor U can reach AS s and is instead chosen to be the new best upstream neighbor (for M(s; d). Now, both u and u0 may explore multiple routes; however, since u0 has already announced a route (about s) to v, the IDPF at v can correctly filter (that is, accept) packetM(s; d), which is forwarded from u0. 3) No feasible upstream neighbors can reach s. Consequently, AS v will also not be able to reach s, and v will no longer be on the best route between s and d. No new packet M(s; d) should be sent through v.

M(s; d) should be sent through v. The other concern of routing dynamics relates to how a newly connected network (or a network recovered from a fail-down event) will be affected. In general, a network may start sending data immediately following the announcement of a (new) prefix, even before the route has had time to propagate to the rest of the Internet. During the time that the route should be propagated, packets from this prefix may be discarded by some IDPFs if the reachability information has not propagated to them. However, the mitigating factor here is that in contrast to the long convergence delay that follows failure, reachability for the new prefix will be distributed far more speedily. In general, the time taken for such new prefix information to reach an IDPF is proportional to the shortest ASpath between the IDPF and the originator of the prefix and independent of the number of alternate paths between the two. Previous work has established this bound with L being the diameter of the AS graph . We believe that in this short timescale, it is acceptable for IDPFs to potentially incorrectly behave (discarding valid packets). It must be noted that during BGP route convergence periods, without IDPF, BGP can also drop packets. One alternative solution is to allow a neighbor

to continue forwarding packets from a source within a grace period, after the corresponding network prefix has been withdrawn by the neighbor. In this case, during this short period, IDPFs may fail to discard spoofed attack packets. However, given that most DDoS attacks require a persistent train of packets to be directed at a victim, not discarding spoofed packets for this short period of time should be acceptable. We plan to further investigate the related issues in the future.

In short, IDPFs can handle the routing dynamics caused by network failures, which may cause long route convergence times. IDPFs may, however, drop packets in the network recovery events. We argue that this is not a big problem, since 1) the network recovery events typically have a short convergence time and 2) such events can also cause service disruptions in the original BGP without IDPF.

CHAPTER 3

METHODOLOGY

3.1 OVERVIEW

The approach used can be classified in 3 steps: (i) understanding the problem and existing solutions, (ii) implementing and evaluating proposed solution

3.2 UNDERSTANDING EXISTING APPROACHES

Route-based Packet Filter:

Route-based distributed packet filtering (DPF) uses routing information to determine if a packet arriving at a router— e.g., border router at an AS—is valid with respect to its inscribed source/destination addresses, given the reach ability constraints imposed by routing and network topology. A single AS can only exert a limited impact with respect to identifying and discarding forged IPflows.

As with routing, route-based packet filtering occurs at two time scales—packet forwarding/discard based on table look-up (fast) and filter table update (slow)—and thus its forwarding/discard function can be performed close to line speed subject to generic processing overhead.

Distributed Denial-of-Service (DDoS) attacks pose an increasingly grave threat to the Internet, as evident in recent DDoS attacks mounted on both popular Internet sites and the Internet infrastructure. Alarmingly, DDoS attacks are observed on a daily basis on most of the large backbone networks . One of the factors that complicate the mechanisms for policing such attacks is IP spoofing, which is the act of forging the source addresses in IP packets. By masquerading as a different host, an attacker can hide its true identity and location, rendering source based packet filtering less effective. It has been shown that a large part of the Internet is vulnerable to IP spoofing. In the existing system where there is no defense mechanism installed, First, IP spoofing makes isolating attack traffic from legitimate traffic harder, packets with spoofed source

addresses may appear to be from all around the Internet. Second, it presents the attacker with an easy way to insert a level of indirection. As a consequence, substantial effort is required to localize the source of the attack traffic. All the Prevention mechanisms in the existing system are thwarted by the ability of attackers to forge or spoof the source addresses in IP packets.

Disadvantages:

- ☐ IP spoofing may easily occur. Because the packet-filtering router permits or denies a network connection based on the source and destination addresses of the packet, any attack that uses valid IP address may not be detected.

- ☐ Packet-filtering rules are comparatively harder to be designed and configured.

3.3 PROPOSED SYSTEM:

Inspired by the route-based packet filters , we propose an interdomain packet filter (IDPF) architecture, a router based packet filter system that can be constructed solely based on the locally exchanged BGP updates, assuming that all ASs employ a set of routing policies that are commonly used today . They showed that packet filters constructed based on the global routing information can significantly limit IP spoofing when deployed in just a small number of ASs. In this work, we extend the idea and demonstrate that filters that are built based on local BGP updates can also be effective. First, we describe how we can practically construct IDPFs at an AS by only using the information in the locally exchanged BGP updates. Second, we establish the conditions under which the proposed IDPF framework works correctly in that it does not discard packets with valid source addresses. Third, to evaluate the effectiveness of the proposed architecture, we conduct extensive simulation studies based on AS topologies and AS paths extracted from real BGP data. The results show that, even with partial deployment, the architecture can proactively limit an attacker's ability to spoof packets. When a spoofed packet cannot be stopped, IDPFs can help localize the attacker to a small number of candidate ASs, which can significantly improve the IP traceback situation . In addition, IDPF-enabled ASs(and their customers) provide better protection against IP spoofing

attacks than the ones that do not support IDPFs. This should give network administrators incentives. to deploy IDPFs.

Definition1: (stable routing state). A routing system is in a Stable state if all the nodes have selected a best route to reach other nodes and no route updates are generated (or propagated).

Definition 2: (route-based packet filtering). Node v accepts packet M(s, d) that is forwarded from node u if and only if e(u,v) belongs to R(s,d). Otherwise, the source address of the packet is spoofed, and the packet is discarded by v.

Definition 3: (correctness of packet filtering). A packet filter is correct if it does not discard packets with valid source addresses when the routing system is stable.

Advantages:

- IDPFs can significantly limit the spoofing capability of an attacker.
- Moreover, they also help pinpoint the true origin of an attack packet to be within a small number of candidate networks, thus simplifying the reactive IP trace back process.

3.4 PROBLEM FORMULATION:

Distributed Denial-of-Service (DDoS) attacks pose an increasingly grave threat to the Internet, as evident in recent DDoS attacks mounted on both popular Internet sites and the Internet infrastructure. Alarmingly, DDoS attacks are observed on a daily basis on most of the large backbone networks. One of the factors that complicate the mechanisms for policing such attacks is IP spoofing, which is the act of forging the source addresses in IP packets. By masquerading as a different host, an attacker can hide its true identity and location, rendering source based packet filtering less effective.

Recently, attackers have increasingly been staging attacks via botnets. In this case, since the attacks are carried out through intermediaries, that is, the compromised "bots," attackers may not utilize the technique of IP spoofing to hide their true identities . IP spoofing will remain popular for a number of reasons. First, IP spoofing makes isolating attack traffic from legitimate traffic harder: packets with spoofed source

addresses may appear to be from all around the Internet. Second, it presents the attacker with an easy way to insert a level of indirection. As a consequence, substantial effort is required to localize the source of the attack traffic. Finally, many popular attacks such as man-in-the-middle attacks, reflector-based attacks, and TCP SYN flood attacks use IP spoofing and require the ability to forge source addresses.

3.5 MODULES

3.5.1Implementation of AS Relationships and Routing Policies.

3.5.2 Design of Inter Domain Packet Filters.

3.5.3 Construction of Inter Domain Packet Filters.

3.5.4 Handling Routing Dynamics & Incremental Deployment.

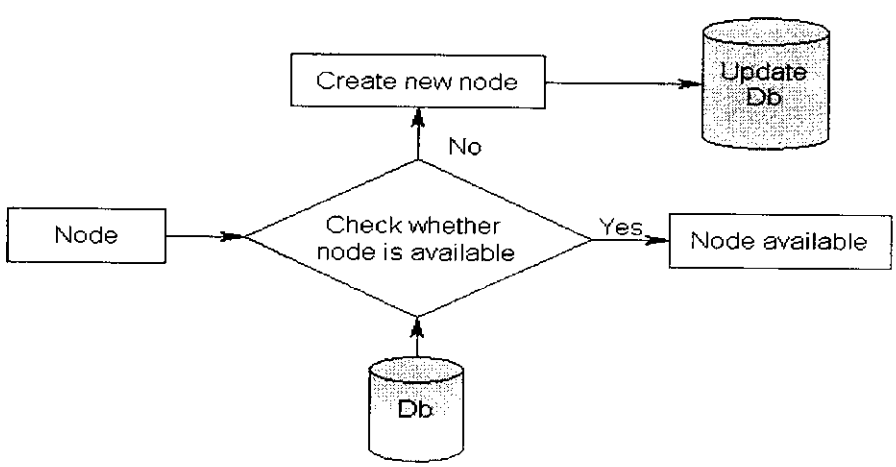
3.5.5 IDPFs with BGP Updates and Non overlapping Prefixes.

3.5.1 Implementation of AS Relationships and Routing Policies.

When there is no confusion, route r and its AS path r:as path are interchangeably used. For convenience, we also consider a specific destination AS d. All route announcements and withdrawals are specific to the network prefixes owned by d. For simplicity, notation d is also used to denote the network prefixes owned by the AS d. As a consequence, a route r that can be used to reach the network prefixes owned by destination d may simply be expressed as a route to reach destination d. Each node only selects and propagates to neighbors a single best route to the destination, if any. Both the selection and the propagation of best routes are governed by locally defined routing policies. Two distinct sets of routing policies are typically employed by a node: import policies and export policies. Neighbor-specific import policies are applied upon routes learned from neighbors, whereas neighbor-specific export policies are imposed on locally selected best routes before they are propagated to the neighbors. Among the set of candidate routes candidate R(v,d) node v selects a single best route to reach the destination based on a well-defined procedure. To aid in description, we shall denote the outcome of the selection procedure at node v, that is, the best route, as , bestR(v,d) which reads the best route to destination d at node v. Having selected bestR(v,d) from

candidateR(v,d), v then exports the route to its neighbors after applying neighbor-specific export policies. The export policies determine if a route should be forwarded to the neighbor, and if so, they modify the route attributes according to the policies. We denote by export (c =>u){r} the route sent to neighbor u by node v after node v applies the export policies on route r.

In this module, we construct a topology structure. Here we use mesh topology because of its unstructured nature. Topology is constructed by getting the names of the nodes and the connections among the nodes as input from the user. While getting each of the nodes, their associated port and ip address is also obtained. For successive nodes, the node to which it should be connected is also accepted from the user. While adding nodes, comparison will be done so that there would be no node duplication. Then we identify the source and the destinations.



MODULE 1:

3.5.2 Design of Inter Domain Packet Filters

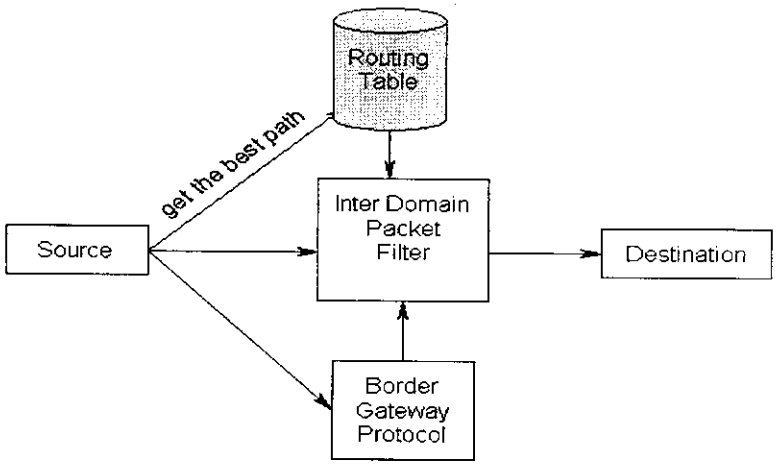
The intuition behind the IDPF architecture, describe how IDPFs are constructed using BGP route updates, and establish the correctness of IDPFs. After that, we discuss

the case where ASs have routing policies that are less restrictive than the others. Let M(s,d) denote a packet whose source address is s (or more generally, the address belongs to AS s) and whose destination address is d. A packet filtering scheme decides whether a packet should be forwarded or dropped based on certain criteria. In the context of preventing IP spoofing, an ideal packet filter should discard spoofed packets while allowing legitimate packets to reach the destinations. Since, even with the perfect routing information, the route-based packet filters cannot identify all spoofed packets, a valid packet filter should focus on not dropping any legitimate packets while providing the ability to limit spoofed packets. A packet filter is correct if it does not discard packets with valid source addresses when the routing system is stable. Clearly, the route-based packet filtering is correct, because valid packets from source s to destination d will only traverse the edges onbestR(s,d). Computing route-based packet filters requires the knowledge of bestR(s,d) on every node, which is impossible in BGP. IDPF overcomes this problem.

3.5.3 Construction of Inter Domain Packet Filters

It is worth noting that IDPFs filter packets based on whether the reachability information of a network prefix is propagated by a neighbor and not on how the BGP updates are propagated. As long as ASs propagate network reachability information, IDPFs work correctly. Moreover, the effectiveness of IDPFs is determined largely by the size of feasibleR(s,d), which is a function of the (relatively static) AS relationships. Hence, how the BGP updates are propagated does not affect both the correctness and the performance of IDPFs. a feasible route r between source s and destination d. Let v belong r.as path and let u be the feasible upstream neighbor of node v along r. When the routing system is stable, export(u -> v) [{bestR(u,s)}] !={}, assuming that all ASs follow the import and export routing policies. It is worth noting that IDPFs filter packets based on whether the reachability information of a network prefix is propagated by a neighbor and not on how the BGP updates are propagated. As long as ASs propagate network reachability information according to the rules, IDPFs work correctly. Moreover, the effectiveness of IDPFs is determined largely by the size of feasibleR(v,d), which is a

function of the (relatively static) AS relationships. Hence, how the BGP updates are propagated does not affect both the correctness and the performance of IDPFs.



3.5.4 Handling Routing Dynamics & Incremental Deployment

IDPFs can independently be deployed in each AS. IDPFs are deployed at the border routers so that IP packets can be inspected before they enter the network. By deploying IDPFs, an AS constrains the set of packets that a neighbor can forward to the AS: a neighbor can only successfully forward a packet M(s,d) to the AS after it announces the reachability information of s. All other packets are identified to carry spoofed source addresses and are discarded at the border router of the AS. In the worst case, even if only a single AS deploys IDPF and spoofed IP packets can get routed all the way to the AS in question, using an IDPF perimeter makes it likely that spoofed packets will be identified and, hence, blocked at the perimeter. Clearly, if the AS is well connected, launching a DDoS attack upon the perimeter itself takes a lot more effort than targeting individual hosts and services within the AS. In contrast, ASs that do not deploy IDPF offer relatively little protection to the internal hosts and services. Therefore, an AS has direct benefits of deploying IDPFs. In general, by deploying IDPFs, an AS can also protect other ASs to which the AS transports traffic, in particular the customer ASs. It

can similarly be understood that an IDPF node limits the set of packets forwarded by a neighbor and destined for a customer of the AS.

Handling Routing Dynamics:

So far, we have assumed that the AS graph is a static structure. In reality, the graph changes, triggering the generation of BGP updates and altering the paths that ASs use in reaching each other. In this section, we examine how routing dynamics affects the operation of IDPFs. We consider two different types of routing dynamics: 1) those caused by network failures and 2) those caused by the creation of a new network (or recovery from a fail-down network event). Routing dynamics caused by routing policy changes can similarly be addressed, and we omit them here. IDPFs are completely oblivious to the specifics of the announced routes. Following a network failure, the set of feasible upstream neighbors will not admit more members during the period of routing convergence, assuming that AS relationships are static, which is true in most cases. Hence, for the first type of routing dynamics (network failure), there is no possibility that the filters will block a valid packet. We illustrate this as follows: Consider an IDPF-enabled AS v that is on the best route from s to d. Let u = bestU (s,d,v) and U = feasible U(s,d,v). A link or router failure between u and s can have three outcomes:

1) ASu can still reach ASs, and u is still chosen to be the best upstream neighbor for packet M(s,d) . In this situation, although u may explore and announce multiple routes to v during the path exploration process, the filtering function of v is unaffected.

2) ASu is no longer the best upstream neighbor for packet M(s,d) and another feasible upstream neighbor u' belong U can reach AS s and is instead chosen to be the new best upstream neighbor (for M(s,d)). Now, both u and u' may explore multiple routes; however, since u0 has already announced a route (about s) to v, the IDPF at v can correctly filter (that is, accept) packet (s,d) which is forwarded from u'.

3) No feasible upstream neighbors can reach s. Consequently, AS v will also not be able to reach s, and v will no longer be on the best route between s and d. No new packet M(s,d) should be sent through v.

3.5.5 IDPFs with BGP Updates and Non overlapping Prefixes

To begin with, we study the performance of IDPFs with BGP updates and non overlapping prefixes. IDPFs cannot completely protect ASs from spoofing-based attacks.

Hence, we focus on its ability to limit the spoofing capability of attackers. IDPF is effective in controlling Victim Fraction (T) , especially with the IDPF VC coverage . The placement of IDPFs plays a key role in the effectiveness of IDPFs in controlling spoofing-based attacks although they recruit a larger number of nodes that support IDPFs. In general, it is more preferable for nodes with large degrees (such as big ISPs) to deploy IDPFs.. We see that overall, similar trends hold for all the years examined. However, it is worth noting that G2004c performs worse than G2004. This is because G2004c contains more edges and more AS paths by incorporating one-year BGP updates Attack Fraction(T) illustrates how effective IDPFs are in limiting the spoofing capability of attackers. In particular, Attack Fraction is the proportion of nodes from which an attacker cannot launch any spoofing-based attacks against any other nodes. Recall that Victim Trace Fraction(T) indicates the proportion of nodes that, under attack by packets with a source IP address, can pinpoint the true origin of the packets to be within at most T nodes.

BGP CONSTRUCTION:

Each node only selects and propagates to neighbors a single best route to the destination, if any. Both the selection and the propagation of best routes are governed by locally defined routing policies.

Two distinct sets of routing policies are typically employed by a node: import policies and export policies. Neighbor-specific import policies are applied upon routes learned from neighbors, whereas neighbor-specific export policies are imposed on locally selected best routes before they are propagated to the neighbors.

In general, import policies can affect the "desirability" of routes by modifying route attributes. Let r be a route (to destination d) received at v from node u. We denote by import (v <- u) [{r}] the possibly modified route that has been transformed by the

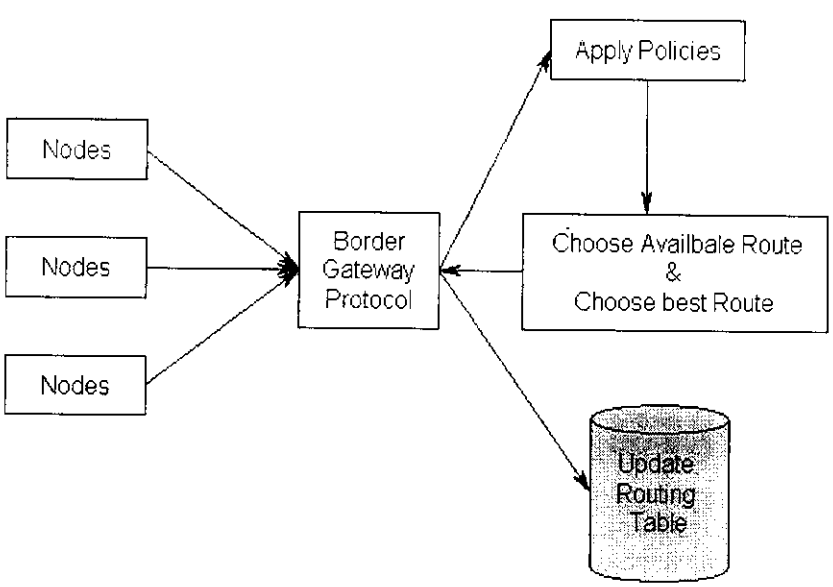
import policies. The transformed routes are stored in v's routing table. The set of all such routes is denoted as candidateR (v, u); CandidateR (v, d) = {r: import (v <- u) [{r}]! = {} r.prefix d ɣu E (v)}

Here, N (v) is the set of v's neighbors.

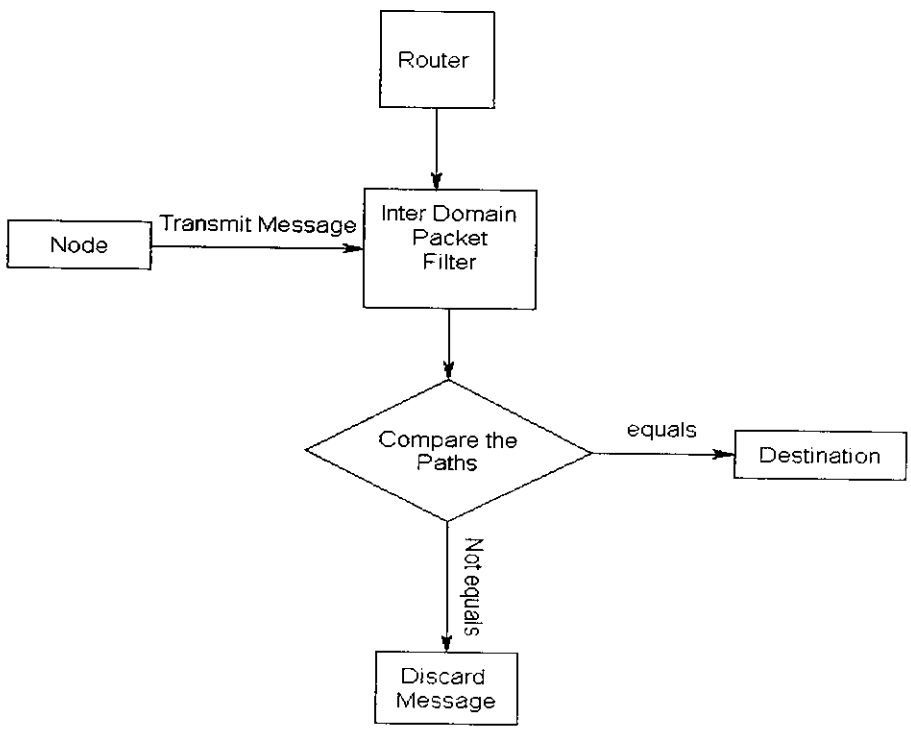
Among the set of candidate routes candidateR (v, d); node v selects a single best route to reach the destination based on a well-defined procedure. To aid in description, we shall denote the outcome of the selection procedure at node v, that is, the best route, as bestR (v, d) which reads the best route to destination d at node v. Having selected bestR (v, d) from candidateR (v, d) v then exports the route to its neighbors after applying neighbor-specific export policies.

The export policies determine if a route should be forwarded to the neighbor and if so, they modify the route attributes according to the policies. We denote by export (v <- u) [{r}] the route sent to neighbor u by node v after node v applies the export policies on route r.

BGP is an incremental protocol: updates are generated only in response to network events. In the absence of any event, no route updates are triggered or exchanged between neighbors, and we say that the routing system is in a stable state.



Based on the IDPF and BGP we will identify the packet will be spoofed or correct. If it's correct the messages allow to the destination or its spoofed means the packets will be discarded. IDPF framework works correctly in that it does not discard packets with valid source addresses.



CHAPTER 4

TECHNOLOGY INFRASTRUCTURE

4.1 CORE JAVA

Java can be used to create two types of programs: application and applet. An application is a program that runs on your computer, under the operating system of that computer. That is, an application created by java is more or less like one created using C or C++. When used to create application, java is not much different from any other computer language. Rather, it is java's ability to create applets that makes it important. An applet is an application designed to be transmitted over the internet and executed by a java-compatible Web Browser.

An applet is actually a tiny java program, dynamically downloaded across the network, just like an image, sound file, or video clip. The important difference is that an applet is an intelligent program, not just an animation or media file. In other words, an applet is a program that can react to user input and dynamically change-not just run the same animation or sound over and over [8].

Java having a major role in internet and the intranet application. The reason for this is quite simple: Java expends the universe pf objects that can move about freely in cyberspace. In a network, two very broad categories ob objects are transmitted between the server and your personal computer: passive information and dynamic, active programs.

Java Technology

Java technology is both a programming language and a platform.

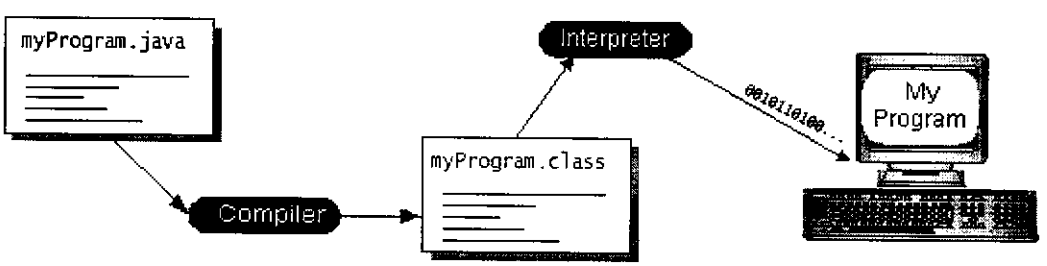
The Java Programming Language

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture neutral

- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



WORKING OF JAVA

You can think of Java byte codes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make "write once, run anywhere" possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any

implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.

Security

As you are likely aware, every time that you download a "normal" program, you are risking viral infection. Prior to java, most users did not download executable programs frequently, and those who did scan them for viruses prior to execution. Even so, most users still worried about the possibility of infecting their system with a virus. When you use a java-compatible web browser, you can safely download java applets without fear of viral infection or malicious intent. Java achieves this protection by confining a java program to the java execution environment and not allowing it access to other parts of computer.

Portability

Many types of computers and operating systems are in use throughout the world-and many are connected to the internet. For program to be dynamically downloaded to all the various type of platforms connected to the Internet, some means of generating portable executable code is needed.

Bytecode

The key that allows java to solve both the security and the portability problems just described is that output of a java compiler is not executable code. Rather, it is BYTECODE. Byte code is a highly optimized set of instruction designed to be executed by the java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreted code.

Simple

Java was designed to be easy for the professional programmer to learn and use effectively. Assuming that you have some programming experience, you will not find

java hard to master. If we know the basic concept of object-oriented programming, learning java will be even easier.

Object-Oriented

Object-Oriented programming is the core of java. In fact, all java programs are object-oriented-this isn't an option the way that it is in C++, for example. OOP is so integral to java that you must understand its basic principles before you can write even simple java programs.

Abstraction

The essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. For example, people do not think of a car as a set of ten of individual parts. They think of it as a well-defined object with its own unique behavior. So this ignore the details of how the engine, transmission, and braking systems work.

4.2 SWING

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit. Swing provides a native look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform.

Swing introduced a mechanism that allowed the look and feel of every component in an application to be altered without making substantial changes to the application code. The introduction of support for a pluggable look and feel allows Swing components to emulate the appearance of native components while still retaining the benefits of platform independence. This feature also makes it easy to make an application written in Swing look very different from native programs if desired. originally distributed as a separately downloadable library, Swing has been included as part of the Java Standard Edition since

release 1.2. The Swing classes and components are contained in the javax.swing package hierarchy[8].

Architecture

Swing is a platform-independent, Model-View-Controller GUI framework for Java. It follows a single-threaded programming model, and possesses the following traits:

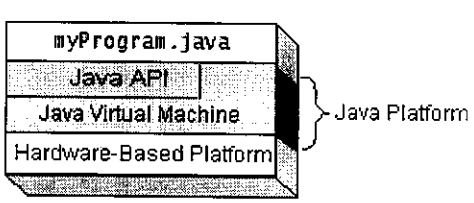
The Java Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.
The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.
The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do?, highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring performance close to that of native code without threatening portability.

Platform independence

Swing is platform independent both in terms of its expression (Java) and its implementation (non-native universal rendering of widgets).

Extensibility

Swing is a highly partitioned architecture, which allows for the "plugging" of various custom implementations of specified framework interfaces: Users can provide their own custom implementation(s) of these components to override the default implementations. In general, Swing users can extend the framework by extending existing (framework) classes and/or providing alternative implementations of core components.

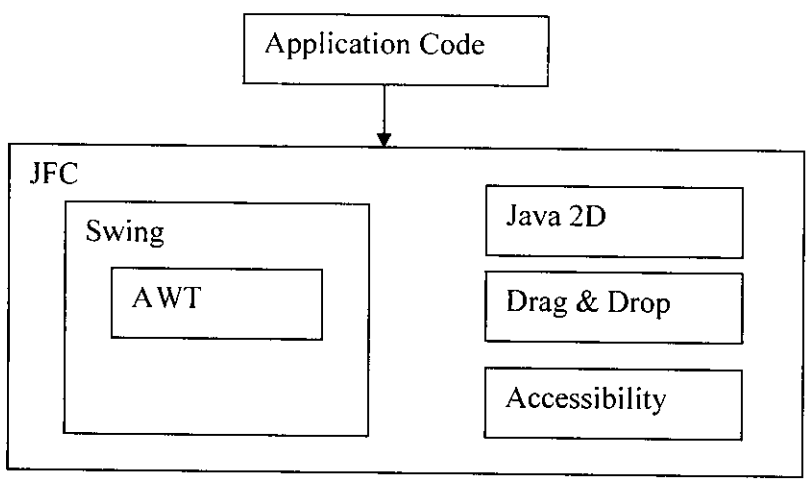
Component-oriented

Swing is a component-based framework. The distinction between objects and components is a fairly subtle point: concisely, a component is a well-behaved object with a known/specified characteristic pattern of behaviour. Swing objects asynchronously fire events, have "bound" properties, and respond to a well-known set of commands (specific to the component.) Specifically, Swing components are Java Beans components, compliant with the Java Beans Component Architecture specifications.

Customizable

Given the programmatic rendering model of the Swing framework, fine control over the details of rendering of a component is possible in Swing. As a general pattern, the visual representation of a Swing component is a composition of a standard set of

elements, such as a "border", "inset", decorations, etc. Typically, users will programmatically customize a standard Swing component (such as a JTable) by assigning specific Borders, Colors, Backgrounds, opacities, etc., as the properties of that component. The core component will then use these property (settings) to determine the appropriate renderers to use in painting its various aspects. However, it is also completely possible to create unique GUI controls with highly customized visual representation.



Swing class

What Can Java Technology Do?

The most common types of programs written in the Java programming language are *applets* and *applications*. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network.

Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a *servlet*. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provide a wide range of functionality. Every full implementation of the Java platform gives you the following features:

The essentials: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.

Applets: The set of conventions used by applets.

Networking: URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.

Internationalization: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.

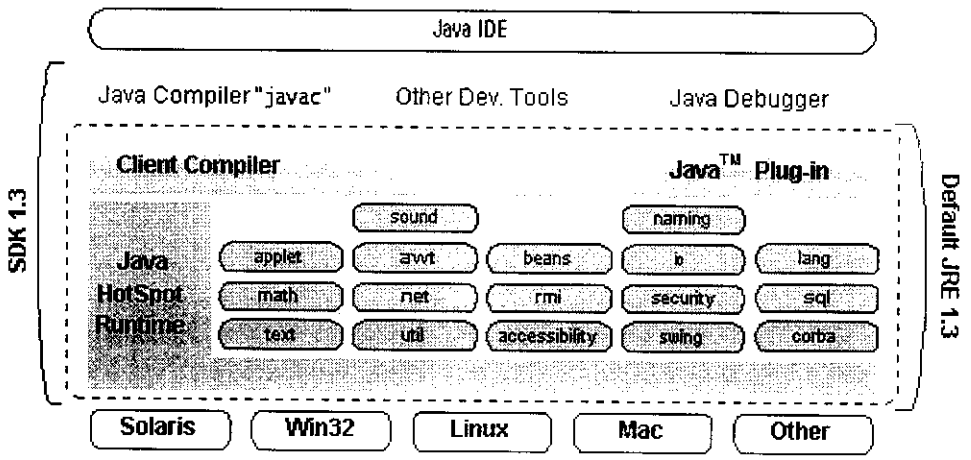
Security: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.

Software components: Known as JavaBeans™, can plug into existing component architectures.

Object serialization: Allows lightweight persistence and communication via Remote Method Invocation (RMI).

Java Database Connectivity (JDBC™): Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.



JAVA 2 SDK

4.3 ODBC

Microsoft Open Database Connectivity (ODBC) is a standard programming interface for application developers and database systems providers. Before ODBC became a *de facto* standard for Windows programs to interface with database systems, programmers had to use proprietary languages for each database they wanted to connect to. Now, ODBC has made the choice of the database system almost irrelevant from a coding perspective, which is as it should be. Application developers have much more important things to worry about than the syntax that is needed to port their program from one database to another when business needs suddenly change.

Through the ODBC Administrator in Control Panel, you can specify the particular database that is associated with a data source that an ODBC application program is written to use. Think of an ODBC data source as a door with a name on it. Each door will lead you to a particular database. For example, the data source named Sales Figures might be a SQL Server database, whereas the Accounts Payable data source could refer to an Access database. The physical database referred to by a data source can reside anywhere on the LAN.

The ODBC system files are not installed on your system by Windows 95. Rather, they are installed when you setup a separate database application, such as SQL Server Client or Visual Basic 4.0. When the ODBC icon is installed in Control Panel, it uses a file called ODBCINST.DLL. It is also possible to administer your ODBC data sources through a stand-alone program called ODBCADM.EXE. There is a 16-bit and a 32-bit

version of this program, and each maintains a separate list of ODBC data sources.

From a programming perspective, the beauty of ODBC is that the application can be written to use the same set of function calls to interface with any data source, regardless of the database vendor. The source code of the application doesn't change whether it talks to Oracle or SQL Server. We only mention these two as an example. There are ODBC drivers available for several dozen popular database systems. Even Excel spreadsheets and plain text files can be turned into data sources. The operating system uses the Registry information written by ODBC Administrator to determine which low-level ODBC drivers are needed to talk to the data source (such as the interface to Oracle or SQL Server). The loading of the ODBC drivers is transparent to the ODBC application program. In a client/server environment, the ODBC API even handles many of the network issues for the application programmer.

The advantages of this scheme are so numerous that you are probably thinking there must be some catch. The only disadvantage of ODBC is that it isn't as efficient as talking directly to the native database interface. ODBC has had many detractors make the charge that it is too slow. Microsoft has always claimed that the critical factor in performance is the quality of the driver software that is used. In our humble opinion, this is true. The availability of good ODBC drivers has improved a great deal recently. And anyway, the criticism about performance is somewhat analogous to those who said that compilers would never match the speed of pure assembly language. Maybe not, but the compiler (or ODBC) gives you the opportunity to write cleaner programs, which means you finish sooner. Meanwhile, computers get faster every year.

4.4 JDBC

In an effort to set an independent database standard API for Java, Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of "plug-in" database connectivity modules, or *drivers*. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC's framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

JDBC Goals

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

SQL Level API

The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to "generate" JDBC code and to hide many of JDBC's complexities from the end user.

SQL Conformance

SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.

1.JDBC must be implemental on top of common database interfaces

The JDBC SQL API must "sit" on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.

2.Provide a Java interface that is consistent with the rest of the Java system

Because of Java's acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.

3.Keep it simple

This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.

4.Use strong, static typing wherever possible

Strong typing allows for more error checking to be done at compile time; also, less errors appear at runtime.

5.Keep the common cases simple

Because more often than not, the usual SQL calls used by the programmer are simple SELECT's, INSERT's, DELETE's and UPDATE's, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

4.6 Networking

TCP/IP stack

The TCP/IP stack is shorter than the OSI one:

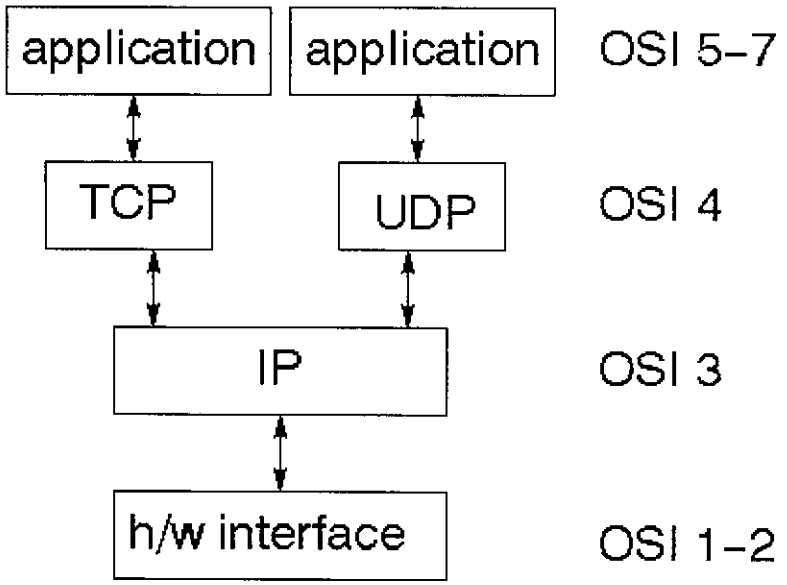


FIGURE 5 – TCP/IP STACK

TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

IP datagram's

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

TCP

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

Internet addresses

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

Network address

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

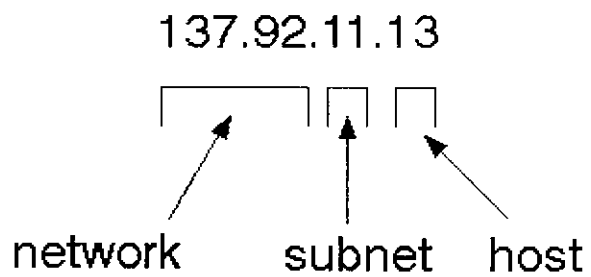
Subnet address

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

Host address

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

Total address



IP ADDRESSING

The 32 bit address is usually written as 4 integers separated by dots.

Port addresses

A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

Sockets

A socket is a data structure maintained by the system to handle network connections. A socket is created using the call socket. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with Read File and Write File functions.

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int family, int type, int protocol);
```

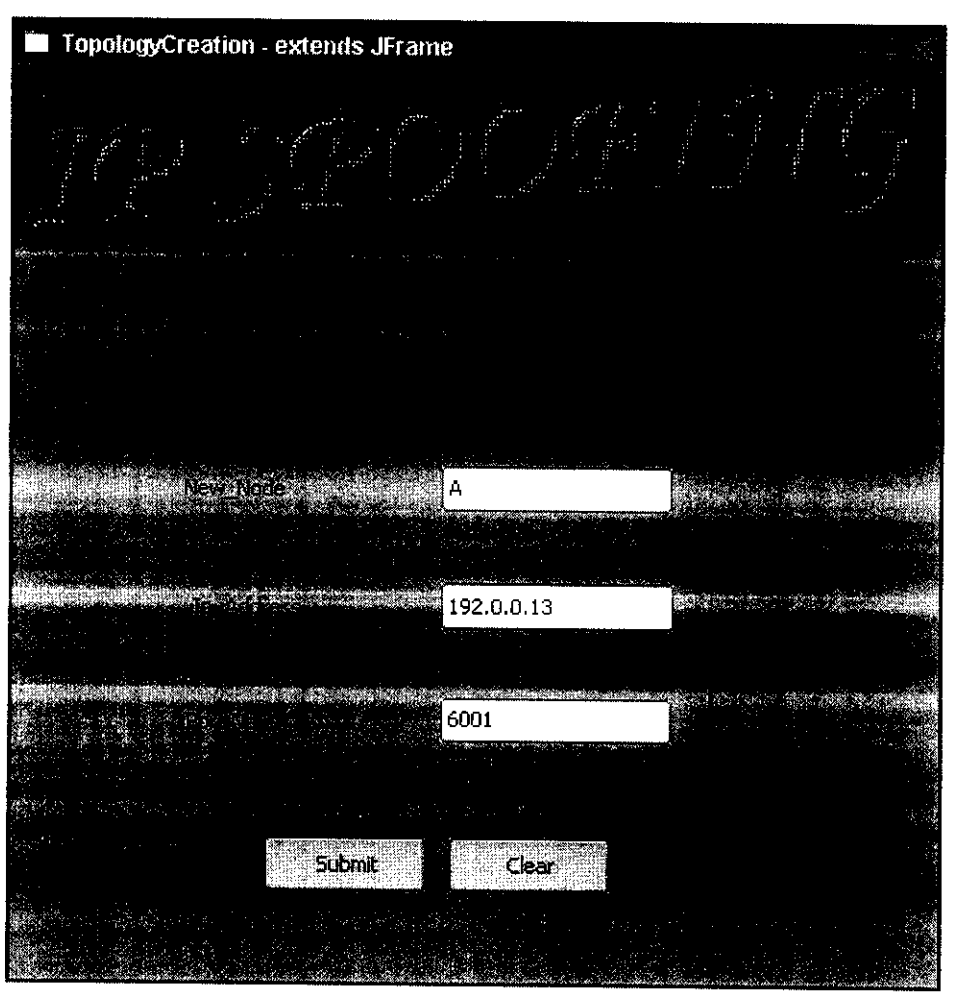
Here "family" will be AF_INET for IP communications, protocol will be zero, and type will depend on whether TCP or UDP is used.

CHAPTER 5

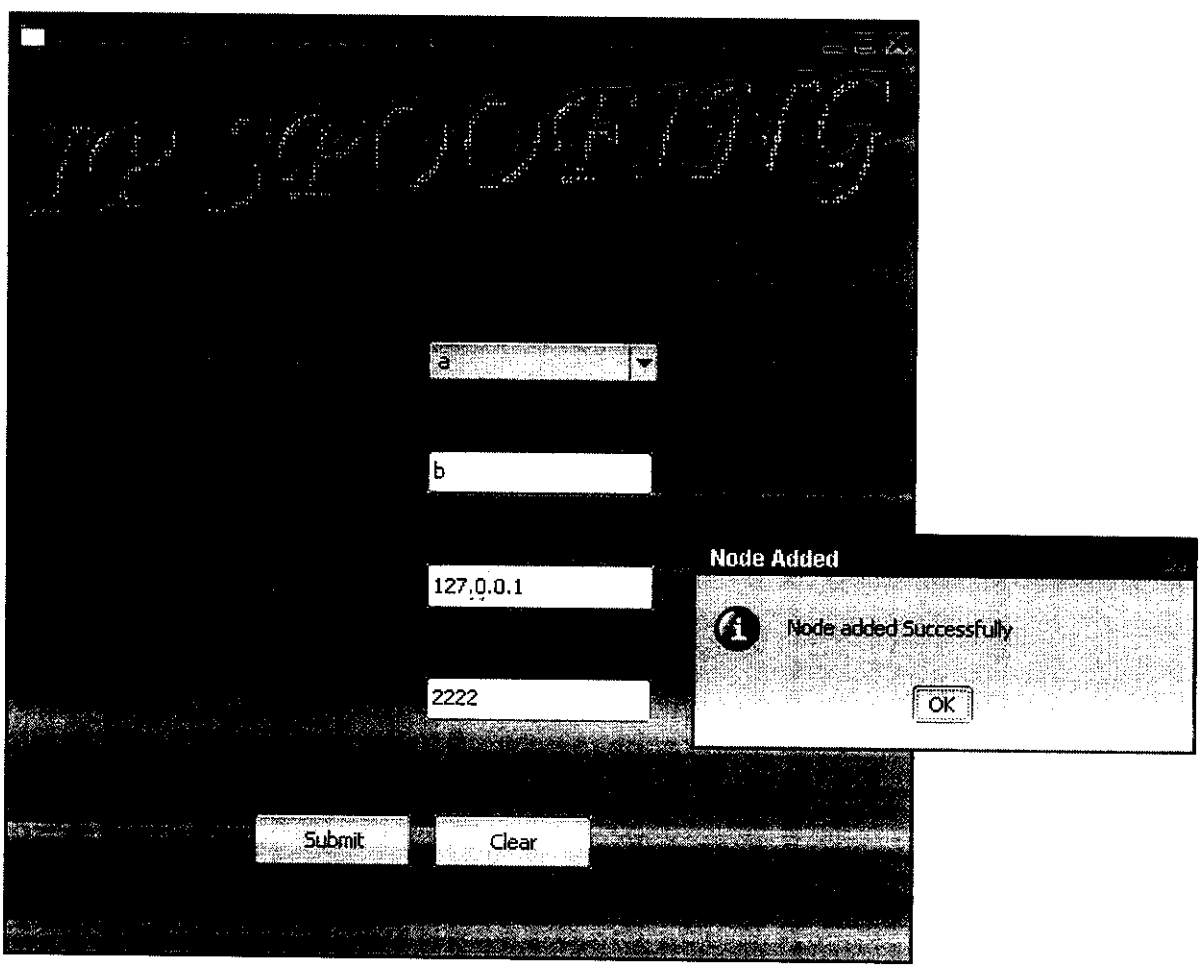
EXPERIMENTAL RESULTS

5.1 TOPOLOGY CREATION

- ☐ *Getting the names of the nodes.*
- ☐ *Port and ip address is obtained.*
- ☐ *For successive nodes, the node to which it should be connected is also accepted from the user.*



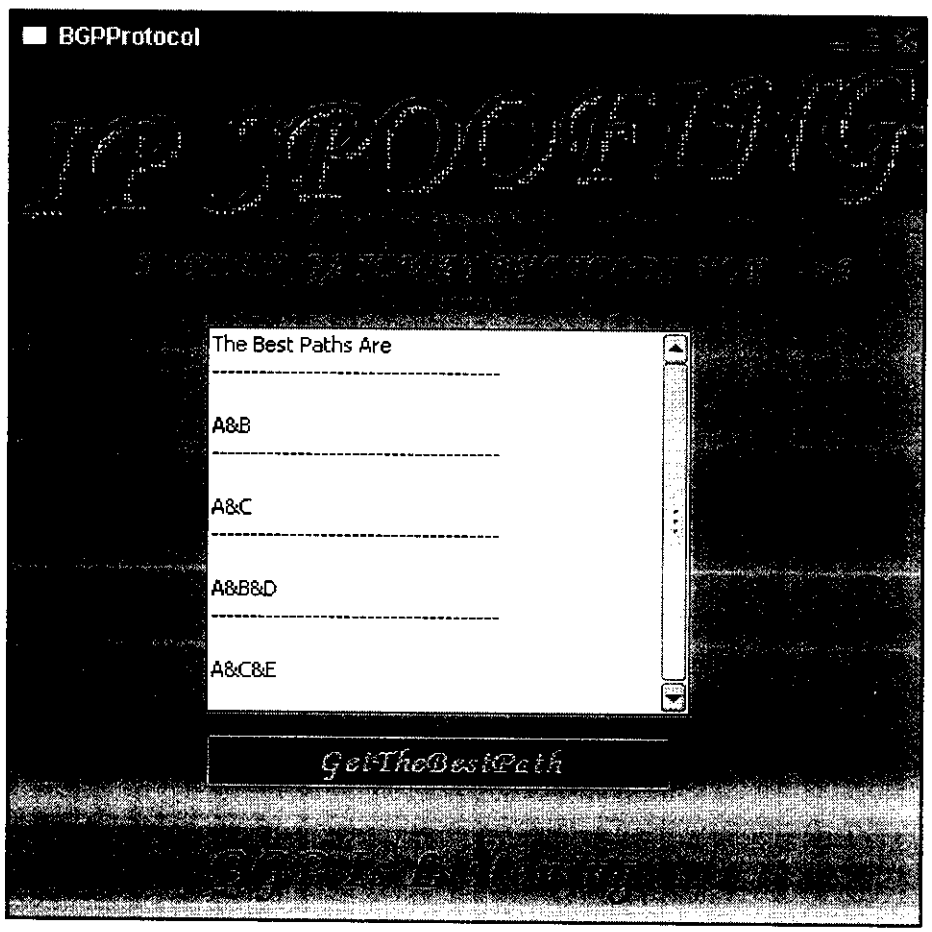
- *For successive nodes, the node to which it should be connected is also accepted from the user.*
- *While adding nodes, comparison will be done so that there would be no node duplication.*



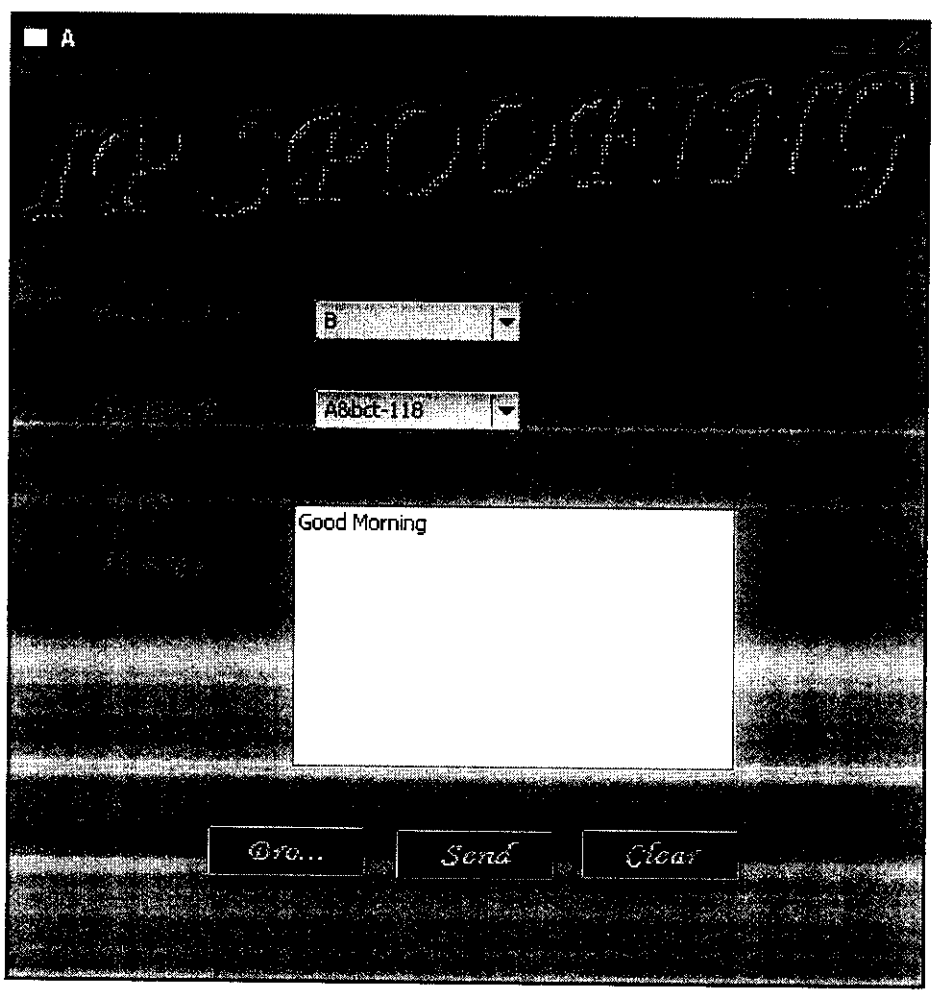
5.2 BGP CONSTRUCTION

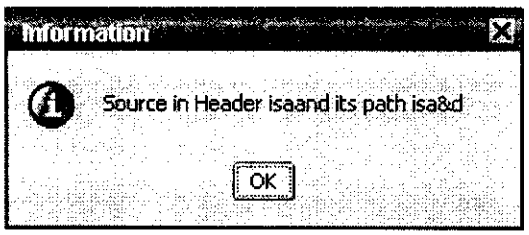
- ☐ ***Each node only selects and propagates to neighbors a single best route to the destination.***
- ☐ ***Both the selection and the propagation of best routes are governed by locally defined routing policies.***

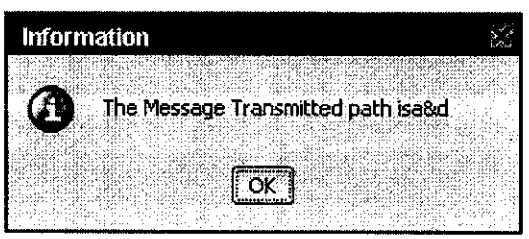
- *Neighbor-specific import policies are applied upon routes learned from neighbors.*
- *export policies are imposed on locally selected best routes before they are propagated to the neighbors.*

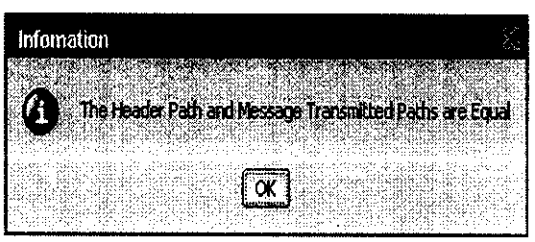


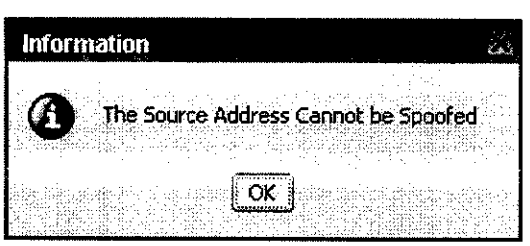
- ***IDPF Construction:***
  - *The IDPF is used to check the message before it will enter into the destination.*
  - *The IDPF will check the source address whether it's correct or its spoofed.*



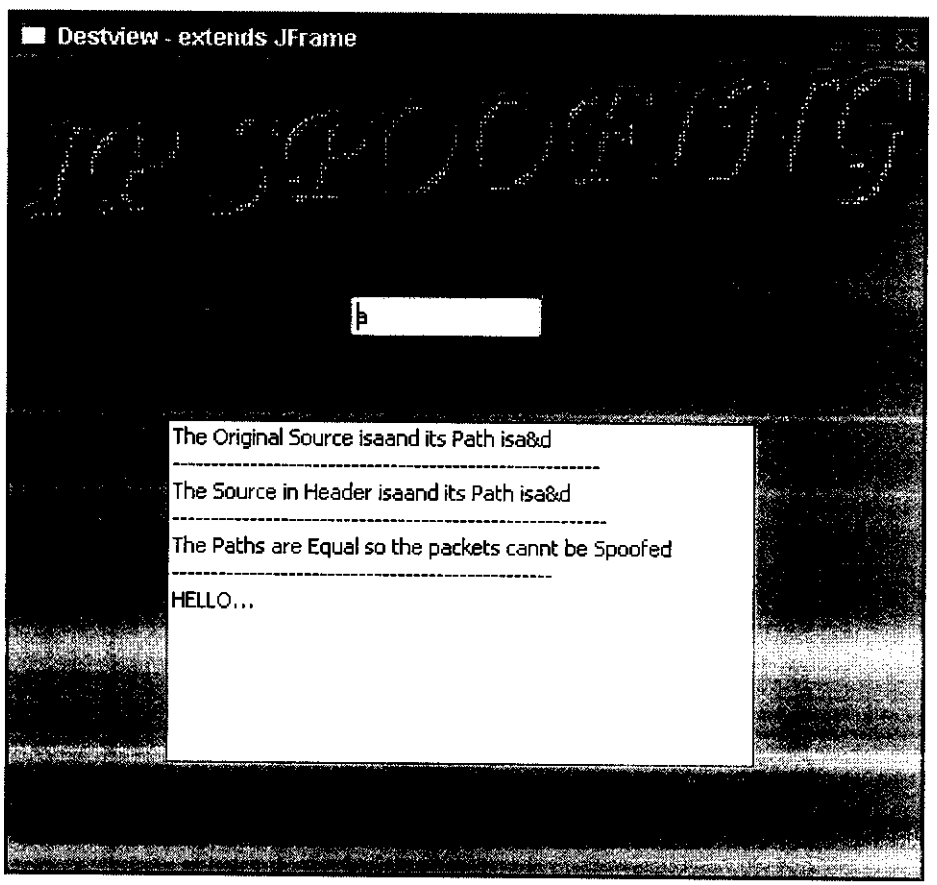

Information
Source in Header isaand its path isa&d
OK


Information
The Message Transmitted path isa&d
OK

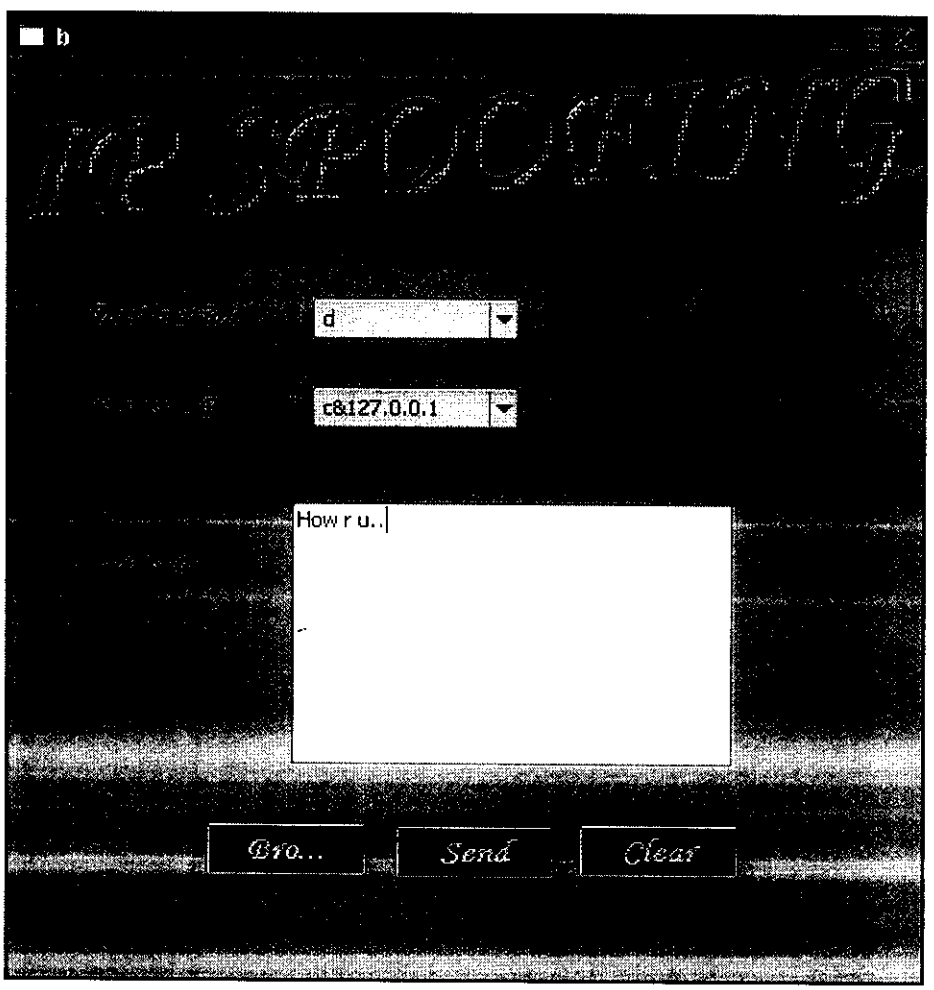

Infomation
The Header Path and Message Transmitted Paths are Equal
OK

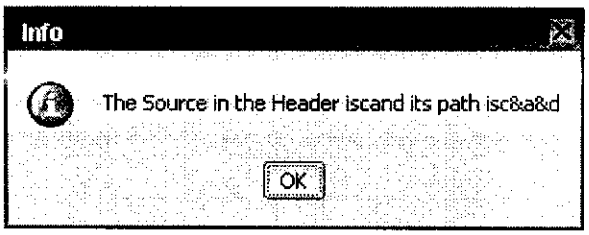

Information
The Source Address Cannot be Spoofed
OK

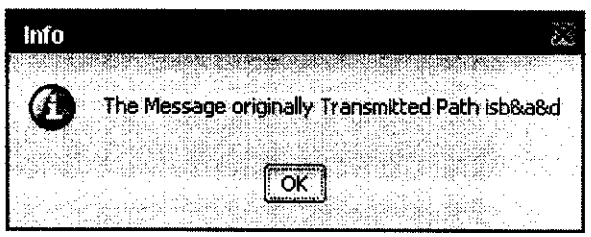
- ***Control the Spoofed Packets:***
    - *Based on the IDPF we will identify the packet will be spoofed or correct.*
    - *If its correct the message allow to the destination or its spoofed means the packets will be discarded.*

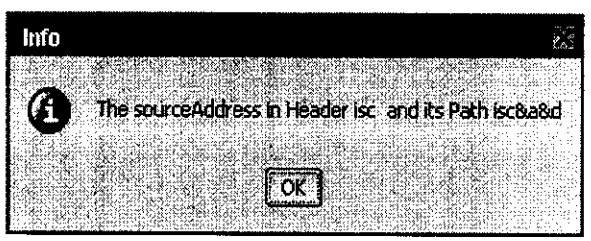


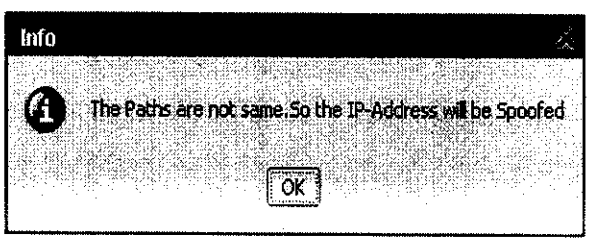
- **Control the Spoofed Packets:**
  - Based on the IDPF we will identify the packet will be spoofed or correct.
  - If its correct the message allow to the destination or its spoofed means the packets will be discarded.

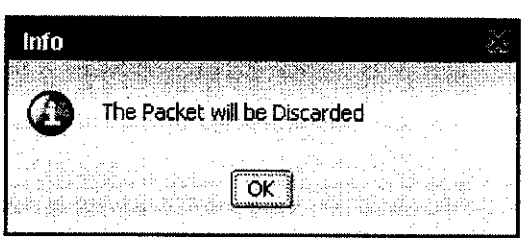



Info
The Source in the Header iscand its path isc&a&d
OK

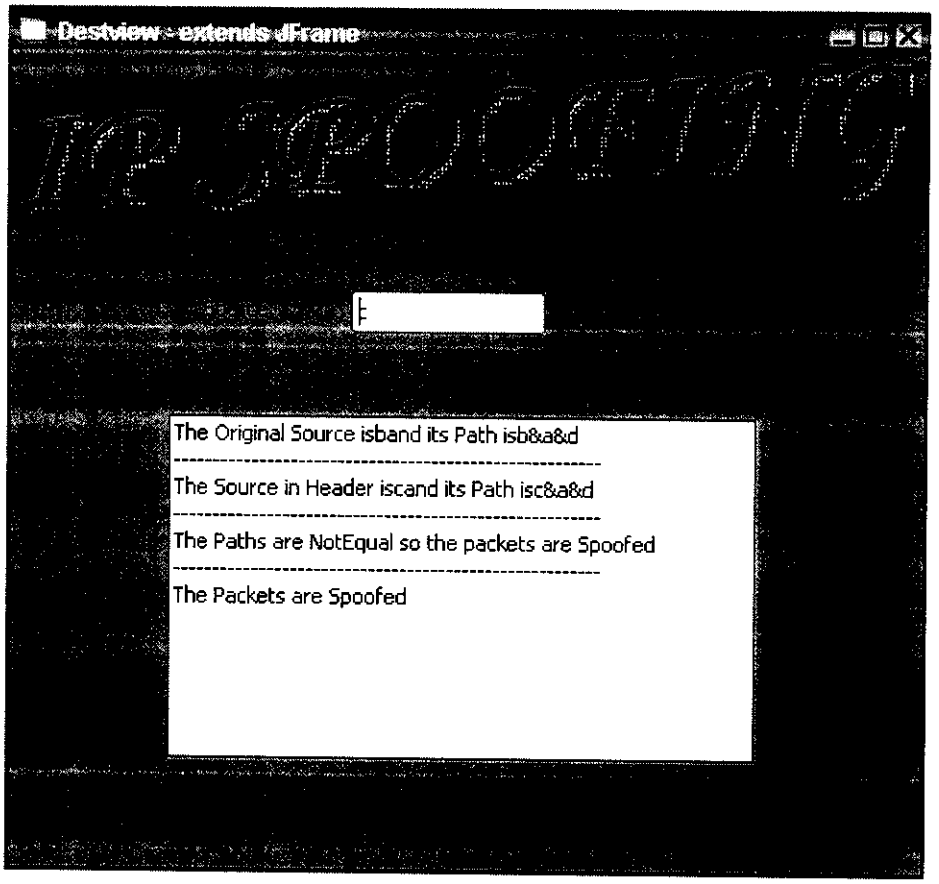

Info
The Message originally Transmitted Path isb&a&d
OK


Info
The sourceAddress in Header isc and its Path isc&a&d
OK


Info
The Paths are not same.So the IP-Address will be Spoofed
OK


Info
The Packet will be Discarded
OK

If packets are spoofed means it display that which as packets are spoofed.



CHAPTER 6

CONCLUSION AND FUTURE OUTLOOK

6.1 CONCLUSION

I have proposed and studied IDPF architecture as an effective countermeasure to the IP spoofing- based DDoS attacks. IDPFs rely on BGP update messages exchanged on the Internet to infer the validity of source address of a packet forwarded by a neighbor. I showed that IDPFs can easily be deployed on the current BGP-based Internet routing architecture. I studied the conditions under which the IDPF framework can correctly work without discarding any valid packets. Our simulation results showed that, even with partial deployment on the Internet, IDPFs can significantly limit the spoofing capability of attackers.

6.2 FUTURE OUTLOOK

It also helps pinpoint the true origin of an attack packet to be within a small number of candidate networks, thus simplifying the reactive IP traceback process.

/***/

APPENDIX 1

/***/

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.net.*;
import java.io.*;

/**
 * Summary description for BGPProtocol
 *
 */

public class BGPProtocol extends JFrame implements Runnable
{
        // Variables declaration
        private JLabel jLabel1;
        private JLabel jLabel2;
        private JLabel jLabel3;
        private JComboBox jComboBox1;
        private JButton jButton1;
        private JButton jButton2;
        private JPanel contentPane;
        private JTextArea jTextArea1;
        private JScrollPane jScrollPane1;
        String listen = "BGP Is Listening";
        Vector allNodes = new Vector();
        private String currentNode;
        Thread t;
        int count = 0;

        private String selectedNode;
        // End of variables declaration
```

```
public BGPProtocol( String curreNodeName,Vector allOtherNodes )
{
        super();

        System.out.println("-----"+curreNodeName);

        currentNode = curreNodeName;

        allNodes = allOtherNodes;

        System.out.println("The Vector elements are"+allNodes);

//      try
//      {
//              UIManager.setLookAndFeel("jane.ui.lf.jane.JaneLookAndFeel");
//      }
//      catch ( Exception ex )
//      {
//              System.out.println("Failed loading L&F: ");
//              System.out.println(ex);
//      }

        getThePathDetails();


        initializeComponent();

        LabelRun();
        //
        // TODO: Add any constructor code after initializeComponent call
        //

        /*try
        {
                UIManager.setLookAndFeel("jane.ui.lf.jane.JaneLookAndFeel");
        }
        catch ( Exception ex )
        {
                System.out.println("Failed loading L&F: ");
                System.out.println(ex);
```

```
        }*/


        this.setVisible(true);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
regenerated
     * by the Windows Form Designer. Otherwise, retrieving design might not work
properly.
     * Tip: If you must revise this method, please backup this GUI file for
JFrameBuilder
     * to retrieve your design properly in future, before revising this method.
     */

    private void initializeComponent()
    {
        System.out.println("vvvvvvvvvvvvvv");
        jLabel1 = new JLabel();
        jLabel2 = new JLabel();
        jLabel3 = new JLabel();
        jComboBox1 = new JComboBox(allNodes);
        jButton1 = new JButton();
        jButton2 = new JButton();
        jTextArea1 = new JTextArea();
        jScrollPane1 = new JScrollPane();
        jLabel3.setFont(new Font("Monotype Corsiva",Font.BOLD,30));
        jLabel3.setForeground(Color.blue);
        contentPane = (JPanel)this.getContentPane();

        //
        // jLabel1
        //
            jLabel1.setIcon(new ImageIcon("spoof.GIF"));
        //
        // jLabel2
        //
```

```
                jLabel2.setText("BORDER GATEWAY PROTOCOL FOR---
>"+currentNode);
                jLabel2.setFont(new Font("Monotype Corsiva",Font.BOLD,20));
                jLabel2.setForeground(Color.blue);
            //
            // jComboBox1
            //
            jComboBox1.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e)
                {
                    jComboBox1_actionPerformed(e);
                }

            });


            jScrollPane1.setViewportView(jTextArea1);
            //
            // jButton1
            //
            jButton1.setText(" Paths");
            jButton1.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e)
                {
                    jButton1_actionPerformed(e);
                }

            });
            //
            // jButton2
            //
            jButton2.setBackground(new Color(0,51,51));
            jButton2.setForeground(new Color(102,255,255));
            jButton2.setFont(new Font("Monotype Corsiva",Font.BOLD,18));
            jButton2.setText("GetTheBestPath");
            jButton2.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e)
                {
                    jButton2_actionPerformed(e);
                }
```

```
            });
            //
            // contentPane
            //
            contentPane.setLayout(null);
            contentPane.setBackground(new Color(166,166,255));
            addComponent(contentPane, jLabel1, 0,0,480,90);
            addComponent(contentPane, jLabel2, 60,60,380,100);
            //addComponent(contentPane, jComboBox1, 204,173,100,22);
        //  addComponent(contentPane, jButton1, 100,273,83,28);
            addComponent(contentPane, jButton2, 100,350,240,28);
            addComponent(contentPane, jScrollPane1,100,140,250,200);
            addComponent(contentPane,jLabel3,100,400,300,40);
            //
            // BGPProtocol
            //
            this.setTitle("BGPProtocol");
            this.setLocation(new Point(200, 200));
            this.setSize(new Dimension(471, 470));
            //this.setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);



            addWindowListener(new java.awt.event.WindowAdapter() {
             public void windowClosing(WindowEvent winEvt) {
    // Perhaps ask user if they want to save any unsaved files first.
    //System.exit(1);
        dispose();

  }
        });

        System.out.println( "sssssssssssssssss" );

        }

        /** Add Component Without a Layout Manager (Absolute Positioning) */
        private void addComponent(Container container,Component c,int x,int y,int
width,int height)
```

```
        {
                c.setBounds(x,y,width,height);
                container.add(c);
        }

        //
        // TODO: Add any appropriate code in the following Event Handling Methods
        //
        private void jComboBox1_actionPerformed(ActionEvent e)
        {
        System.out.println("\njComboBox1_actionPerformed(ActionEvent e) called.");

                Object o = jComboBox1.getSelectedItem();

                selectedNode = o.toString().trim();
                //System.out.println(">>" + ((o==null)? "null" : o.toString()) + " is
selected.");
                // TODO: Add any handling code here for the particular object being
selected

        }

        private void jButton1_actionPerformed(ActionEvent e)
        {

                getAllPossiblePaths();

        }

        private void jButton2_actionPerformed(ActionEvent e)
        {

                getTheBestPath();
        }

        //
        // TODO: Add any method code to meet your needs in the following area
        //
```

```
public void getAllPossiblePaths()
{

        /*Socket clientSocketObj;

        DataInputStream dis;

        DataOutputStream dos;

        String getSelectedItem = jComboBox1.getSelectedItem().toString().trim();

//      selectedNode = getSelectedItem;

        System.out.println("The Selected Node----->"+getSelectedItem);

        try
        {


                //Address addObj = new Address();

                //String ipAdd = addObj.getAddress();

                //System.out.println("The Ip address"+ipAdd);

                if(true)
                {

                        clientSocketObj = new Socket("localhost",3333);

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

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

                        dos.writeUTF("getPath");

                        dos.writeUTF(currentNode);
```

```
                    dos.writeUTF(getSelectedItem);



                }


        }
        catch (Exception e)
        {

                e.printStackTrace();
        }*/


}


public void LabelRun()
{


        t = new Thread(this,"Mon");

        t.start();


}
public void run()
{

        while(true)
        {

        listen = listen+"."      ;

                jLabel3.setText(listen);
```

```
                count++;

                if(count == 10)
                {

                        listen = "BGP Is Listening";

                        count = 0;
                }
                else
                {
                try
                        {
                                t.sleep(500);
                        }
                        catch (Exception e)
                        {

                                e.printStackTrace();
                        }

                }


                }


    }


public void getTheBestPath()
    {


                try
                {

                        Socket sockObj;

                        DataInputStream dis;
```

```
DataOutputStream dos;

String otherNode = "";

Iterator i = allNodes.iterator();

while(i.hasNext())
{

        otherNode += i.next()+"#";

//System.out.println("inside the while loop"+otherNode);

}

if(true)
{


sockObj = new Socket("localhost",3333);

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

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

dos.writeUTF("getBestPath");

dos.writeUTF(currentNode);

dos.writeUTF(otherNode);

System.out.println("The OtherNodes are"+otherNode);

String txtPath = dis.readUTF();

System.out.println("The text Path is"+txtPath);


String[] bestPath = txtPath.split("#");
```

```
                    jTextArea1.append("The Best Paths Are\n");

                    for(int j = 0; j < bestPath.length; j++)
                    {


                    jTextArea1.append("-----------------------------------\n");



                            jTextArea1.append("\n"+bestPath[j]+"\n");
                    }
                    }

                }
                catch (Exception e)
                {

                    e.printStackTrace();
                }


}


public void getThePathDetails()
{
        try
        {

                Socket sockObj;

                DataInputStream dis;

                DataOutputStream dos;

                String otherNode = "";

                Iterator i = allNodes.iterator();
```

```
                while(i.hasNext())
                {

                  otherNode += i.next()+"#";

                        //System.out.println("inside the while loop"+otherNode);

                }

                if(true)
                {


                        sockObj = new Socket("localhost",3333);

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

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

                        dos.writeUTF("getPath");

                        dos.writeUTF(currentNode);

                        dos.writeUTF(otherNode);

                        System.out.println("The OtherNodes are"+otherNode);

                }
            }
            catch(Exception e)
            {
            e.printStackTrace();
            }
        }

//============================== Testing
==================================//
//=                                                =//
//= The following main method is just for testing this class you built.=//
```

```
//= After testing,you may simply delete it.                    =//
//=============================================================
========//
        public static void main(String[] args)
        {


                JFrame.setDefaultLookAndFeelDecorated(true);
                JDialog.setDefaultLookAndFeelDecorated(true);
                /*try
                {
                UIManager.setLookAndFeel( "com.birosoft.liquid.LiquidLookAndFeel" );
                }
                catch ( Exception ex )
                {
                        System.out.println("Failed loading L&F: ");
                        System.out.println(ex);
                }*/
                //new BGPProtocol();


        }
//= End of Testing =

}
```

REFERENCES

[1] ICANN/SSAC, .ICANN SSAC Advisory SAC008 DNS Distributed Denial of Service (DDoS) Attacks,. Mar. 2006.

[2] C. Labovitz, D. McPherson, and F. Jahanian, .Infrastructure attack detection and mitigation,. SIGCOMM 2005, August 2005, tutorial.

[3] R. Beverly and S. Bauer, .The Spoofer Project: Inferring the extent of Internet source address filtering on the internet,. in *Proceedings of Usenix SRUTI*, Cambridge, MA, Jul. 2005.

[4] S. Kandula, D. Katabi, M. Jacob, and A. Berger, .Botz-4-Sale: Surviving Organized DDoS Attacks that Mimic Flash Crowds,. in *NSDI*, 2005.

[5] D. Moore, C. Shannon, D. Brown, G. Voelker, and S. Savage, .Inferring internet Denial-of-Service activity,. *ACM Transactions on Computer Systems*, vol. 24, no. 2, May 2006.

[6] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, .Characteristics of internet background radiation,. in *Proceedings of ACM Internet Measurement Conference*, Oct. 2004.

[7] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, .Practical network support for IP trace back,. *ACM SIGCOMM Computer Communication Review*, vol. 30, no. 4, Oct. 2000.

[8] James Gosling, Bill Joy," Java Language Specification", Sun MicroSystems.Inc. pp. 456-478,200

[9] V. Paxson, .An analysis of using reflectors for distributed denial of-service attacks,. *ACM Computer Communications Review (CCR)*, vol. 31, no. 3, Jul. 2001.

[10] CERT, .Cert advisory ca-1996-21 TCP SYN flooding and IP spoofing attacks, 1996, ttp://www.cert.org/advisories/CA-1996-21.tml.

[11] K. Park and H. Lee, .On the effectiveness of route-based packet filtering for distributed DDoS attack prevention in power-law internets,. in *Proc.ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[12] Y. Rekhter and T. Li, .A border gateway protocol 4 (BGP-4),. RFC 1771, Mar. 1995.