Chapter 1

Introduction

1.1 Introduction

The Internet is the basis of number of innovative technologies like the World Wide Web, Email, P2P applications, VOIP etc. It has enabled instant access to vast and diverse resources. But, Internet is also vulnerable to number of attacks from different sources. Major categories of attacks during 2006-07 were viruses, insider abuse of access, unauthorized access to information, and denial of service (DoS) attacks [1]. It is often much easier to disrupt the operation of a network or system than to actually gain access to a network. There are number of freely available tools on Internet, from covertly exchanged exploit programs to publicly released vulnerability assessment software, to degrade performance or even disable vital network services [2].

The aim of DoS attack is to prevent legitimate users access to system resources by shutting down or seriously slowing down a service provided by a computer system. DoS first received large scale public attention in February 2000 when major Internet sites including Amazon, Yahoo, CNN and e bay were brought down by DoS attacks. CNN and other victims claimed that the attack caused damages totaling $1.7 billion [3].

In distributed DoS (DDoS) attack, the attacker uses hundreds or thousands of compromised hosts, often residing on different networks, to overload and crash target system [4]. Currently, it is not possible to prevent DoS/DDoS attacks because they are based on exploiting weaknesses in the core internet protocols which are embedded in the underlying network technology.

Denial of Service (DoS) attacks has proved to be a serious threat to users, organizations and infrastructures of the internet. There are two major reasons that make DoS attacks attractive for attackers. The first reason is that there are effective automatic tools available for attacking any victim, so expertise is not necessarily required [5]. Secondly, the attackers use fake source IP addresses to make tracing and stopping of DoS difficult. This technique is called IP Spoofing and it involves the manipulation of source IP address in the IP header of the transmitted packet. This gives the attacker a form of anonymity. It is difficult to solve the problem of IP Spoofing because of lack of security features in TCP/IP specifications.

1.1.1 DoS Incidents

DoS incidents in the Internet between the years 1989 and 1995 were investigated in [6]. The three most typical effects were the following: 51% of these incidents filled a disk, 33% of the incidents degraded network service, and 26% of the incidents deleted some critical files. A single incident was able to cause several types of damages at the same time [5].

The first reported large-scale DDoS attack occurred in August, 1999, against a university [7]. This attack shut down the victim's network for more than two days. In February 7, 2000, several high-profile Web sites were attacked, which caused them to go offline for several hours [7]. In some cases these DDoS attacks were able to produce about 1 Gbps of attack traffic against a single victim.

The backscatter analysis was used to assess the number, duration, and focus of DoS attacks in the Internet. Backscatter is called the unsolicited response traffic which the victim sends in response to attack packets with spoofed IP source addresses. The results indicate more than 12 000 attacks against more than 5000 distinct victims during the 3-week period examined in February, 2001 [5].

Packet fragmentation in real networks was studied in [8]. Bugs in the fragment handling software are exploited in many logic DoS attacks, and the results of this study still indicate the presence of these kinds of DoS attacks in the Internet. The Coordination Center of the Computer Emergency Response Team (CERT) was attacked in May, 2001. A DDoS attack caused its web site to be available only intermittently for more than two days [5].

Internet service providers (ISP) in UK have been targets for DDoS attacks during 2002. Some customers experienced a downtime of 12 hours in one of these attacks [9].

The Domain Name System (DNS) is a continuous target for DoS attacks. In October, 2002, all root name servers experienced an exceptionally intensive DoS attack. Some DNS requests were not able to reach a root name server due to congestion caused by the DoS attack. Another major DoS attack on June 15, 2004, against name servers in Akamai's Content Distribution Network (CDN) blocked nearly all access to many sites for more than two hours. The affected sites included Apple Computer, Google, Microsoft,

and Yahoo. These companies have outsourced their DNS service to Akamai to enhance service performance [5].

In UK online bookmaking, betting, and gambling sites have been extorted with DoS attacks at least during 2004 by unidentified attackers. The Internet-based business service of Al Jazeera was brought down due to a DoS attack in January, 2005 [10]. Al Jazeera provides many Arabic-language news services.

The text-to-speech translation application running in the Sun Microsystems's Grid Computing system was disabled with a DoS attack in March, 2006. This attack was carried out during the opening day of this service [5].

1.2 General Security Terminology

The subject of this thesis is related to security in computer networks, that is network security. Generally the word security can be preceded by practically any asset to be protected, such as software security and computer security.

Security

Security can be defined as "A continuous process towards reasonable resistance against known risks" [5].

Information Security

Information security can be defined as **"**A continuous process towards reasonable protection of information against unauthorized disclosure, transfer, modification, destruction or control" [11].

Computer Security

The definition of computer security is based mostly on "Computer Security" written by Dieter Gollmann. It is defined as "A continuous process towards a reasonably good prevention and detection of unauthorized actions by users of a computer system" [5].

Network Security

Since most of the existing computers are connected to networks, so network security is important. It can be defined as "A continuous process towards meeting reasonable objectives of providing confidentiality, integrity, availability and access for legitimate users of network resources" [12]. Thus network security measures are needed to protect data during their transmission.

Defending against DoS attacks is mostly network security, as these attacks are mostly carried out from a remote location through a network. Separate network security devices are typically required for defending against DoS attacks. Defending against DoS attacks can also be seen to be part of computer security, as DoS attacks ultimately exploit weaknesses in computer systems [5].

Confidentiality

The property that information is not made available to or disclosed to unauthorized individuals, entities or processes [5]. Public or Private Key Cryptography is commonly employed to ensure confidentiality of data being transmitted.

Integrity

The assurance that data received are exactly as sent by an authorized entity (i.e, contains no modification, insertion, deletion or replay) [13].

Availability

The property of a system or a system resource, being accessible and usable upon demand by an authorized entity, according to performance specifications for the system (i.e, a system is available if it provides services according to the system design whenever users request them) [5].

Access Control

In the context of network security, access control is the ability to limit and control the access to host systems and applications via communication links. To achieve this, to

achieve this each entity must first be identified or authenticated, so that access rights can be tailored to the individual [13].

Authentication

The assurance that the communicating entity is the one that it claims to be. ***Peer Entity Authentication*** is used in association with a logical connection to provide confidence in the identity of the entities connected. ***Data Origin Authentication*** is used in association with connectionless transfer and provides assurance that the source of received data is as claimed [13].

Non-Repudiation

It provides protection against the denial of transmission or reception of information by the source or destination respectively. Digital Signatures are used to ensure non-repudiation.

1.3 Denial of Service Terminology

International Organization for Standardization (ISO) has given the following definition for denial of service in the standard ISO 7498-2:1989.

Denial of Service (DoS)

The prevention of authorized access to resources or the delaying of time-critical operations [14].

Examples of resources in this definition are network bandwidth, processing capacity, disk space, memory and static memory structures [5].

An attack (which doesn't have to be successful) is defined in ANSI's Telecom Glossary 2000 [15] to be an attempt to violate security. This will be used as the basis for defining a DoS attack.

Denial of Service Attack

DoS attacks can be defined as: An intentional attempt to prevent or degrade the availability of resources [5].

It should be noted that DoS can also result from unintentional human errors, design faults, or software bugs [5].

1.4 Objectives of the Research

The primary objective of this research is to provide IP traceback support for computer networks against DoS attacks. However, this thesis concentrates on the following areas.

- To study and provide a comprehensive description of all types of DoS/DDoS attacks in computer networks.
- To study and provide a comprehensive description of all the existing IP traceback techniques and to carry out a comparison of these techniques.
- To propose a new effective IP traceback technique capable of tracing any type of DoS attack.
- Extending the proposed technique for IPv6 and Mobile IP.

1.5 Structure of the Thesis

Chapter 2, Networking Fundamentals. This chapter provides a brief description of OSI and TCP/IP reference models.

Chapter 3, IPv4 and IPv6 Basics. This chapter provides a complete description of IPv4 and IPv6 headers. It includes the functionality of different fields in the IPv4 and IPv6 headers along with the addressing methodology for both IPv4 and IPv6 networks.

Chapter 4, Attack Methods. This chapter describes the various types of DoS and DDoS attacks in IP networks.

Chapter 5, Existing IP Traceback Techniques. This chapter gives a brief description of existing IP traceback methods. A comparison of existing techniques is also carried out in this chapter.

Chapter 6, Proposed IP Traceback Technique. This chapter gives a complete description of our proposed traceback technique (for both IPv4 and IPv6 networks) supported by the simulation results.

Chapter 7, Mobile IP. This chapter provides a brief description of Mobile IPv4.

Chapter 8, IP Traceback for Mobile IP. This chapter provides a complete description of our proposed traceback technique for Mobile IPv4 networks. It also provides a brief overview of DoS attacks in case of Mobile IP networks.

Chapter 9, Conclusion and Future work.

References

[1] Lawrence A. Gordon, Martin P. Loeb, William Lucyshyn and Robert Richardson, CSI/FBI Computer Crime and Security Survey, 2006.

[2] Computer Emergency Response Team. CERT advisory CA-1999- 17 denial of service tools. http://www.cert.org/advisories/CA-1999-17.html.

[3] Computer Emergency Response Team. CERT advisory CA-2000- 01: denial of service developments. http://www.cert.org/advisories/CA-2000-01.html, 2000.

[4] Matthew Hutchinson, "*Study of Denial of Service*", MS Dissertation, Queen's University of Belfast, Aug 2003.

[5] Jarmo Molsa, "*Mitigating Denial of Service Attacks in Computer Networks*", Doctoral Dissertation, Helsinki University of Technology, 2006.

[6] J. D. Howard, "*An analysis of the security incidents on the Internet 1989-1995*", Ph.D. Dissertation, Carnegie Melon University, April 1997.

[7] P. Albitz and C. Liu, "*DNS and BIND"*, 4th ed. Sebastopol, USA: O'Reilly & Associates, Inc., Apr. 2001.

[8] C. Shannon, D. Moore, and K. C. Claffy, "*Beyond folklore: Observations on fragmented traffic*", IEEE/ACM Trans. Networking, vol. 10, no. 6, pp. 709-720, Dec. 2002.

[9] J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner, "*State of the practice of intrusion detection technologies*", Carnegie Mellon University, Software Engineering Institute, Tech. Rep. CMU/SEI-99-TR-028, Jan. 2000.

[10] Haymarket Media, "*Al-Jazeera hacked in DoS attack*," Jan. 2005. [Online]. Available: http://www.itnews.com.au/newsstory.aspx?CIaNID=17603, [accessed Mar. 10, 2007].

[11] American National Standards Institute, Inc., "*American National Standard T1.523-2001, Telecom Glossary 2000"*, Alliance for Telecommunications Industry Solutions, Feb. 2001.

[12] G. B. White, E. A. Fisch, and U. W. Pooch, "*Computer System and Network Security"*. Boca Raton, USA: CRC Press, 1996.

[13] W. Stallings, "*Cryptography and Network Security"*, 4th ed, Pearson Education, pp 17-19, 2006.

[14] International Organization for Standardization, "*ISO 7498-2:1989, Information processing systems - Open Systems Interconnection - Basic Reference Model Part 2: Security Architecture"*, Geneva, Switzerland, 1989.

[15] American National Standards Institute, Inc., "*American National Standard T1.523-2001, Telecom Glossary 2000"*, Alliance for Telecommunications Industry Solutions, Feb. 2001.

Chapter 2

Networking Fundamentals

2.1 Networking Fundamentals

A network is simply a collection of computers or other hardware devices that are connected together, either physically or logically, using special hardware and software, to allow them to exchange information and cooperate. Networks are used for an incredible variety of different purposes. The widespread networking of personal computers is a relatively new phenomenon. For the first decade or so of their existence, PCs were very much independent and were rarely connected together. In the early 1990s, PC networking began to grow in popularity as businesses realized the advantages that networking could provide. By the late 1990s, networking in homes with two or more PCs started to really take off as well.

2.1.1 The Advantages (Benefits) of Networking

Networks allow computers, and hence their users, to be connected together. They also allow for the easy sharing of information and resources, and cooperation between the devices in other ways. Some of the major benefits are [1]:

1. ***Connectivity and Communication:*** Networks connect computers and the users of those computers. Individuals within a building or work group can be connected into LAN.
2. ***Data Sharing*:** One of the most important uses of networking is to allow the sharing of data. True networking allows thousands of employees to share data much more easily and quickly.
3. ***Hardware Sharing*:** Networks facilitate the sharing of hardware devices. For example, instead of giving each of 10 employees in a department an expensive color printer, one printer can be placed on the network for everyone to share.
4. ***Internet Access:*** The Internet is itself an enormous network. The significance of the Internet on modern society is hard to exaggerate, especially for technical fields.
5. ***Internet Access Sharing*:** Small computer networks allow multiple users to share a single Internet connection. Special hardware devices allow the bandwidth of the connection to be easily allocated to various individuals as they need it.

6. ***Data Security and Management*:** In a business environment, a network allows the administrators to much better manage the company's critical data. Instead of having this data spread over dozens or even hundreds of small computers in a haphazard fashion as their users create it.

2.1.2 The Disadvantages (Costs) of Networking

There are some points against the advantages of networking as well [1]:

1. ***Network Hardware, Software and Setup Costs*:** Setting up a network requires an investment in hardware and software, as well as funds for planning, designing and implementing the network.
2. ***Hardware and Software Management and Administration Costs*:** In all, small or large implementations, ongoing maintenance and management of the network require the care and attention of an IT professional.
3. ***Undesirable Sharing*:** While networking allows the easy sharing of useful information, it also allows the sharing of undesirable data.
4. ***Illegal or Undesirable Behavior*:** Networking facilitates useful connectivity and communication, but also brings difficulties with it. Typical problems include abuse of company resources, distractions that reduce productivity, downloading of illegal or illicit materials, and even software piracy.
5. ***Data Security Concerns*:** If a network is implemented properly, it is possible to greatly improve the security of important data. In contrast, a poorly-secured network put critical data at risk, exposing it to the potential problems associated with hackers, unauthorized access and even sabotage.

2.1.3 Performance Measurements: Bandwidth, Throughput and Latency

There are a number of terms that are commonly used to refer to various aspects of network performance. Most important are [2]:

1. ***Bandwidth*:** Bandwidth is a widely-used term that usually refers to the data-carrying capacity of a network or data transmission medium. It indicates the maximum amount of data that can pass from one point to another in a unit of

time. Bandwidth is still used in these two senses: frequency band width and data capacity.

2. ***Throughput*:** Throughput is a measure of how much actual data can be sent per unit of time across a network, channel or interface. While throughput can be a theoretical term like bandwidth, it is more often used in a practical sense, for example, to measure the amount of data actually sent across a network in the real world.
3. ***Latency***: This refers to the timing of data transfer on a communications channel or network. One important aspect of latency is how long it takes from the time a request for data is made until it starts to arrive. Low latency is considered better than high latency.

2.1.4 International Networking Standards Organizations

In order to facilitate the development of open standards, organizations are needed that will coordinate the creation and publishing of documents. The most important are [2]:

1. ***International Organization for Standardization (ISO):*** Probably the biggest standards organization in the world, the ISO is really a federation of standards organizations from dozens of nations. In the networking world, the ISO is best known for its OSI Model.
2. ***American National Standards Institute (ANSI*):** ANSI is the main organization responsible for coordinating and publishing computer and information technology standards in the United States.
3. ***Institute of Electrical and Electronics Engineers (IEEE):*** The IEEE is a well-known professional organization for those in the electrical or electronics fields, including computers and networking.
4. ***Electronic Industries Alliance (EIA):*** The EIA is an international industry association that is best known for publishing electrical wiring and transmission standards.

5. ***Telecommunications Industry Association (TIA*):** The TIA is the communications sector of the EIA, and is responsible for developing communications standards.
6. ***International Telecommunication Union - Telecommunication Standardization Sector (ITU-T):*** ITU-T is another large international body that develops standards for the telecommunications industry. The ITU-T was formerly named the International Telephone and Telegraph Consultative Committee or CCITT.

2.2 The Open System Interconnection (OSI) Reference Model

Models are useful because they help us understand difficult concepts and complicated systems. When it comes to networking, there are several models that are used to explain the roles played by various technologies, and how they interact. The most popular and commonly used is the Open System Interconnect (OSI) Reference Model presented by ISO. The idea behind the OSI Reference Model is to provide a framework for both designing networking systems and for explaining how they work.

The Open System Interconnection (OSI) reference model describes how information from software application of one computer moves through the network medium to a software application in another computer. The OSI reference model is a conceptual model composed of seven layers, each specifying particular network functions. The model was developed by the International Organization for Standardization (ISO) in 1984, and it's now considered as primary architectural model for inter computer communication. The OSI model divides the tasks involved with moving information between networked computers into seven smaller, more manageable task groups. A task or group of tasks is then assigned to each of the seven OSI layers. Each layer is reasonably self-contained so that the tasks assigned to each layer can be implemented independently. This enables the solutions offered by one layer to be updated without adversely affecting the other layers [3].

Table 2.1: OSI Reference Model

Layer – 1	***Application***
Layer – 2	***Presentation***
Layer – 3	***Session***
Layer – 4	***Transport***
Layer - 5	***Network***
Layer - 6	***Data link***
Layer - 7	***Physical***

The seven layers of the OSI Model can be divided into two categories

***a) Upper Layers*:** The upper layers of the OSI model deal with application issues and generally are implemented only in software. The highest layer, the application layer, is closest to the user end. Both users and application layer possesses interaction with software application that contains a communication component. The term upper layer is sometimes used to refer to any layer above another layer in the OSI model.

***b) Lower Layers*:** The lower layers of the OSI model handle the data transport issues. The physical and the data link layer are implemented in hardware and software. The lowest layer, physical layer, is closest to physical network medium and is responsible for actually placing information on the medium.

Table 2.2: Two set of layers making OSI Model

Layer – 1	***Application***	***Upper layers - Application***
Layer – 2	***Presentation***	
Layer – 3	***Session***	
Layer – 4	***Transport***	***Lower layers - Data Transport***
Layer – 5	***Network***	
Layer – 6	***Data link***	
Layer – 7	***Physical***	

2.2.1 OSI Model Physical Layer

The physical layer defines the electrical, mechanical, procedural and functional specifications for activating, maintaining and deactivating the physical link between communicating network systems. Physical layer specifications define characteristics such as voltage levels, timing of voltage changes, physical data rates, maximum transmission distances and physical connectors. In general, then, physical layer technologies are ones that are at the very lowest level and deal with the actual ones and zeroes that are sent over the network [3].

The following are the main responsibilities of the physical layer in the OSI Reference Model [1]:

1. ***Definition of Hardware Specifications*:** The details of operation of cables, connectors, wireless radio transceivers, network interface cards and other hardware devices are generally a function of the physical layer.
2. ***Encoding and Signaling*:** The physical layer is responsible for various encoding and signaling functions that transform the data from bits that reside within a computer or other device into signals that can be sent over the network.
3. ***Data Transmission and Reception:*** After encoding the data appropriately, the physical layer actually transmits the data and receives it.
4. ***Topology and Physical Network Design:*** The physical layer is also considered the domain of many hardware-related network design issues, such as LAN and WAN topology.

2.2.2 OSI Model Data Link Layer

The second-lowest layer in the OSI Reference Model stack is the data link layer, often abbreviated DLL. The data link layer provides reliable transmit of data across physical network link. The different data link layer specifications define different network and protocol characteristics, including physical addressing, network topology, error notification, sequencing of frames, flow control. Network topology consists of data link layer specifications that often define how devices are to be physically connected, such as in a bus or ring topology. Error notification alerts the upper layer protocols that a

transmission error has occurred. Finally, flow control moderates the transmission of the data so that the receiving device is not crowded with more traffic than it can handle in one time [3].

The Institute of Electrical and Electronics Engineers (IEEE) has sub divided the data link layers into two layers [1]:

Table 2.3: Layer 2 sub-division

Data Link Layer	***LLC Sub layer***
	MAC Sub layer

Logical Link Control (LLC)

The Logical Link Control (LLC) sub layer of the data link layer manages communications between devices over a single link of network. LLC is defined in IEEE 802.2 specification and supports both connectionless and connection oriented services used by higher layer protocols to share a single physical data link.

Media Access control (MAC)

The Media Access Control (MAC) sub layer of data link layer manages protocol access to physical network medium. The IEEE MAC specifications define MAC addresses, which enable multiple devices to uniquely identify one another at the data link layer.

The following are the key tasks performed at the data link layer [1]:

1. ***Data Framing*:** The data link layer is responsible for the final encapsulation of higher-level messages into frames that are sent over the network at the physical layer.
2. ***Addressing*:** The data link layer is the lowest layer in the OSI model that is concerned with addressing: labeling information with a particular destination location. Each device on a network has a unique number, usually called a hardware address or MAC address that is used by the data link layer protocol to ensure that data intended for a specific machine gets to it properly.
3. ***Error Detection and Handling*:** The data link layer handles errors that occur at the lower levels of the network stack. For example, a cyclic redundancy

check (CRC) field is often employed to allow the station receiving data to detect if it was received correctly

2.2.3 OSI Model Network Layer

The third-lowest layer of the OSI Reference Model is the network layer. The network layer defines the network addresses, which differ from MAC addresses. Some network layer implementations, such as Internet Protocol (IP), define network addresses in a way that route selection can be determined systematically by comparing the source network address with the destination network address and applying the subnet mask. Because this layer defines the logical network layout, routers can use this layer to determine how to forward the packets [3].

Some of the specific jobs normally performed by the network layer include [1]:

1. ***Logical Addressing:*** Every device that communicates over a network has associated with it a logical address, sometimes called a layer three address. For example, on the Internet, the internet protocol (IP) is the network layer protocol and every machine has an IP address.
2. ***Routing:*** Moving data across a series of interconnected networks is probably the defining function of the network layer. It is the job of the devices and software routines that function at the network layer to handle incoming packets from various sources, determine their final destination, and then figure out where they need to be sent to get them where they are supposed to go.
3. ***Datagram Encapsulation*:** The network layer normally encapsulates messages received from higher layers by placing them into datagrams (also called packets) with a network layer header.
4. ***Fragmentation and Reassembly*:** The network layer must send messages down to the data link layer for transmission. Some data link layer technologies have limits on the length of any message that can be sent. If the packet that the network layer wants to send is too large, the network layer must split the packet up, send each piece to the data link layer, and then have pieces reassembled once they arrive at the network layer on the destination machine.

5. ***Error Handling and Diagnostics*:** Special protocols are used at the network layer to allow devices that are logically connected, or that are trying to route traffic, to exchange information about the status of hosts on the network or the devices themselves

2.2.4 OSI Model Transport Layer

The fourth and the middle layer of the OSI Reference Model protocol stack is the transport layer. It is more often associated with the lower layers, because it concerns itself with the transport of data, but its functions are also somewhat high-level. The transport layer really acts as a link between the abstract world of applications at the higher layers, and the concrete functions of layers one to three. Due to this role, the transport layer's overall job is to provide the necessary functions to enable communication between software application processes on different computers. The transport layer is charged with providing a means by which these applications can all send and receive data using the same lower-layer protocol implementation. Thus, the transport layer is sometimes said to be responsible for end-to-end or host-to-host transport [3].

The specific functions often performed at the transport layer are [1]:

1. ***Port Addressing*:** This enables many different software programs to use a network layer protocol simultaneously. The best example of transport-layer process-level addressing is the TCP/UDP used in TCP/IP, which allows applications to be individually referenced on any TCP/IP device.
2. ***Multiplexing and Demultiplexing:*** Transport layer protocols on a sending device multiplex the data received from many application programs for transport, combining them into a single stream of data to be sent. The same protocols receive data and then demultiplex it from the incoming stream of datagrams, and direct each package of data to the appropriate recipient application processes.
3. ***Segmentation, Packaging and Reassembly*:** The transport layer segments the large amounts of data it sends over the network into smaller pieces on the source machine, and then reassembles them on the destination machine.

4. ***Connection Establishment, Management and Termination:*** Transport layer connection-oriented protocols are responsible for the series of communications required to establish a connection, maintain it as data is sent over it, and then terminate the connection when it is no longer required.
5. ***Acknowledgments and Retransmissions:*** The transport layer is where many protocols are implemented that guarantee reliable delivery of data. This is done using a variety of techniques, most commonly the combination of acknowledgments and retransmission timers.
6. ***Flow Control*:** Transport layer protocols that offer reliable delivery also often implement flow control features.

2.2.5 OSI Model Session Layer

The fifth layer in the OSI Reference Model is the session layer. The session layer is the first one where much practical matters related to the addressing, packaging and delivery of data are left behind; they are functions of layers four and below. It is the lowest of the three upper layers, which collectively are concerned mainly with software application issues and not with the details of network and internet implementation [3].

2.2.6 OSI Model Presentation Layer

The presentation layer is the sixth layer of the OSI Reference Model protocol stack. It is different from the other layers in two key respects. First, it has a much more limited and specific function than the other layers. Second, it is used much less often than the other layers; in many types of connections it is not required. The name of this layer suggests its main function that it deals with the presentation of data. More specifically, the presentation layer is charged with taking care of any issues that might arise where data sent from one system needs to be viewed in a different way by the other system. It also takes care of any special processing that must be done to data from the time an application tries to send it until the time it is sent over the network [3]. Its main functions are [1]:

1. ***Translation*:** Networks can connect very different types of computers together: PCs, Macintoshes, UNIX systems, servers and mainframes can all

exist on the same network. These systems have many distinct characteristics and represent data in different ways; they may use different character sets for example. The presentation layer handles the job of hiding these differences between machines.

2. ***Compression:*** Compression (and decompression) may be done at the presentation layer to improve the throughput of data.
3. ***Encryption*:** Some types of encryption (and decryption) are performed at the presentation layer. This ensures the security of the data as it travels down the protocol stack. For example, one of the most popular encryption schemes that are usually associated with the presentation layer is the secure socket layer (SSL) protocol. Not all encryption is done at layer 6, however; some encryption is often done at lower layers in the protocol stack.

2.2.7 OSI Model Application Layer

At the top of the OSI Reference Model stack of layers is the application layer. This too is named very appropriately: the application layer is the one that is used by network applications. These programs are what actually implement the functions performed by users to accomplish various tasks over the network. In the OSI model, the application layer provides services for user applications to employ. For example, web browser; it makes use of the services offered by a protocol that operates at the application layer, which is called the Hypertext Transfer Protocol (HTTP). As the top of the stack layer, the application layer is the only one that does not provide any services to the layer above it in the stack. Instead, it provides services to programs that want to use the network. So the responsibilities at this layer are simply to implement the functions that are needed by users of the network. And to issue the appropriate commands to make use of the services provided by the lower layers [1].

2.3 TCP/IP Protocol Suite and Architecture

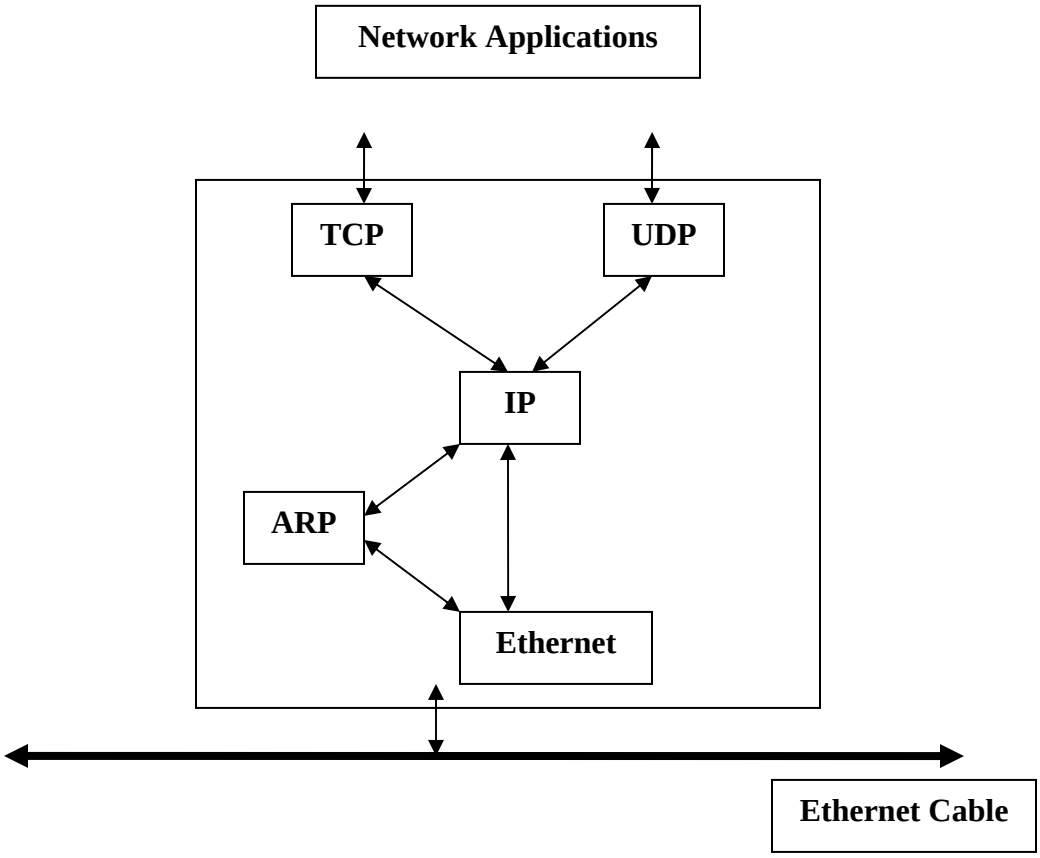

Figure 2.1: TCP/IP Reference Model [4]

Modern internetworking is dominated by the suite known as TCP/IP. Named for two key protocols of the many that comprise it, TCP/IP has been in continual development and use for about three decades. In that time, it has evolved from an experimental technology used to hook together a handful of research computers, to the powerhouse of the largest and most complex computer network in history: the global Internet, connecting together millions of networks and end devices.

The OSI Model consists of seven layers that represent a functional division of the tasks required to implement a network. The TCP/IP protocol suite was in fact created before the OSI Reference Model; as such, its inventors didn't use the OSI model to explain TCP/IP architecture. The TCP/IP model uses four layers that logically span the

equivalent of the top six layers of the OSI reference model. The following are the TCP/IP model layers, starting from the bottom [4].

Table 2.4: OSI Reference Model and TCP/IP Model Layers [4]

Layer - 1	**Application**	***APPLICATION***
Layer - 2	**Presentation**	
Layer - 3	**Session**	
Layer - 4	**Transport**	***(Host-to-Host) TRANSPORT***
Layer - 5	**Network**	***INTERNET***
Layer - 6	**Datalink**	***NETWORK INTERFACE***
Layer - 7	**Physical**	***(Hardware)***
	OSI Model	***TCP/IP Model***

2.3.1 Network Interface Layer

This layer represents the place where the actual TCP/IP protocols running at higher layers interface to the LAN. None of the core IP protocols run at this layer. Despite this, the network interface layer is part of the architecture. It is equivalent to the data link layer of OSI Model and is also sometimes called the link layer [4].

On many TCP/IP networks, there is no TCP/IP protocol running at all on this layer, because it is simply not needed. For example running TCP/IP over an Ethernet, then Ethernet handles layer two (and layer one) functions. However, the TCP/IP standards do define protocols for TCP/IP networks that do not have their own layer two implementation. These protocols, the Serial Line Internet Protocol (SLIP) and the Point-to-Point Protocol (PPP), serve to fill the gap between the network layer and the physical layer. They are commonly used to facilitate TCP/IP over direct serial line connections (such as dial-up telephone networking) and other technologies that operate directly at the physical layer [1].

2.3.2 Internet Layer

This layer corresponds to the network layer in the OSI Reference Model and for that reason is sometimes called the network layer even in TCP/IP model. It is responsible for typical layer three jobs, such as logical device addressing, data packaging, manipulation and delivery, and last but not least, routing. At this is the Internet Protocol (IP), arguably the heart of TCP/IP, as well as support protocols such as ICMP and the routing protocols (RIP, OSFP, BGP, etc.) The new version of IP, called IP version 6, will be used for the Internet of the future and is also at this layer [4].

2.3.3 (Host-to-Host) Transport Layer

The primary job of this layer is to facilitate end-to-end communication over an internetwork. It is in charge of allowing logical connections to be made between devices to allow data to be sent either unreliably or reliably. It is also here that identification of the specific source and destination application process is accomplished

The formal name of this layer is often shortened to just the transport layer; the key TCP/IP protocols at this layer are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The TCP/IP transport layer corresponds to the layer of the same name in the OSI model (layer four) but includes certain elements that are arguably part of the OSI session layer. For example, TCP establishes a connection that can persist for a long period of time, which some people say makes a TCP Connection more like a session [4].

2.3.4 Application Layer

This is the highest layer in the TCP/IP model. It is a rather broad layer, encompassing layers five through seven in the OSI model. The TCP/IP model better reflects the blended nature of the divisions between the functions of the higher layers in the OSI model, which in practical terms often seem rather arbitrary. It really is hard to separate some protocols in terms of which of layers five, six or seven they encompass. Numerous protocols reside at the application layer. These include application protocols such as HTTP, FTP and SMTP for providing end users services, as well as administrative protocols like SNMP, DHCP and DNS [4].

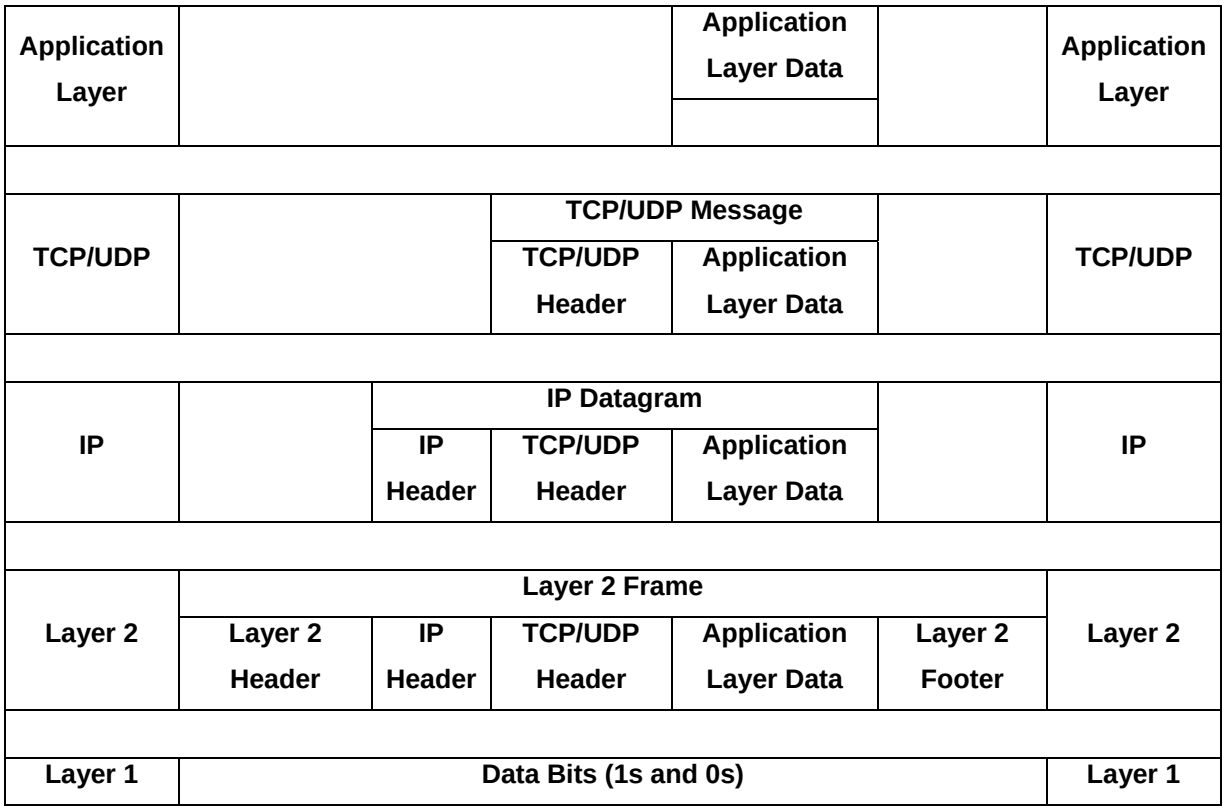

Figure 2.2: Data Movement in TCP/IP Model [1].

References

[1] The TCP/IP Guide: Version 3.0 by Charles M. Kozierok, 2005 Publicaton available at http://www.tcpipguide.com/index.htm

[2] Behrouz A. Forouzan, "*Data Communications and Networking*", 3rd ed, Pearson Education.

[3] Cisco Systems, "*Internetworking Technologies Handbook*", 4th Edition, Pearson Education.

[4] T. Scolofsky and Kale, "*A TCP/IP Tutorial*" Request for Comments 1180, Network Working Group, Jan 1991.

Chapter 3

IPv4 and IPv6 Basics

3.1 Internet Protocol (IP)

The Internet Protocol (IP) is the core of the TCP/IP protocol suite and its main protocol at the network layer. The network layer is primarily concerned with the delivery of data between devices that may be on different networks that are interconnected in an arbitrary manner. IP is the mechanism by which this data is sent on TCP/IP networks. As the layer three protocol, it provides a service to layer four in the TCP/IP stack, represented mainly by the TCP and UDP protocols. This service is to take data that has been packaged by either TCP or UDP, manipulate it as necessary, and send it out. This service is sometimes called network datagram delivery. In a nutshell IP sends data from point A to point B over a network of connected networks. The key characteristics used to describe IP and the general manner in which it operates are [1]:

- ***Universally-Addressed*:** In order to send data from point A to point B, it is necessary to ensure that devices know how to identify which device is point B. IP defines the addressing mechanism for the network and uses these addresses for delivery purposes.
- ***Underlying-Protocol Independent*:** IP is designed to allow the transmission of data across any type of underlying network that is designed to work with a TCP/IP stack. It includes provisions to allow it to adapt to the requirements of various lower-level protocols.
- ***Connectionless*:** IP is a connectionless protocol. This means that when A wants to send data to B, it doesn't first set up a connection to B and then send the data, it just makes the datagram and sends it.
- ***Delivered Without Acknowledgments*:** IP unreliable in nature. It doesn't use acknowledgements. When device B gets a datagram from device A, it doesn't send back an acknowledgement to tell A that the datagram was received.

3.1.1 IP Functions

IP performs four basic functions at the network layer, namely [2]:

1. ***Addressing:*** In order to perform the job of delivering datagrams, IP must know where to deliver them. For this reason, IP includes a mechanism for host

addressing. Furthermore, since IP operates over networks, it is designed to allow unique addressing of devices across arbitrarily large networks. It also contains a structure to facilitate the routing of datagrams to distant networks if required.

2. ***Data Encapsulation and Formatting/Packaging*:** As the TCP/IP network layer protocol, IP accepts data from the transport layer protocols UDP and TCP. It then encapsulates this data into an IP datagram using a special format prior to transmission.
3. ***Fragmentation and Reassembly:*** IP datagrams are passed down to the data link layer for transmission on the network. However, the maximum frame size of each physical/data-link network using IP may be different. For this reason, IP includes the ability to fragment IP datagrams into pieces so they can each be carried on the local network. The receiving device uses the reassembly function to recreate the whole IP datagram again.
4. ***Routing / Indirect Delivery:*** When an IP datagram must be sent to a destination on the same local network, this can be done easily using the network's underlying protocol using direct delivery. However, in most cases the final destination is on a distant network not directly attached to the source. In this situation the datagram must be delivered indirectly. This is accomplished by routing the datagram through intermediate devices called routers. IP accomplishes this with support from the other protocols.

3.2 Internet Protocol Version 4 (IP or IPv4)

Internet Protocol, version 4 of IP was the first that was widely used in modern TCP/IP. IPv4, as it is sometimes called to differentiate it from the newer IPv6, is the Internet Protocol version in use on the Internet today, and an implementation of the protocol is running on hundreds of millions of computers. It provides the basic datagram delivery capabilities upon which all of TCP/IP works and it has proven its quality in use over a period of more than two decades [3].

3.2.1 IPv4 Datagram Format

Data transmitted over an internet using IP is carried in messages called IP datagrams. Like all network protocol messages, IP uses a specific format for its datagrams. The IPv4 datagram is conceptually divided into two pieces [4]:

- The Header.
- The Payload.

The header contains addressing and control fields, while the payload carries the actual data to be sent over the network. Unlike some message formats, IP datagrams do not have a footer following the payload.

Even though IP is a relatively simple, connectionless, unreliable protocol, the IPv4 header carries a fair bit of information, which makes it rather large. At a minimum, it is 20 bytes long, and with options can be significantly longer.

<table>
<tr><td>Version</td><td>Header Length</td><td colspan="2">Type of Service (TOS)</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Flags</td><td>Fragment Offset</td></tr>
<tr><td colspan="2">TTL</td><td colspan="2">Protocol</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="6">Source Address</td></tr>
<tr><td colspan="6">Destination Address</td></tr>
<tr><td colspan="6">Options & Padding</td></tr>
<tr><td colspan="6">Data</td></tr>
</table>

Figure 3.1: IPv4 datagram Format [4]

Version (4 bits): Identifies the version of IP used to generate the datagram. For IPv4, this is of course the number 4. The purpose of this field is to ensure compatibility between devices that may be running different versions of IP.

***Internet Header Length (4 bits*):** Specifies the length of the IP header, in 32-bit words. This includes the length of any options fields and padding. The normal value of this field when no options are used is 5.

***Type of Service –ToS (1 byte*):** A field designed to carry information to provide quality of service features, such as prioritized delivery, for IP datagrams.

Total Length – TL (2 bytes): Specifies the total length of the IP datagram, in bytes. Since this field is 16 bits wide, the maximum length of an IP datagram is 65,535 bytes, though most are much smaller.

Identification (2 bytes): This field contains a 16-bit value that is common to each of the fragments belonging to a particular message, for datagrams originally sent unfragmented it is still filled in, so it can be used if the datagram must be fragmented by a router during delivery. This field is used by the recipient to reassemble messages without accidentally mixing fragments from different messages. This is needed because fragments may arrive from multiple messages mixed together, since IP datagrams can be received out of order from any device.

Flags (3 bits): Bits reserved for flag indications.

Fragment Offset (13 bits): When fragmentation of a message occurs, this field specifies the offset, or position, in the overall message where the data in this fragment goes. It is specified in units of 8 bytes (64 bits). The first fragment has an offset of 0.

Time to live - TTL (1 byte): Specifies how long the datagram is allowed to live on the network, in terms of router hops. Each router decrements the value of the TTL field (reduces it by one) prior to transmitting it. If the TTL field drops to zero, the datagram is assumed to have taken too long a route and is discarded.

Protocol (1 byte): Type of upper layer protocol.

Header Checksum (2 bytes): A checksum computed over the header to provide basic protection against corruption in transmission. This is not the complex CRC code typically used by data link layer technologies but it's just a 16-bit checksum. It is calculated by dividing the header bytes into words (a word is two bytes) and then adding them together. The data is not check summed, only the header. At each hop the device receiving the datagram does the same checksum calculation and on a mismatch, discards the datagram as damaged.

Source Address (4 bytes): The 32-bit IP address of the originator of the datagram. Note that even though intermediate devices such as routers may handle the datagram, they do not normally put their address into this field—it is always the device that originally sent the datagram.

***Destination Address (4 bytes*)**: The 32-bit IP address of the intended recipient of the datagram. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.

***Options (variable*)**: One or more of several types of options may be included after the standard headers in certain IP datagrams.

Padding (variable): If one or more options are included, and the number of bits used for them is not a multiple of 32, enough zero bits are added to pad out the header to a multiple of 32 bits (4 bytes).

Data (variable): The data to be transmitted in the datagram, either an entire higher-layer message or a fragment of one.

3.2.2 IP Datagram Options and Option Format

All IP datagrams must include the standard 20-byte header, which contains key information such as the source and destination address of the datagram, fragmentation control parameters, length information and more. In addition to these invariable fields, the creators of IPv4 included the ability to add options that provide additional flexibility in how IP handles datagrams. Use of these options is optional. However, all devices that handle IP datagrams must be capable of properly reading and handling them. The IP datagram may contain zero, one or more options, which makes the total length of the Options field in the IP header variable. Each of the options can be either a single byte long, or multiple bytes in length, depending on how much information the option needs to convey. When more than one option is included they are just concatenated together and put into the Options field as a whole. Since the IP header must be a multiple of 32 bits, a Padding field is included if the number of bits in all options together is not a multiple of 32 bits [4].

The description below lists most common IPv4 options, showing the option class, option number and length for each [1].

Security (option 2 – 11 bytes): An option provided for the military to indicate the security classification of IP datagrams.

Time Stamp (option 4 – variable): This option is similar to the Record Route option. However, instead of each device that handles the datagram inserting its IP address into the option, it puts in a timestamp, so the recipient can see how long it took for the datagram to travel between routers. As with the Record Route option, the length of this option is set by the originating device and cannot be enlarged by intermediate devices.

Record Route (option 7 – variable): This option allows the route used by a datagram to be recorded within the header for the datagram itself. If a source device sends a datagram with this option in it, each router that handles the datagram adds its IP Address to this option. The recipient can then extract the list of IP Addresses to see the route taken by the datagram.

Traceroute (option 18 – variable): Used in the enhanced implementation of the traceroute utility.

3.2.3 IP Routing and the Process of Next-Hop Routing

When a datagram is sent between source and destination devices that are not on the same physical network, the datagram must be delivered indirectly between the devices, a process called routing. It is this ability to route information between devices that may be far away that allows IP to create the equivalent of a virtual network that spans potentially thousands of physical networks.

Even though IP lets devices connect over the network using indirect delivery, all of the actual communication of datagrams occurs over physical networks using routers. We rely on intermediate devices that are each physically connected to each other in a variety of ways to form a mesh containing millions of paths between networks. To get the datagram where it needs to go, it needs to be handed off from one router to the next, until it gets to the physical network of the destination device. The general term for this is next hop routing [2].

In IP networks routing is done on a step-by-step basis, one hop at a time. When we decide to send a datagram to a device on a distant network, we don't know the exact

path that the datagram will take; we only have enough information to send it to the correct router to which we are attached. That router, in turn, looks at the IP Address of the destination and decides where the datagram should next hop to. This process continues until the datagram reaches the destination host's network, when it is delivered. Next hop routing makes IP powerful. On each step of the journey to any other host, a router only needs to know where the next step for the datagram is. Without this concept, each device and router would need to know what path to take to every other host on the internet, which would be impractical [4].

3.2.4 IP Routes and Routing Tables

Routers are responsible for forwarding traffic on an IP network. Each router accepts datagrams from a variety of sources, examines the IP of the destination and decides what the next hop is that the datagram needs to take to get it that much closer to its final destination. This is done by the fact that each router maintains a set of information that provides a mapping between different network IDs and the other routers to which it is connected. This information is contained in a data structure normally called a routing table. Each entry in the table, unsurprisingly called a routing entry, provides information about one network. Each time a datagram is received the router checks its destination IP address against the routing entries in its table to decide where to send the datagram, and then sends it on its next hop [1].

3.3 Internet Protocol Version 6 (IPv6) / IP Next Generation (IPng)

Since 1981, TCP/IP has been built on version 4 of the Internet Protocol. IPv4 has done its job admirably. At the same time, it has been apparent for many years that certain limitations in this vulnerable protocol would hold back the future growth of both Internet size and services if not addressed. Due to the key role that IP plays, changing it is no simple feat. It means a substantial modification to the way that nearly everything in TCP/IP operates. For the last several years, development of a new version of IP has been underway, officially called Internet Protocol version 6 (IPv6) and also sometimes referred

to as IP Next Generation or IPng. IPv6 is poised to take over from IPv4, and will be the basis for the Internet of the future [6].

3.3.1 Design Goals of IPv6

The problem of addressing was the main motivation for creating IPv6. Unfortunately, this has caused many people to think that the address space expansion is the only change made in IP, which is not true. Since making a change to IP is such a big deal, it's something done rarely. It made sense to correct not just the addressing issue but to update the protocol in a number of other respects as well, to ensure its viability. In fact, even the addressing changes in IPv6 go far beyond just adding more bits to IP address fields. Some of the most important goals in designing IPv6 include [1]:

- ***Larger Address Space*:** IPv6 had to provide more addresses for the growing Internet. This is done by 128-bits address space.
- ***Better Management of Address Space:*** It was desired that IPv6 not only include more addresses, but a more capable way of dividing the address space and using the bits in each address.
- ***Easier TCP/IP Administration:*** The designers of IPv6 hoped to resolve some of the current labor intensive requirements of IPv4, such as the need to configure IP addresses.
- ***Modern Design for Routing:*** In contrast to IPv4, which was designed before we all had any idea what the modern Internet would be like, IPv6 was created specifically for efficient routing in our current Internet, and with the flexibility for the future.
- ***Better Support for Multicasting:*** Multicasting was an option under IPv4 from the start, but support for it has been slow in coming.
- ***Better Support for Security:*** IPv4 was designed at a time when security wasn't much of an issue, because there were a relatively small number of networks on the internet, and their administrators often knew each other. Today, security on the public Internet is a big issue, and the future success of the Internet requires that security concerns be resolved.

- ***Better Support for Mobility:*** When IPv4 was created, there really was no concept of mobile IP devices. The problems associated with computers that move between networks led to the need for Mobile IP. IPv6 builds on Mobile IP and provides mobility support within IP itself.

3.3.2 Major Changes and Additions in IPv6

Primary motivator for creating a new version of IP was to fix the problems with addressing under IPv4. But numerous other design goals existed for the new protocol as well. Once the decision was made to take the significant step of creating a new version of a protocol as important as IP, it made sense to use the opportunity to make as many improvements as possible. The following list provides a summary of the most important changes between IPv4 and IPv6, showing some of the ways that the IPv6 met the design goals for the new protocol [6]:

1. ***Larger Address Space:*** IPv6 addresses are 128 bits long instead of 32 bits. This expands the address space.
2. ***Hierarchical Address Space*:** One reason why the IPv6 address size was expanded so much was to allow it to be hierarchically divided to provide a large number of each of many classes of addresses.
3. ***Hierarchical Assignment of Unicast Addresses:*** A special global unicast address format was created to allow addresses to be easily allocated across the entire Internet.
4. ***Better Support for Non-Unicast Addressing*:** Support for multicasting is improved, and support is added for a new type of addressing: anycast addressing. This new kind of addressing basically is to deliver this message to the easiest reach member of this group, and potentially enables new types of messaging functionality.
5. ***Auto Configuration and Renumbering:*** A provision is included to allow easier auto configuration of hosts and renumbering of the IP addresses in networks and subnetworks as needed. A technique also exists for renumbering router addresses.
6. ***New Datagram Format*:** The IP datagram format has been redefined and given new capabilities. The main header of each IP datagram has been streamlined, and

support added for easily extending the header for datagrams requiring more control information.

7. ***Support for Quality of Service:*** IPv6 datagrams include QoS features, allowing better support for multimedia and other applications requiring quality of service.
8. **Security Support:** Security support is designed into IPv6 using the authentication and encryption extension headers and other features.
9. ***Updated Fragmentation and Reassembly Procedures*:** The way that fragmentation and reassembly of datagrams works has been changed in IPv6, to improve efficiency of routing and better reflect the realities of today's networks.
10. ***Modernized Routing Support*:** The IPv6 protocol is designed to support modern routing systems, and to allow expansion as the Internet grows.
11. ***Transition Capabilities:*** Since it was recognized from the start that going from IPv4 to IPv6 is a big move, support for the IPv4/IPv6 transition has been provided in numerous areas. This includes a plan for interoperating IPv4 and IPv6 networks, mapping between IPv4 and IPv6 addresses etc.

3.3.3 IPv4-IPv6 Transition Methods

Due to the time that change takes, IETF has been working on specific provisions to allow a smooth transition from version 4 to version 6, and hardware and software interoperability solutions to let newer IPv6 devices access IPv4 hosts. A technique was included in IPv6 to allow administrators to embed IPv4 addresses within IPv6 addresses. Special methods are defined to handle interoperability, including [1]:

***Dual Stack Devices*:** Routers and some other devices may be programmed with both IPv4 and IPv6 implementations to allow them to communicate with both types of hosts.

***IPv4/IPv6 Translation*:** Dual stack devices may be designed to accept requests from IPv6 hosts, convert them to IPv4 datagrams, and send the datagrams to the IPv4 destination and then process the return datagrams similarly.

IPv4 Tunneling of IPv6: IPv6 devices that don't have a path between them consisting entirely of IPv6-capable routers may be able to communicate by encapsulating IPv6 datagrams within IPv4. In essence, they would be using IPv6 on top of IPv4; two

network layers. The encapsulated IPv4 datagrams would travel across conventional IPv4 routers.

3.3.4 IPv6 Address Types

One important change in the addressing model of IPv6 is the address types supported. IPv6 also supports three address types, but with some changes [6]:

***Unicast Addresses*:** These are standard unicast addresses as in IPv4, one per host interface.

***Multicast Addresses*:** These are addresses that represent various groups of IP devices: a message sent to a multicast address goes to all devices in the group. IPv6 includes much better multicast features and many more multicast addresses than IPv4. Since multicast under IPv4 was hampered in large part due to lack of support of the feature by many hardware devices, support for multicasting is a required, not optional, part of IPv6.

***Anycast Addresses*:** Anycast addressing is used when a message must be sent to any member of a group, but does not need to be sent to them all. Usually the member of the group that is easiest to reach will be sent the message. One common example of how anycast addressing could be used is in load sharing amongst a group of routers in an organization.

3.3.5 IPv6 Address Size

In IPv4, IP addresses are 32 bits long; these are usually grouped into four octets of eight bits each. The theoretical IPv4 address space is 2^{32}, or 4,294,967,296 addresses. To increase this address space we simply increase the size of addresses; each extra bit we give to the address size doubles the address space. Based on this, some folks expected the IPv6 address size to increase from 32 to 48 bits, or perhaps 64 bits. Either of these numbers would have given a rather large number of addresses. However, IPv6 addressing doesn't use either of these figures; instead, the IP address size jumps all the way to 128 bits, or sixteen 8-bit octets/bytes. This represents a truly remarkable increase in the address size, which surprised a lot of people [5].

3.3.6 IPv6 General Datagram Structure

IPv6 datagrams now includes a main header format and zero or more extension headers. The overall structure therefore is like [6].

Main Header (40 Bytes)	Extension Header (Variable)	Data (Variable)

Figure 3.2: IPv6 General Datagram Structure

3.3.7 Main Header Format

The IPv6 main header is required for every datagram. It contains addressing and control information that are used to manage the processing and routing of the datagram. The main header format of IPv6 datagrams is described below [1].

Version	Traffic Class	Flow Label	
Pay Load Length		Next Header	Hop Limit
Source Address (128 Bits)			
Destination Address (128 Bits)			

Figure 3.3: IPv6 Main Header Format

Version (4 bits): Identifies the version of IP used to generate the datagram. This field is used the same way as in IPv4, except that it carries the value 6 (0110 binary).

Traffic Class (1 byte): This field replaces the Type of Service (TOS) field in the IPv4 header

Flow Label (20 bits): This large field was created to provide additional support for real-time datagram delivery and quality of service features. The concept of a flow is defined as a sequence of datagrams sent from a source device to one or more destination devices. A unique flow label is used to identify all the datagrams in a particular flow, so that routers between the source and destination all handle them the same way, to help ensure uniformity in how the datagrams in the flow are delivered. For example, if a video stream is being sent across an IP

internetwork, the datagrams containing the stream could be identified with a flow label to ensure that they are delivered with minimal latency. Not all devices and routers may support flow label handling, and use of the field by a source device is entirely optional.

Payload Length (2 bytes): This field replaces the Total Length field from the IPv4 header, but it is used differently. Rather than measuring the length of the whole datagram, it only contains the number of bytes of the payload. However, if extension headers are included, their length is counted here as well. In simpler terms, this field measures the length of the datagram less the 40 bytes of the main header itself.

Next Header (1 byte): This field replaces the Protocol field and has two uses. When a datagram has extension headers, this field specifies the identity of the first extension header, which is the next header in the datagram. When a datagram has just this main header and no extension headers, it serves the same purpose as the old IPv4 Protocol field and has the same values, though new numbers are used for IPv6 versions of common protocols. In this case the next header is the header of the upper layer message the IPv6 datagram is carrying.

Hop Limit (1 byte): This replaces the Time to Live (TTL) field in the IPv4 header; its name better reflects the way that TTL is used in modern networks, since TTL is really used to count hops, not time.

Source Address (16 bytes): The 128-bit IP address of the originator of the datagram. As with IPv4, this is always the device that originally sent the datagram.

Destination Address (16 bytes): The 128-bit IP address of the intended recipient of the datagram; unicast, anycast or multicast. Again, even though devices such as routers may be the intermediate targets of the datagram, this field is always for the ultimate destination.

3.3.8 IPv6 Extension Header Types

IPv6 options supplement extension headers, in fact, they are actually implemented as extension headers. There are two different ones used to encode options. These two

headers only differ in terms of how the options they contain are to be processed by devices otherwise, they are formatted the same and used in the same way.

The two extension header types are [1]:

Destination Options: Contains options that are intended only for the ultimate destination of the datagram and perhaps a set of routers specified in a Routing header.

Hop-By-Hop Options: Contains options that carry information for every device (router) between the source and destination.

3.3.9 IPv6 Datagram Delivery and Routing

IP functions such as addressing, datagram encapsulation and if necessary, fragmentation and reassembly, all lead up to the ultimate objective of the protocol, the actual delivery of datagrams from a source device to one or more destination devices. Most of the changes in routing in IPv6 are directly related to changes in other areas of the protocol. Some of the main issues of note related to routing and routers in IPv6 include the following [6]:

Hierarchical Routing and Aggregation: One of the goals of the structure used for organizing unicast addresses was to improve routing. The unicast addressing format is designed to provide a better match between addresses and Internet topology, and to facilitate route aggregation. Classless addressing in IPv4 was an improvement, but lacked any formal mechanism for creating a scalable hierarchy.

***Scoped Local Addresses*:** Local-use addresses including site-local and link-local are defined in IPv6, and routers must be able to recognize them. They must route them or not route them when appropriate. Multicast addresses also have various levels of scope.

***Multicast and Anycast Routing*:** Multicast is standard in IPv6, not optional as in IPv4, so routers must support it. Anycast addressing is a new type of addressing in IPv6.

More Support Functions: Capabilities must be added to routers to support new features in IPv6.

***New Routing Protocols*:** Routing protocols such as RIP must be updated to support IPv6.

***Transition Issues*:** Routers play a major role in supporting the transition from IPv4 to IPv6. They will be responsible for connecting together IPv6 and performing translation to allow IPv4 and IPv6 devices to communicate with each other during the multi-year migration to the new protocol.

References

[1] The TCP/IP Guide: Version 3.0 by Charles M. Kozierok, 2005 Publicaton available at http://www.tcpipguide.com/index.htm.

[2] Behrouz A. Forouzan, "*Data Communications and Networking*", 3rd ed, Pearson Education.

[3] Cisco Systems, "*Internetworking Technologies Handbook*", 4th Edition, Pearson Education.

[4] Mariana Dell. Ray, "*Internet Protocol, Version 4 (IPv4) Specification*", Request for Comments 0791, Internet Engineering Task Force, Sep 1981.

[5] T. Scolofsky and Kale, "*A TCP/IP Tutorial*" Request for Comments 1180, Network Working Group, Jan 1991.

[6] S. Deering and R. Hinden, "*Internet Protocol, Version 6 (IPv6) Specification*", Request for Comments 2460, Internet Engineering Task Force, Dec 1998.

Chapter 4

Attack Methods

4.1 Classification of DoS Attacks

DoS attacks can be classified as either ***flooding attacks*** or ***logic attacks*** [1]. However a basic mechanism in all DoS attacks to hide the location of an attacker is ***IP spoofing*** which means sending packets with a false source IP address. It is possible to carry out DoS attacks without source address spoofing if the attacker has compromised enough hosts, or if a chain of compromised hosts is used.

4.1.1 Flooding DoS Attacks

A flooding DoS attack is based on brute force. Real-looking but unnecessary data is sent as much as possible to a victim. As a result, network bandwidth is wasted, disk space is filled with unnecessary data, or processing power is spent for unuseful purposes [2]. Flooding DoS attacks are carried out as either ***direct*** or ***reflector*** attacks [3]. Example of flooding attack is TCP SYN flooding.

4.1.2 Logic DoS Attacks

The objective of logic DoS attacks is to build a small number of specific packets exploiting vulnerabilities which cause the victim to do abnormal things. The packets are normally sent directly to a victim because special knowledge about vulnerability is required for building the attack packets [2]. Example of logic attack is the LAND attack.

4.2 Types of DoS/DDoS Attacks

4.2.1 TCP SYN Flooding Attacks

This attack exploits the three-way handshake mechanism of connection establishment at the transport layer. Basically, whenever a client wants to make a connection to a server it sends a TCP packet with the SYN flag set (request to open a connection). The server responds with a TCP packet with both the SYN and ACK flags set. The client completes the connection by sending a TCP packet with the ACK flag set. However, in TCP SYN flooding attacks, TCP packets with SYN flag set are sent at maximum rate to the victim with spoofed source addresses (non-existing host address). The server responds but it would never receive ACK packets. These half open connections consume some of the resources at the server side and since the SYN packets

are sent at maximum rate, all the connections at the server side are reserved for unresolved clients. Thus legitimate users are denied connections.

A solution for TCP SYN flooding attack is to use ***SCTP (Stream Control Transmission Protocol)*** at the transport layer instead of TCP. SCTP allocates resources after the connection is complete by using a feature called ***cookie*** [4].

4.2.2 ICMP Echo Flooding

ICMP (Internet Control Message Protocol) is a network layer protocol for the exchange of control packets between a router and a host or between hosts. In this type of attack the victim is forced to handle large number of ICMP Echo packets (ping packets). The Echo packets require the recipient to respond with an echo-reply to check that communication is possible between entities [5]. The attacker floods the victim with so much ping traffic that legitimate traffic fails to reach the system. Sending a ping packet of size greater than 64Kbytes can crash the target system and is known as ***Ping of Death***.

4.2.3 Application Layer Attacks

Application layer attacks refer to all the attacks performed at the layer 7 of the OSI model. This is the bulk of all attacks on the Internet today, and the vulnerabilities that enable these attacks represent the source of most of the insecurities in today's networks. General attacks such as buffer overflows, Web application attacks, and viruses and worms all fall into this category.

4.2.4 LAND Attack

It involves sending connection opening TCP packet (SYN flag set) to the victim. The source and destination address fields of this packet (TCP packets are encapsulated in IP datagrams) contain the IP address of the target system. The victim eventually crashes by replying to itself continuously.

4.2.5 Broadcast Amplification Attacks

Broadcast amplification attacks, commonly referred to as ***smurf*** attacks are a DoS attack tool that take advantage of the ability to send an echo-request message with a

destination address of a subnet broadcast and a spoofed source address, using the victim's IP. All end hosts on the subnet respond to the spoofed source address and flood the victim with echo-reply messages [6]. This attack can be mitigated if IP directed broadcasts are disabled on the router. The command '***no ip directed broadcasts'*** is the default for Cisco IOS Software version 12.0 and later.

4.2.6 Teardrop Attack

The Teardrop attack involves sending IP fragments with overlapping oversized payloads to the target machine. A bug in the TCP/IP fragmentation reassembly code causes the fragments to be improperly handled, crashing the operating system as a result of this [7].

4.2.7 UDP Data Flooding

Instead of directly overloading the victim, it may be possible to cause the victim or a machine on the same subnet as the victim to overload itself.

An example of such an attack is UDP data flooding, where the attacker spoofs the source address on a packet sent to the victim's UDP echo port. The source address is that of another machine that is running a UDP chargen server (a chargen server sends a character pattern back to the originating source). The result is that the two machines bounce packets back and forth as fast as they can, overloading either the network between them or one of the end-systems itself [6].

4.2.8 DoS Attacks on Routers

Many of the denial-of-service attacks that can be launched against end-systems can also be launched against the control processor of an IP router, for example, by flooding the command and control access ports. In case of a router, these attacks may cause the router to stall, or may cause the router to cease processing routing packets. Even if the router does not stop servicing routing packets, it may become sufficiently slow that routing protocols time out. In any of these circumstances, the consequence of routing failure is not only that the router ceases to forward traffic, but also that it causes routing protocol churn that may have further side effects [6].

4.2.9 Distributed Denial of Service (DDoS) Attack

An attempt to prevent or degrade the availability of resources by using multiple source hosts at the same time to send attack traffic. In the context of DDoS attack, an **agent (or daemon or zombie or bot)** is defined as a compromised host used to send attack traffic in a DoS attack. A **master (or handler)** is defined as a compromised host used to control operation of a large set of agents. A **DDoS network** is defined as a hierarchically structured set of masters and agents to make it easier to control a DDoS attacks by an attacker [2]. DDoS attacks can be classified as either direct or reflector attacks.

4.2.10 Direct DDoS Attacks

In direct DDoS attack, the attacker is able to implant zombie software on a number of hosts distributed throughout the internet. Often the DDoS attack involves two levels of zombie machines: master zombies and slave zombies. The hosts of both the machines have been infected with malicious software. The attacker coordinates and triggers the master zombies (handlers), which in turn coordinate and trigger the slave zombies (agents). The use of two levels of zombies makes it more difficult to trace the attack back to its source and provides for a more resilient network of attackers [5].

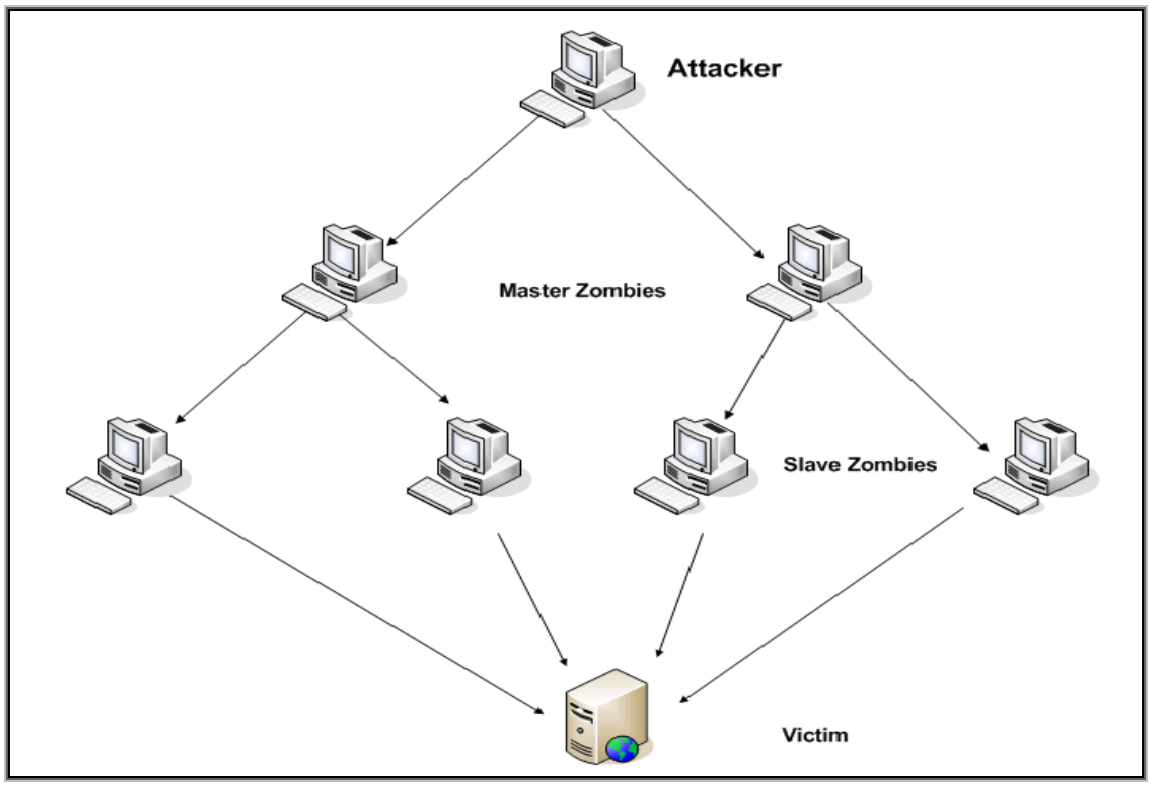

Figure 4.1: Direct DDoS Attack

4.2.11 Reflector DDoS Attacks

In reflector attacks, packets with the victim's address in the source IP address field are sent by slave zombies (agents) to innocent third parties (uninfected machines like web servers, DNS server etc), which in turn will send the reply to the victim (flood the victim). Reflector attacks thus have at least two victims at the same time [2]. Reflector attacks can be more damaging since they involve more machines and thus more traffic and they are more difficult to trace as well.

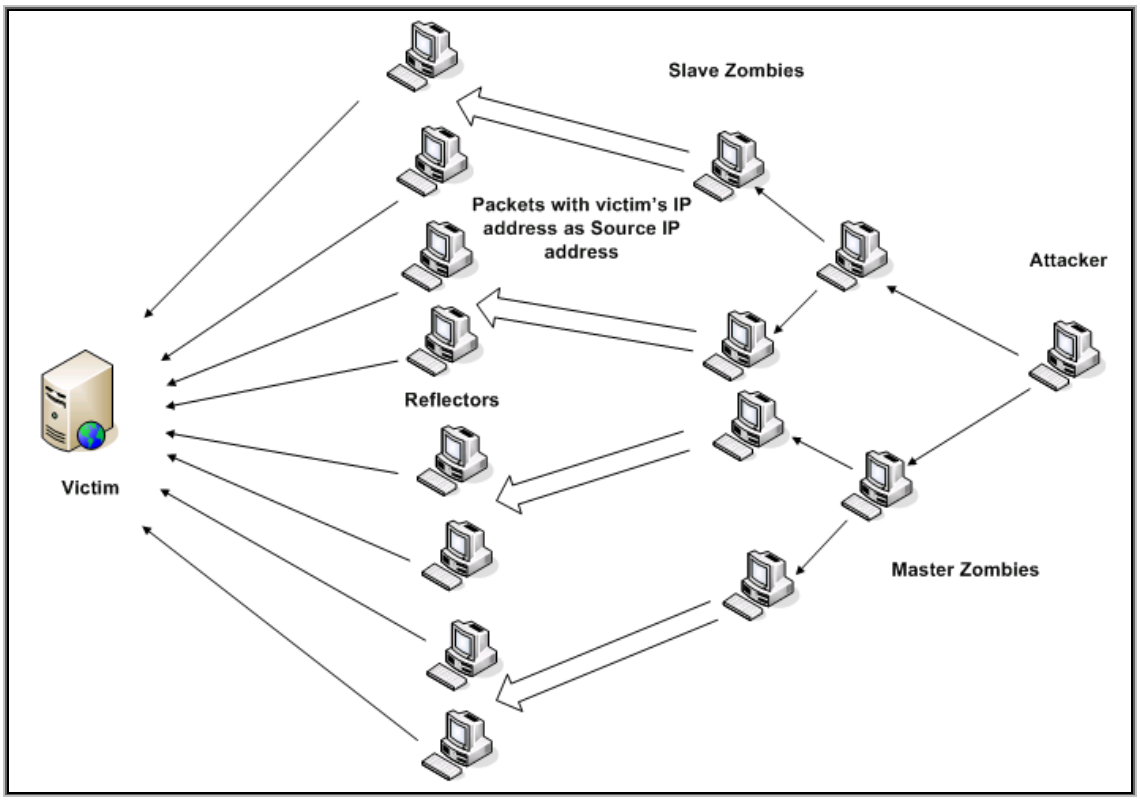

Figure 4.2: Reflector DDoS Attack

References

[1] D. Moore, G. M. Voelker, and S. Savage, "*Inferring Internet Denial-of-Service Activity*", in Proceedings of the 10th USENIX Security Symposium, Washington, D.C., Aug. 2001.

[2] Jarmo Molsa, "*Mitigating Denial of Service Attacks in Computer Networks*", Doctoral Dissertation, Helsinki University of Technology, 2006.

[3] R. K. Chang, "*Defending against Flooding-based Distributed Denial-of-Service Attacks: A Tutorial*," IEEE Communication. Magazine, vol. 40, no. 10, pp. 42–51, Oct. 2002.

[4] B. A. Forouzan, "*Data Communications and Networking*", 4th ed, Tata McGraw Hill pp. 724-726, 2006.

[5] W. Stallings, "*Cryptography and Network Security"*, 4th ed, Pearson Education, pp. 616-618, 2006.

[6] M. Handley and E. Rescorla, "*Internet Denial-of-Service Considerations*", Request for Comments 4732, Internet Engineering Task Force, Nov 2006.

[7] CERT Advisory CA-1997-28, "*Denial of Service Attacks*", http://www.cert.org/advisories/CA-1997-28.html, Dec 1997.

Chapter 5

Existing IP Traceback Techniques

5.1 IP Traceback

IP traceback is a name given to any method for reliably determining the origin of a packet on the Internet. The datagram nature of the Internet makes it difficult to determine the originating host of a packet, the source id supplied in an IP packet can be falsified (IP spoofing) allowing for Denial of Service attacks (DoS) or one-way attacks where the response from the victim host is so well known that return packets need not be received to continue the attack. The problem of finding the source of a packet is called the IP traceback problem. IP Traceback is a critical ability for identifying sources of attacks and instituting protection measures for the Internet.

In DoS/DDoS attack, attacker uses fake source IP addresses to make tracing and stopping of DoS difficult. This technique is called ***IP spoofing***. This technique involves the manipulation of the source IP address in the IP header of a transmitted packet. This gives the attacker a form of anonymity. It is difficult to solve problem of IP Spoofing because of lack of security features in TCP/IP specifications. Ingress filtering, Use of cryptographic authentication , IP trace back are some of the approaches used to handle forged IP source addresses [1]. The purpose of IP traceback is to identify the true IP address of a host originating attack packets. IP trace back is vital for quickly restoring normal network functionality and preventing reoccurrences [2].

5.1.1 Existing IP Traceback Techniques

There is no intrinsic support to identify the real sources of IP packets in the Internet architecture, so different techniques have been proposed to provide traceback capability. Existing trace back schemes can be roughly categorized into three distinct categories viz. traditional, marking and logging. In traditional scheme, victim develops an attack signature, consisting of some data common and unique to the attack traffic. A query including the attack signature is then sent hop-by-hop to each router along the path. Examples of this type of technique are input debugging and controlled flooding [3]. In packet logging, the IP packet is logged at each router through which it passes. Routers are queried in order to reconstruct the network path. SPIE (Source Path Isolation Engine) is an example of this type of technique [4]. In packet marking [5], the router marks IP

packets with its identification information. The network path can be reconstructed by combining packets containing marks. The marking information may be inscribed in the same attack packets called inbound marking or extra ICMP packets called outbound marking. Current traceback schemes based on marking include variants of PPM (Probabilistic Packet Marking), ATA (Algebraic Based Traceback Approach) [6], DPM (Deterministic Packet Marking) [7], and schemes that use ICMP (Internet Control Message) messages, such as iTrace [8].

5.2 Classification of IP Traceback Techniques

Traditional Approach

1. Input Debugging
2. Controlled Flooding

Logging

1. SPIE (Source Path Isolation Engine)

Packet Marking

1. Probabilistic Packet Marking
2. Deterministic Packet Marking
3. ICMP (Internet Control Message Protocol) Traceback
4. ATA (Algebraic Based Traceback Approach)

5.3 Traditional Approach

Traditional approach also referred as link-testing methods or hop-by-hop tracing work by testing network links between routers to determine the origin of the attacker's traffic.

5.3.1 Input Debugging

Input debugging start from the router closest to the victim and interactively tests its incoming (upstream) links to determine which one carries the attack traffic. This process repeats recursively on the upstream routers until reaching the traffic's source.

5.3.2 Controlled Flooding

Controlled flooding works by generating a burst of network traffic from the victim's network to the upstream network segments and observing how this intentionally generated flood affects the attack traffic's intensity. From changes in the attack traffic's frequency and intensity, the victim can deduce the incoming network link on the upstream router and repeat the same process on the router one level above. Traditional approach is not very efficient as it requires a lot of human effort and other network providers' support. For a successful trace, attack must last far a long enough duration. Compatibility with existing protocols, routers and network infrastructure is an advantage of this method [9].

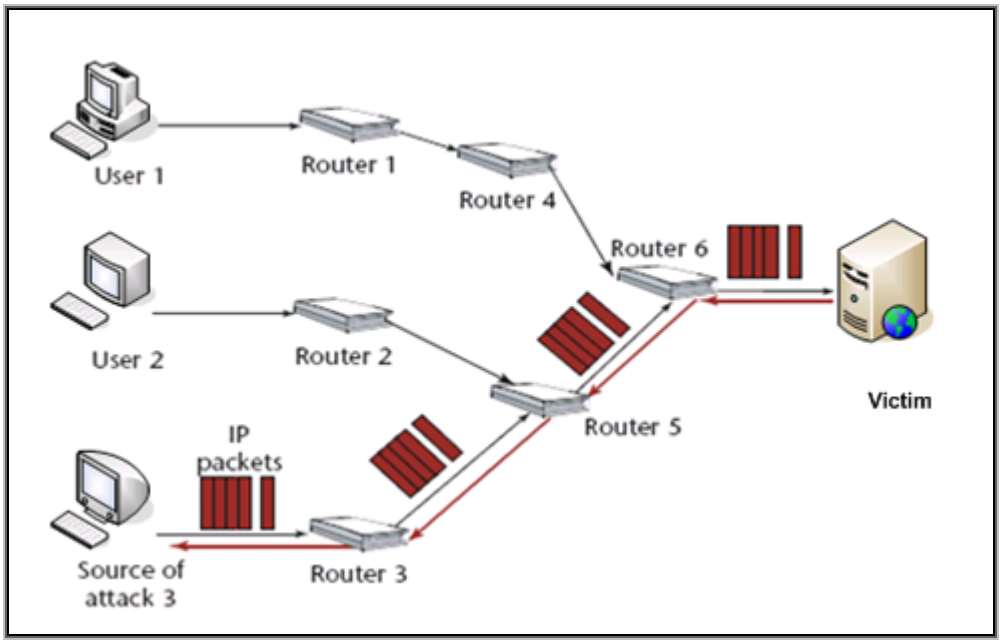

Figure 5.1: Link-testing Traceback

Advantages of Traditional Approach

- Both controlled flooding and input debugging are compatible with existing protocols.
- Both are compatible with existing routers and network infrastructure as well.
- Both support incremental implementation [9].

Disadvantages of Traditional Approach

- ISP cooperation is required for both controlled flooding and input debugging.
- Controlled flooding serves as a DoS attack.
- Both techniques are less suitable for DDoS attacks.
- Attack must last long enough for a successful traceback in both the cases [9].

5.4 Logging

The main idea of IP traceback approach based on packet logging is to log packets at each router through which they pass.

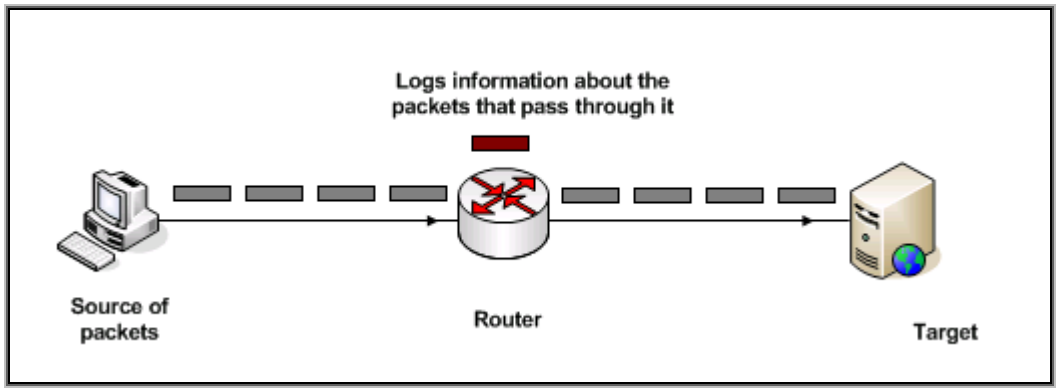

Figure 5.2: Logging approach

5.4.1 SPIE (Source Path Isolation Engine)

SPIE stores packet digests, instead of packets themselves, in a space-efficient data structure, to decrease the required storage space. For each arriving packet, the router uses the first 24 invariant byte of the packet (20-byte IP header with 4 bytes masked out plus

the first 8 bytes of payload) as input to the digesting function. The 32-bit packet digest is stored into the time-stamped digest table. The digest table is paged out before it becomes saturated. Digest tables are archived for one minute for potential traceback operation. During the traceback process, routers are queried in the reverse-path flooding (RPF) manner and the digest tables at queried routers are examined to reconstruct the network path [10]. Logging approach is resource-intensive in terms of processing and storage requirements. This scheme is not scalable. It is difficult to extend this scheme to complete Internet. Sharing of the logging information can lead to logistic and legal issues. Using Hash based IP traceback can reduce the storage overhead significantly [7].

Advantages of Logging

- This method is compatible with existing protocols.
- This method is compatible with existing routers and network infrastructure.
- It allows post-attack analysis.
- Insignificant network traffic overhead.
- Can trace a single packet [9].

Disadvantages of Logging

- This technique is resource-intensive in terms of processing and storage requirements.
- The sharing of the logging information among several ISPs leads to logistic and legal issues.
- It is less suitable for DDoS attacks [9].

5.5 Packet Marking

The basic idea of IP traceback approach based on packet marking is that the router marks packets with its identification information as they pass through that router.

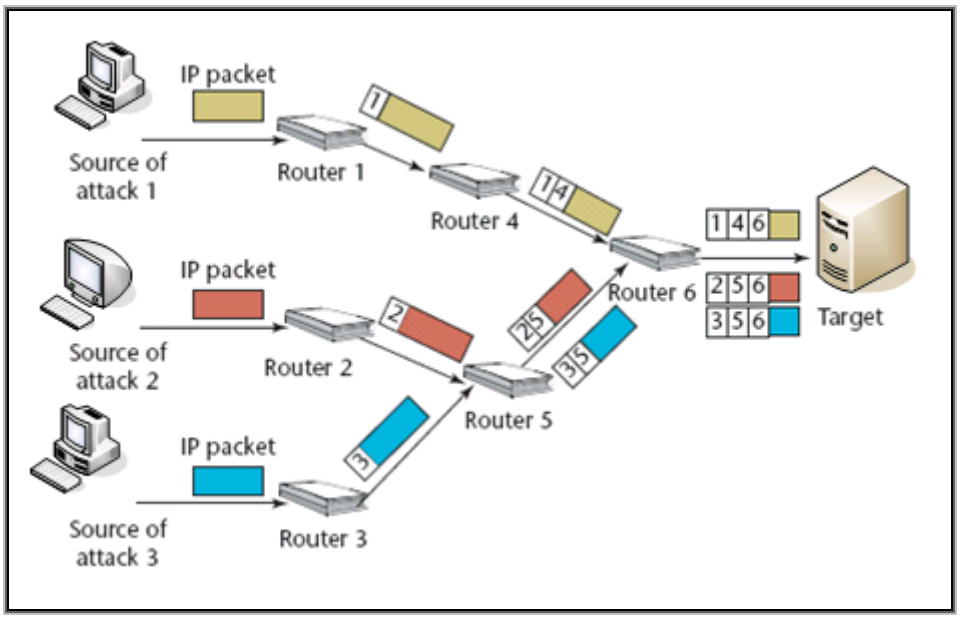

Figure 5.3: Packet marking

5.5.1 PPM (Probabilistic Packet Marking)

In Probabilistic Packet Marking the mark overloads a rarely used field in IP packet header, i.e., 16-bit IP identification field. The identification of a router could be 32-bit IP address, hash value of IP address or uniquely assigned number. In the last two cases, the length of identification information is variable and could be less than 16 bits. Since the marking space in packet header is too small to record the entire path, routers mark packets with some probability so that each marked packet carries the information of one node in the path. In addition, based on the length of router identification and the implementation of marking procedure, the router may only write part of its identification information into the marking space. While each marked packet represents only a small portion of the path it has traversed, the whole network path can be reconstructed by combining a modest number of such packets [11].

The PPM approach does not incur any storage overhead at routers and the marking procedure (a write and checksum update) can be easily and efficiently executed at current routers. But due to its probabilistic nature, it can only trace the traffic that consists of a large volume of packets. However, this method increases the packet's length at each router hop and can lead to additional fragmentation [9].

5.5.2 DPM (Deterministic Packet Marking)

In DPM only ingress edge routers perform the marking. All other routers are exempted from the marking task. Basic DPM uses the 16-bit IP identification field of the IP header and one reserved bit to record the marking information. The IP address of every ingress edge router is split into two segments with 16 bits each. One segment is randomly selected when a packet traverses this router. The idea is that the victim is capable of recovering the whole IP address of an ingress edge router once it obtains both segments from the same router. For the victim to figure out which portion of the IP address the current packet carries, one bit is used as a flag. Therefore, the marking information comprises two parts, the 16-bit partial IP address of the edge router and a 1-bit flag [7].

There are two main differences between DPM and PPM. DPM only marks the first ingress edge router, while PPM marks all routers along an attack path. PPM marks probabilistically, while DPM marks every packet at the ingress edge router. The task of ingress address reconstruction in DPM is much simpler than the task of path reconstruction in PPM.

Advantages of Packet Marking

- This technique can be deployed incrementally and appears to be low cost.
- It can work with existing routers and network infrastructure.
- It is effective against DDoS attacks.
- ISP cooperation not required.
- It allows post-attack analysis [9].

Disadvantages of Packet Marking

- It requires modifications to the protocol.
- PPM can produce false positive paths (that are not part of the attack path).
- DPM cannot handle fragmentation.

5.5.3 ICMP (Internet Control Message Protocol)

In ICMP traceback method, iTrace, each router selects one packet per 20,000 packets and then generates an ICMP message. The ICMP message has the same

destination IP address as the traced packet. The ICMP message also contains the IP header of the traced packet, and the IP addresses of the incoming interface and the outgoing interface of the current router. As long as the victim receives sufficient ICMP messages, it may recover the whole attack path.

The marking procedure of iTrace is very similar to PPM. Therefore, it shares similar pros and cons. Unlike PPM, ICMP traceback belongs to outbound marking, because of which ICMP traceback requires additional bandwidth to convey the marking information [7].

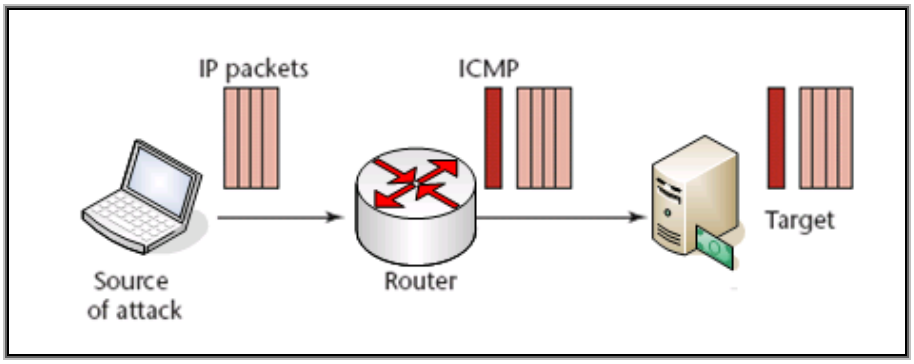

Figure 5.4: ICMP traceback

Advantages of ICMP

- This method is compatible with existing protocols.
- It can support incremental implementation.
- It allows post-attack analysis.
- If implemented with encryption and key distribution schemes presents a very promising and expandable technology for of dealing with denial-of-service attacks
- ISP cooperation is not required
- Compatible with existing routers and network infrastructure [9].

Disadvantages of ICMP

- ICMP traffic is increasingly differentiated and may itself be filtered in a network under attack.

- The ICMP Traceback message relies on an input debugging capability (i.e., the ability to associate a packet with the input port and/or MAC address on which it arrived) that is not available in some router architectures
- If only some of the routers participate it seems difficult to positively "connect" traceback messages from participating routers separated by a nonparticipating router.
- It requires a key distribution infrastructure to deal with the problem of attackers sending false ICMP Traceback messages [12].

5.5.4 ATA (Algebraic Based Traceback Approach)

ATA is a modified PPM method that uses algebraic techniques from the fields of coding theory and machine learning to encode and decode path information as points on polynomials. The encoded path information is stored in the Fragment ID field. At the victim side, algebraic methods are used to reconstruct the polynomials [13].

5.6 Challenges for IP Traceback

IP traceback has several limitations, such as the problem with tracing beyond corporate firewalls. To accomplish IP traceback, we need to reach the host where the attack originated. It is difficult; however, to trace packets through firewalls into corporate intranets, the last-traced IP address might be the firewall's address. Knowing the IP address of the organization's network entry point, however, allows us to obtain information about the organization where the attacker's host is located, such as the organization's name and the network administrator's e-mail address. If we can identify the organization from which the attack originated, the organization can often identify the user who launched the attack. Another limitation relates to the deployment of traceback systems. Most traceback techniques require altering the network, including adding router functions and changing packets. To promote traceback approaches, we need to remove any drawbacks to implementing them. Moreover, even if IP traceback reveals an attack's source, the source itself might have been used as a stepping-stone in the attack. IP traceback methods cannot identify the ultimate source behind the stepping-stone;

however, techniques to trace attacks exploiting stepping-stones are under study. Some operational issues must also be solved before IP traceback can be widely deployed. To trace an attack packet through different networks, for example, there must be a common policy for traceback. We also need guidelines for dealing with traceback results to avoid infringing on privacy. Furthermore, we need to consider how to use information about an attack source identified by IP traceback [14].

Table 5.1: Comparison of existing traceback methods

	Management overhead	**Network overhead**	**Router overhead**	**Logic attack detection**	**Nature**
Input debugging	High	Low	High	Poor	Reactive
Controlled flooding	Low	High	Low	Poor	Reactive
Logging	High	Low	High	Good	Proactive
PPM	Low	Low	Low	Poor	Proactive
DPM	Low	Low	Low	Excellent	Proactive
ICMP traceback	Low	Low	Low	Poor	Proactive

References

[1] J. Li, J. Mirkovic, M. Wang, P. Reiher and L. Zhang, "*SAVE: Source Address Validity Enforcement Protocol*", in Proc IEEE INFOCOM, 2002, pp 1557-1566.

[2] S.C. Lee and C. Shields, "*Tracing the Source of Network Attack: A Technical, Legal and Societal Problem*", Proc. 2001, IEEE Workshop on Information Assurance and Security, IEEE Press, 2001, pp. 239-246.

[3] Hassan Aljifri, "*IP Traceback: A New Denial of Service Deterrent*", IEEE Computer Society, 2003.

[4] A.C. Snoeren, C. Patriage, L. A. Sanchez and S. Kent, "*Hash-based IP Traceback*", Proc. of ACM SIGCOMM conference, 2001, San Diego, CA, Computer Communication Review vol 31, no 24, Oct 2001, pp. 3-14.

[5] Chao Gong and Kamil Sarac, "*IP Traceback based on Packet Marking and logging"*, Proc. of International Conference on Communication, 2005, Seoul, Korea, IEEE Communications Magazine, vol 2, May 2005, pp 1043-1047.

[6] D. Dean, M. Franklin and A. Stubblefield, "*An Algebraic Approach to IP Traceback"*, ACM Trans. Info. and Sys. Sec., vol. 5, May 2002, pp. 119-137.

[7] Zhiqiang Gao and Nirwan Ansari, "*Tracing Cyber Attacks from Practical Perspective*", IEEE Communications Magazine, 2005.

[8] Vadim Kuznetsov, Andrei Simkin and Helena Standstorm, "*An evaluation of different IP Traceback Approaches*", Proc. of 4th International Conference on Information and Communication Security, Singapore, 2002, pp 37-48.

[9] Hassan Aljifri, "*IP Traceback: A New Denial of Service Deterrent*", IEEE Computer Society, 2003.

[10] A.C. Snoeren, C Patriage, L. A. Sanchez and S. Kent, "*Hash-based IP Traceback*", Proc. of ACM SIGCOMM conference, 2001, San Diego, CA, Computer Communication Review vol 31, no 24, Oct. 2001, pp. 3-14.

[11] Chao Gong and Kamil Sarac, "*IP Traceback based on Packet Marking and logging"*, Proc. of International Conference on Communication, 2005, Seoul, Korea, IEEE Communications Magazine, vol. 2, May 2005, pp 1043-1047.

[12] Stefan Savage, David Wetherall, Anna Karlin and Tom Anderson, "*Network support for IP Traceback*", IEEE/ACM Transactions on Networking, vol 9, no 3, 2001, pp 226-237.

[13] D. Dean, M. Franklin, and A. Stubblefield, "*An Algebraic Approach to IP Traceback*," ACM Trans. Information and System Security, vol. 5, no. 2, 2002, pp. 119–137.

[14] Tatsuya Baba and Shigeyuki Matsuda, "*Tracing Network Attacks To Their Sources*", IEEE Computer society, vol. 6, no 3, 2002, pp 20-26.

Chapter 6

Proposed IP Traceback Technique

6.1 A Hybrid IP Traceback Technique Based on TTL Identification

We propose a new IP traceback technique which is a variant of packet marking and is based on TTL identification. This technique was initially proposed for IPv4 networks.

Different IP traceback techniques have been proposed so far. Some of them are compatible with existing infrastructure and some require modification to it, but the effectiveness of any traceback technique can be measured by the following characteristics.

1. Capability of tracing any type of DoS attack.
2. Minimum overhead in terms of storage requirements.
3. Minimum processing requirements on the routers.
4. Least complexity in path reconstruction algorithm (if any).
5. Faster convergence.

This hybrid technique was proposed by taking into account all the above mentioned characteristics.

The aim of all the traceback techniques is to identify the sources of attacking traffic but path reconstruction algorithms actually reveal the identity of first router on the path. A better approach would be to find an algorithm that reveals the identity of first router without requiring the participation of all the routers on the path [1].

Since the attacker can forge any field in the IP header, he can't falsify the Time to live (TTL) field. The TTL is an 8-bit field that determines the maximum number of hops a datagram can traverse. Each router decrements the TTL value by 1, after forwarding the datagram. The problem of determining the first router on the path can be solved by using this field.

The TTL field is different for different operating systems and is not universally selected, but all the packets sent by a particular operating system will have the same initial TTL value [2]. Default TTL values for different operating systems are shown in Table 1.

We propose to create a TTL vs. operating system (OS) table and store it on the routers. The router should read the TTL value of all the packets passing through it. If the TTL value matches any entry of the table, the router should mark the packet with its identification. This router would obviously be of the subnet from which the packets originated. All the other routers on the path cannot mark the packet, since the TTL value would not match any entry of the TTL vs. OS table. There is one exception to this. Consider a packet originating with a TTL of 64. It would be marked by its subnet router. But after four hops, its TTL would be 60. Since 60 is an entry of stored table, the packet would again be marked. If this packet was sent by an attacker, traceback would lead to a subnet four hops away from the subnet from which the attack originated. To overcome this problem, we propose to use the reserved flag in IP header (there are 3 flags in IP header, 2 are used with fragmentation and one is reserved). The first router on the path should set the flag to '1' after marking the packet. All the other routers on the path should read both the TTL and reserved flag. The packet should only be marked if a match of TTL is found and the value of reserved flag is '0'. It is however not possible that a packet originating with a TTL of 255 would reach a router with its TTL set to 128 since 95% of the traffic in the internet reaches its destination before 30 hops. The storage requirements on the routers can be minimized by having only a 5 entry TTL table (since most of the values in table 6.1 are occurring more than once).

The proposed marking scenario obviously has the first two characteristics of an effective IP traceback technique. We can trace any type of DoS attack because each and every packet is marked by its subnet router. It is possible to trace DoS attacks which require only a single packet, which may not be possible with other marking techniques like PPM and iTrace.

Table 6.1: Default initial TTL values for different operating systems

OS	Version	Platform	TTL value
Windows	9x/NT	Intel	32
Windows	9x/NT	Intel	128
Windows	2000	Intel	128

Digital Unix	4.0	Alpha	60
Unisys	x	Mainframe	64
Linux	2.2.x	Intel	64
FTX (UNIX)	3.3	STRATUS	64
SCO	R5	Compaq	64
Netware	4.11	Intel	128
AIX	4.3.x	IBM / RS6000	60
AIX	4.2.x	IBM / RS6000	60
Cisco	11.2	7507	60
Cisco	12.0	2514	255
IRIX	6.x	SGI	60
Free BSD	3.x	Intel	64
Open BSD	2.x	Intel	64
Solaris	8	Intel / Sparc	64
Solaris	2.x	Intel / Sparc	255

The second challenge is obviously to store the marking information. The question arises where to store the marking information in the IP header. Basic DPM [3] uses the 16-bit 'Identification' field of the IP header. However, choosing Identification field may not be a good idea because it is used for fragmentation purposes. Fragmented traffic constitutes between 0.25% and 0.5% of the total IP traffic [4]. Though the amount of fragmented traffic is small, it does exist and for the worst case scenario, Identification field should be reserved only for fragmentation purposes. Carrying marking information in outbound packets (iTrace) would obviously increase router and network overhead and would require a new and complex protocol to be implemented as well.

We propose to use the Record Route (RR) optional field in the IPv4 header. The IP address of the router would be stored in the first 4 bytes of route data in RR field. Thus the router appends the marking information with the packets. If the RR field is already present, the router should overwrite the first 4 bytes of route data in it. Thus even if the attacker forges the RR field with wrong IP addresses or unnecessary data, it would still be

overwritten with the true IP address of the router. The minimum required length of RR field is 7 bytes (4 bytes for route data, 1 byte for option type code, 1 byte for option length and 1 byte for pointer into the route data). The remaining space in the optional field can be used for other options like 'Strict Source Route', 'Loose Source Route', 'Stream Identifier' etc, if required.

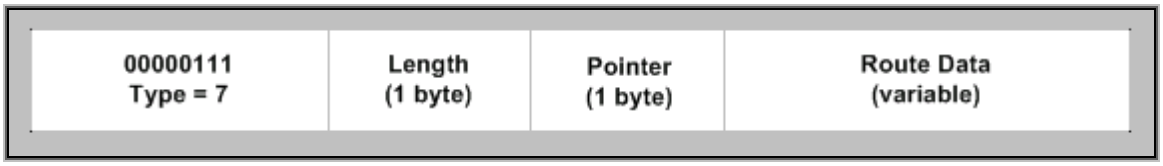

Figure 6.1: Record Route Option

```
#define net_id
#define router_ip

struct datagram
{
/*
   this structure contains different fields of IP Header
   e.g, int ttl;
*/

main() {
datagram D;
int ttl_table[18];  /*
                        initial ttl values for different
                        operating systems
                    */
for (each D)
{
 for i=0:17
 {

  if D.ttl==table[i]          // ingress router identified
  {
   y=compute_netid(D.source_address);  /*
                                          compute net id
                                          from source IP
                                          address
                                       */
   if y==net_id
   {
    write router_ip into D.record_route
    forward(D)      // forward the datagram after marking
    }
   else if y!=net_id
    discard(D)        /*
                        discard datagram if invalid source
                        IP address
                      */

   }

   else if D.ttl!=table[i]          // not ingress router
   forward(D)         // forward datagram without marking
   }

  }
 }
```

Figure 6.2: Algorithm for the proposed IP traceback technique

The problem of source address spoofing can be solved by a technique called ***Ingress Filtering*** [5], in which the router discards the incoming packets with invalid source IP address. A serious limitation of this technique arises when the attacker forges the address to the one that belongs to the same network as the attacker's host. Ingress filtering is commonly done at the border router. So this technique is not effective for internal attacks.

We propose ***Egress Filtering*** at the subnet router, in which the router discards the outgoing packets with illegitimate source IP address. The legitimacy of source IP address can be checked from the network id part of the 32-bit IP address. An effective IP traceback technique can result by combining this filtering technique with the proposed packet marking (based on TTL identification) technique, and thus the name 'Hybrid'. The algorithm for this hybrid technique is shown in figure 6.2.

The packet is first checked for spoofing and is discarded if the source address is forged (doesn't have a valid network id). If the source address has a valid network id, the packet is marked with router's identity. It should be noted that spoofing attacks from the same subnet cannot be stopped because the router is checking only the network id of the IP address.

This hybrid technique is significantly different from the basic DPM. The basic DPM needs an address construction algorithm since the 32-bit IP address is divided into two 16-bit IP addresses which are stored in alternate packets. This technique however doesn't require any address reconstruction algorithm. Also, the Identification field is retained for fragmentation purposes. This technique doesn't require a path reconstruction algorithm since only the first router on the path is participating in marking process. It provides faster convergence as well, since a single packet can provide the traceback capability at the victim side.

As far as DDoS attacks are concerned, this technique can prevent reflector DDoS attacks; since the slave zombies cannot send spoofed packets to reflectors (spoofed packets would be discarded). This technique however can trace the direct DDoS attacks till slave zombies only. Once the slave zombies are identified, suitable measures can be taken to prevent reoccurrences of DDoS attacks using these compromised machines.

This technique has the additional advantage of tracing the internal attacks in a corporation's network (figure 6.3). This technique can trace beyond corporate firewall.

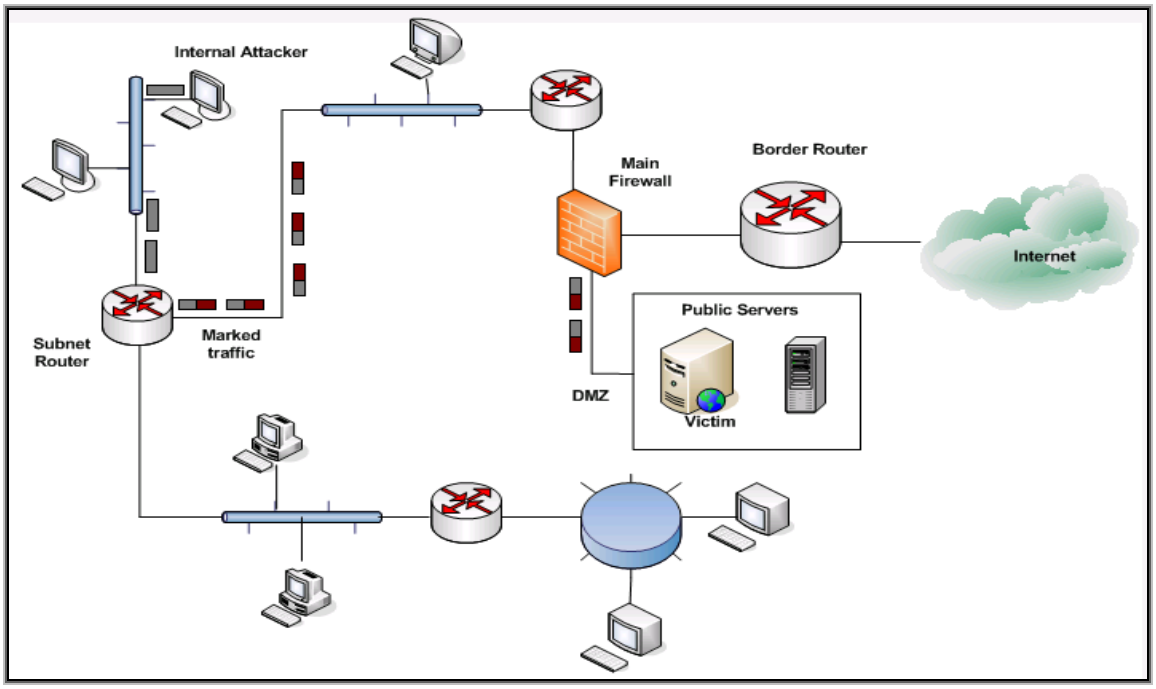

Figure 6.3: Tracing internal attackers

6.2 Performance Evaluation

This hybrid technique was simulated in the Network Simulator (ns-2.31) [6] at network layer to measure the delay for marked traffic as compared to normal (unmarked) traffic. The simulator was running on an Intel based machine having 1.7 GHz processor and 512 MB of main memory. The internal files of ns-2.31 were modified to incorporate packet marking in it. The modified internal files along with the Tcl code are given in Appendix A. Appendix B contains a detailed tutorial on ns-2. The simulated scenario and simulated topology are shown in figures 6.4 and 6.5 respectively.

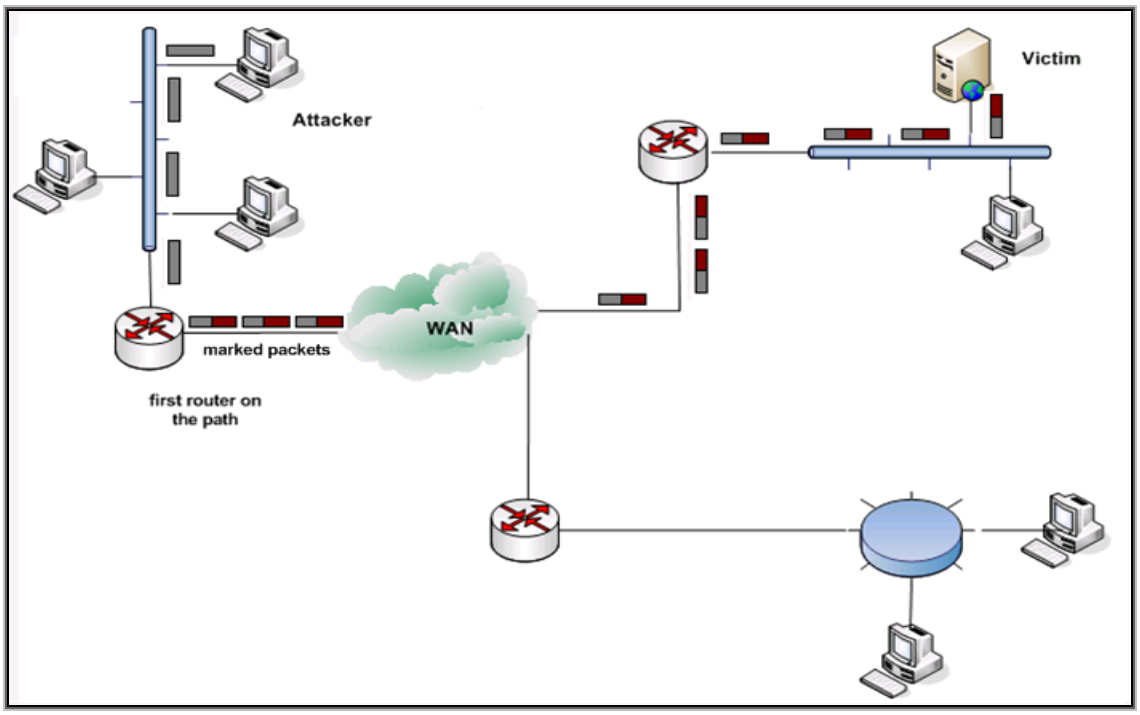

Figure 6.4: Simulated Scenario

The size of the topology doesn't matter because all the delay is incurred at the first router only. Traffic originated from node 7 and was destined for node 3. For this traffic, node 0 acts as the first router on the path. Traffic consisted of TCP packets of 1040 bytes carrying FTP data. The comparison between marked and normal traffic is shown in figure 6.6. The additional time taken by marked traffic is just 0.8 milliseconds. Similar delay is also observed for traffic from node 13 to node 8 for which node 12 acts as the first router on the path.

6.3 IPv6 Compatibility

IPv6 was proposed to overcome different problems of IPv4 like address depletion, lack of security features, lack of support for real-time audio and video etc. IPv6 addresses are 128 bits long as compared to 32 bit addresses of IPv4 [7].The Hop limit field in IPv6 base header serves the same purpose as TTL field in case of IPv4. The length of base header is fixed at 40 bytes. However, to achieve greater functionality, extension headers

are used. Most of the extension headers are options in IPv4. In this section we describe, how our proposed technique would work with IPv6.

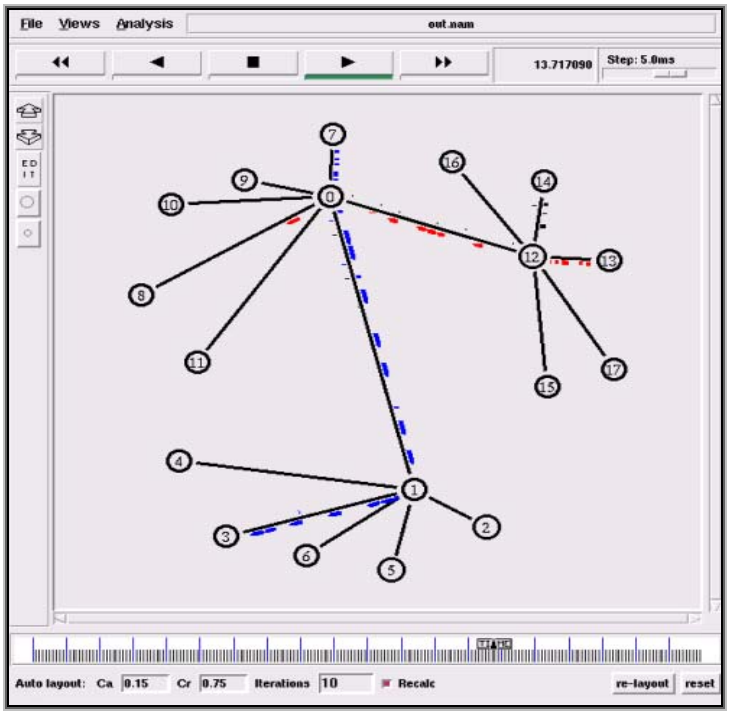

Figure 6.5: Simulated topology

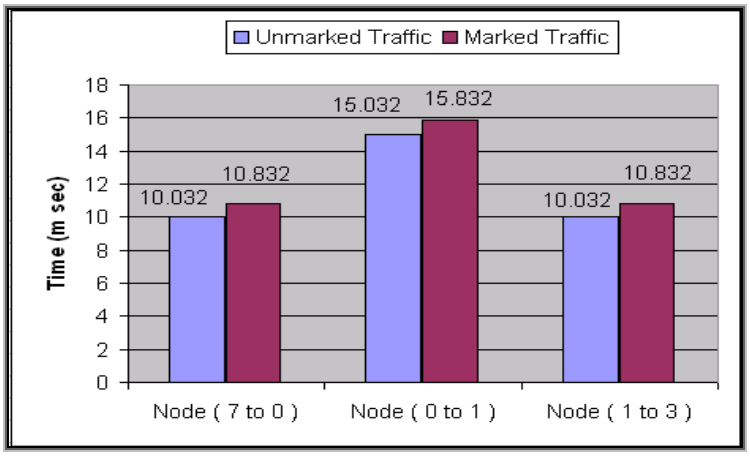

Figure 6.6: Graph showing the delay for marked traffic as compared to normal traffic

The matching of Hop limit value with any entry of stored table (Hop limit vs. OS table in this case) will identify the first router on the path. This router should then mark the packet with its identification (128 bit IP address). Since Record Route option is not

implemented in IPv6, we propose to use 'Destination Option' extension header which is identified by a Next header value of 60. The Destination Option is used when the source needs to pass information to the destination only. Intermediate routers are not permitted to access this information.

The 128 bit Destination address field in IPv6 header is not necessarily the final destination. It contains the address of next hop when source routing is used. It means that the intermediate routers can have access to Destination Option field which is not desired.

This problem can be solved by placing the Destination Option extension header after the Routing header and before the upper layer header [7]. In this case only the final destination is allowed to process the Destination Option header.

The legitimacy of the source IP address, in this case can be checked from the 32-bit subnet identifier portion of the 128-bit IP address. The packet is discarded if it doesn't have a valid source IP address.

References

[1] Vadim Kuznetsov, Andrei Simkin and Helena Standstorm, "*An evaluation of different IP Traceback Approaches*", Proc. of 4th International Conference on Information and Communication Security, Singapore, 2002, pp 37-48.

[2] Ankit Fadia, "*Network Security: A Hacker's Perspective*", pp. 125-127, Course Technology Inc, 2006. ISBN 1598631632.

[3] A. Belenky and N. Ansari, "*IP Traceback with Deterministic Packet Marking*", IEEE Communications Letters, vol. 7, no. 4, pp. 162-164, April 2003.

[4] C. Shannon et al, "*Beyond folklore: observations on fragmented traffic*", IEEE/ACM Trans. Networking, vol. 10, no. 6, pp.709-720, Dec 2002.

[5] A.C. Snoeren, C Patriage, L. A. Sanchez and S. Kent, "*Hash-based IP Traceback*", Proc. of ACM SIGCOMM conference, 2001, San Diego, CA, Computer Communication Review vol. 31, no 24, Oct 2001, pp. 3-14.

[6] The Network Simulator – ns-2, http://www.isi.edu/nsnam/ns

[7] S. Deering and R. Hinden, "*Internet Protocol, Version 6 (IPv6) Specification*", Request for Comments 2460, Internet Engineering Task Force, Dec 1998.

Chapter 7

Mobile IP

7.1 Internet Protocol Mobility Support (Mobile IP)

The Internet Protocol (IP) is the most successful network layer protocol in computing due to its strength, but it also has some weaknesses, most of which have become more important as networks have evolved over time. While mobile devices can certainly use IP, the way that devices are addressed and datagrams routed causes a problem when they are moved from one network to another. To support IP in a mobile environment, a new protocol called IP Mobility Support, or more simply Mobile IP was developed.

IP addresses are divided into two portions, a network identifier (network ID) and a host identifier (host ID). The network ID specifies which network a host is on, and the host ID uniquely specifies hosts within a network. This structure is fundamental to datagram routing, because devices use the network ID portion of the destination address of a datagram to determine if the recipient is on a local network or a remote one, and routers use it to determine how to route the datagram.

The tight binding of network identifier and host IP Address means that there are only two real options under conventional IP when a mobile device moves from one network to another:

Change IP Address: We can change the IP address of the host to a new address that includes the network ID of the network to which it is moving.

Decouple IP Routing From Address: We can change the way routing is done for the device, so that instead of router sending datagrams to it based on its network ID, they route based on its entire address [1].

Unfortunately, they are both inefficient, often impractical, and neither is scalable, meaning, and practical when thousands or millions of devices try them,

- Changing the IP address each time a device moves is time-consuming and normally requires manual intervention. In addition, the entire TCP/IP stack would need to be restarted, breaking any existing connections.
- If we change the mobile device's IP address, how do we communicate the change of IP to other devices on the Internet. These devices will only have the mobile

node's original home address, which means they won't be able to find it even if we give it a new address matching its new location.

- Routing based on the entire address of a host would mean the entire Internet would be flooded with routing information for each and every mobile device.

To ensure its success, Mobile IP's designers had to meet a number of important goals. The resulting protocol has these key attributes and features [2]:

1. **Seamless Device Mobility Using Existing Device Address:** Mobile devices can change their physical network attachment method and location while continuing to use their existing IP address.
2. **No New Addressing or Routing Requirements:** The overall scheme for addressing and routing as in regular IP is maintained. IP addresses are still assigned in the conventional way, by the owner of each device. No new routing requirements are placed on the network, such as host-specific routes.
3. **Interoperability:** Mobile IP devices can still send to and receive from existing IP devices that do not know how Mobile IP works, and vice-versa.
4. **Layer Transparency:** The changes made by Mobile IP are confined to the network layer. Transport layer and higher layer protocols and applications are able to function as in regular IPv4, and existing connections can even be maintained across a move.
5. **Limited Hardware Changes:** Changes are required to the software in the mobile device, as well as to routers used directly by the mobile device. Other devices, however, do not need changes, including routers between the ones on the home and visited networks.
6. **Scalability:** Mobile IP allows a device to change from any network to any other, and supports this for an arbitrary number of devices. The scope of the connection change can be global.
7. **Security:** Mobile IP works by redirecting messages, and includes authentication procedures to prevent an unauthorized device from causing problems.

7.2 Mobile IP Terminology

The different terminologies used in the context of Mobile IP are as follows [3]:

1. **Mobile Node (MN).** A node that, as part of normal use, changes its point of attachment to the internet.
2. **Mobility Agent.** A node (typically a router) that offers support services to mobile nodes. A mobility agent can either be home agent or a foreign agent.
3. **Home Network.** The network at which the mobile node seems reachable, to the rest of internet, by virtue of its assigned IP address.
4. **Home Address (HoA).** The IP address assigned to the mobile node, making it logically appear attached to its home network.
5. **Care-of-Address (CoA).** An IP address at the mobile node's current point of attachment to the internet, when the mobile node is not attached to the home network. In case of Mobile IPv4, the CoA remains same for a number of mobile nodes attached to the foreign agent (all MNs attached to a FA use one CoA, which is the IP address of the FA). A ***collocated care of address*** is a care of address assigned to one of the mobile node's network interfaces, instead of one being offered by a foreign agent.
6. **Foreign Network.** The network to which the mobile node is attached when it is not attached to its home network, and on which care of address is reachable from rest of the internet.
7. **Home Agent (HA).** A node on the home network that effectively causes the mobile node to be reachable to its home address even when the mobile node is not attached to its home network.
8. **Foreign Agent (FA).** A mobility agent on the foreign network that can assist the mobile node in receiving datagrams delivered to the care of address.
9. **Binding.** The triplet of numbers that contains the mobile node's home address, care of address and registration life time.
10. **Correspondent Node (CN).** A node that sends or receives a packet to a mobile node. A CN may be another MN or a non-mobile internet node.

7.3 Mobile IP Functions

Mobile IP includes a host of special functions that are used to set up and manage datagram forwarding. To see how these support functions work, following describes the general operation of Mobile IP as a simplified series of steps [2]:

1. **Agent Communication:** The mobile node finds an agent on its LAN by engaging in the Agent Discovery process. It listens for Agent Advertisement messages sent out by agents and from this can determine where it is located. If it doesn't hear these messages it can ask for one using an Agent Solicitation message.
2. **Network Location Determination:** The mobile node determines whether it is on its home network or a foreign one by looking at the information in the Agent Advertisement message.

If it is on its home network it functions using regular IP. For the device which just moved to a foreign network, the steps are [2]:

3. **Care-Of Address Acquisition:** The device obtains a temporary address called a care-of address. This either comes from the Agent Advertisement message from the foreign agent, or through some other means. This address is used only as the destination point for forwarding datagrams, and for no other purpose.
4. **Agent Registration:** The mobile node informs the home agent on its home network of its presence on the foreign network and enables datagram forwarding, by registering with the home agent. This may be done either directly between the node and the home agent, or indirectly using the foreign agent.
5. **Datagram Forwarding:** The home agent captures datagrams intended for the mobile node and forwards them. It may send them either directly to the node or indirectly to the foreign agent for delivery, depending on the type of care-of address in use.

Datagram forwarding continues until the current agent registration expires. The device can then renew it. If it moves again, it repeats the process to get a new care-of address and then registers its new location with the home agent. When the mobile node

returns to its home network, it deregisters with the FA to cancel datagram forwarding and resumes normal IP operation.

7.3.1 Agent Discovery Process

When a mobile node is first turned on, it cannot assume that it is still at home the way normal IP devices do. It must first determine where it is, and if it is not at home, begin the process of setting up datagram forwarding from its home network. This process is accomplished by communicating with a local router serving as an agent, through the process called agent discovery [1].

***Agent/Node Communication*:** Agent discovery is the method by which a mobile node first establishes contact with an agent on the LAN to which it is attached. Messages are sent from the agent to the node containing important information about the agent, a message can also be sent from the node to the agent asking for this information to be sent.

***Orientation*:** The node uses the agent discovery process to determine where it is. Specifically, it learns whether it is on its home network or a foreign network by identifying the agent that sends it messages.

***Care-Of Address Assignment*:** The agent discovery process is the method used to tell a mobile node the care of address it should use, when foreign agent care-of addressing is used.

Mobile IP agents are routers that have been given additional programming to make them Mobile IP aware. The communication between a mobile node and the agent on its local network is basically the same as the normal communication required between a device on an IP network and its local router, except more information needs to be sent when the router is an agent.

***Agent Advertisement*:** This is a message transmitted regularly by a router acting as a Mobile IP agent. It consists of a regular Router Advertisement message that has one or more extensions added that contain Mobile-IP-specific information for mobile nodes.

***Agent Solicitation*:** This message can be sent by a mobile IP device to urge a local agent to send an Agent Advertisement [1].

7.3.2 Mobile IP Home Agent Registration

Once a mobile node has completed agent discovery, it knows whether it is on its home network or a foreign network. If on its home network it communicates as a regular IP device, but if on a foreign network it must activate Mobile IP. This requires that it communicate with its home agent so information and instructions can be exchanged between the two. This process is called home agent registration, or more simply, just registration.

The main purpose of registration is to actually start Mobile IP working. The mobile node must contact the home agent and tell it that it is on a foreign network and request that datagram forwarding be turned on. It also must let the home agent know its CoA so the home agent knows where to send the forwarded datagrams. The home agent in turn needs to communicate various types of information back to the mobile node when registration is performed. Foreign agent is not really involved in registration.

Successful registration establishes what is called in the standard a mobility binding between a home agent and a mobile node. For the duration of the registration, the mobile node's regular home address is tied to its current care-of address and the home agent will encapsulate and forward datagrams addressed to the home address over to the care-of address. The mobile node is supposed to manage its registration and handle various events using several actions:

***Registration*:** The mobile node initiates a registration when it first detects it has moved from its home network to a foreign network.

Deregistration: When the mobile node returns home, it should tell the home agent to cancel forwarding, a process called deregistration.

***Re-registration*:** If the mobile node moves from one foreign network to another, or if its care-of-address changes, it must update its registration with the home agent. It also must do so if it’s current registration is about to expire, even if it remains stationary on one foreign network.

Each registration is established only for a specific length of time, which is why regular re-registration is required whether the device moves or not. Registrations are time-limited to ensure that they do not become stale. If, for example, a node forgets to de-register when it returns home, the datagram forwarding will eventually stop when the registration expires.

7.3.3 Mobile IP Data Encapsulation and Tunneling

Once a mobile node on a foreign network has completed a successful registration with its home agent, the home agent will intercept datagrams intended for the mobile node as they are routed to its home network, and forward them to the mobile node. This is done by encapsulating the datagrams and then sending them to the node's care-of address.

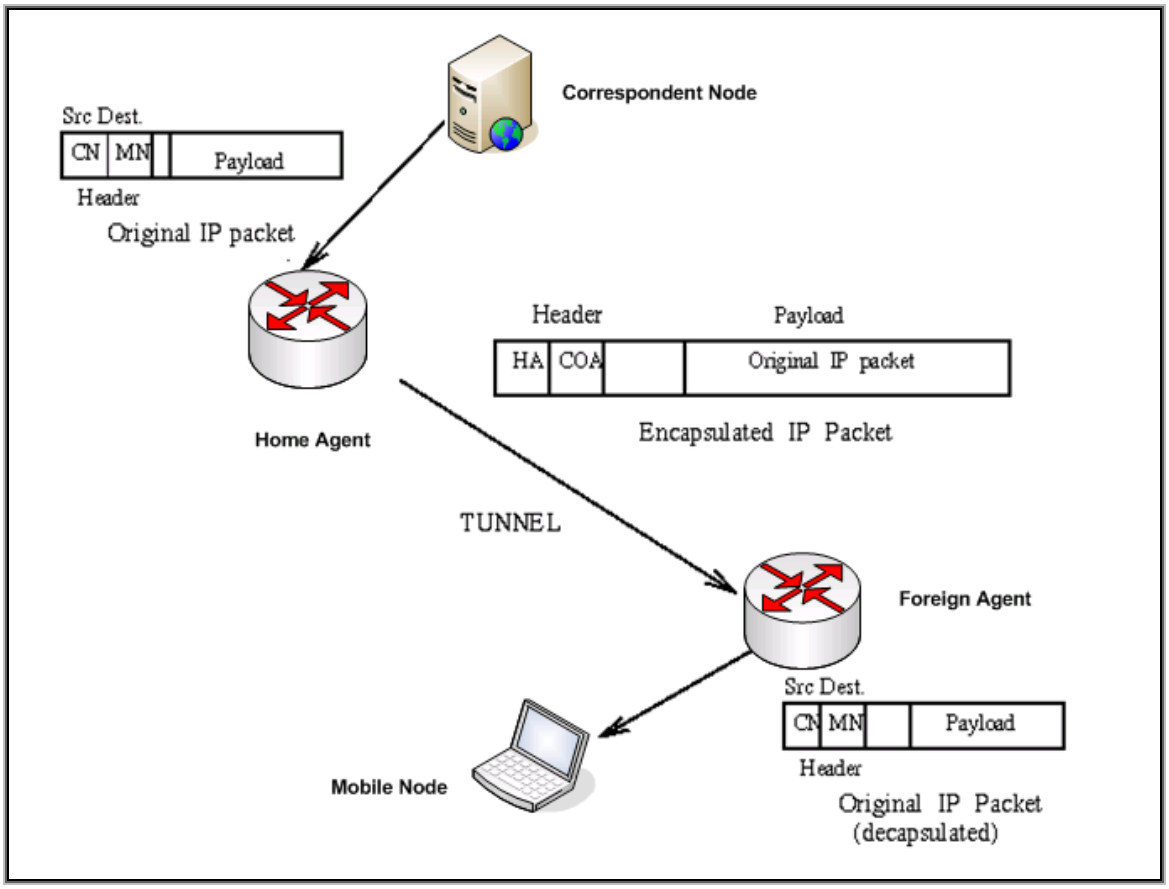

Figure 7.1: Tunneling operation in Mobile IP

Encapsulation is required because each datagram we intercept and forward needs to be resent over the network to the device's care-of address. Its not just having the home agent to change the destination address and stick it back out on the network, but there are various complications that make this unwise. It makes more sense to take the entire datagram and wrap it in a new set of headers before retransmitting.

The encapsulation process creates a logical construct called a tunnel between the device that encapsulates and the one that decapsulates. The tunnel represents a path over which datagrams are forwarded across an arbitrary network, with the details of the encapsulated datagram (meaning the original IP headers) temporarily hidden.

In Mobile IP, the start of the tunnel is the home agent, which does the encapsulation. The end of the tunnel depends on what sort of care-of address is being used. Either mobile node itself is the end of the tunnel and strips off the outer header or the foreign agent is the end of the tunnel. It receives encapsulated messages from the home agent, strips off the outer IP header and then delivers the datagram to the mobile node. This is generally done using layer two, because the mobile node and foreign agent are on the same local network, and of course, the mobile node does not have its own IP address on that network, it is using that of the foreign agent.

7.3.4 Mobile IP Conventional Tunneling

Normally, the tunnel described above is used only for datagrams that have been sent to the mobile node and captured by the home agent. When the mobile node wants to send a datagram, it doesn't tunnel it back to the home agent which makes it inefficient. Instead it just sends out the datagram directly using whatever router it can find on its current network, which may or may not be a foreign agent. When it does this, it uses its own home address as the source address for any requests it sends. As a result, any response to those requests will go back to the home network. This sets up a sort of ***routing triangle*** for these kinds of transactions [1]:

1. The mobile node sends a request from the foreign network to some third party device somewhere on the network.
2. The third party device responds back to the mobile node. However, this sends the reply back to the mobile node's home address on its home network.

3. The home agent intercepts the response on the home network and tunnels it back to the mobile node.

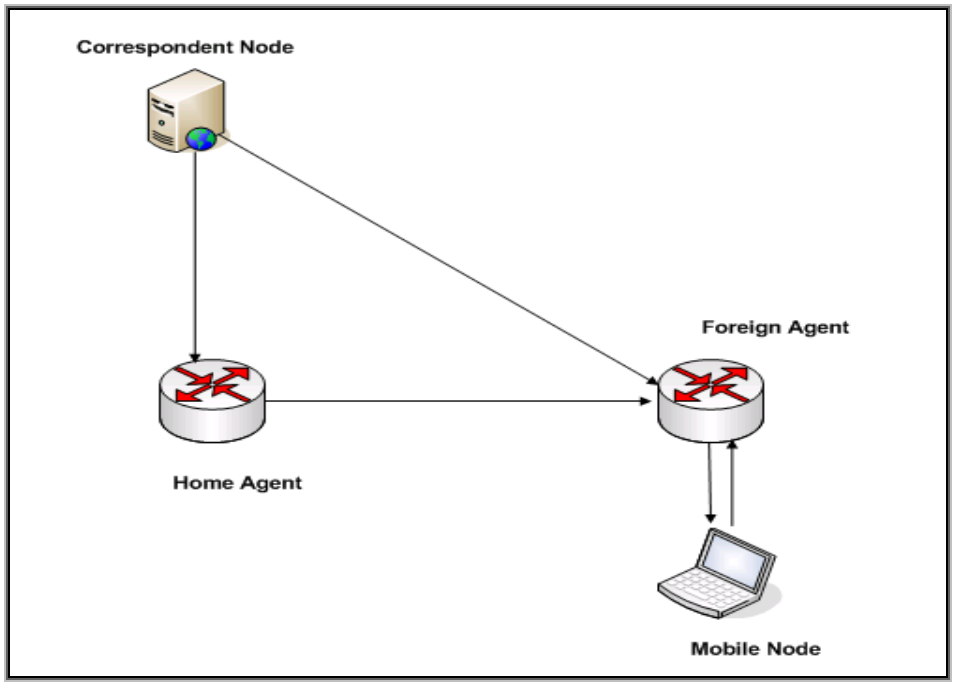

Figure 7.2: Triangle Routing

The reverse transaction would be the same, just in the reverse order. In that case the third party i-e, Internet device would send a request to mobile node, which would be received and forwarded by the home agent. The mobile node would reply back directly to the internet host.

7.3.5 Mobile IP Reverse Tunneling

There may be situations where it is not feasible or desired to have the mobile node send datagrams directly to the network using a router on the foreign network. In this case, an optional feature called reverse tunneling may be deployed, if it is supported by the mobile node, the home agent and if relevant, the foreign agent.

When this is done, a reverse tunnel to complement the normal one is set up between the mobile node and the home agent, or between the foreign agent and the home agent, depending on care-of address type. All transmissions from the mobile node are tunneled back to the home network where the home agent transmits them over the network, resulting in a more symmetric operation rather than the triangle.

One situation where reverse tunneling may be required is if the network where the mobile node is located has implemented certain security measures that prohibit the node from sending datagrams using its normal IP address. In particular, a network may be set up not to allow outgoing datagrams with a source address that doesn't match its network prefix. This is often done to prevent spoofing [4].

References

[1] C. Perkins, "IP Mobility Support for IPv4", Request for Comments 3344, Internet Engineering Task Force, Aug 2002.

[2] The TCP/IP Guide: Version 3.0 by Charles M. Kozierok, 2005 available at http://www.tcpipguide.com/index.html

[3] C. Perkins, "*Mobile Networking through Mobile IP*", IEEE Internet Computing, Jan-Feb, 1998.

[4] G. Montenegro, "Reverse Tunneling for Mobile IP" Request for Comments 3024, Internet Engineering Task Force, Jan 2001.

Chapter 8

IP Traceback for Mobile IP

8.1 DoS Attacks in Mobile IP Networks

DoS attacks in Mobile IP networks can be classified as either ***authentication-based*** attacks or ***flooding-based*** attacks [1].

8.1.1 Authentication-based Attacks

In case of authentication-based attacks, an unauthorized node spoofs the identity of a legitimate MN and redirects the packets destined for MN to other network locations. So a service is denied to the legitimate MN. In current wireless networks, authentication-based attacks form the basis for most DoS attacks. Example of authentication-based attack is ***Man-in -the-Middle*** attack [2], in which a rogue device (devices introduced into the network that are not authorized. Although this could most easily be a simple unauthorized laptop, more interesting for an adversary would be a rogue wireless access point, DHCP server or a switch) authenticates with an access point (AP) as a valid user and then presents itself to other users as a viable AP. When a legitimate user attempts to authenticate with the corporate network via the rogue AP, the rogue node will tunnel the authentication session to the real network and steal the session encryption keys when they are passed between the client and the authentication server. These keys can then be used to authenticate anywhere in the network. In case of Mobile IP networks, authentication-based attack occurs when an unauthorized user manages to do a bogus registration of a new Care-of-Address (CoA) for a particular mobile node. Such a bogus registration gives rise to two problems: disconnection of legitimate mobile node and monitoring its traffic.

8.1.2 Flooding-based Attacks

Flooding-based attacks in Mobile IP networks are similar to flooding attacks in wired IP networks, in which a mobile node acts as a source of attack traffic. The only difference is that, in case of Mobile IP networks, the mobile node can roam among different foreign networks while flooding the victim, due to which it becomes difficult to trace the attacker. A typical example is shown in figure 8.1.

IP traceback techniques can only trace flooding-based attacks in Mobile IP networks. Authentication-based attacks are difficult to trace because the victim

(legitimate MN) has no attack traffic to analyze and carry out traceback. It is however possible to prevent authentication-based attacks by strong authentication on all registration messages which are exchanged during the registration process.

8.2 Extending Our Proposed Technique for Mobile IPv4

In case of Mobile IP networks, two scenarios are of particular importance, which are as follows:

1. When the MN is acting as a source of DoS attack and is residing on the home network.
2. When the MN is acting as a source of DoS attack and residing on the foreign network.

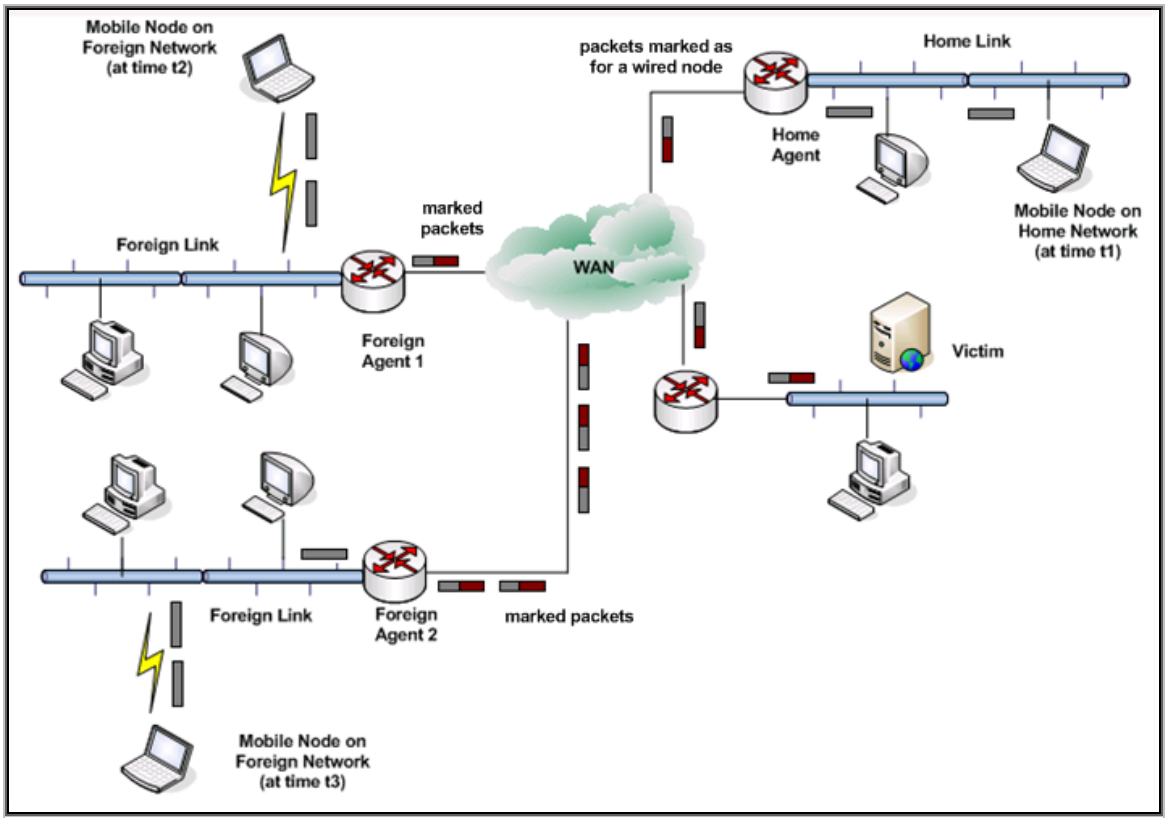

Figure 8.1: Mobile node flooding the victim while roaming among different foreign networks

The first case is not specific to Mobile IPv4. It can be treated as a normal attack for IPv4, since the mobile node operates without mobility services, when residing on the home network. So the traffic of the MN is marked in a similar manner as proposed for a wired node.

The second case is specific to Mobile IPv4. We propose to mark the traffic of mobile node (MN) at the foreign agent (FA). The FA first checks the TTL value. If a match of TTL value is found with the stored table, the packet is marked. But here, the marking is done with home agent (HA) address, rather than CoA. The primary reason for this is that the MN can roam among different foreign networks, while attacking the victim. Marking with HA address is recommended because it remains fixed unlike the CoA which changes at each new foreign network. Thus the victim can trace the attacker even if is not residing on its home network. For marking with HA agent address, the FA must know the MN's HA address. This problem is solved by the visitor list (table 8.1) on the foreign agent. The FA should read the media address of packets and retrieve the corresponding HA address in the visitor list. For marking purposes, the record route field is used in the same manner as proposed for wired networks. After marking the packet, the reserved flag in the IP header should be set to '1'.

A serious problem arises when filtering of packets with invalid source addresses is considered for Mobile IP networks. The MN, when residing on the foreign network, uses its home address (HoA) as source IP address when sending packets to any other node in the internet [3]. So the traffic of MN would be discarded at the foreign network. Since filtering is essential part of out technique as well, we propose conditional filtering at the FA. If the packet is not having a valid source IP address, the source address should be checked in the visitor list. If there is an entry for the source address, the packet should be marked with the HA address, otherwise the packet should be discarded.

Table 8.1: Visitor List on the Foreign Agent

HoA	HA Address	Media Address	Lifetime (sec)
100.10.1.9	100.10.1.0	00-E1-66-95-65-50	200
200.20.2.18	200.20.2.0	00-E1-66-93-76-49	100
----------	----------	----------	----------

Consider the scenario depicted in figure 8.1. If the MN acted as a source of DoS attack when residing on the home network, its traffic would be marked by the HA with the HA address. In this case there is no need to mark the packets again, if the MN moved to a foreign network after some time. The packet is already marked with the HA address. This is where the importance of setting the reserved flag to '1' comes. The packet is marked only if reserved flag is '0', meeting other conditions. The complete marking process, in case of hybrid wired / mobile IP networks, follows the following sequence of actions at the router / HA / FA:

1. ***Check the TTL value in the IP header. If a match of TTL value is found with the stored table, the packet is valid for marking. If match not found, simply forward the packet.***
2. ***Check the Reserved Flag. If its value is '1', forward the packet without marking. If its value is '0', the packet is valid for marking.***
3. ***Check the source IP address. If it is having a valid network id, mark the packet. That is, write IP address of router / HA into first four bytes of route data in Record Route (RR) field.***
4. ***If the source address is not having a valid network id, check the IP address in visitor list (on FA). If a match found, retrieve the corresponding HA address from visitor list. Then, write HA address into first four bytes of route data in RR field.***
5. ***If source IP address is not having a valid network id and there is no entry of the source IP address in the visitor list, simply discard the packet.***

It should be noted that the marking scenario of Mobile IPv4, when reverse tunneling [4] is used, changes to the general IPv4 marking scenario but the packet should be marked before encapsulation.

References

[1] Tarik Taleb et al, "*Securing Hybrid Wired/Mobile IP Networks from TCP-Flooding Based Denial-of-Service Attacks*", IEEE GLOBECOM, 2005, pp 2907-2911.

[2] N. Asokan et al, "*Man-in-the-Middle in Tunneled Authentication Protocols*", In Proc. 11th Int. Workshop on Security Protocols, Cambridge, UK, Apr. 2003.

[3] C. Perkins, "*Mobile Networking through Mobile IP*,"IEEE Internet Computing, Jan-Feb 1998.

[4] G. Montenegro, "*Reverse Tunneling for Mobile IP*" Request for Comments 3024, Internet Engineering Task Force, Jan 2001.

Chapter 9

Conclusion and Future Work

Conclusion

The development of IP traceback techniques is motivated by different DoS attacks in recent years. With the development of Mobile IP, more complex DoS attacks can be launched. However, IP traceback is the first step in identifying the attacker behind the attacks. The effectiveness of any traceback technique depends primarily on its overhead, convergence and the ability to trace any type of DoS attack. The hybrid technique presented here is capable of tracing any type of DoS attack because we can trace even a single packet. It can even trace beyond the corporate firewalls which is a challenge faced by most of the existing traceback techniques. It also eliminates the reflector DDoS attacks. It is particularly designed for networks supporting both wired and mobile nodes. Today there is a need for practical implementation of an effective technique so that IP traceback could be carried out in real time across the internet.

Future work

There is a lack of research on IP traceback for Mobile IPv6. The packet marking technique presented in this thesis is compatible with Mobile IPv4; however it can be extended for Mobile IPv6.

Appendix A

Modified ns-2 Files

Tcl Code

```
#Open the new simulator
set ns [new Simulator]

set nf [open out.nam w]
$ns namtrace-all $nf

#Open the output files
set f0 [open out.tr w]
#$ns trace-all $f0


#Define a 'finish' procedure
proc finish {} {
      global ns nf f0 f
        $ns flush-trace
        close $nf
        exec nam out.nam &
      #Close the output files
      close $f0
      #Call xgraph to display the results
     exec xgraph -x time -y throughput(mbps) out.tr -geometry 800x400 &

#for the RED function
    global tchan_
    set awkCode {
      {
          if ($1 == "Q" && NF>2) {
            print $2, $3 >> "temp.q";
            set end $2
          }
          else if ($1 == "a" && NF>2)
          print $2, $3 >> "temp.a";
      }
    }
    set f [open temp.queue w]
    puts $f "TitleText: red"
    puts $f "Device: Postscript"

    if { [info exists tchan_] } {
      close $tchan_
    }
    exec rm -f temp.q temp.a
    exec touch temp.a temp.q

    exec awk $awkCode all.q

    puts $f \"queue
    exec cat temp.q >@ $f
    puts $f \n\"ave_queue
    exec cat temp.a >@ $f
    close $f
    exec xgraph -bb -tk -x time -y queue temp.queue &
        exit 0
}
```

```
$ns rtproto DV
for {set i 0} {$i < 18} {incr i} {
    set n($i) [$ns node]
}

for {set i 2} {$i < 7} {incr i} {
    $ns duplex-link $n(1) $n($i) 10Mb 10ms DropTail
}

for {set i 7} {$i < 12} {incr i} {
    $ns duplex-link $n(0) $n($i) 10Mb 10ms DropTail
}

for {set i 13} {$i < 18} {incr i} {
    $ns duplex-link $n(12) $n($i) 10Mb 10ms DropTail
}

$ns duplex-link $n(0) $n(1) 10Mb 15ms DropTail
$ns duplex-link $n(0) $n(12) 10Mb 15ms DropTail

set tcp [new Agent/TCP]
$tcp set class_ 1
$ns attach-agent $n(7) $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n(3) $sink

set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.5 "$ftp start"

set tcp1 [new Agent/TCP]
$tcp1 set class_ 2
$ns attach-agent $n(13) $tcp1

set sink1 [new Agent/TCPSink]
$ns attach-agent $n(8) $sink1


set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 1.0 "$ftp1 start"

set tcp2 [new Agent/TCP]
$tcp2 set class_ 3
$ns attach-agent $n(14) $tcp2

set sink2 [new Agent/TCPSink]
$ns attach-agent $n(6) $sink2


set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ns at 10.0 "$ftp2 start"

$ns connect $tcp $sink
```

```
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2

$ns color 1 Blue
$ns color 2 Red
$ns color 3 Black

#Define a procedure which periodically records the bandwidth received
proc record {} {
        global sink f0
      #Get an instance of the simulator
      set ns [Simulator instance]
      #Set the time after which the procedure should be called again
        set time 0.5
      #How many bytes have been received by the traffic sinks?
        set bw0 [$sink set bytes_]
      #Get the current time
        set now [$ns now]
      #Calculate the bandwidth (in MBit/s) and write it to the files
        puts $f0 "$now [expr $bw0/$time*8/261000]"
      #Reset the bytes_ values on the traffic sinks
        $sink set bytes_ 0
      #Re-schedule the procedure
        $ns at [expr $now+$time] "record"
}



#Start logging the received bandwidth
$ns at 0.0 "record"
#Start the traffic sources
$ns at 0.5 "$ftp start"
$ns at 1.0 "$ftp1 start"
$ns at 10.0 "$ftp2 start"


#Stop the traffic sources
$ns at 19.5 "$ftp stop"
$ns at 19.5 "$ftp1 stop"
$ns at 19.5 "$ftp2 stop"


#Call the finish procedure after 60 seconds simulation time
$ns at 20.0 "finish"


#Run the simulation
$ns run


# Tracing a queue
$ns queue-limit $n(1) $n(3) 15
set redq [[$ns link $n(1) $n(3)] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
```

```
$redq attach $tchan_


#Run the simulation
$ns run

#.....................
```

Modified Internal Files of ns-2

----------ip.h----------

```
/* -*-      Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t
-*- */
/*
 * Copyright (c) 1997 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
the
 *    documentation and/or other materials provided with the
distribution.
 * 3. All advertising materials mentioning features or use of this
software
 *    must display the following acknowledgement:
 *    This product includes software developed by the MASH Research
 *    Group at the University of California Berkeley.
 * 4. Neither the name of the University nor of the Research Group may
be
 *    used to endorse or promote products derived from this software
without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS''
AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY
```

```
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
 * SUCH DAMAGE.
 *
 * @(#) $Header: /cvsroot/nsnam/ns-2/common/ip.h,v 1.16 2006/02/22
13:32:23 mahrenho Exp $
 */

/* a network layer; basically like IPv6 */
#ifndef ns_ip_h
#define ns_ip_h

#include "config.h"
#include "packet.h"


#define IP_HDR_LEN      24
#define IP_DEF_TTL      32

// The following undef is to suppress warnings on systems were
// IP_BROADCAST is defined.
#ifdef IP_BROADCAST
#undef IP_BROADCAST
#endif

// #define IP_BROADCAST ((u_int32_t) 0xffffffff)
static const u_int32_t IP_BROADCAST = ((u_int32_t) 0xffffffff);


struct hdr_ip {
      /* common to IPv{4,6} */
      ns_addr_t   src_;
      ns_addr_t   dst_;
      int32_t     ingressrecord_;
      int         ttl_;


      /* Monarch extn */
//    u_int16_t   sport_;
//    u_int16_t   dport_;

      /* IPv6 */
      int         fid_; /* flow id */
      int         prio_;

      static int offset_;
      inline static int& offset() { return offset_; }
      inline static hdr_ip* access(const Packet* p) {
            return (hdr_ip*) p->access(offset_);
      }

      /* per-field member acces functions */
      ns_addr_t& src() { return (src_); }
      nsaddr_t& saddr() { return (src_.addr_); }

        int32_t& sport() { return src_.port_;}
```

```
      ns_addr_t& dst() { return (dst_); }
      nsaddr_t& daddr() { return (dst_.addr_); }

        int32_t& dport() { return dst_.port_;}
      int32_t& ingressrecord() { return (ingressrecord_); }
      int& ttl() { return (ttl_); }
      /* ipv6 fields */
      int& flowid() { return (fid_); }
      int& prio() { return (prio_); }
};

#endif
```

```
----------------------------------ip.cc-----------------------------
/* -*-      Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t
-*- */
/*
 * Copyright (c) 1997 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
the
 *    documentation and/or other materials provided with the
distribution.
 * 3. All advertising materials mentioning features or use of this
software
 *    must display the following acknowledgement:
 *    This product includes software developed by the MASH Research
 *    Group at the University of California Berkeley.
 * 4. Neither the name of the University nor of the Research Group may
be
 *    used to endorse or promote products derived from this software
without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS''
AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
```

```
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
 * SUCH DAMAGE.
 */

#ifndef lint
/*static const char rcsid[] =
    "@(#) $Header: /cvsroot/nsnam/ns-2/common/ip.cc,v 1.8 1998/08/12
23:41:05 gnguyen Exp $";*/
#endif

#include "packet.h"
#include "ip.h"

int hdr_ip::offset_;


static class IPHeaderClass : public PacketHeaderClass {
public:
        IPHeaderClass() : PacketHeaderClass("PacketHeader/IP",
                                  sizeof(hdr_ip)) {
            bind_offset(&hdr_ip::offset_);
      }
      void export_offsets() {
            field_offset("src_", OFFSET(hdr_ip, src_));
            field_offset("dst_", OFFSET(hdr_ip, dst_));
            field_offset("ingressrecord_", OFFSET(hdr_ip,
ingressrecord_));
            field_offset("ttl_", OFFSET(hdr_ip, ttl_));
            field_offset("fid_", OFFSET(hdr_ip, fid_));
            field_offset("prio_", OFFSET(hdr_ip, prio_));
      }
} class_iphdr;
```

```
--------------------------------packet.h-------------------------------
/* -*-      Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t
-*- */
/*
 * Copyright (c) 1997 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
the
 *    documentation and/or other materials provided with the
distribution.
 * 3. All advertising materials mentioning features or use of this
software
 *    must display the following acknowledgement:
```

```
 *    This product includes software developed by the Computer Systems
 *    Engineering Group at Lawrence Berkeley Laboratory.
 * 4. Neither the name of the University nor of the Laboratory may be
used
 *    to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS''
AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
 * SUCH DAMAGE.
 *
 * @(#) $Header: /cvsroot/nsnam/ns-2/common/packet.h,v 1.100 2006/02/22
13:32:23 mahrenho Exp $ (LBL)
 */

#ifndef ns_packet_h
#define ns_packet_h

#include <string.h>
#include <assert.h>

#include "config.h"
#include "scheduler.h"
#include "object.h"
#include "lib/bsd-list.h"
#include "packet-stamp.h"
#include "ns-process.h"

// Used by wireless routing code to attach routing agent
#define RT_PORT         255   /* port that all route msgs are sent to
*/

#define HDR_CMN(p)      (hdr_cmn::access(p))
#define HDR_ARP(p)      (hdr_arp::access(p))
#define HDR_MAC(p)      (hdr_mac::access(p))
#define HDR_MAC802_11(p) ((hdr_mac802_11 *)hdr_mac::access(p))
#define HDR_MAC_TDMA(p) ((hdr_mac_tdma *)hdr_mac::access(p))
#define HDR_SMAC(p)     ((hdr_smac *)hdr_mac::access(p))
#define HDR_LL(p)       (hdr_ll::access(p))
```

```
#define HDR_HDLC(p)     ((hdr_hdlc *)hdr_ll::access(p))
#define HDR_IP(p)       (hdr_ip::access(p))
#define HDR_RTP(p)      (hdr_rtp::access(p))
#define HDR_TCP(p)      (hdr_tcp::access(p))
#define HDR_SCTP(p)     (hdr_sctp::access(p))
#define HDR_SR(p)       (hdr_sr::access(p))
#define HDR_TFRC(p)     (hdr_tfrc::access(p))
#define HDR_TORA(p)     (hdr_tora::access(p))
#define HDR_IMEP(p)     (hdr_imep::access(p))
#define HDR_CDIFF(p)    (hdr_cdiff::access(p))  /* chalermak's
diffusion*/
//#define HDR_DIFF(p)     (hdr_diff::access(p))  /* SCADD's diffusion
ported into ns */
#define HDR_LMS(p)           (hdr_lms::access(p))

/* ----------------------------------------------------------------------
*/

enum packet_t {
      PT_TCP,
      PT_UDP,
      PT_CBR,
      PT_AUDIO,
      PT_VIDEO,
      PT_ACK,
      PT_START,
      PT_STOP,
      PT_PRUNE,
      PT_GRAFT,
      PT_GRAFTACK,
      PT_JOIN,
      PT_ASSERT,
      PT_MESSAGE,
      PT_RTCP,
      PT_RTP,
      PT_RTPROTO_DV,
      PT_CtrMcast_Encap,
      PT_CtrMcast_Decap,
      PT_SRM,
      /* simple signalling messages */
      PT_REQUEST,
      PT_ACCEPT,
      PT_CONFIRM,
      PT_TEARDOWN,
      PT_LIVE,    // packet from live network
      PT_REJECT,

      PT_TELNET,  // not needed: telnet use TCP
      PT_FTP,
      PT_PARETO,
      PT_EXP,
      PT_INVAL,
      PT_HTTP,

      /* new encapsulator */
      PT_ENCAPSULATED,
      PT_MFTP,
```

```
        /* CMU/Monarch's extnsions */
        PT_ARP,
        PT_MAC,
        PT_TORA,
        PT_DSR,
        PT_AODV,
        PT_IMEP,

        // RAP packets
        PT_RAP_DATA,
        PT_RAP_ACK,

        PT_TFRC,
        PT_TFRC_ACK,
        PT_PING,

        // Diffusion packets - Chalermek
        PT_DIFF,

        // LinkState routing update packets
        PT_RTPROTO_LS,

        // MPLS LDP header
        PT_LDP,

        // GAF packet
          PT_GAF,

        // ReadAudio traffic
        PT_REALAUDIO,

        // Pushback Messages
        PT_PUSHBACK,

#ifdef HAVE_STL
        // Pragmatic General Multicast
        PT_PGM,
#endif //STL

        // LMS packets
        PT_LMS,
        PT_LMS_SETUP,

        PT_SCTP,
        PT_SCTP_APP1,

        // SMAC packet
        PT_SMAC,

        // XCP packet
        PT_XCP,

        // HDLC packet
        PT_HDLC,

        // Bell Labs Traffic Trace Type (PackMime OL)
```

```
	PT_BLTRACE,

	// insert new packet types here
	PT_NTYPE // This MUST be the LAST one
};

class p_info {
public:
	p_info() {
		name_[PT_TCP]= "tcp";
		name_[PT_UDP]= "udp";
		name_[PT_CBR]= "cbr";
		name_[PT_AUDIO]= "audio";
		name_[PT_VIDEO]= "video";
		name_[PT_ACK]= "ack";
		name_[PT_START]= "start";
		name_[PT_STOP]= "stop";
		name_[PT_PRUNE]= "prune";
		name_[PT_GRAFT]= "graft";
		name_[PT_GRAFTACK]= "graftAck";
		name_[PT_JOIN]= "join";
		name_[PT_ASSERT]= "assert";
		name_[PT_MESSAGE]= "message";
		name_[PT_RTCP]= "rtcp";
		name_[PT_RTP]= "rtp";
		name_[PT_RTPROTO_DV]= "rtProtoDV";
		name_[PT_CtrMcast_Encap]= "CtrMcast_Encap";
		name_[PT_CtrMcast_Decap]= "CtrMcast_Decap";
		name_[PT_SRM]= "SRM";

		name_[PT_REQUEST]= "sa_req";
		name_[PT_ACCEPT]= "sa_accept";
		name_[PT_CONFIRM]= "sa_conf";
		name_[PT_TEARDOWN]= "sa_teardown";
		name_[PT_LIVE]= "live";
		name_[PT_REJECT]= "sa_reject";

		name_[PT_TELNET]= "telnet";
		name_[PT_FTP]= "ftp";
		name_[PT_PARETO]= "pareto";
		name_[PT_EXP]= "exp";
		name_[PT_INVAL]= "httpInval";
		name_[PT_HTTP]= "http";
		name_[PT_ENCAPSULATED]= "encap";
		name_[PT_MFTP]= "mftp";
		name_[PT_ARP]= "ARP";
		name_[PT_MAC]= "MAC";
		name_[PT_TORA]= "TORA";
		name_[PT_DSR]= "DSR";
		name_[PT_AODV]= "AODV";
		name_[PT_IMEP]= "IMEP";

		name_[PT_RAP_DATA] = "rap_data";
		name_[PT_RAP_ACK] = "rap_ack";

		name_[PT_TFRC]= "tcpFriend";
		name_[PT_TFRC_ACK]= "tcpFriendCtl";
```

```
            name_[PT_PING]="ping";

            /* For diffusion : Chalermek */
            name_[PT_DIFF] = "diffusion";

            // Link state routing updates
            name_[PT_RTPROTO_LS] = "rtProtoLS";

            // MPLS LDP packets
            name_[PT_LDP] = "LDP";

            // for GAF
                name_[PT_GAF] = "gaf";

            // RealAudio packets
            name_[PT_REALAUDIO] = "ra";

            //pushback
            name_[PT_PUSHBACK] = "pushback";

#ifdef HAVE_STL
            // for PGM
            name_[PT_PGM] = "PGM";
#endif //STL

            // LMS entries
            name_[PT_LMS]="LMS";
            name_[PT_LMS_SETUP]="LMS_SETUP";

            name_[PT_SCTP]= "sctp";
            name_[PT_SCTP_APP1] = "sctp_app1";

            // smac
            name_[PT_SMAC]="smac";

            // HDLC
            name_[PT_HDLC]="HDLC";

            // XCP
            name_[PT_XCP]="xcp";

            // Bell Labs (PackMime OL)
            name_[PT_BLTRACE]="BellLabsTrace";

            name_[PT_NTYPE]= "undefined";
      }
      const char* name(packet_t p) const {
            if ( p <= PT_NTYPE ) return name_[p];
            return 0;
      }
      static bool data_packet(packet_t type) {
            return ( (type) == PT_TCP || \
                  (type) == PT_TELNET || \
                  (type) == PT_CBR || \
                  (type) == PT_AUDIO || \
                  (type) == PT_VIDEO || \
                  (type) == PT_ACK || \
```

```
                  (type) == PT_SCTP || \
                  (type) == PT_SCTP_APP1 || \
                  (type) == PT_HDLC \
                 );
      }
private:
      static char* name_[PT_NTYPE+1];
};
extern p_info packet_info; /* map PT_* to string name */
//extern char* p_info::name_[];


#define DATA_PACKET(type) ( (type) == PT_TCP || \
                            (type) == PT_TELNET || \
                            (type) == PT_CBR || \
                            (type) == PT_AUDIO || \
                            (type) == PT_VIDEO || \
                            (type) == PT_ACK || \
                            (type) == PT_SCTP || \
                            (type) == PT_SCTP_APP1 \
                            )


//#define OFFSET(type, field) ((long) &((type *)0)->field)
#define OFFSET(type, field) ( (char *)&( ((type *)256)->field )  -
(char *)256)

class PacketData : public AppData {
public:
      PacketData(int sz) : AppData(PACKET_DATA) {
            datalen_ = sz;
            if (datalen_ > 0)
                  data_ = new unsigned char[datalen_];
            else
                  data_ = NULL;
      }
      PacketData(PacketData& d) : AppData(d) {
            datalen_ = d.datalen_;
            if (datalen_ > 0) {
                  data_ = new unsigned char[datalen_];
                  memcpy(data_, d.data_, datalen_);
            } else
                  data_ = NULL;
      }
      virtual ~PacketData() {
            if (data_ != NULL)
                  delete []data_;
      }
      unsigned char* data() { return data_; }

      virtual int size() const { return datalen_; }
      virtual AppData* copy() { return new PacketData(*this); }
private:
      unsigned char* data_;
      int datalen_;
};
```

```
//Monarch ext
typedef void (*FailureCallback)(Packet *,void *);

class Packet : public Event {
private:
      unsigned char* bits_;   // header bits
//    unsigned char* data_;   // variable size buffer for 'data'
//    unsigned int datalen_;  // length of variable size buffer
      AppData* data_;         // variable size buffer for 'data'
      static void init(Packet*);     // initialize pkt hdr
      bool fflag_;
protected:
      static Packet* free_;   // packet free list
      int   ref_count_; // free the pkt until count to 0
public:
      Packet* next_;          // for queues and the free list
      static int hdrlen_;

      Packet() : bits_(0), data_(0), ref_count_(0), next_(0) { }
      inline unsigned char* const bits() { return (bits_); }
      inline Packet* copy() const;
      inline Packet* refcopy() { ++ref_count_; return this; }
      inline int& ref_count() { return (ref_count_); }
      static inline Packet* alloc();
      static inline Packet* alloc(int);
      inline void allocdata(int);
      // dirty hack for diffusion data
      inline void initdata() { data_  = 0;}
      static inline void free(Packet*);
      inline unsigned char* access(int off) const {
            if (off < 0)
                  abort();
            return (&bits_[off]);
      }
      // This is used for backward compatibility, i.e., assuming user
data
      // is PacketData and return its pointer.
      inline unsigned char* accessdata() const {
            if (data_ == 0)
                  return 0;
            assert(data_->type() == PACKET_DATA);
            return (((PacketData*)data_)->data());
      }
      // This is used to access application-specific data, not limited
      // to PacketData.
      inline AppData* userdata() const {
            return data_;
      }
      inline void setdata(AppData* d) {
            if (data_ != NULL)
                  delete data_;
            data_ = d;
      }
      inline int datalen() const { return data_ ? data_->size() : 0; }

      // Monarch extn
```

```
      static void dump_header(Packet *p, int offset, int length);

      // the pkt stamp carries all info about how/where the pkt
        // was sent needed for a receiver to determine if it correctly
        // receives the pkt
        PacketStamp     txinfo_;

      /*
         * According to cmu code:
       * This flag is set by the MAC layer on an incoming packet
         * and is cleared by the link layer.  It is an ugly hack, but
         * there's really no other way because NS always calls
         * the recv() function of an object.
       *
         */
        u_int8_t        incoming;

      //monarch extns end;
};

/*
 * static constant associations between interface special (negative)
 * values and their c-string representations that are used from tcl
 */
class iface_literal {
public:
      enum iface_constant {
            UNKN_IFACE= -1, /*
                             * iface value for locally originated packets
                             */
            ANY_IFACE= -2   /*
                             * hashnode with iif == ANY_IFACE_
                             * matches any pkt iface (imported from TCL);
                             * this value should be different from
                             * hdr_cmn::UNKN_IFACE (packet.h)
                             */
      };
      iface_literal(const iface_constant i, const char * const n) :
            value_(i), name_(n) {}
      inline int value() const { return value_; }
      inline const char * const name() const { return name_; }
private:
      const iface_constant value_;
      /* strings used in TCL to access those special values */
      const char * const name_;
};

static const iface_literal UNKN_IFACE(iface_literal::UNKN_IFACE, "?");
static const iface_literal ANY_IFACE(iface_literal::ANY_IFACE, "*");

/*
 * Note that NS_AF_* doesn't necessarily correspond with
 * the constants used in your system (because many
 * systems don't have NONE or ILINK).
 */
enum ns_af_enum { NS_AF_NONE, NS_AF_ILINK, NS_AF_INET };
```

```
struct hdr_cmn {
      enum dir_t { DOWN= -1, NONE= 0, UP= 1 };
      packet_t ptype_;  // packet type (see above)
      int   size_;            // simulated packet size
      int   uid_;       // unique id
      int   error_;           // error flag
      int     errbitcnt_;     // # of corrupted bits jahn
      int     fecsize_;
      double      ts_;        // timestamp: for q-delay measurement
      int   iface_;           // receiving interface (label)
      dir_t direction_; // direction: 0=none, 1=up, -1=down
      // source routing
        char src_rt_valid;
      double ts_arr_; // Required by Marker of JOBS

      //Monarch extn begins
      nsaddr_t prev_hop_;     // IP addr of forwarding hop
      nsaddr_t next_hop_;     // next hop for this packet
      int      addr_type_;    // type of next_hop_ addr
      nsaddr_t last_hop_;     // for tracing on multi-user channels

        // called if pkt can't obtain media or isn't ack'd. not called
if
        // droped by a queue
        FailureCallback xmit_failure_;
        void *xmit_failure_data_;

        /*
         * MONARCH wants to know if the MAC layer is passing this back
because
         * it could not get the RTS through or because it did not
receive
         * an ACK.
         */
        int     xmit_reason_;
#define XMIT_REASON_RTS 0x01
#define XMIT_REASON_ACK 0x02

        // filled in by GOD on first transmission, used for trace
analysis
        int num_forwards_;    // how many times this pkt was forwarded
        int opt_num_forwards_;   // optimal #forwards
      // Monarch extn ends;

      // tx time for this packet in sec
      double txtime_;
      inline double& txtime() { return(txtime_); }

      static int offset_;     // offset for this header
      inline static int& offset() { return offset_; }
      inline static hdr_cmn* access(const Packet* p) {
            return (hdr_cmn*) p->access(offset_);
      }

        /* per-field member functions */
      inline packet_t& ptype() { return (ptype_); }
      inline int& size() { return (size_); }
```

```
        inline int& uid() { return (uid_); }
        inline int& error() { return error_; }
        inline int& errbitcnt() {return errbitcnt_; }
        inline int& fecsize() {return fecsize_; }
        inline double& timestamp() { return (ts_); }
        inline int& iface() { return (iface_); }
        inline dir_t& direction() { return (direction_); }
        // monarch_begin
        inline nsaddr_t& next_hop() { return (next_hop_); }
        inline int& addr_type() { return (addr_type_); }
        inline int& num_forwards() { return (num_forwards_); }
        inline int& opt_num_forwards() { return (opt_num_forwards_); }
          //monarch_end
};


class PacketHeaderClass : public TclClass {
protected:
        PacketHeaderClass(const char* classname, int hdrsize);
        virtual int method(int argc, const char*const* argv);
        void field_offset(const char* fieldname, int offset);
        inline void bind_offset(int* off) { offset_ = off; }
        inline void offset(int* off) {offset_= off;}
        int hdrlen_;            // # of bytes for this header
        int* offset_;           // offset for this header
public:
        virtual void bind();
        virtual void export_offsets();
        TclObject* create(int argc, const char*const* argv);
};


inline void Packet::init(Packet* p)
{
        bzero(p->bits_, hdrlen_);
}

inline Packet* Packet::alloc()
{
        Packet* p = free_;
        if (p != 0) {
                assert(p->fflag_ == FALSE);
                free_ = p->next_;
                assert(p->data_ == 0);
                p->uid_ = 0;
                p->time_ = 0;
        } else {
                p = new Packet;
                p->bits_ = new unsigned char[hdrlen_];
                if (p == 0 || p->bits_ == 0)
                        abort();
        }
        init(p); // Initialize bits_[]
        (HDR_CMN(p))->next_hop_ = -2; // -1 reserved for IP_BROADCAST
        (HDR_CMN(p))->last_hop_ = -2; // -1 reserved for IP_BROADCAST
        p->fflag_ = TRUE;
        (HDR_CMN(p))->direction() = hdr_cmn::DOWN;
```

```
	/* setting all direction of pkts to be downward as default;
	   until channel changes it to +1 (upward) */
	p->next_ = 0;
	return (p);
}

/*
 * Allocate an n byte data buffer to an existing packet
 *
 * To set application-specific AppData, use Packet::setdata()
 */
inline void Packet::allocdata(int n)
{
	assert(data_ == 0);
	data_ = new PacketData(n);
	if (data_ == 0)
		abort();
}

/* allocate a packet with an n byte data buffer */
inline Packet* Packet::alloc(int n)
{
	Packet* p = alloc();
	if (n > 0)
		p->allocdata(n);
	return (p);
}


inline void Packet::free(Packet* p)
{
	if (p->fflag_) {
		if (p->ref_count_ == 0) {
			/*
			 * A packet's uid may be < 0 (out of a event queue),
or
			 * == 0 (newed but never gets into the event queue.
			 */
			assert(p->uid_ <= 0);
			// Delete user data because we won't need it any
more.
			if (p->data_ != 0) {
				delete p->data_;
				p->data_ = 0;
			}
			init(p);
			p->next_ = free_;
			free_ = p;
			p->fflag_ = FALSE;
		} else {
			--p->ref_count_;
		}
	}
}

inline Packet* Packet::copy() const
{
```

```
      Packet* p = alloc();
      memcpy(p->bits(), bits_, hdrlen_);
      if (data_)
            p->data_ = data_->copy();
      p->txinfo_.init(&txinfo_);

      return (p);
}

inline void
Packet::dump_header(Packet *p, int offset, int length)
{
        assert(offset + length <= p->hdrlen_);
        struct hdr_cmn *ch = HDR_CMN(p);

        fprintf(stderr, "\nPacket ID: %d\n", ch->uid());

        for(int i = 0; i < length ; i+=16) {
                fprintf(stderr, "%02x %02x %02x %02x %02x %02x %02x
%02x %02x %02x %02x %02x %02x %02x %02x %02x\n",
                        p->bits_[offset + i],     p->bits_[offset + i +
1],
                        p->bits_[offset + i + 2], p->bits_[offset + i +
3],
                        p->bits_[offset + i + 4], p->bits_[offset + i +
5],
                        p->bits_[offset + i + 6], p->bits_[offset + i +
7],
                        p->bits_[offset + i + 8], p->bits_[offset + i +
9],
                        p->bits_[offset + i + 10], p->bits_[offset + i
+ 11],
                        p->bits_[offset + i + 12], p->bits_[offset + i
+ 13],
                        p->bits_[offset + i + 14], p->bits_[offset + i
+ 15]);
        }
}

#endif
```

```
----------------------------packet.cc--------------------------------

/* -*-      Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t
-*- */
/*
 * Copyright (c) 1994-1997 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
```

```
 *    notice, this list of conditions and the following disclaimer in
the
 *    documentation and/or other materials provided with the
distribution.
 * 3. All advertising materials mentioning features or use of this
software
 *    must display the following acknowledgement:
 *    This product includes software developed by the Computer Systems
 *    Engineering Group at Lawrence Berkeley Laboratory.
 * 4. Neither the name of the University nor of the Laboratory may be
used
 *    to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS''
AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
 * SUCH DAMAGE.
 */

#ifndef lint
static const char rcsid[] =
    "@(#) $Header: /cvsroot/nsnam/ns-2/common/packet.cc,v 1.18
2000/09/15 20:32:57 haoboy Exp $ (LBL)";
#endif

#include "packet.h"
#include "flags.h"

p_info packet_info;
char* p_info::name_[PT_NTYPE+1];

int Packet::hdrlen_ = 0;            // size of a packet's header
Packet* Packet::free_;              // free list
int hdr_cmn::offset_;               // static offset of common header
int hdr_flags::offset_;             // static offset of flags header


PacketHeaderClass::PacketHeaderClass(const char* classname, int hdrlen)
:
```

```
      TclClass(classname), hdrlen_(hdrlen), offset_(0)
{
}


TclObject* PacketHeaderClass::create(int, const char*const*)
{
      return (0);
}

void PacketHeaderClass::bind()
{
      TclClass::bind();
      Tcl& tcl = Tcl::instance();
      tcl.evalf("%s set hdrlen_ %d", classname_, hdrlen_);
      export_offsets();
      add_method("offset");
}

void PacketHeaderClass::export_offsets()
{
}

void PacketHeaderClass::field_offset(const char* fieldname, int offset)
{
      Tcl& tcl = Tcl::instance();
      tcl.evalf("%s set offset_(%s) %d", classname_, fieldname,
offset);
}

int PacketHeaderClass::method(int ac, const char*const* av)
{
      Tcl& tcl = Tcl::instance();
      int argc = ac - 2;
      const char*const* argv = av + 2;
      if (argc == 3) {
            if (strcmp(argv[1], "offset") == 0) {
                  if (offset_) {
                        *offset_ = atoi(argv[2]);
                        return TCL_OK;
                  }
                  tcl.resultf("Warning: cannot set offset_ for %s",
                              classname_);
                  return TCL_OK;
            }
      }
      else if (argc == 2) {
            if (strcmp(argv[1], "offset") == 0) {
                  if (offset_) {
                        tcl.resultf("%d", *offset_);
                        return TCL_OK;
                  }
            }
      }
      return TclClass::method(ac, av);
}
```

```
class CommonHeaderClass : public PacketHeaderClass {
public:
      CommonHeaderClass() : PacketHeaderClass("PacketHeader/Common",
                                    sizeof(hdr_cmn)) {
            bind_offset(&hdr_cmn::offset_);
      }
      void export_offsets() {
            field_offset("ptype_", OFFSET(hdr_cmn, ptype_));
            field_offset("size_", OFFSET(hdr_cmn, size_));
            field_offset("uid_", OFFSET(hdr_cmn, uid_));
            field_offset("error_", OFFSET(hdr_cmn, error_));
      };
} class_cmnhdr;

class FlagsHeaderClass : public PacketHeaderClass {
public:
      FlagsHeaderClass() : PacketHeaderClass("PacketHeader/Flags",
                                    sizeof(hdr_flags)) {
            bind_offset(&hdr_flags::offset_);
      }
} class_flagshdr;


/* manages active packet header types */
class PacketHeaderManager : public TclObject {
public:
      PacketHeaderManager() {
            bind("hdrlen_", &Packet::hdrlen_);
      }
};

static class PacketHeaderManagerClass : public TclClass {
public:
      PacketHeaderManagerClass() : TclClass("PacketHeaderManager") {}
      TclObject* create(int, const char*const*) {
            return (new PacketHeaderManager);
      }
} class_packethdr_mgr;
```

```
-----------------------------------ttl.cc------------------------------
/* -*-      Mode:C++; c-basic-offset:8; tab-width:8; indent-tabs-mode:t
-*- */
/*
 * Copyright (c) 1996-1997 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
the
```

```
 *    documentation and/or other materials provided with the
distribution.
 * 3. All advertising materials mentioning features or use of this
software
 *    must display the following acknowledgement:
 *    This product includes software developed by the MASH Research
 *    Group at the University of California Berkeley.
 * 4. Neither the name of the University nor of the Research Group may
be
 *    used to endorse or promote products derived from this software
without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS''
AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE
LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
 * SUCH DAMAGE.
 */

#ifndef lint
static const char rcsid[] =
    "@(#) $Header: /cvsroot/nsnam/ns-2/common/ttl.cc,v 1.11 1998/08/12
23:41:24 gnguyen Exp $";
#endif

#include "packet.h"
#include "ip.h"
#include "connector.h"
#include "ip.cc"

class TTLChecker : public Connector {
public:
      TTLChecker() : noWarn_(1), tick_(1) {}
      int command(int argc, const char*const* argv) {
            if (argc == 3) {
                  if (strcmp(argv[1], "warning") == 0) {
                        noWarn_ = ! atoi(argv[2]);
                        return TCL_OK;
                  }
                  if (strcmp(argv[1], "tick") == 0) {
```

```
                        int tick = atoi(argv[2]);
                        if (tick > 0) {
                              tick_ = tick;
                              return TCL_OK;
                        } else {
                              Tcl& tcl = Tcl::instance();
                              tcl.resultf("%s: TTL must be positive
(specified = %d)\n",
                                          name(), tick);
                              return TCL_ERROR;
                        }
                  }
            }
            return Connector::command(argc, argv);
      }
      void recv(Packet* p, Handler* h) {
            hdr_ip* iph = hdr_ip::access(p);
            int ttl = iph->ttl() - tick_;
            if (ttl <= 0) {
                  /* XXX should send to a drop object.*/
                  // Yes, and now it does...
                  // Packet::free(p);
                  if (! noWarn_)
                        printf("ttl exceeded\n");
                  drop(p);
                  return;
            }
            int
table[18]={128,128,60,64,64,64,64,128,60,60,60,255,60,64,64,64,255,32};
            iph->ttl() = ttl;
            for (int i=0;i<18;i++)
            {
            if (ttl == table[i]-1)
            {iph->ingressrecord() = nodeid_;

            }}
            send(p, h);
      }
protected:
      int noWarn_;
      int tick_;
};

static class TTLCheckerClass : public TclClass {
public:
      TTLCheckerClass() : TclClass("TTLChecker") {}
      TclObject* create(int, const char*const*) {
            return (new TTLChecker);
      }
} ttl_checker_class;


class SessionTTLChecker : public Connector {
public:
      SessionTTLChecker() {}
      int command(int argc, const char*const* argv);
      void recv(Packet* p, Handler* h) {
```

```
            hdr_ip* iph = hdr_ip::access(p);
            int ttl = iph->ttl() - tick_;
            if (ttl <= 0) {
                  /* XXX should send to a drop object.*/
                  // Yes, and now it does...
                  // Packet::free(p);
                  printf("ttl exceeded\n");
                  drop(p);
                  return;
            }
            iph->ttl() = ttl;
            int
table[18]={128,128,60,64,64,64,64,128,60,60,60,255,60,64,64,64,255,32};
            iph->ttl() = ttl;
            for (int i=0;i<18;i++)
            {
            if (ttl == table[i]-1)
            {iph->ingressrecord() = nodeid_;
                        }}
            send(p, h);
      }
protected:
      int tick_;
};

static class SessionTTLCheckerClass : public TclClass {
public:
      SessionTTLCheckerClass() : TclClass("TTLChecker/Session") {}
      TclObject* create(int, const char*const*) {
            return (new SessionTTLChecker);
      }
} session_ttl_checker_class;

int SessionTTLChecker::command(int argc, const char*const* argv)
{
      if (argc == 3) {
            if (strcmp(argv[1], "tick") == 0) {
                  tick_ = atoi(argv[2]);
                  return (TCL_OK);
            }
      }
      return (Connector::command(argc, argv));
}
```

A Small Portion of Trace File

```
+ 0.408238 7 0 rtProtoDV 18 ------- 0 7.2 0.1 -1 104
- 0.408238 7 0 rtProtoDV 18 ------- 0 7.2 0.1 -1 104
r 0.409722 5 1 rtProtoDV 18 ------- 0 5.1 1.1 -1 103
+ 0.418162 11 0 rtProtoDV 18 ------- 0 11.1 0.1 -1 105
- 0.418162 11 0 rtProtoDV 18 ------- 0 11.1 0.1 -1 105
r 0.418382 7 0 rtProtoDV 18 ------- 0 7.2 0.1 -1 104
r 0.428306 11 0 rtProtoDV 18 ------- 0 11.1 0.1 -1 105
+ 0.429864 16 12 rtProtoDV 18 ------- 0 16.1 12.1 -1 106
- 0.429864 16 12 rtProtoDV 18 ------- 0 16.1 12.1 -1 106
```

```
r 0.440008 16 12 rtProtoDV 18 ------- 0 16.1 12.1 -1 106
+ 0.5 7 0 tcp 40 ------- 1 7.0 3.0 0 107
- 0.5 7 0 tcp 40 ------- 1 7.0 3.0 0 107
0.5 0.0
r 0.51032 7 0 tcp 40 ------- 1 7.0 3.0 0 107
+ 0.51032 0 1 tcp 40 ------- 1 7.0 3.0 0 107
- 0.51032 0 1 tcp 40 ------- 1 7.0 3.0 0 107
r 0.52596 0 1 tcp 40 ------- 1 7.0 3.0 0 107
+ 0.52596 1 3 tcp 40 ------- 1 7.0 3.0 0 107
- 0.52596 1 3 tcp 40 ------- 1 7.0 3.0 0 107
r 0.53628 1 3 tcp 40 ------- 1 7.0 3.0 0 107
+ 0.53628 3 1 ack 40 ------- 1 3.0 7.0 0 108
- 0.53628 3 1 ack 40 ------- 1 3.0 7.0 0 108
r 0.5466 3 1 ack 40 ------- 1 3.0 7.0 0 108
+ 0.5466 1 0 ack 40 ------- 1 3.0 7.0 0 108
- 0.5466 1 0 ack 40 ------- 1 3.0 7.0 0 108
r 0.56224 1 0 ack 40 ------- 1 3.0 7.0 0 108
+ 0.56224 0 7 ack 40 ------- 1 3.0 7.0 0 108
- 0.56224 0 7 ack 40 ------- 1 3.0 7.0 0 108
r 0.57256 0 7 ack 40 ------- 1 3.0 7.0 0 108
+ 0.57256 7 0 tcp 1040 ------- 1 7.0 3.0 1 109
- 0.57256 7 0 tcp 1040 ------- 1 7.0 3.0 1 109
+ 0.57256 7 0 tcp 1040 ------- 1 7.0 3.0 2 110
- 0.58088 7 0 tcp 1040 ------- 1 7.0 3.0 2 110
r 0.59088 7 0 tcp 1040 ------- 1 7.0 3.0 1 109
+ 0.59088 0 1 tcp 1040 ------- 1 7.0 3.0 1 109
- 0.59088 0 1 tcp 1040 ------- 1 7.0 3.0 1 109
r 0.5992 7 0 tcp 1040 ------- 1 7.0 3.0 2 110
+ 0.5992 0 1 tcp 1040 ------- 1 7.0 3.0 2 110
- 0.60752 0 1 tcp 1040 ------- 1 7.0 3.0 2 110
r 0.62252 0 1 tcp 1040 ------- 1 7.0 3.0 1 109
+ 0.62252 1 3 tcp 1040 ------- 1 7.0 3.0 1 109
- 0.62252 1 3 tcp 1040 ------- 1 7.0 3.0 1 109
r 0.63916 0 1 tcp 1040 ------- 1 7.0 3.0 2 110
+ 0.63916 1 3 tcp 1040 ------- 1 7.0 3.0 2 110
- 0.63916 1 3 tcp 1040 ------- 1 7.0 3.0 2 110
r 0.64084 1 3 tcp 1040 ------- 1 7.0 3.0 1 109
+ 0.64084 3 1 ack 40 ------- 1 3.0 7.0 1 111
- 0.64084 3 1 ack 40 ------- 1 3.0 7.0 1 111
r 0.65116 3 1 ack 40 ------- 1 3.0 7.0 1 111
+ 0.65116 1 0 ack 40 ------- 1 3.0 7.0 1 111
- 0.65116 1 0 ack 40 ------- 1 3.0 7.0 1 111
r 0.65748 1 3 tcp 1040 ------- 1 7.0 3.0 2 110
+ 0.65748 3 1 ack 40 ------- 1 3.0 7.0 2 112
- 0.65748 3 1 ack 40 ------- 1 3.0 7.0 2 112
r 0.6668 1 0 ack 40 ------- 1 3.0 7.0 1 111
+ 0.6668 0 7 ack 40 ------- 1 3.0 7.0 1 111
- 0.6668 0 7 ack 40 ------- 1 3.0 7.0 1 111
r 0.6678 3 1 ack 40 ------- 1 3.0 7.0 2 112
+ 0.6678 1 0 ack 40 ------- 1 3.0 7.0 2 112
- 0.6678 1 0 ack 40 ------- 1 3.0 7.0 2 112
r 0.67712 0 7 ack 40 ------- 1 3.0 7.0 1 111
+ 0.67712 7 0 tcp 1040 ------- 1 7.0 3.0 3 113
- 0.67712 7 0 tcp 1040 ------- 1 7.0 3.0 3 113
+ 0.67712 7 0 tcp 1040 ------- 1 7.0 3.0 4 114
r 0.68344 1 0 ack 40 ------- 1 3.0 7.0 2 112
+ 0.68344 0 7 ack 40 ------- 1 3.0 7.0 2 112
```

```
- 0.68344 0 7 ack 40 ------- 1 3.0 7.0 2 112
- 0.68544 7 0 tcp 1040 ------- 1 7.0 3.0 4 114
r 0.69376 0 7 ack 40 ------- 1 3.0 7.0 2 112
+ 0.69376 7 0 tcp 1040 ------- 1 7.0 3.0 5 115
+ 0.69376 7 0 tcp 1040 ------- 1 7.0 3.0 6 116
- 0.69376 7 0 tcp 1040 ------- 1 7.0 3.0 5 115
r 0.69544 7 0 tcp 1040 ------- 1 7.0 3.0 3 113
+ 0.69544 0 1 tcp 1040 ------- 1 7.0 3.0 3 113
- 0.69544 0 1 tcp 1040 ------- 1 7.0 3.0 3 113
- 0.70208 7 0 tcp 1040 ------- 1 7.0 3.0 6 116
r 0.70376 7 0 tcp 1040 ------- 1 7.0 3.0 4 114
+ 0.70376 0 1 tcp 1040 ------- 1 7.0 3.0 4 114
r 0.71208 7 0 tcp 1040 ------- 1 7.0 3.0 5 115
+ 0.71208 0 1 tcp 1040 ------- 1 7.0 3.0 5 115
- 0.71208 0 1 tcp 1040 ------- 1 7.0 3.0 4 114
r 0.7204 7 0 tcp 1040 ------- 1 7.0 3.0 6 116
+ 0.7204 0 1 tcp 1040 ------- 1 7.0 3.0 6 116
r 0.72708 0 1 tcp 1040 ------- 1 7.0 3.0 3 113
+ 0.72708 1 3 tcp 1040 ------- 1 7.0 3.0 3 113
- 0.72708 1 3 tcp 1040 ------- 1 7.0 3.0 3 113
- 0.72872 0 1 tcp 1040 ------- 1 7.0 3.0 5 115
r 0.74372 0 1 tcp 1040 ------- 1 7.0 3.0 4 114
+ 0.74372 1 3 tcp 1040 ------- 1 7.0 3.0 4 114
- 0.74372 1 3 tcp 1040 ------- 1 7.0 3.0 4 114
- 0.74536 0 1 tcp 1040 ------- 1 7.0 3.0 6 116
r 0.7454 1 3 tcp 1040 ------- 1 7.0 3.0 3 113
+ 0.7454 3 1 ack 40 ------- 1 3.0 7.0 3 117
- 0.7454 3 1 ack 40 ------- 1 3.0 7.0 3 117
r 0.75572 3 1 ack 40 ------- 1 3.0 7.0 3 117
+ 0.75572 1 0 ack 40 ------- 1 3.0 7.0 3 117
- 0.75572 1 0 ack 40 ------- 1 3.0 7.0 3 117
r 0.76036 0 1 tcp 1040 ------- 1 7.0 3.0 5 115
```

Appendix B

NS-2 Tutorial

General Structure and Architecture of NS

NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkeley written in C++ and OTcl. NS is primarily useful for simulating local and wide area networks.

NS is an event driven network simulator developed at UC Berkeley that simulates variety of IP networks. It implements network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, CBR and router queue management mechanism such as Drop Tail, RED, routing algorithms such as Dijkstra, and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations. The NS project is now a part of the VINT project that develops tools for simulation results display, analysis and converters that convert network topologies generated by well-known generators to NS formats. Currently, NS (version 2) written in C++ and OTcl (Tcl script language with Object-oriented extensions developed at MIT) is available.

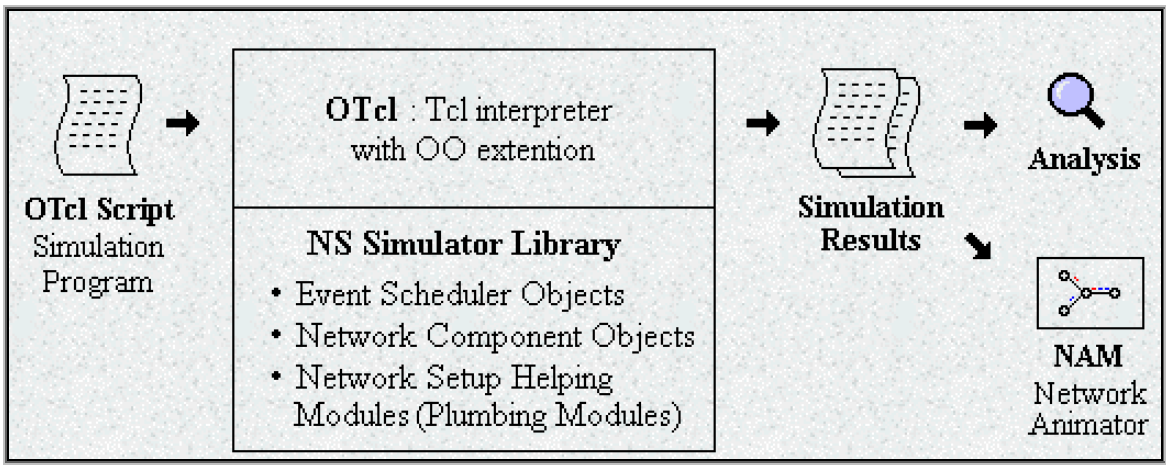

Figure B.1: Simplified User's View of NS

NS is Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network setup (plumbing) module libraries (actually, plumbing modules are implemented as member functions of the base simulator object). In other words, to use NS, we program in OTcl script language. To setup and run a simulation network, a user should write an OTcl script that initiates an

event scheduler, sets up the network topology using the network objects and the plumbing functions in the library, and tells traffic sources when to start and stop transmitting packets through the event scheduler. The term plumbing is used for a network setup, because setting up a network is plumbing possible data paths among network objects by setting the neighbor pointer of an object to the address of an appropriate object. When a user wants to make a new network object, he or she can easily make an object either by writing a new object or by making a compound object from the object library, and plumb the data path through the object. This may sound like complicated job, but the plumbing OTcl modules actually make the job very easy.

The power of NS comes from this plumbing. Another major component of NS beside network objects is the event scheduler. An event in NS is a packet ID that is unique for a packet with scheduled time and the pointer to an object that handles the event. In NS, an event scheduler keeps track of simulation time and fires all the events in the event queue scheduled for the current time by invoking appropriate network components, which usually are the ones who issued the events, and let them do the appropriate action associated with packet pointed by the event. Network components communicate with one another passing packets; however this does not consume actual simulation time. All the network components that need to spend some simulation time handling a packet (i.e. need a delay) use the event scheduler by issuing an event for the packet and waiting for the event to be fired to itself before doing further action handling the packet. For example, a network switch component that simulates a switch with 20 microseconds of switching delay issues an event for a packet to be switched to the scheduler as an event 20 microsecond later. The scheduler after 20 microseconds dequeues the event and fires it to the switch component, which then passes the packet to an appropriate output link component. Another use of an event scheduler is timer. For example, TCP needs a timer to keep track of a packet transmission time out for retransmission (transmission of a packet with the same TCP packet number but different NS packet ID). Timers use event schedulers in a similar manner that delay does. The only difference is that timer measures a time value associated with a packet and does an appropriate action related to that packet after a certain time goes by, and does not simulate a delay.

NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object. In this way, the controls of the C++ objects are given to OTcl. It is also possible to add member functions and variables to a C++ linked OTcl object.

The objects in C++ that do not need to be controlled in a simulation or internally used by another object do not need to be linked to OTcl. Likewise, an object (not in the data path) can be entirely implemented in OTcl. Figure 2 shows an object hierarchy example in C++ and OTcl. One thing to note in the figure is that for C++ objects that have an OTcl linkage forming a hierarchy, there is a matching OTcl object hierarchy very similar to that of C++.

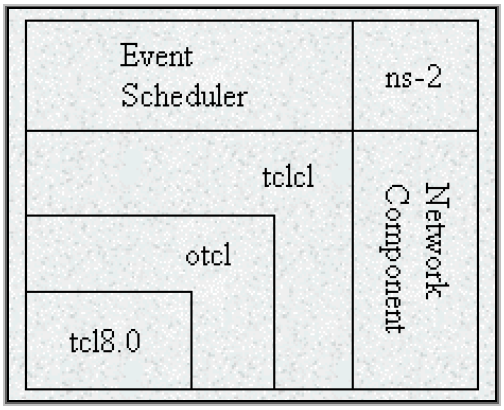

Figure B.2: Architectural View of NS

In the figure above, a general user (not an NS developer) can be thought of standing at the left bottom corner, designing and running simulations in Tcl using the simulator objects in the OTcl library. The event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using tclcl. The whole thing together makes NS, which is an object oriented extended Tcl interpreter with network simulator libraries.

At this point it's important to know how to obtain NS simulation results. When a simulation is finished, NS produces one or more text-based output files that contain detailed simulation data, if specified to do so in the input Tcl (or more specifically, OTcl) script. The data can be used for simulation analysis (two simulation result analysis examples are presented in later sections) or as an input to a graphical simulation display tool called Network Animator (NAM) that is developed as a part of VINT project. NAM has a nice graphical user interface similar to that of a CD player (play, fast forward, rewind, pause and so on), and also has a display speed controller. Furthermore, it can graphically present information such as throughput and number of packet drops at each link, although the graphical information cannot be used for accurate simulation analysis.

Downloading/Installing ns & nam on LINUX

1. Download ns-allinone-2.31.tar.gz from http://www.isi.edu/nsnam
2. Put the file in /usr/src by typing, cd /usr/src
3. Unpack file by typing, tar -xvzf nsallinone-2.31.tar.gz
4. cd ns-allinone -2..31
5. type, ls -lr to check if we are in correct directory (a dot means correct directory)
6. type, ./install
7. type, ./validate to validate

To Run NS

1. Place the following bash file in user work directory, /home/usr then, ns-abc.sh

********** Script File Start **********

```
#!/bin/sh

p1=/usr/src/ns-allinone-2.31/bin
p2=/usr/src/ns-allinone-2.31/otcl-1.0a8
p3=/usr/src/ns-allinone-2.31/lib
p4=/usr/src/ns-allinone-2.31/lib/tcl8.3
p5=/usr/src/ns-allinone-2.31/lib/tk8.3
```

PATH=$PATH:$p1:$p2:$p3:$p4:$p5
export PATH

$PATH

LD_LIBRARY_PATH=$p3:$p4:$p5
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH

$LD_LIBRARY_PATH

ns $1

********** Script File End **********

2. Now, ./ns-abc.sh simple.tcl
Use a file browser and select the properties button to set attributes.
Ns$1 invokes the simulator and passes in the file name as a parameter.
We can also create a script to start Network Animator , only change to be done is ns$1 to nam$1 and save as run-nam.sh by gedit abc.sh
3. To run, ./run-nam.sh

****

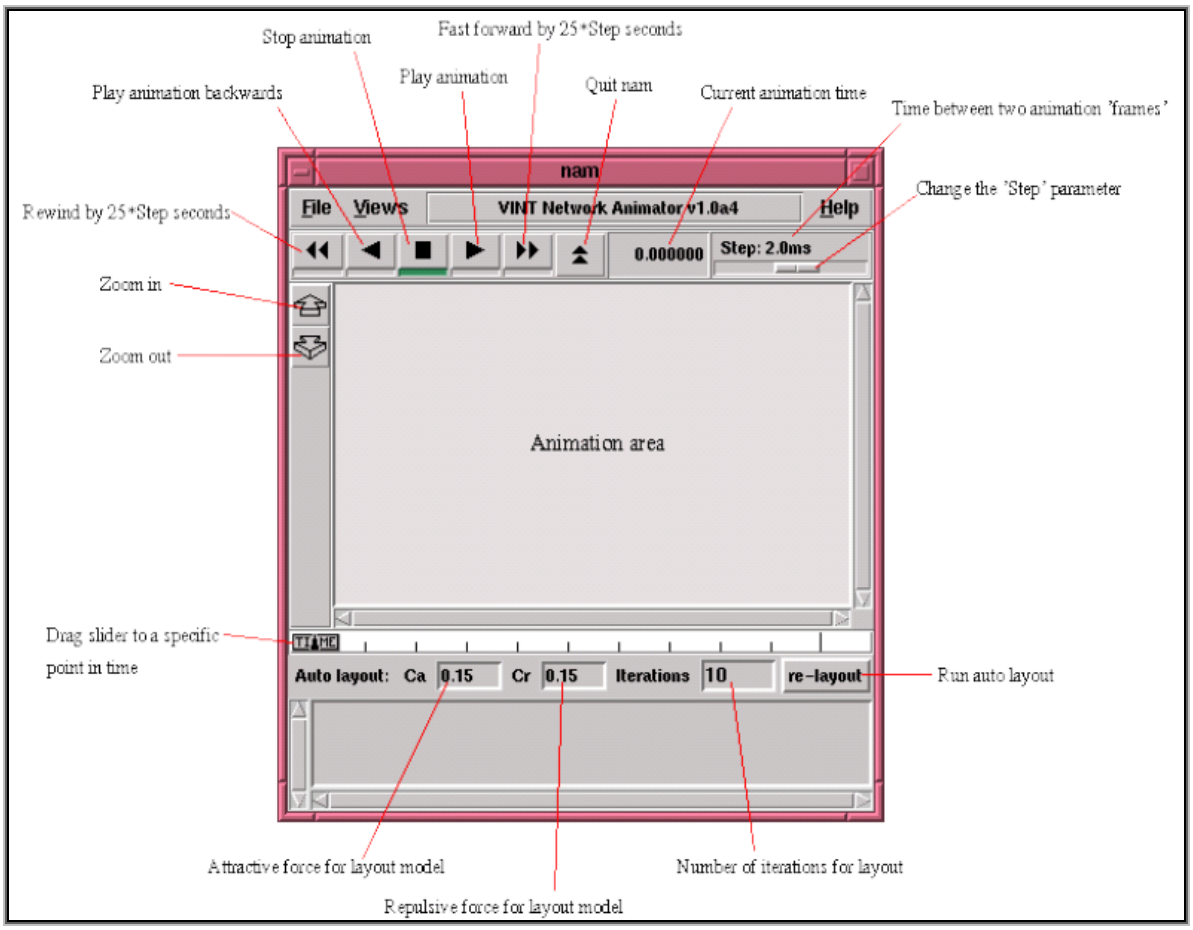

Figure B.3: nam window

Starting ns

NS starts with the command 'ns <tclscript>' (assuming that we are in the directory with the ns executable, or that were path points to that directory), where '<tclscript>' is the name of a Tcl script file which defines the simulation scenario (i.e. the topology and the events). We could also just start ns without any arguments and enter the Tcl commands in the Tcl shell, but that is definitely less comfortable. Everything else depends on the Tcl script. The script might create some output, it might write a trace file or it might start nam to visualize the simulation.

Starting nam

We can either start nam with the command 'nam <nam-file>' where '<nam-file>' is the name of a nam trace file that was generated by ns, or we can execute it directly out

of the Tcl simulation script for the simulation which we want to visualize. Below we can see a screenshot of a nam window where the most important functions are being explained.

How to start Tcl Scripts

We can write wer Tcl scripts in any text editor like joe or emacs. First of all, we need to create a simulator object. This is done with the command

```
set ns [new Simulator]
```

Now we open a file for writing that is going to be used for the nam trace data.

```
set nf [open out.nam w]
$ns namtrace-all $nf
```

The first line opens the file 'out.nam' for writing and gives it the file handle 'nf'. In the second line we tell the simulator object that we created above to write all simulation data that is going to be relevant for nam into this file. The next step is to add a 'finish' procedure that closes the trace file and starts nam.

```
proc finish {} {
global ns nf
$ns flush-trace
close $nf
exec nam out.nam &
exit 0
}
```

The next line tells the simulator object to execute the 'finish' procedure after 5.0 seconds of simulation time.

```
$ns at 5.0 finish
```

We can understand what this line does just by looking at it. ns provides we with a very simple way to schedule events with the 'at' command. The last line finally starts the simulation.

```
$ns run
```

Network Components

The root of the hierarchy is the TclObject class that is the superclass of all OTcl library objects (scheduler, network components, timers and the other objects including NAM related ones). As an ancestor class of TclObject, NsObject class is the superclass of all basic network component objects that handle packets, which may compose compound network objects such as nodes and links. The basic network components are further divided into two subclasses, Connector and Classifier, based on the number of the possible output data paths. The basic network objects that have only one output data path are under the Connector class, and switching objects that have possible multiple output data paths are under the Classifier class.

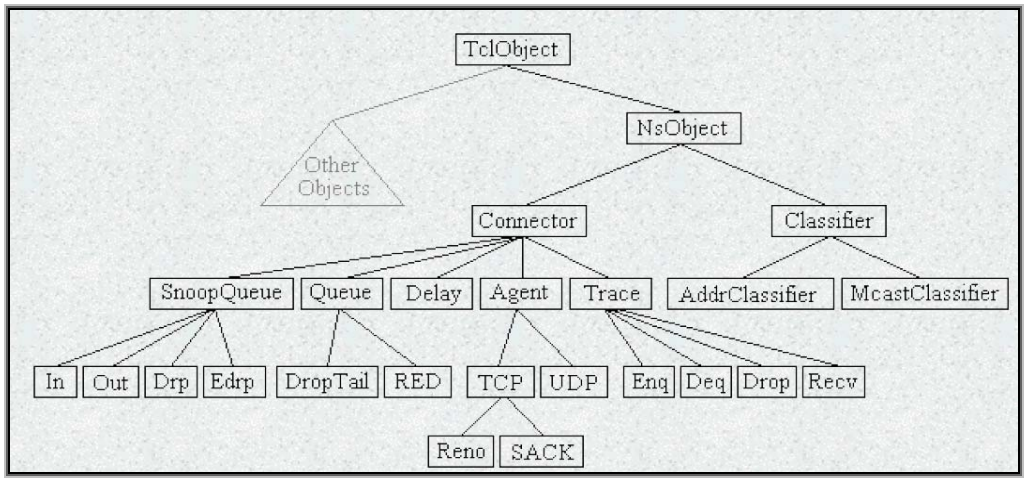

Figure B.4: class hierarchy (partial)

Node and Routing

There are two types of nodes in NS.

- **A unicast node** has an address classifier that does unicast routing and a port classifier.

  $ns rtproto type

  type: Static, Session, DB, cost, multi-path

- **A multicast node**, in addition, has a classifier that classify multicast packets from unicast packets and a multicast classifier that performs multicast routing.

  $ns multicast (right after set $ns [new Scheduler])

  $ns mrtproto type

type: CtrMcast, DM, ST, BST

In NS, Unicast nodes are the default nodes. To create Multicast nodes the user must explicitly notify in the input OTcl script, right after creating a scheduler object, that all the nodes that will be created are multicast nodes. After specifying the node type, the user can also select a specific routing protocol other than using a default one.

Link

A link is another major compound object in NS. When a user creates a link using a duplex-link member function of a Simulator object, two simplex links in both directions are created. One thing to note is that an output queue of a node is actually implemented as a part of simplex link object. Packets dequeued from a queue are passed to the Delay object that simulates the link delay, and packets dropped at a queue are sent to a Null Agent and are freed there. Finally, the TTL object calculates Time To Live parameters for each packet received and updates the TTL field of the packet.

Tracing

In NS, network activities are traced around simplex links. If the simulator is directed to trace network activities (specified using $ns trace-all file or $ns namtrace-all file), the links created after the command will have the following trace objects inserted. Users can also specifically create a trace object of type type between the given src and dst nodes using the create-trace {type file src dst} command. When each inserted trace object (i.e. EnqT, DeqT, DrpT and RecvT) receives a packet, it writes to the specified trace file without consuming any simulation time, and passes the packet to the next network object.

Queue Monitor

Basically, tracing objects are designed to record packet arrival time at which they are located. Although a user gets enough information from the trace, he might be interested in what is going on inside a specific output queue. For example, a user interested in RED queue behavior may want to measure the dynamics of average queue size and current queue size of a specific RED queue (i.e. need for queue monitoring).

Queue monitoring can be achieved using queue monitor objects and snoop queue objects. When a packet arrives, a snoop queue object notifies the queue monitor object of this event. The queue monitor using this information monitors the queue.

Packet

A NS packet is composed of a stack of headers, and an optional data space. A packet header format is initialized when a Simulator object is created, where a stack of all registered (or possibly useable) headers, such as the common header that is commonly used by any objects as needed, IP header, TCP header, RTP header (UDP uses RTP header) and trace header, is defined, and the offset of each header in the stack is recorded. What this means is that whether or not a specific header is used, a stack composed of all registered headers is created when a packet is allocated by an agent, and a network object can access any header in the stack of a packet it processes using the corresponding offset value.

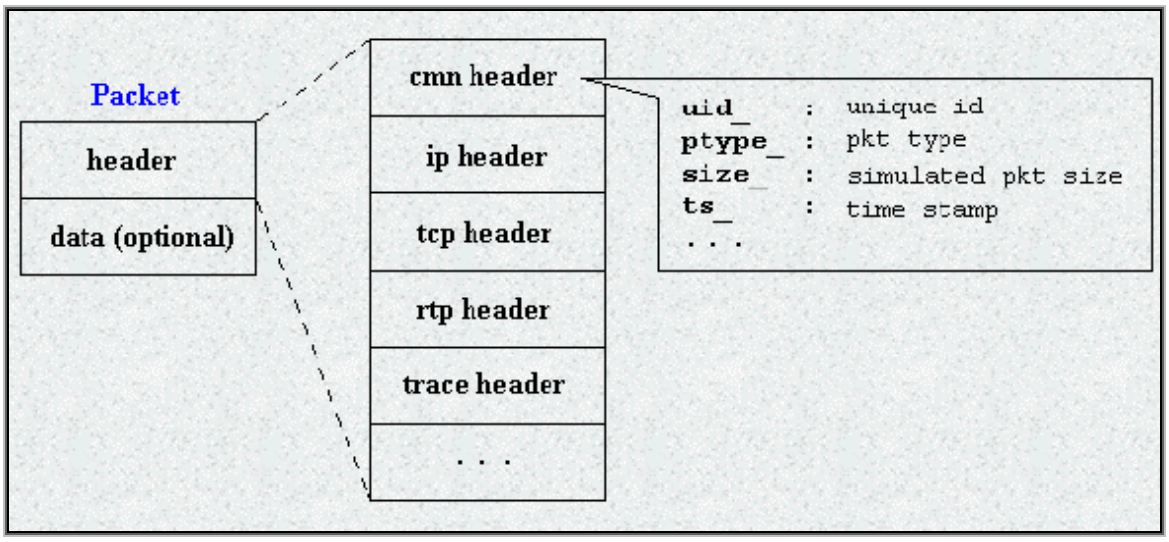

Figure B.5: NS Packet Format

Usually, a packet only has the header stack (and a data space pointer that is null). Although a packet can carry actual data (from an application) by allocating a data space, very few application and agent implementations support this. This is because it is meaningless to carry data around in a non-real-time simulation. However, if we want to

implement an application that talks to another application cross the network, we might want to use this feature with a little modification in the underlying agent implementation. Another possible approach would be creating a new header for the application and modifying the underlying agent to write data received from the application to the new header.

References

[1] "*NS Manual*", The VINT Project from http://www.isi.edu/nsnam/ns/doc

[2] "*NS tutorial*" by Marc Greis. http://www.isi.edu/nsnam/ns/tutorial/nsmenu.html

[3] "*NS by Examples*" by Jae Chung and Mark Claypool.

[4] John K. Ousterhout, "*Tcl and the Tk Toolkit"*', Pearson Education, 1994