Network Based Packet Watermarking Using TCP/IP Protocol Suite

By

Maitrik Shah

10MCEC15

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

MAY 2012

Network Based Packet Watermarking Using TCP/IP Protocol Suite

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

Maitrik Shah

(10MCEC15)

Guided By

Prof. Samir B. Patel

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

MAY 2012

Declaration

I, **Maitrik K. Shah**, **10MCEC15**, give undertaking that the Major Project entitled **"Network based packet watermarking using TCP/IP protocol suite"** submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in Institute of Technology of Nirma University, Ahmedabad, is the original work carried out by me and I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

Maitrik Shah

Certificate

This is to certify that the Major Project entitled "Network Based Packet Watermarking Using TCP/IP Protocol Suite" submitted by Maitrik Shah (10MCEC15), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Samir B. Patel
Guide, Associate Professor,
Department of C.S.E.,
Institute of Technology,
Nirma University, Ahmedabad.

Dr.S.N.Pradhan
Professor and PG-Coordinator,
Department of C.S.E,
Institute of Technology,
Nirma University, Ahmedabad.

Prof.D.J.Patel
Professor and Head,
Department of C.S.E,
Institute of Technology,
Nirma University, Ahmedabad.

Dr.K.Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad.

Abstract

Watermarking is defined as the art and science of embedding information in some cover. It takes one piece of information and embeds it within another. Mostly digital data is used for embedding. In this paper, we present a way to embed this digital information into the data packet. TCP/IP packets are transmitted over a network in large quantity. Within TCP/IP header there are number of fields that are not used for normal transmission or are "optional" fields which can be used by the sender of the datagrams. An analysis of the areas of a typical IP header shows that there are few fields which are either unused or optional reveals many possibilities where data can be stored and transmitted. Options field of IP header is one of those fields. By embedding information in the options field of IP packet, we can make use of unused field of the IP header and can transmit our digital data within TCP/IP packets from sender to the receiver. One advantage of transmitting data in the header is that intruders will not have any idea that data is transmitted in the header, intruders will concentrate only on the payload of the packet. In order to implement this idea we will make our packet(which contains hidden data)to look like as normal packet(which does not contain hidden data) so that intruders will not be able to identify that our packet contains hidden data. Encryption and compression however are optional in performing watermarking, but by making use of such techniques the security in the design can be incorporated. Challenges like loss of packets and many others are also handled in the design.

Index Terms - Watermarking, Steganography, Packet-Watermarking,IP options,Linux Kernel.

Acknowledgements

My deepest thanks to **Prof. Samir B. Patel**, Associate Professor, Department of Computer Science and Engineering, Institute of Technology, Nirma University, Ahmedabad the Guide of the project that I undertook for giving his valuable inputs and correcting various documents of mine with attention and care. He has taken the pain to go through the project and make necessary amendments as and when needed.

My deep sense of gratitude to **Dr. S.N.Pradhan**, Professor and PG-Coordinator of Department of Computer Engineering, Institute of Technology, Nirma University, Ahmedabad for an exceptional support and continual encouragement throughout part one of the Major project.

I also express my profound gratitude to **Prof. Zunnun Narmawala**, **Prof. Vijay Ukani** and **Prof. Priyanka Sharma** for their timely suggestions, continuous support and help to solve my doubts.

I would like to thanks **Dr. Ketan Kotecha,** Hon'ble Director, Institute of Technology, Nirma University, Ahmedabad for his unmentionable support, providing basic infrastructure and healthy research environment.

I would also thank my Institution, all my colleagues and all my faculty members in Department of Computer Science and my colleagues without whom this project would have been a distant reality. Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete my dissertation work successfully.

- Maitrik Shah

10MCEC15

Contents

List of Tables

List of Figures

Abbreviations

DF	Do Not Fragment
DOD	Department of Defence
ihl	IP header length
IP	Internet Protocol
IPC	Inter-process Communication
IPv4	Internet Protocol Version-4
MM	Memory Manager
NET	Network Interface
SCHED	Schedular
TCP	Transmission Control protocol
TTL	Time To Live
UDP	User Datagram Protocol
VFS	Virtual File System

Chapter 1

Project Introduction

1.1 Watermarking

One of the reasons that intruders can be successful is that most of the information they acquire from a system is in a form that they can read and comprehend. Intruders may reveal the information to others, modify it to misrepresent an individual or organization, or use it to launch an attack or change the ownership, make the copy of it and distribute it to the others. One solution to this problem is Watermarking.

Watermarking is the technique to embed the information in the carrier medium. The medium that contains a digital watermark is called a carrier. Carrier medium may be image, Audio, Video, Network protocols such as TCP, IP, UDP . We can embed information in the carrier medium in visible form which is called visible watermark or in invisible form which is called invisible watermark. We generally do not use digital watermark as separate file or link. We will embed our digital watermark which is nothing but information directly into the carrier medium. We can provide security and confidentiality with the use of the key. A person who wants to access and modify the embedded watermark must possess a security key. Without security key embedded watermark can't be accessed and modified. Digital watermarking can be characterized by a number of requirements. The most important are [3]:

- Transparency: The watermark must be imperceptible, image and audio quality must be unchanged.
- Robustness: Measures, how stable a watermark against alterations of the carrier medium is.

- Capacity: Describes the amount of information that can be embedded into a carrier medium by a watermarking algorithm.
- Security: Without knowledge of the secret key, the embedded information may not be extracted, altered or destroyed without rendering the carrier medium unusable.
- Performance: In real-world systems embedding and retrieval process need to be sufficiently fast. Depending on the application this can be multiple times real-time.

1.2 Packet Watermarking

Here in this paper, we describe packet watermarking technique by transmitting data (encrypted) in the options field of TCP/IP packet header. We describe that a more important problem that has received little attention to date is that of Packet Watermarking. In Image watermarking we embed our watermarked data in the image. Similarly in packet watermarking we will embed our information in the packet. And for that we will make use of header field of IP packet. By embedding information in the options field we can make use of unused options field of the IP header and can transmit the watermarked data.

1.3 Objective

This project comprehends the following objectives:
(i) To develop secure system for transmitting hidden data.
(ii)To make the use of unused options field of the IP header.

1.4 Basic Model Of Watermarking

Here in this section basic model of watermarking is described. It consists of Carrier object, Message(we want to send) and secret key which is optional. Carrier object is also known as cover-object, within which we will hide our message. Basically, the model for Watermarking is shown on Fig.1 [3]. Message is the confidential information that sender wants to send to the receiver. It can be plain text(if not encrypted), cipher text(if encrypted), other image, or anything that can be embedded in a bit stream such as a copyright mark, a covert communication, or a serial number. Secret-key is also known as Watermark-key, which ensures that only recipient who know the corresponding decoding key will be able to extract the message from a cover-object. The cover-object with the secretly embedded message is then called the Watermarked-object.

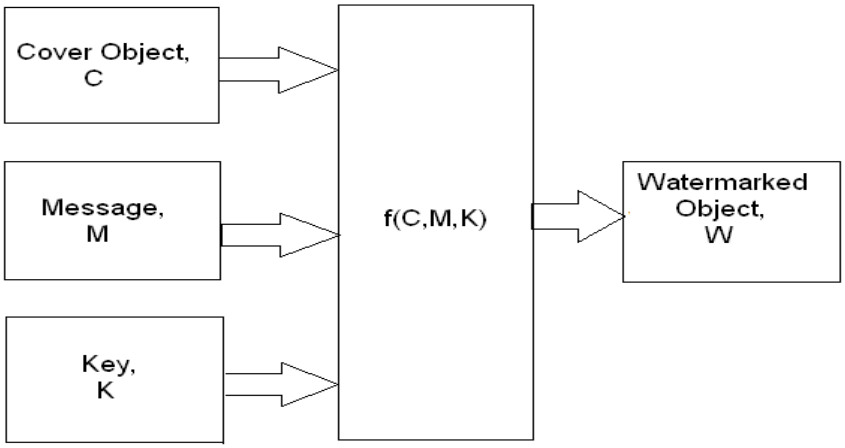

Figure 1.1: Basic Model Of Watermarking

To recover message from the watermarked object receiver requires the cover-object itself and a corresponding decoding key if a Watermark-key was used during the encoding process.

To recover message from the watermarked object receiver requires the cover-object itself and a corresponding decoding key if a Watermark-key was used during the encoding process.

There are several suitable carriers below to be the cover-object [3]:

1. Network Protocols such as TCP, IP and UDP

2. Audio that using digital audio formats such as wav, midi, avi, mpeg, mpi and voc.

3. File and Disk

4. Images file

1.5 Thesis Organization

The rest of the thesis is organized as follows.

Chapter 2: *Literature Survey*, This section describes different methods on how hidden data can be transmitted across the network. It also describes the problems in those methods.

Chapter 3: *Our Approach And Method*, This section includes the details of different approaches about how packet watermarking can be used to send hidden data using unused options field of the IP header.

Chapter 4: *Linux Kernel*, This section includes the details of linux kernel and TCP/IP stack in linux kernel. It also includes what will happen in TCP/IP stack when sender sends the packet and receiver receives the packet.

Chapter 5: *Implementation and Results*, This section includes the flowchart and description of what happens at sender side and receiver side when packet comes to IP layer. It also includes snapshots as a result of implementation of our approach.

Chapter 6: *Conclusion and Future Work*

Chapter 2

Literature Survey and Important observations

2.1 Previous Work

TCP/IP packet header contains number of fields that are reserved for future use or unused. So there are many possibilities to transmit hidden data in TCP/IP packet. Similar approaches have been published in [1],[2],[4],[5].

- In [1] the author has used Identification field of the IP header which is of 16 bits. It is the random number which is assigned by the sender when the packet is constructed. This field is used when fragmentation of the packet occurs. With the value in this field receiver will identify frames of the same packet. Here author has used one number 256 as key. Encoding is done like this:

  Encoding: Suppose we want to send H. ASCII of H is 72.
  72*256 = 18432 this value will be sent in the Identification field.

  Decoding: 18432/256 = 72(H).

- In [2] author has used Do Not Fragment Bit of IP header to send the hidden data. This field is used to specify whether to do fragmentation or not. So we can use this field to send our data only if we are sure packet will not be fragmented. In this work the problem is the size of data we can use to send our data. Do not Fragment field

is of one bit only so here we can transmit only one bit for each datagram. Suppose our packet does not carry anything in the payload. It contains only data. Since IP header is of 20 bytes if options field is unused than the ratio useful information to total data is 1:160, it means that if you want to transmit the phrase "hello India" then there will be overhead of almost 2 Kb for just 11 bytes.

- In [4] they have made the use of type-of-service field of IP packet to send data. Type-of-service field is of 8 bits. In those 8 bits 2 bits are unused and that 2 bits can be used to transmit the hidden data. In another approach they have made the use of reserved bits of TCP header for the transmission of the hidden data. 6 bits of the TCP header are reserved. So they have made the use of those 6 bits. So if we combine these two approach we can transmit one byte of hidden data per packet transmitted.

- In [5] they have used two steganographic method. In one method they have used number of fragments to send hidden data. SS is the steganogram sender and SR is the steganogram receiver. Here, original IP packet is divided into predetermined number of fragments. If the number of fragments are even then it means that binary "0" is transmitted otherwise binary "1" as shown in the following figure.

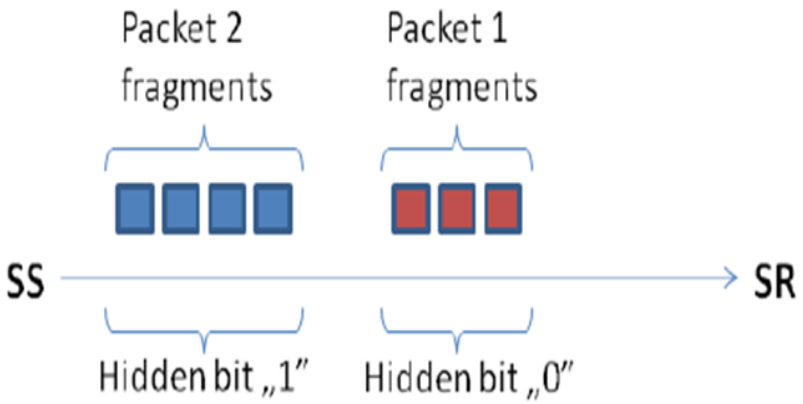

Figure 2.1: Steganographic method example1

- In another method they have used fragmentation offset field of IP header to transmit hidden data. If fragmentation offset is even then it means that binary "0" is transmitted otherwise binary "1" as shown in the following figure.

Sequence	Identifier	Total Length	DF Flag	MF Flag	Fragment Offset	Hidden data
0-0	345	1300	0	1	0	-
0-1	345	1340	0	1	160	1
0-2	345	1340	0	1	325	0
0-3	345	1220	0	0	490	1

Figure 2.2: Steganographic method example2

2.2 Problems

The main problem in second paper which makes the use of DF field of IP header is the Packet Fragmentation. There are two possibilities:

1. Packet will be fragmented.

2. Packet will not be fragmented.

Since as a sender we do not know what will be the network between sender and receiver. So we do not know whether fragmentation will occur or not. We can use DF field of IP header if and only if we are sure that our packet will not be fragmented. Even if we are sure that our packet will not be fragmented we can send only 1 bit of hidden data using this field.

Problem in [1],[4],[5] is very few number of hidden bits per packet are transmitted.

Chapter 3

Our Approach And Method

3.1 IP Packet Structure

TCP/IP is the protocol used in Internet. TCP/IP were developed by a Department of Defense (DOD) research project to connect a number different networks designed by different vendors into a network of networks (the "Internet"). IP (Internet Protocol) is responsible for moving packet of data from node to node, and TCP (Transmission Control Protocol) is responsible for verifying the correct delivery of data from client process to server process. The basic unit of data transfer is Packet. At sender side the data is partitioned into IP packets and packets are transmitted over the network. At receiver side packets are reassembled to get the data. Each packet begins with a header containing addressing and system control information. The IP packet header consists of 20 bytes(if options field not used) of data divided in several fields. Each field has a special purpose, depending on the type of data contained in the packet payload. Following figure shows the structure of the IP packet.

- Version(4 bits): The Version field indicates the format of the internet header. This document describes version 4.
- IHL(4 bits): Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.
- Type of Service(8 bits): The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to

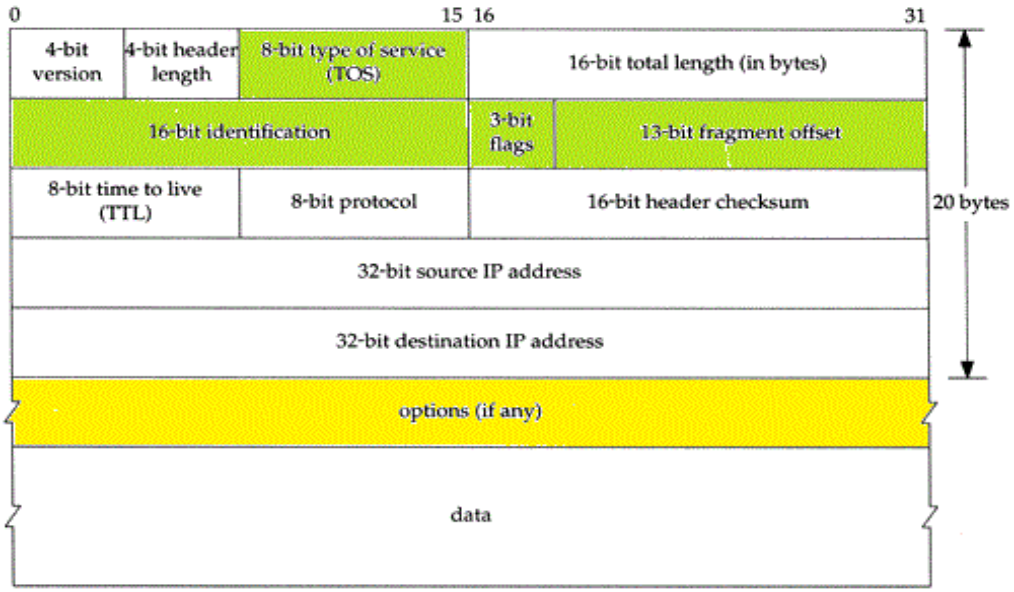

Figure 3.1: IP Packet Structure: Green coloured fields are those which are used in previous implementation.

guide the selection of the actual service parameters when transmitting a datagram through a particular network. The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

- Total Length(16 bits): Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments).

- Flags(3 bits): Various Control Flags.

- Fragment Offset(13 bits): This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

- Time to Live(8 bits): This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed.

- Protocol(8 bits): This field indicates the next level protocol used in the data portion of the internet datagram.

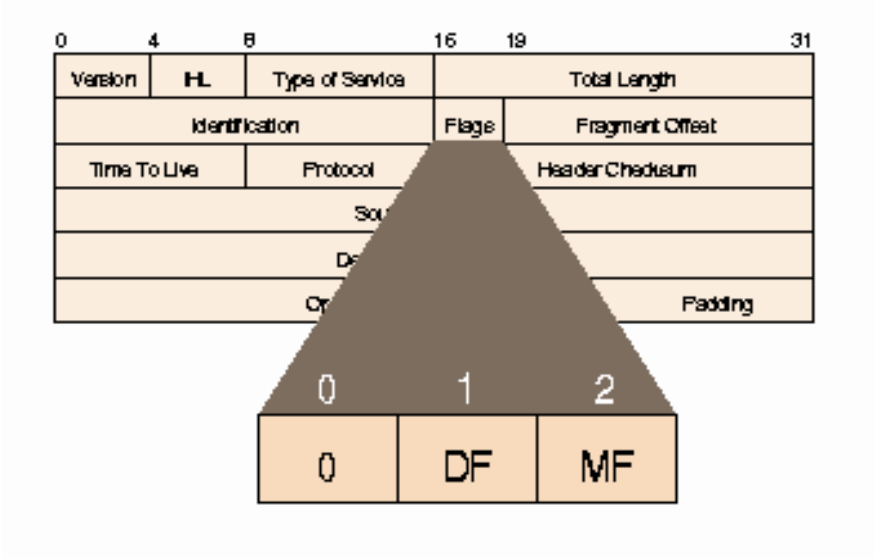

Figure 3.2: IP flags

- Header Checksum(16 bits): of checksum of the header is inserted into this field. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.
- Source Address(32 bits): The source address.
- Destination Address(32 bits): The destination address.
- Options(variable): The options may appear or may not appear in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation.

3.2 IP Options

When a packet is sent to the IP layer, then it normally includes all required information in the packet's protocol header. However, there could be times when packets require additional information in the protocol header for example, for diagnostics purposes or if a packet's path across the Internet is specified before it is sent. For these purposes, an Option field with variable length can be added to each IP packet header.

The Option field can take one or several packet options, where an option can be given in either of two formats [17]:

1. Single byte options: One byte options field which describes only the option type. The length of these options is always exactly one byte.

2. Multi byte options: The first byte includes the option type and the second byte contains the length of this packet option. The following bytes include the actual data of that option.

Following figure lists all IP packet options defined in RFC 791 [20], including their lengths and their defined option numbers and option classes. There are four option classes in total but only two are currently used. Option class 0 includes packet options for control and management; option class 2 includes debugging and measurement options. The option classes 1 and 3 are reserved for future IP packet-option classes.

Class	Number	Length	Name
0	0	-	End of Option List
0	1	-	No Operation
0	2	11	Security
0	3	var	Loose Source Routing
0	9	var	Strict Source Routing
0	7	var	Record Route
0	8	4	Stream ID
2	4	var	Internet Timestamp

Figure 3.3: Defined IP packet options

- **End of Option List :** This packet option marks the end of a series of options; it is appended to the last packet option and must never be between any other pair of options.

Figure 3.4: End of Options List

- **No operation :** No Operation can be between any two packet options, for example to let the second option begin at a 32-bit boundary.

Figure 3.5: No Operation

- **Security :** It comprises a total of 11 bytes. The Security option allows end systems to send security parameters or define own (controllable) groups of communication partners, which want to exchange IP packets in isolation from all other traffic. The two-byte Security field can be used to state 16 security levels for an IP packet; of these, the original RFC 791 defines eight levels, including Unclassified, Confidential, Restricted, Secret or Top Secret. The other security levels are reserved for future use.

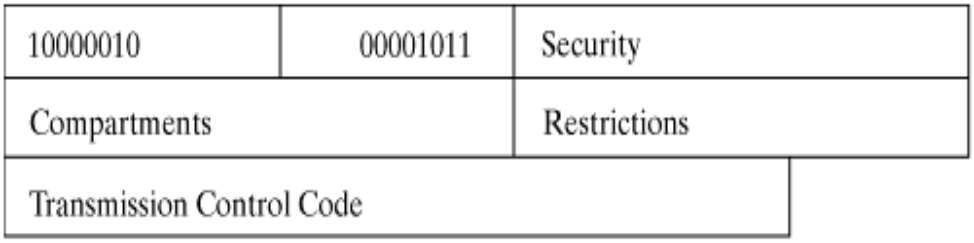

Figure 3.6: Security bit sequence

- **Loose Source Routing :** This option is used to specify all routers an IP packet has to visit on its way across the network. In addition, it accepts data about the packet's path. The third byte includes a pointer to the address of the next router that the packet has to pass.

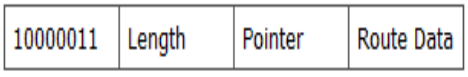

Figure 3.7: Loose Source Routing bit sequence

- **Strict Source Routing :** The Strict Source Routing option differs in only one point from the Loose Source Routing option: The packet may pass exactly those routers specified in the Route Data list.As with the previous option, if fragmentation is required, then the Strict Source Routing option has to be copied in each single fragment, which means that One is in the first position of this option.

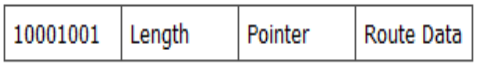

Figure 3.8: Strict Source Routing bit sequence

- **Record Route :** The Record Route option can be used to register the addresses of all intermediate systems an IP packet will pass on its way to the destination. The third byte includes a pointer to the field that is to accept the next address.

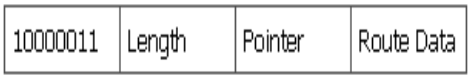

Figure 3.9: Record Route bit sequence

- **Stream Identifier :** This option enables the transport of SATNET Stream Identifiers across the Internet. The Stream Identifier packet option is always 4 bytes long and has to be copied to all fragments, if fragmentation is used. However, this option currently has no practical use.

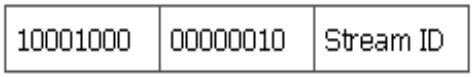

Figure 3.10: Stream Identifier bit sequence

- **Internet Timestamp :** This option can be used to store time stamps of selected or all network nodes. A 4-bit flag determines the data to be stored here, and it can take either of the following values:

  – 0 – time stamps only, stored in consecutive 32-bit words,

– 1 – each timestamp is preceded with internet address of the registering entity,

– 3 – the internet address fields are prespecified. An IP module only registers its timestamp if it matches its own address with the next specified internet address.

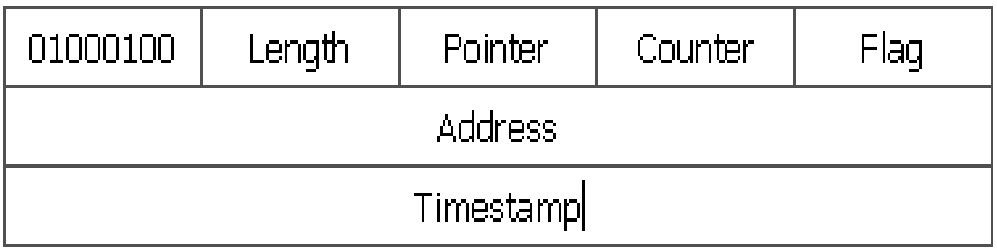

Figure 3.11: Internet Timestamp bit sequence1

3.3 Approach

The objective is to hide information in addition to make the use of unused options field of IP header and to securely transmit the data. As discussed in section literature survey, we can use IP identification field and Do Not Fragment bit of IP header, if and only if fragmentation does not occur. Unfortunately we are not sure whether fragmentation will occur or not. In this design we will neither hide data in the identification field of IP nor in the offset field but, we will hide our data in the options field of the IP header. Resulting in a situation in which we do not have to bother whether the packet will be fragmented or not. Secondly options field is of the variable length in which maximum 40 bytes can be used. So we can send good amount of data instead of few bytes.

Most of the times IP options field is not used. But there could be times when packets require additional information in the protocol header for diagnostics purposes or if a packet's path across the Internet is specified before it is sent.In that case options field will be included.

So there are two cases:

1. Options will be included

2. Options will not be included

Following are the five approaches that can be considered.

- **Approach-I : Use of End of options field** Here we will make use of End of options list field as shown in the following figure.

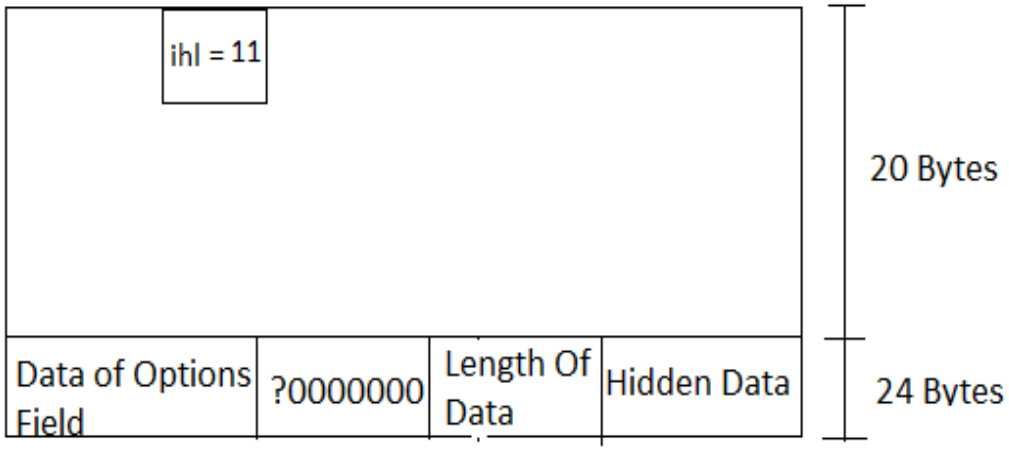

Figure 3.12: Use of End of options field-1

In first case we have set value of ihl field to its actual value. Receiver will search for the sequence 0/10000000. After this sequence hidden data will be there. But the problem with this approach is intruder will easily read our data. Because normally after end of options field there will be no operation field to maintain 32 bit boundry. Here after end of options sequence we are sending hidden data so intruder easily conclude that there is something in the header.

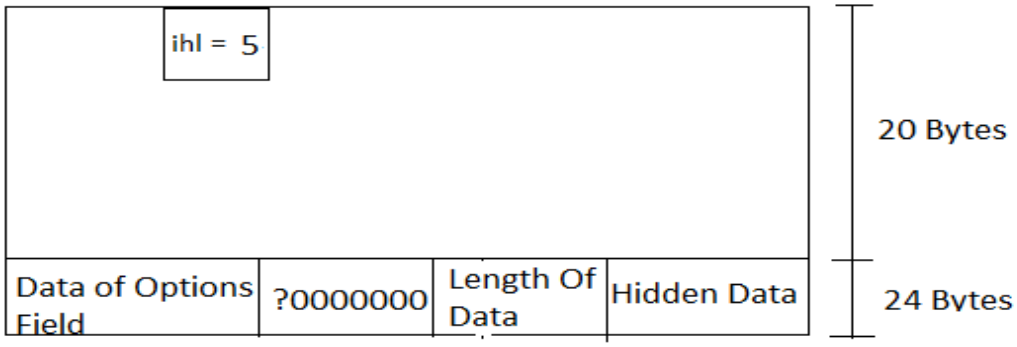

Figure 3.13: Use of End of options field-2

In second case we set value of ihl field to 5. In this case intruder will not be able

to conclude that there is something in the packet. But intermediate routers will not be able to read data of options field. This will cause problem in cases when we have used strict source routing, loose source routing, etc.

- **Approach-II : Use of specific bit sequence** To solve the problem of previous approach we have used specific bit sequence as shown in the following figures.

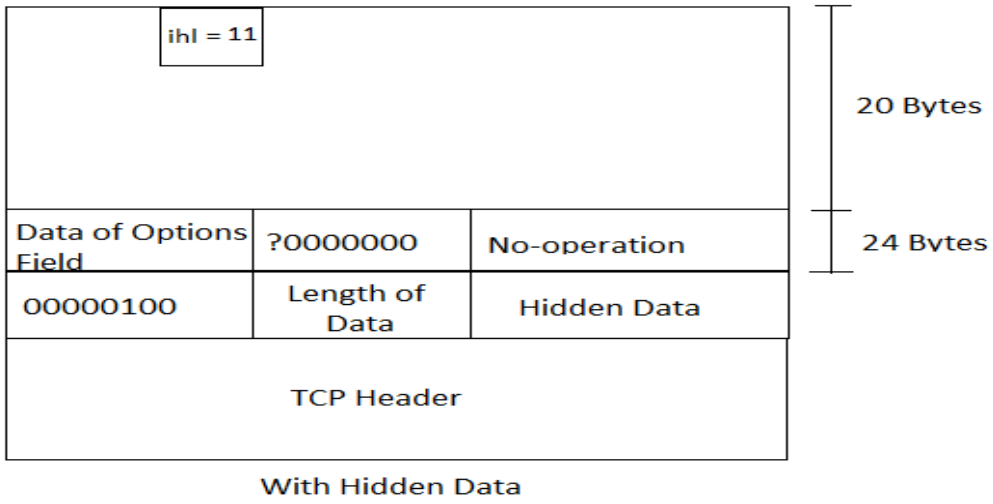

Figure 3.14: Use of specific bit sequence-1

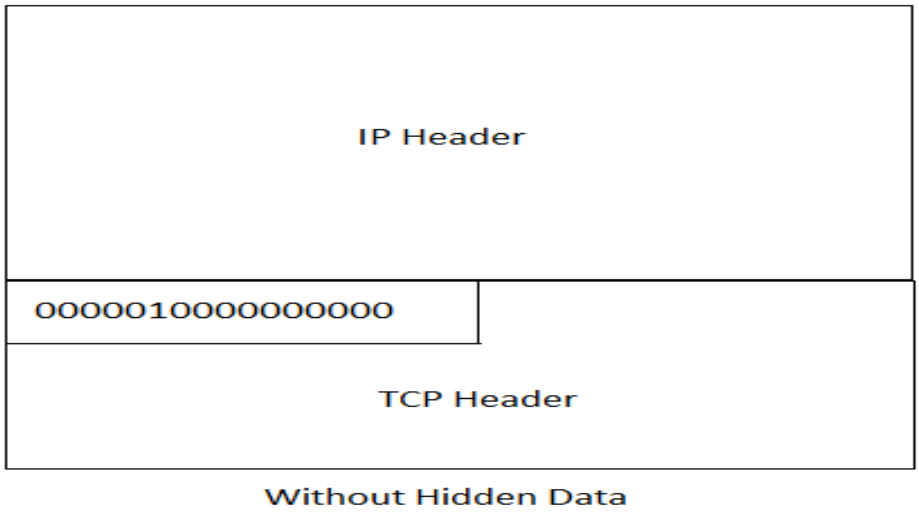

Figure 3.15: Use of specific bit sequence-2

Here, intermediate routers will be able to read header correctly. We will use specific bit sequence(e.g. 00000100). So that receiver can understand by reading this sequence that there is hidden data in the header. But it may be possible that same

sequence will be there in TCP header as shown in second case. So receiver will misinterpret this packet. Here receiver will not be able to distinguish which packet has hidden data and which does not have.

- **Approach-III : Use of Code bits of IP options field**

  Following figure shows the format of code bits.

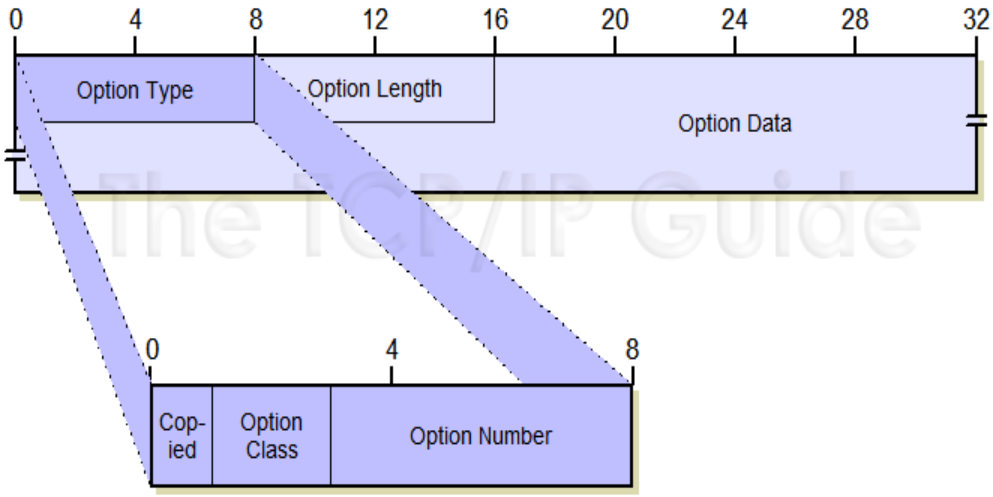

Figure 3.16: Format of code field of IP options

Here, the class field which of 2 bits can have 4 values.

00 - datagram control

10 - Debugging and Management

Other two values 01 and 11 are unused. So we can set 11 - for our hidden data as shown in the following figure.

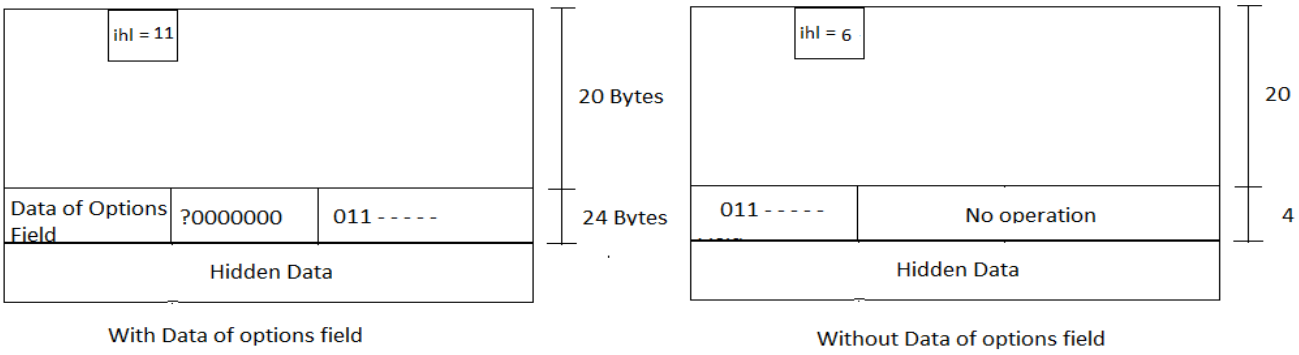

Figure 3.17: Use of Code bits of IP options field

So in sequence 011 - - - - -, 011 will tell the receiver there is hidden data. And remaining five bits will specify the length of the hidden data. But by reading 011 - - - - - sequence intruder can easily conclude that there is something abnormal in the header.

- **Approach-IV : Use of Identification field with same value for each packet**

  Identification field is of 16 bits. It is the random number generated by sender. For each and every packet in which sender wants to send hidden information sender can set this field to some value(e.g. 593). When receiver get the packet it will read this field. If it is 593 receiver will conclude that hidden data is there in the messsage. Length of the hidden field is specified by the length field which is of 1 byte.

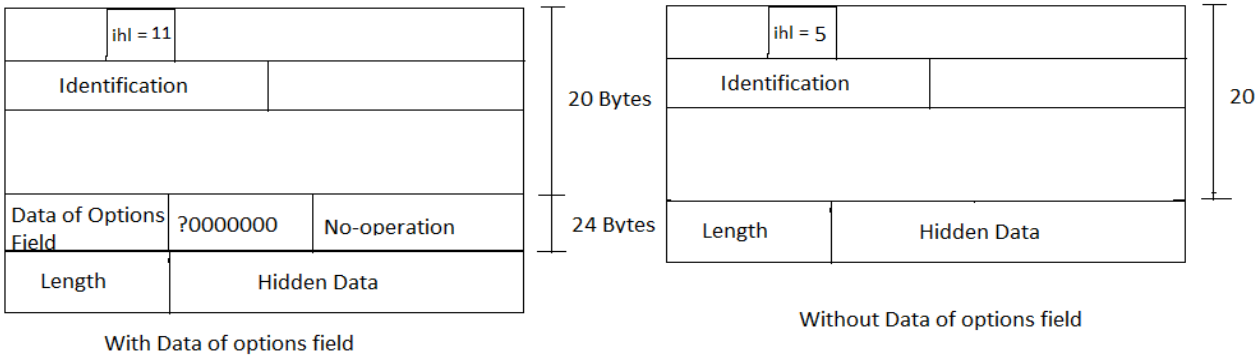

Figure 3.18: Use of Identification field-1

In normal cases value of identification field increases by 1 for each packet but in our case we copy same value in identification field. Since for each and every packet identification field is same. By observing large number of packets from sender to the receiver intruder will get an idea that there is something wrong in the packet. By analysis of those packets he will be able to get our pattern.

Following table describes the summary of all the previous approaches.

Table 3.1: Summary of Approaches

Approach	**Description**	**Problem**
Approach-1(a)	Use of End of options field with actual ihl value	Intruder will identify that there is something hidden in the header.
Approach-1(b)	Use of End of options field with ihl value 5	Intermediate routers will not be able to read the complete header.
Approach-2	Use of specific code sequence	Receiver will not be able to identify the packets with or without hidden data.
Approach-3	Use of Class bits of the options field	Intruder will identify that there is something hidden in the header.
Approach-4	Use of Identification field	Intruder can find the the pattern because same value used in all packets.

- **Approach-V : Use Of Identification Field with different value for each packet**

  The problem in the previous approach can be eliminated if we choose different value of identification field for each packet. We will divide this field in two parts as shown in the figure 3.19.

  Here the first part will be same for each packet(e.g.00111000). But for the first packet of the sequence random number will be generated for the remaining part. And that number will be incremented by 1 for each of the following packet. So each packet will have different value in the identification field. Receiver will search for 00111000 sequence. If it is there in the packet it means packet contains hidden data. So receiver will search for the hidden data in the packet.

  Since each and every packet has different value for the identification field intruder will not have any idea that there is something hidden in the header.

Following two sections shows you the flowchart of processing at sender side and processing at receiver side when packet comes to the IP layer.

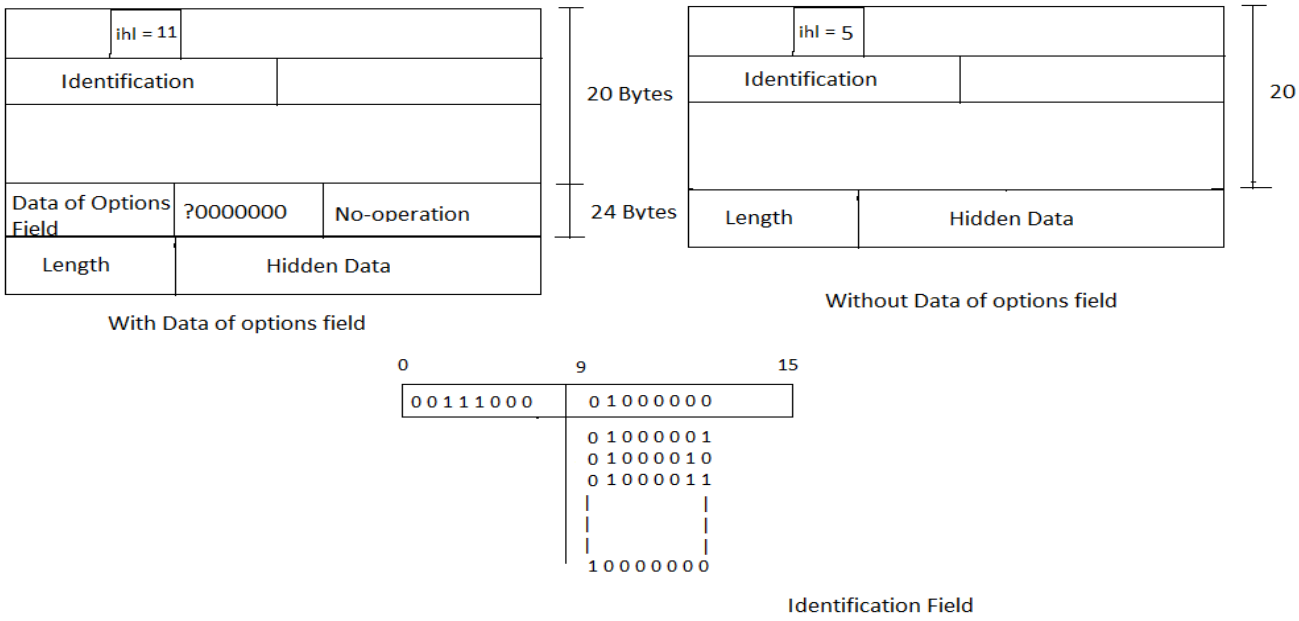

Figure 3.19: Use of Identification field-2

3.4 Processing at Sender side using Approach-V

Figure 3.20 shows the flowchart of what happens when sender wants to send packets using approach-V.

As shown in the figure when packet comes from transport layer to IP layer sender will add standard IP header which is of 20 bytes.Then two possibilities are there:

- Sender may send options data. If sender wants to send options data then ihl field will be updated according to the number of bytes added in the header. Maximum value of ihl field is 15. So if after addition of options data value of ihl becomes 15 then sender can not send hidden data. If it is less then 15 then sender can send hidden data.

- Sender may not send options data. Here, again there are two possibilities based on whether sender wants to send hidden data or not. If he does not want to send hidden data then packet will be forwarded to the lower layer. If sender wants to send hidden data then he will specify length(nbyte) of hidden data at (ihl*4+1)th byte of the IP header. Then sender reads nbyte of data from the file and insert it into IP header as shown in figure 3.19.

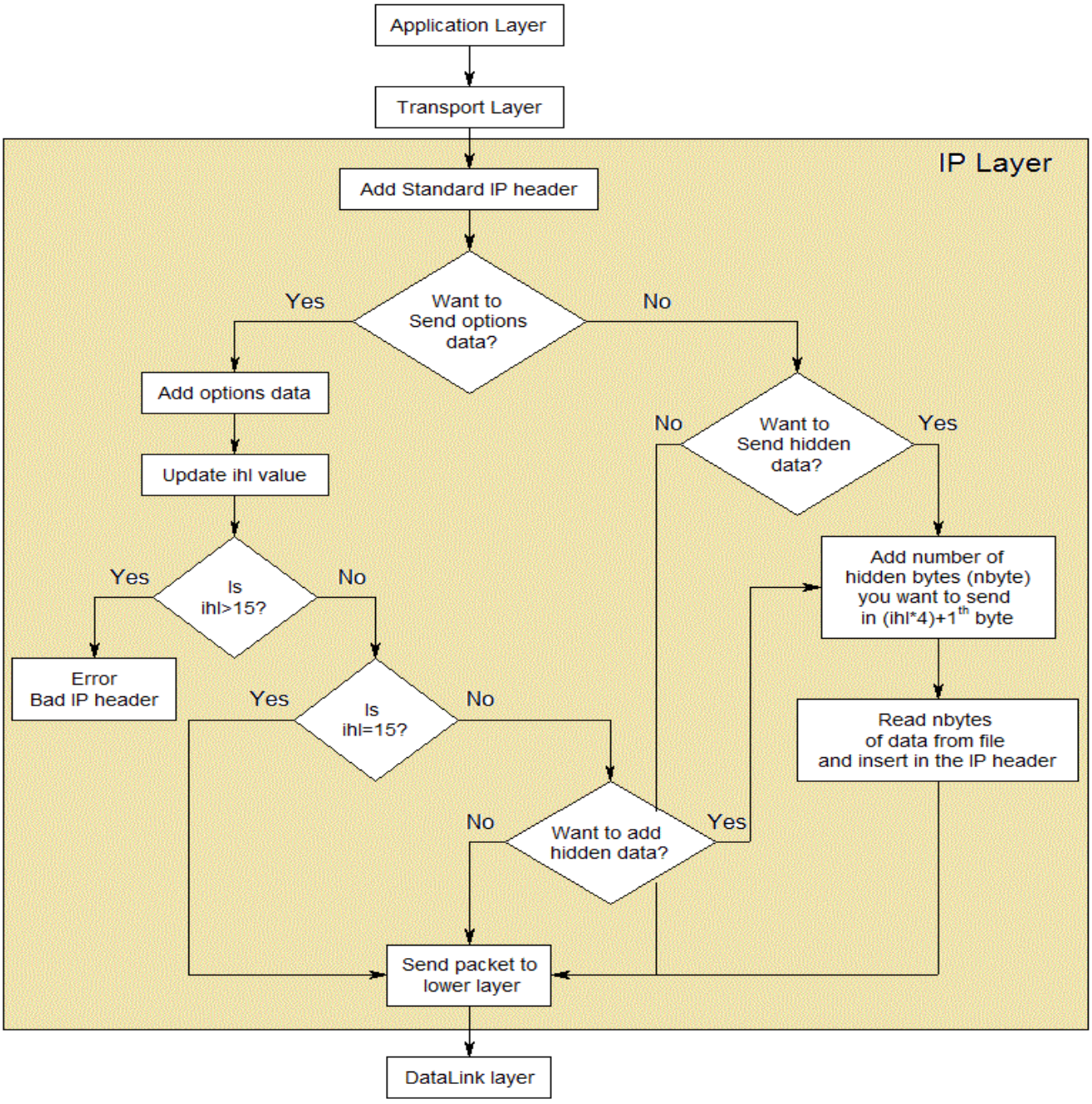

Figure 3.20: Processing at Sender side

3.5 Processing at Receiver side using Approach-V

Figure 3.21 shows the flowchart of what happens when receiver receives the packet.

- First of all when packet comes from the data link layer to the IP layer receiver will check ihl field of IP header. If it is greater than 5 it means that options data is there. If it is equal to 5 no options data is there in the packet.
- Then receiver will read first 8 bits of identification field. If hexadecimal value of first 8 bits of identification field is 38(in hexadecimal), then it means that hidden data is there in the packet. Receiver can get length(nbyte) of hidden data by reading (id*4+1)th byte of IP header. Receiver will write nbyte of data in the file and forward this packet to the transport layer.
- If hexadecimal value of first 8 bits of identification field is not equal to 38, then it means that no hidden data in the packet and packet will be forwarded to the transport layer as shown in the figure 3.21.

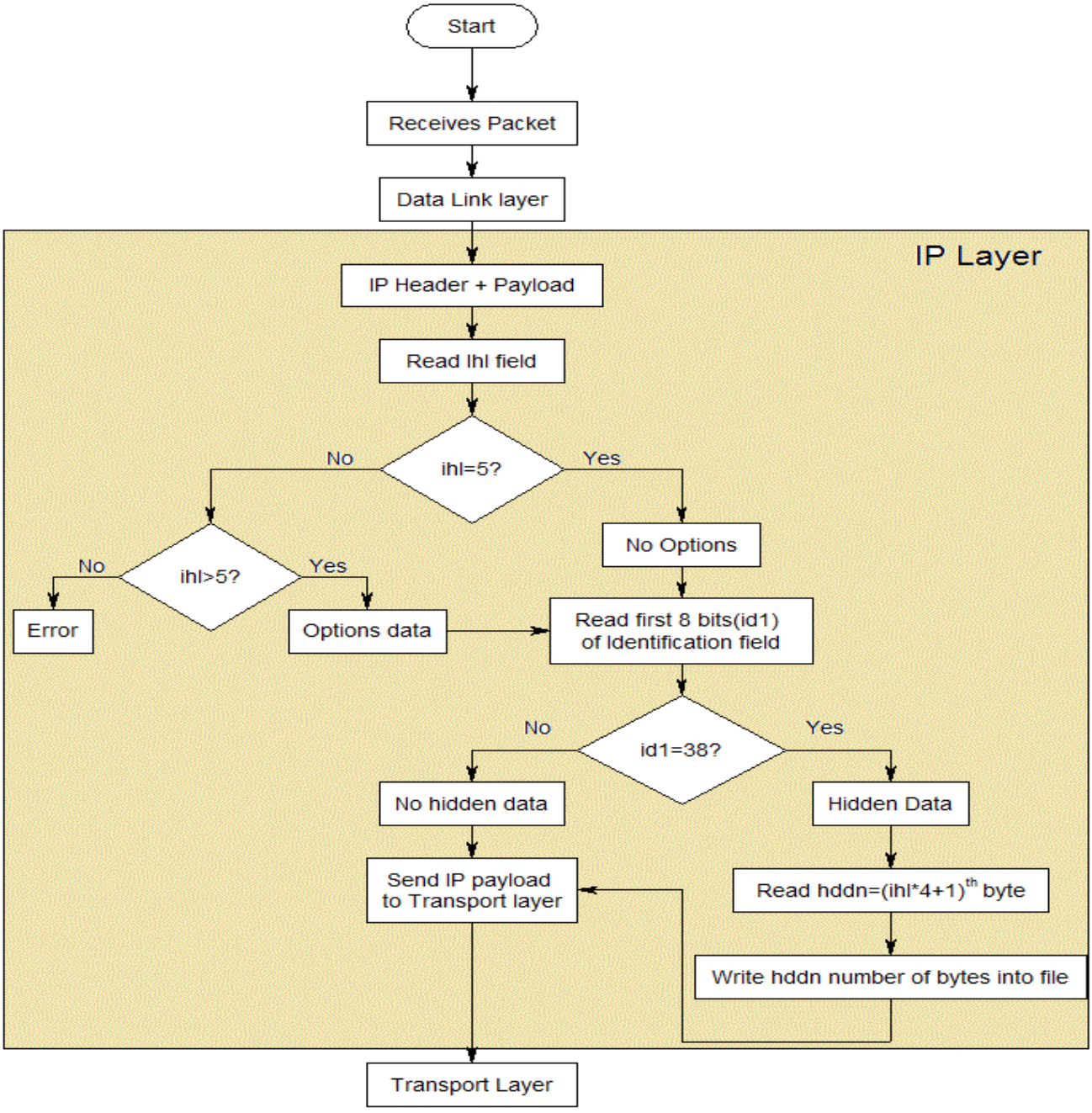

Figure 3.21: Processing at Receiver side

Chapter 4

Linux Kernel

4.1 Linux Kernel structure Overview

The Linux kernel is composed of five main subsystems [22]:

1. The Process Scheduler (SCHED) is responsible for controlling process access to the CPU. The scheduler enforces a policy that ensures that processes will have fair access to the CPU, while ensuring that necessary hardware actions are performed by the kernel on time.

2. The Memory Manager (MM) permits multiple process to securely share the machine's main memory system. In addition, the memory manager supports virtual memory that allows Linux to support processes that use more memory than is available in the system. Unused memory is swapped out to persistent storage using the file system then swapped back in when it is needed.

3. The Virtual File System (VFS) abstracts the details of the variety of hardware devices by presenting a common file interface to all devices. In addition, the VFS supports several file system formats that are compatible with other operating systems.

4. The Network Interface (NET) provides access to several networking standards and a variety of network hardware.

5. The Inter-Process Communication (IPC) subsystem supports several mechanisms for process-to-process communication on a single Linux system.

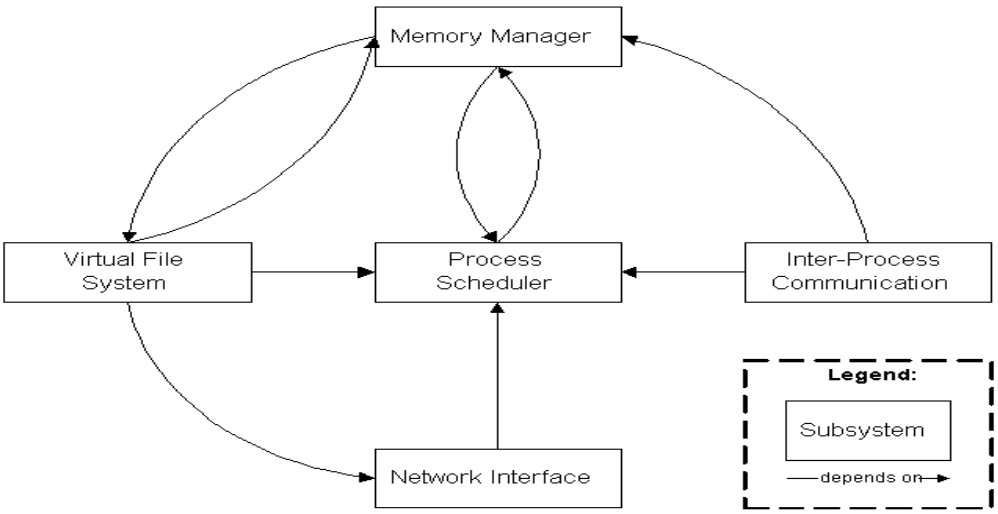

Figure 4.1: Kernel Subsystem Overview

This diagram emphasizes that the most central subsystem is the process scheduler: all other subsystems depend on the process scheduler since all subsystems need to suspend and resume processes. Usually a subsystem will suspend a process that is waiting for a hardware operation to complete, and resume the process when the operation is finished. For example, when a process attempts to send a message across the network, the network interface may need to suspend the process until the hardware has completed sending the message successfully. After the message has been sent (or the hardware returns a failure), the network interface then resumes the process with a return code indicating the success or failure of the operation. The other subsystems (memory manager, virtual file system, and inter-process communication) all depend on the process scheduler for similar reasons.

4.2 The Linux Source Tree

Linux source code is usually in the /usr/src directory (if installed). Otherwise it should be manually installed. Next section describes the steps to download and install kernel source code.This is an overview of the Linux source directory structure (not all branches are shown):

- arch - architecture specific code, by processor
  - i386 - code for Intel processors (including 486 and Pentium lines)
    - boot - location of newly compiled kernels
- drivers - code for drivers of all sorts
  - block - block device drivers (e.g., hard drives)
  - cdrom - CD ROM device drivers
  - net - network device drivers
  - pci - PCI bus drivers
- fs - code for different file systems (EXT2, MS-DOS, etc.)
- include - header files used throughout the code
  - asm asm-i386 - processor dependent headers
  - config - general configuration headers
  - linux - common headers
  - net - networking headers
- kernel - code for the kernel specific routines
- lib - code for errors, strings, and printf
- mm - code for memory management
- modules - object files and references for the kernel to load as required
- net - code for networking

– core - protocol independent code

– ipv4 - code specific to IPv4

– packet - protocol independent packet code

– sched - code for scheduling network actions

4.3 Sending Messages

This chapter presents the sending side of message trafficking. It provides an overview of the process, examines the layers packets travel through, details the actions of each layer, and summarizes the implementation code within the kernel.

4.3.1 Overview

An outgoing message begins with an application system call to write data to a socket. The socket examines its own connection type and calls the appropriate send routine (typically INET). The send function verifies the status of the socket, examines its protocol type, and sends the data on to the transport layer routine (such as TCP or UDP). This protocol creates a new buffer for the outgoing packet (a socket buffer, or struct skbuff skb), copies the data from the application buffer, and fills in its header information (such as port number, options, and checksum) before passing the new buffer to the network layer (usually IP). The IP send functions fill in more of the buffer with its own protocol headers (such as the IP address, options, and checksum). It may also fragment the packet if required. Next the IP layer passes the packet to the link layer function, which moves the packet onto the sending device's xmit queue and makes sure the device knows that it has traffic to send. Finally, the device (such as a network card) tells the bus to send the packet.

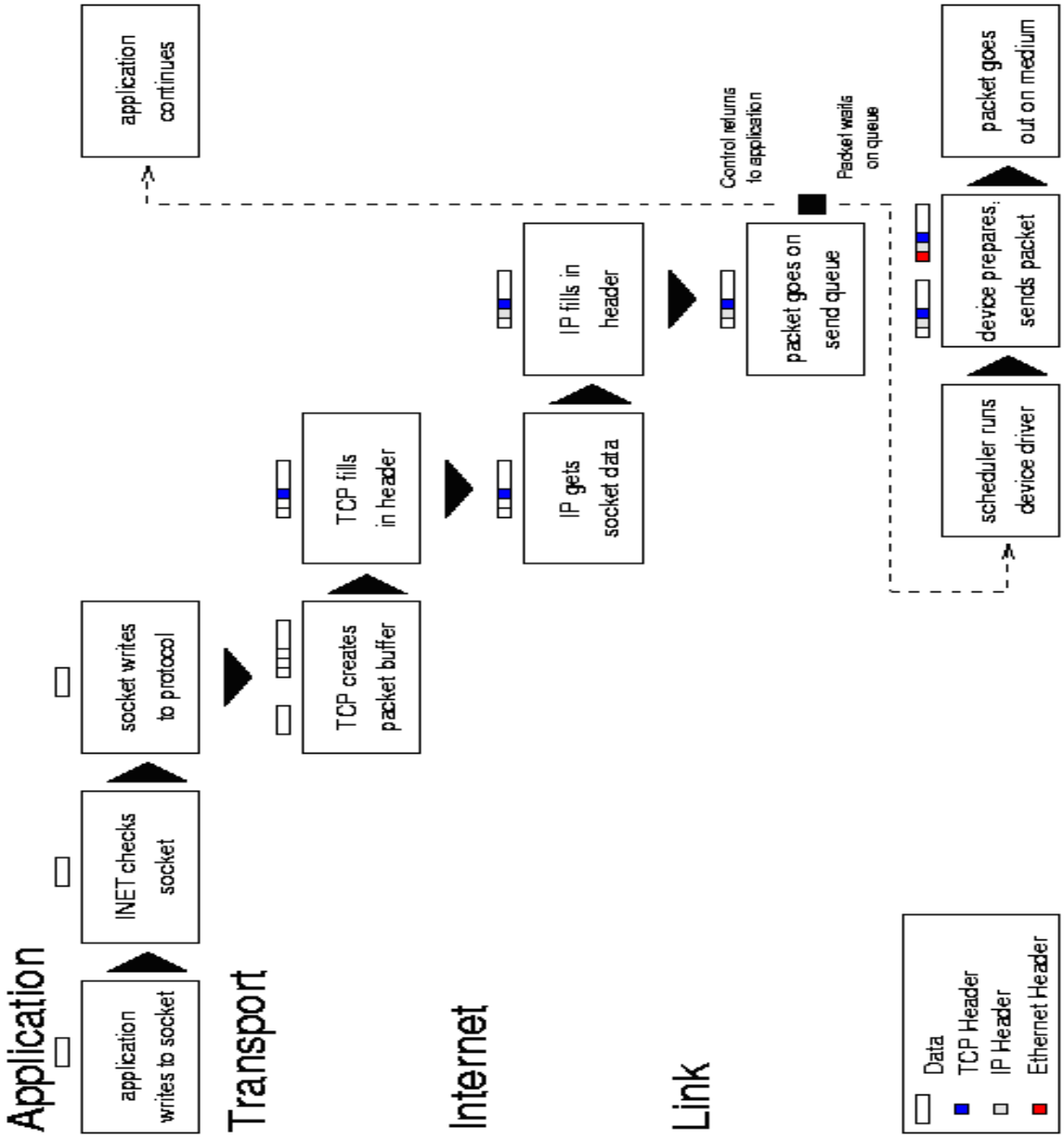

Figure 4.2: Message Transmission [12]

4.3.2 Processing At Sender Side

Writing to a Socket

- Write data to a socket (application)
- Fill in message header with location of data (socket)
- Check for basic errors - is socket bound to a port? can the socket send messages? is there something wrong with the socket?
- Pass the message header to appropriate transport protocol (INET socket)

Creating a Packet with TCP

- Check connection - is it established? is it open? is the socket working?
- Create a packet buffer
- Copy the payload from user space
- Add the packet to the outbound queue

Wrapping a Packet in IP

- Look up route to destination (if necessary - TCP)
- Fill in the packet IP header
- Copy the transport header and the payload from user space
- Send the packet to the destination route's device output funtion

Transmitting the Packet

- Put the packet on the device output queue
- Wake up the device
- Wait for the scheduler to run the device driver
- Test the medium (device)
- Send the link header
- Tell the bus to transmit the packet over the medium

4.3.3 List Of Functions

Following table gives information about few functions that are invoked when packet is sent.Table shows the function name , What they do and location of those functions in the linux kernel.

Table 4.1: Functions invoked during message transmission

Function Name	**What they do?**	**Location in the Kernel**
ssock_write()	creates and fills in message header with data size/addresses returns sock_sendmsg()	net/socket.c
sock_sendmsg()	calls scm_sendmsg() [socket control message]	net/socket.c
tcp_do_sendmsg()	waits for connection, if necessary adds data to waiting packet and checks window status calls csum_and_copy_from_user() to copy packet and do checksum calls tcp_send_skb()	net/socket.c
tcp_send_skb()	calls _skb_queue_tail() to add packet to queue calls tcp_transmit_skb()	net/ipv4/tcp_output.c
tcp_transmit_skb()	builds TCP header and adds checksum checks ACKs,SYN	net/ipv4/tcp_output.c
tcp_v4_sendmsg()	checks for IP address type, opens connection, port addresses	net/ipv4/tcp_ipv4.c
ip_build_xmit	calls sock_alloc_send_skb() to establish memory for skb sets up skb header	net/ipv4/ip_output.c
ip_queue_xmit()	looks up route builds IP header fragments if required adds IP checksum calls dev_queue_xmit()	net/ipv4/ip_output.c
dev_queue_xmit()	if device has a queue calls enqueue() to add packet to queue calls qdisc_wakeup() to wake device else calls hard_start_xmit()	net/core/dev.c
hard_start_xmit()	tests to see if medium is open sends header tells bus to send packet	drivers/net/DEVICE.c

4.4 Receiving Messages

This chapter presents the receiving side of message trafficking. It provides an overview of the process, examines the layers packets travel through, details the actions of each layer, and summarizes the implementation code within the kernel.

4.4.1 Overview

An incoming message begins with an interrupt when the system notifies the device that a message is ready. The device allocates storage space and tells the bus to put the message into that space. It then passes the packet to the link layer, which puts it on the backlog queue, and marks the network flag for the next "bottom-half" run.

The bottom-half is a Linux system that minimizes the amount of work done during an interrupt. Doing a lot of processing during an interrupt is not good precisely because it interrupts a running process; instead, interrupt handlers have a "top-half" and a "bottom-half". When the interrupt arrives, the top-half runs and takes care of any critical operations, such as moving data from a device queue into kernel memory. It then marks a flag that tells the kernel that there is more work to do - when the processor has time - and returns control to the current process. The next time the process scheduler runs, it sees the flag, does the extra work, and only then schedules any normal processes.

When the process scheduler sees that there are networking tasks to do it runs the network bottom-half. This function pops packets off of the backlog queue, matches them to a known protocol (typically IP), and passes them to that protocol's receive function. The IP layer examines the packet for errors and routes it; the packet will go into an outgoing queue (if it is for another host) or up to the transport layer (such as TCP or UDP). This layer again checks for errors, looks up the socket associated with the port specified in the packet, and puts the packet at the end of that socket's receive queue.

Once the packet is in the socket's queue, the socket will wake up the application process that owns it (if necessary). That process may then make or return from a read system call that copies the data from the packet in the queue into its own buffer. (The process may also do nothing for the time being if it was not waiting for the packet, and get the data off the queue when it needs it.)

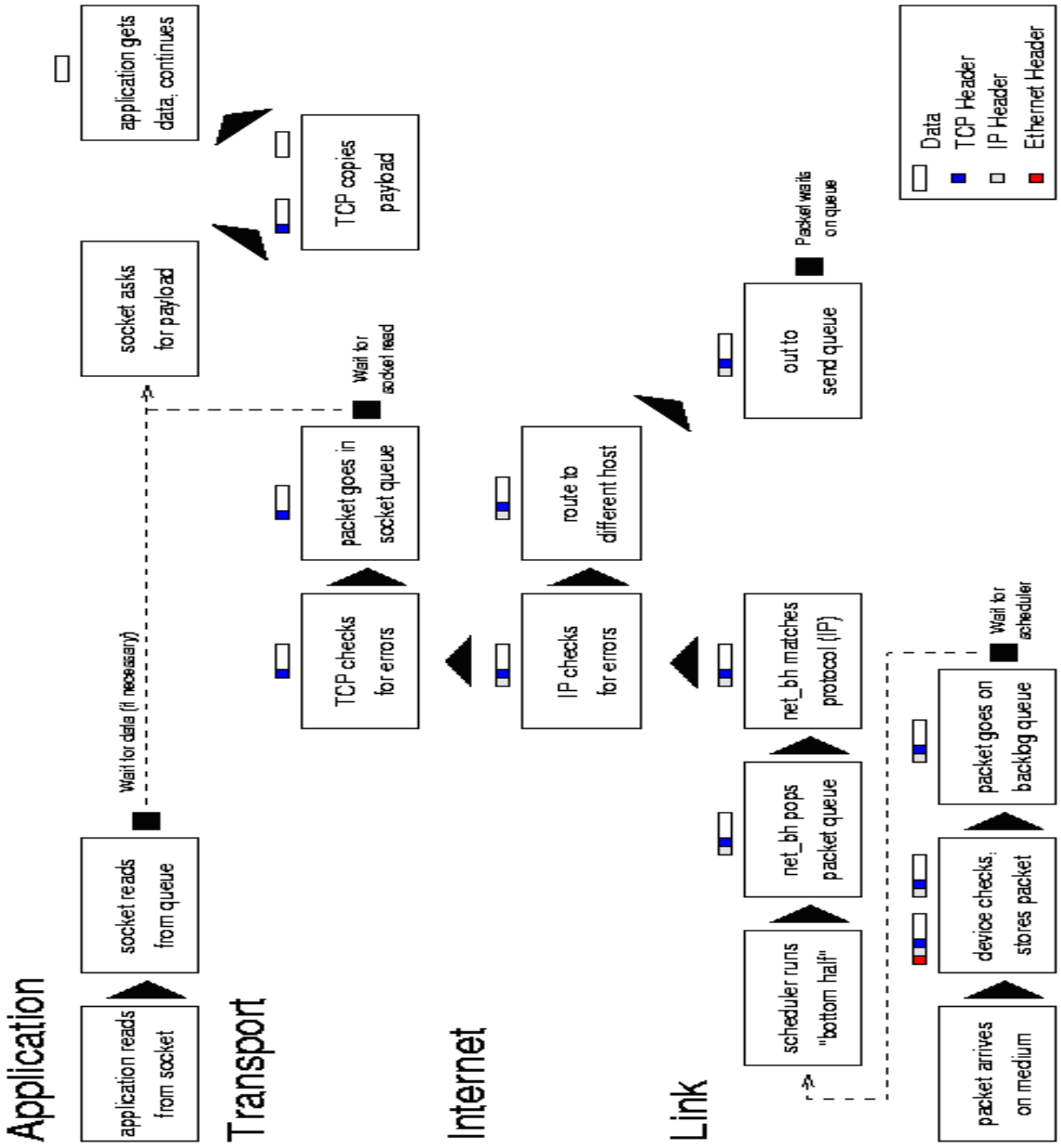

Figure 4.3: Message Reception [12]

4.4.2 Processing At Receiver Side

- Receiving a Packet
  - Wake up the receiving device (interrupt)
  - Test the medium (device)
  - Receive the link header
  - Allocate space for the packet
  - Tell the bus to put the packet into the buffer
  - Put the packet on the backlog queue
  - Set the flag to run the network bottom half when possible
  - Return control to the current process
- Running the Network "Bottom Half"
  - Run the network bottom half (scheduler)
  - Send any packets that are waiting to prevent interrupts (bottom half)
  - Loop through all packets in the backlog queue and pass the packet up to its Internet reception protocol - IP
  - Flush the sending queue again
  - Exit the bottom half
- Unwrapping a Packet in IP
  - Check packet for errors - too short? too long? invalid version? checksum error?
  - Defragment the packet if necessary
  - Get the route for the packet (could be for this host or could need to be forwarded)
  - Send the packet to its destination handling routine (TCP or UDP reception, or possibly retransmission to another host)
- Accepting a Packet in TCP

  - Check sequence and flags; store packet in correct space
  - If already received, send immediate ACK and drop packet
  - Determine which socket packet belongs to
  - Put packet into appropriate socket receive queue
  - Wake up and processes waiting for data from that socket

- Reading from a Socket
  - Wake up when data is ready (socket)
  - Call transport layer receive function
  - Move data from receive queue to user buffer (TCP/UDP)
  - Return data and control to application (socket)

4.4.3 List of Functions

Following table gives information about few functions that are invoked when packet is recieved.Table shows the function name , What they do and location of those functions in the linux kernel.

Table 4.2: Functions invoked during message reception

Function Name	**What they do?**	**Location in the Kernel**
DEVICE_rx()	performs status checks to make sure it should be receiving calls dev_alloc_skb() to reserve space for packet gets packet off of system bus	drivers/net/DEVICE.c
ip_rcv()	examines packet for errors:invalid checksum invalid length (too short or too long) and incorrect version (not 4) defrags packet if necessary calls ip_route_input() to route packet	net/ipv4/ip_input.c
tcp_recvmsg()	checks for errors wait until there is at least one packet available calls cleanup_rbuf() to release memory and send ACK if necessary	net/ipv4/tcp.c
tcp_data()	calls tcp_data_queue() to queue packet	net/ipv4/tcp_input.c
tcp_data_queue()	if packet is out of sequence: if old, discards immediately else calculates appropriate storage location calls _skb_queue_tail() to put packet in socket receive queue	net/ipv4/tcp_input.c
inet_recvmsg()	extracts pointer to socket sock checks socket to make sure it is accepting	net/ipv4/af_inet.c
sock_read()	sets up message headers returns sock_recvmsg() with result of read	net/socket.c

Chapter 5

Implementation and Results

5.1 Implementation Setup

For implementation fedora-10, kernel version-2.6.27 and i686 architecture has been used. To implement approach-V few changes in TCP/IP stack in linux kernel has been made and programs at sender side and receiver side are also developed. After making changes to the linux kernel it is necessary to build and install the kernel. Following subsections give the details of changes made in kernel, steps to download and install the kernel and steps to recompile the kernel.

5.1.1 Changes made in TCP/IP stack of linux kernel

Following files in the TCP/IP stack has been modified.

- `/kernel-2.6.27/linux-2.6.27.i686/include/linux/ip.h`

  This file contains the structure of IP header. Modification to that structure has been made to include our data in the options field and to divide identification field in two parts.

- `/kernel-2.6.27/linux-2.6.27.i686/include/net/ip.h`

  This file contains declaration and definition of functions which are used at network layer when packet is dent or received. Here, we have declared our function named ip_my_ident() which uses our logic of identification field.

- `/kernel-2.6.27/linux-2.6.27.i686/net/ipv4/inetpeer.c`

This file contains definition of our function named ip_my_ident(). This function contains the main logic of identification field.

- `/kernel-2.6.27/linux-2.6.27.i686/net/ipv4/ip_output.c`

  This file will be executed when packet comes to IP layer from transport layer. In this file ip_build_and_send_pkt() is there which assigns values to the IP header field. Here, in this function we have called our function named ip_my_ident().

- `/kernel-2.6.27/linux-2.6.27.i686/net/ipv4/ip_input.c`

  This file will be executed when packet comes from data link layer to IP layer. This file contains ip_rcv function which is main IP receive routine. Few changes has been made to implement our approach.

5.1.2 Steps to download and install kernel source code

There are 3 basic steps involved in installing the kernel source [6].

1. Download the desired kernel source (matching your current kernel if required)
2. Installing the SRC.RPM package
3. Using rpmbuild to prepare the source into a usable state

Following two commands will download the latest kernel for your fedora.

- yum install yum-utils
- yumdownloader –source kernel
- Kernel will be stored in root directory.

  ex:for fedora 10 /root/kernel-2.6.27.41-170.2.117.fc10.src.rpm

- Then enter following command.

  rpm -ivh kernel-2.6.27.41-170.2.117.fc10.src.rpm

- It may show you that tools like xmlto etc are missing you can downlaod it using normal yum command

  for ex: yum install xmlto

- After installing that again enter following command.

  rpm -ivh kernel-2.6.27.41-170.2.117.fc10.src.rpm

- Code will be saved in /root/rpmbuild/BUILD/kernel-2.6.27/linux-2.6.27.noarch/net

5.1.3 Steps to recompile kernel

After getting the source and installing kernel following are the steps for recompilation of linux kernel [8]:

- Prepare the kernel source tree using the following commands:
  - `cd ~/rpmbuild/SPECS`
  - `rpmbuild -bp --target=$(uname -m) kernel.spec`
  - The kernel source tree is now located in the

    `~/rpmbuild/BUILD/`
    `kernel-2.6.27.41-170.2.117.fc10/linux-2.6.27.41-170.2.117.fc10.<arch>`
    `directory.`

- Configure kernel options:
  - Change to the kernel source tree directory:

    `cd ~/rpmbuild/BUILD/kernel-2.6.27.41-170.2.117.fc10/linux-2.6.27.41-170.2.1`

  - Select the desired configuration file from

    `~/rpmbuild/BUILD/kernel-2.6.$ver/linux-2.6.$ver.$arch/configs.`

  - Copy the desired config file to

    `to ~/rpmbuild/BUILD/kernel-2.6.$ver/linux-2.6.$ver.$arch/.config:`
    `cp configs/<desired-config-file> .config`

- Run the following command.

  `make oldconfig`

Then run the following command, selecting and saving the desired kernel options from the text-based UI:

```
make menuconfig
```

For GUI run the following command:

```
make xconfig
```

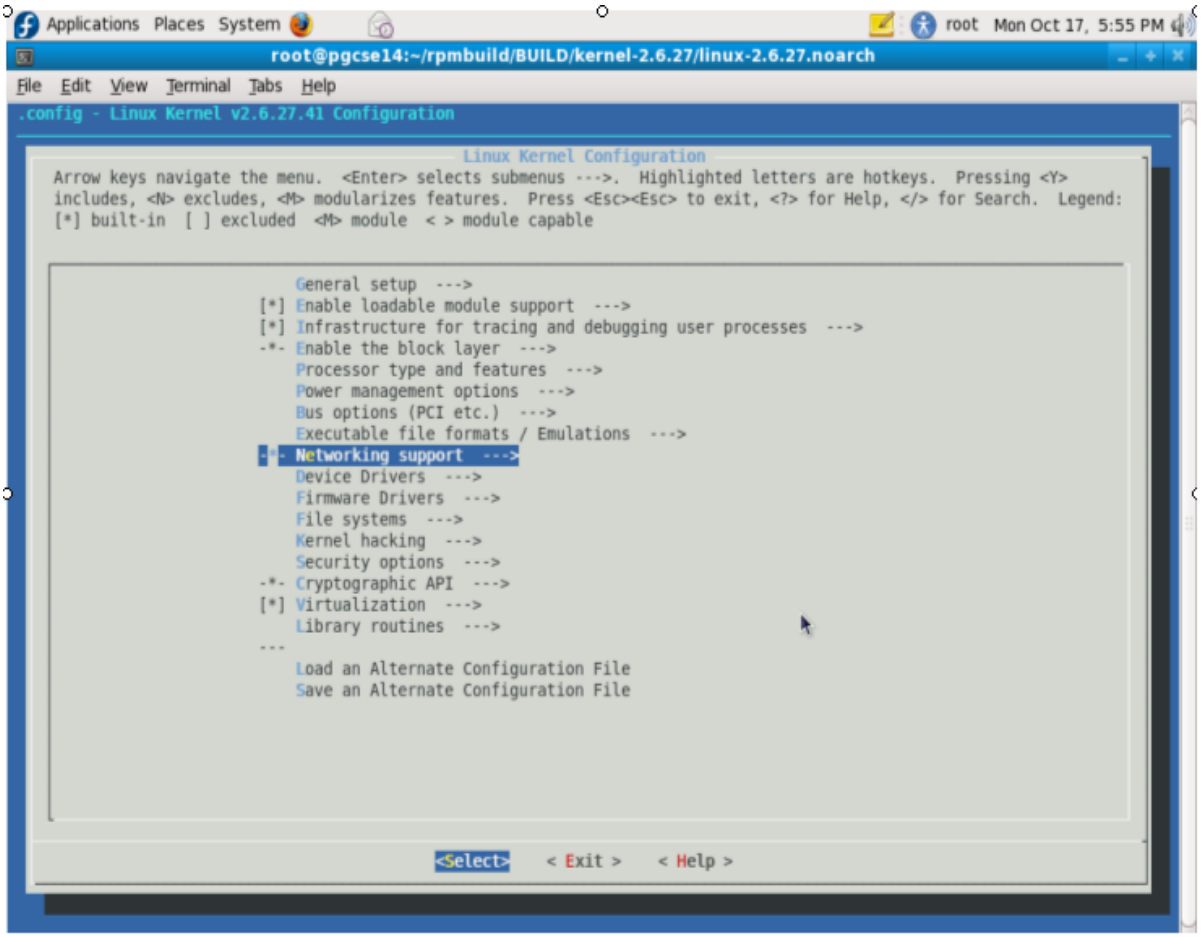

Figure 5.1: GUI for Linux kernel Configuration

- Then Build the kernel using following command: To build all kernel flavors:

```
rpmbuild -bb --target=‘uname -m‘kernel.spec
```

- Then run the following command to install the kernel. This step actually installs the new kernel into the running system.

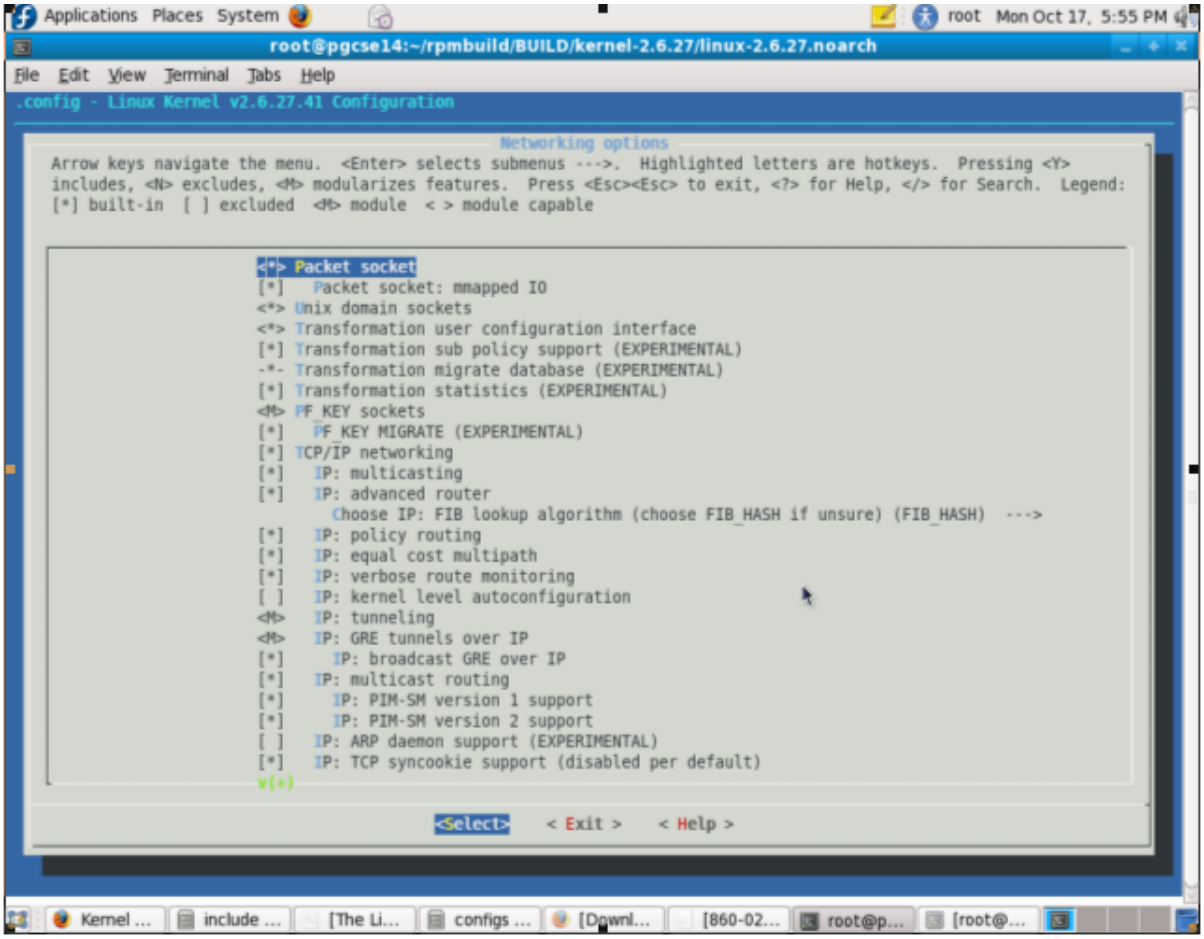

Figure 5.2: GUI for Linux kernel Configuration

```
su -c "rpm -ivh --force
$HOME/rpmbuild/RPMS/<arch>/kernel-<version>.<arch>.rpm"
```

5.2 Results

Using approach-V we have developed programs at sender side and receiver side. In this section results of those programs which has been implemented using approach-V has been shown. We have used RSA encryption algorithm to provide security to our data.

- Program at sender side sends data as shown in the figure 3.19 using approach-V. One file contains confidential data which sender wants to send hiddenly and another file contains non confidential data which sender wants to send in the payload. Sender will specify number of hidden bytes(nbyte) he wants to send hiddenly in each packet and then sends packets.

- Program at receiver side reads packet data as shown in the figure 3.21 using approach-V. When packet come to the receiver receiver will check whether hidden data is there in the packet or not. If hidden data is there in the packet receiver will write nbyte of hidden data in the file and data of payload in anothe file. Figure 5.3 shows the snapshot of output of program at receiver side. At the same time receiver will create one log file which shows all the information about packet. Log file contains details of IP header,hidden data in the options field, TCP header and data in the payload. Figure 5.4 shows the log file. Figure 5.5 shows the log file for encrypted data. Here, the IP address of sender is 10.1.3.18 and IP address of the receiver is 10.1.3.14.

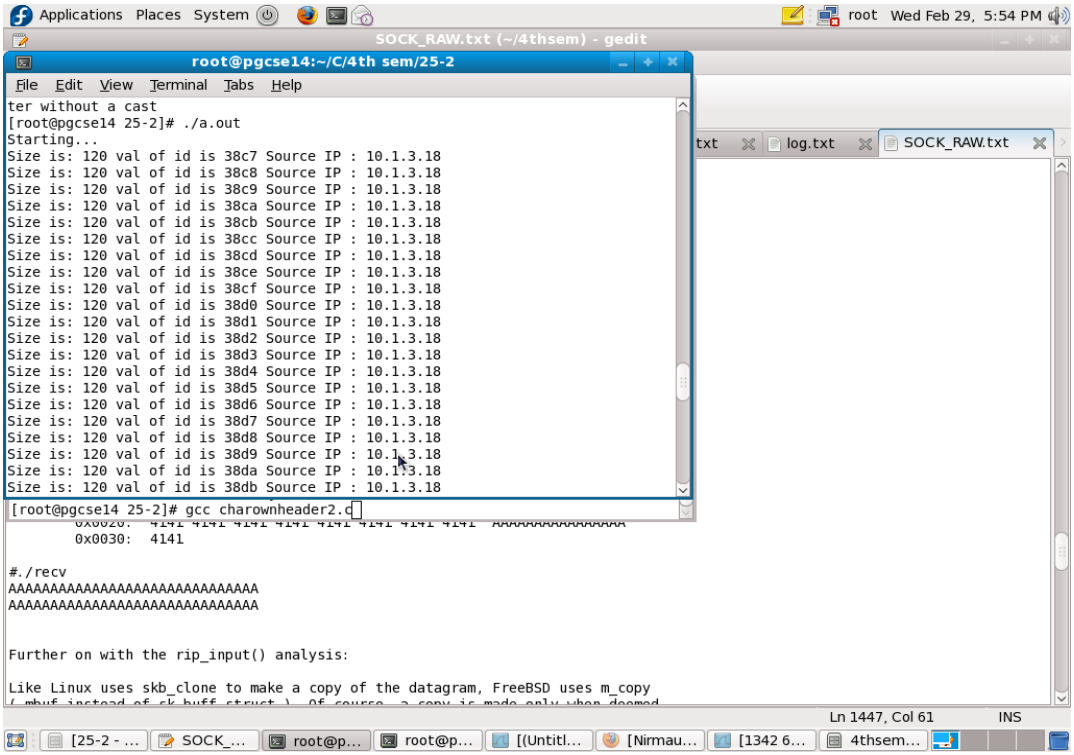

Figure 5.3: Output of program at receiver side

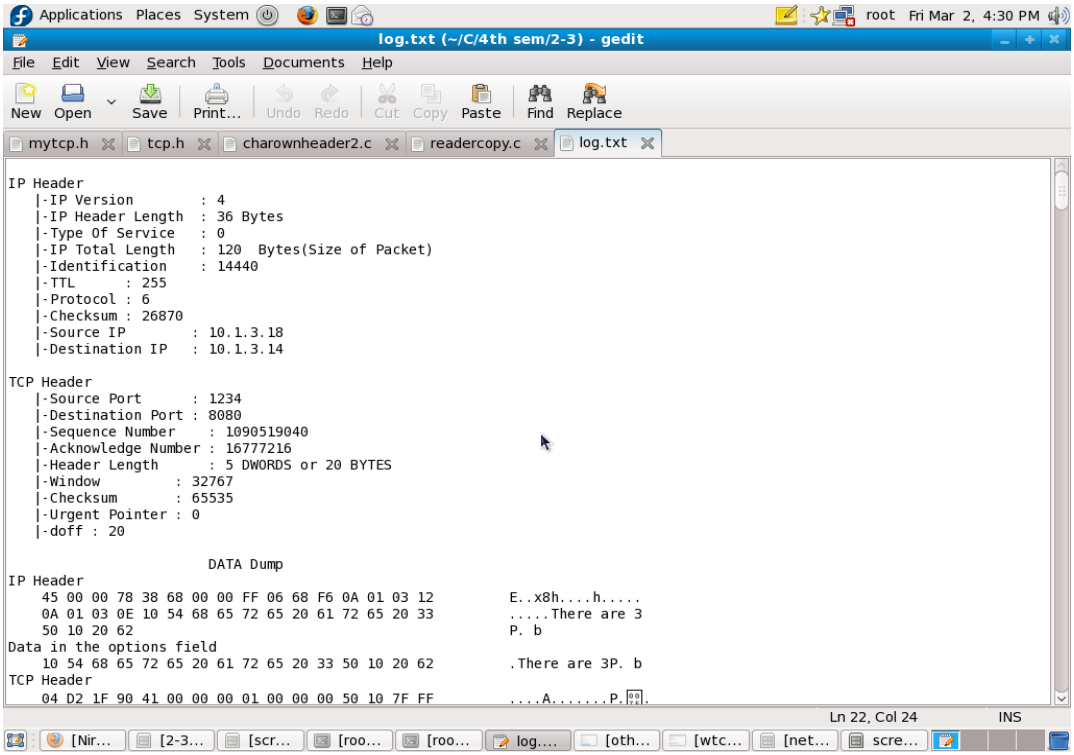

Figure 5.4: Log file at receiver side

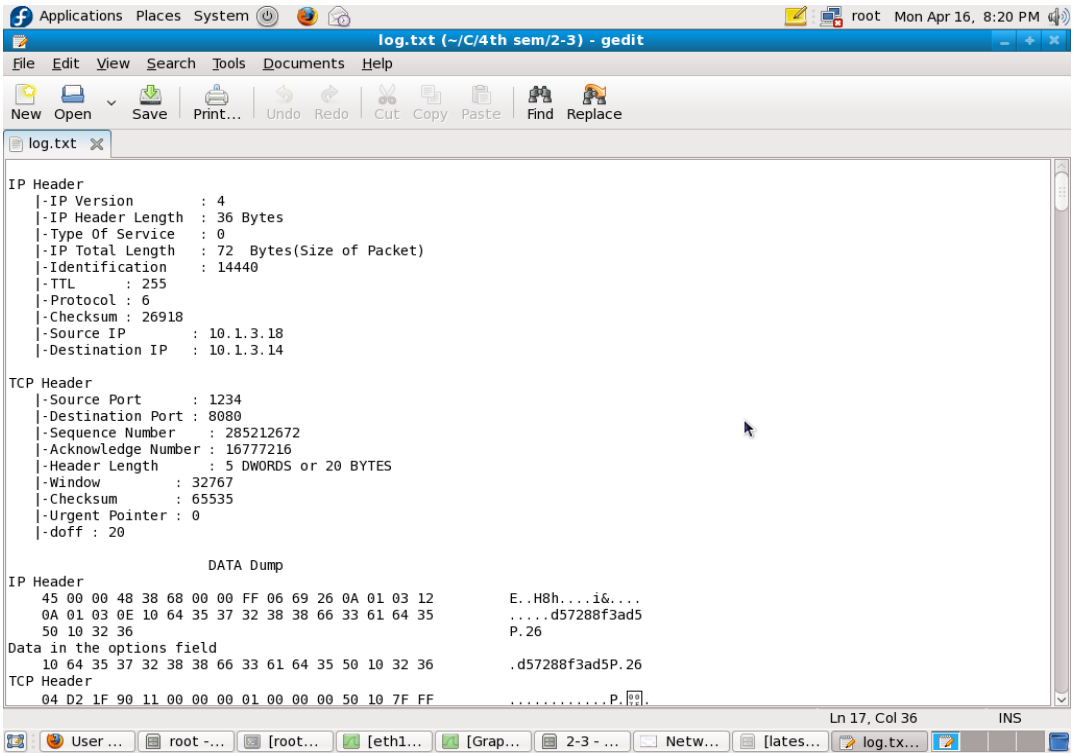

Figure 5.5: Log file at receiver side for encrypted data

- I have captured packets sent by sender using sniffer program and two packet sniffing tools tcpdump and wireshark. Figures 5.6, 5.7, 5.8 shows the output of them respectively.

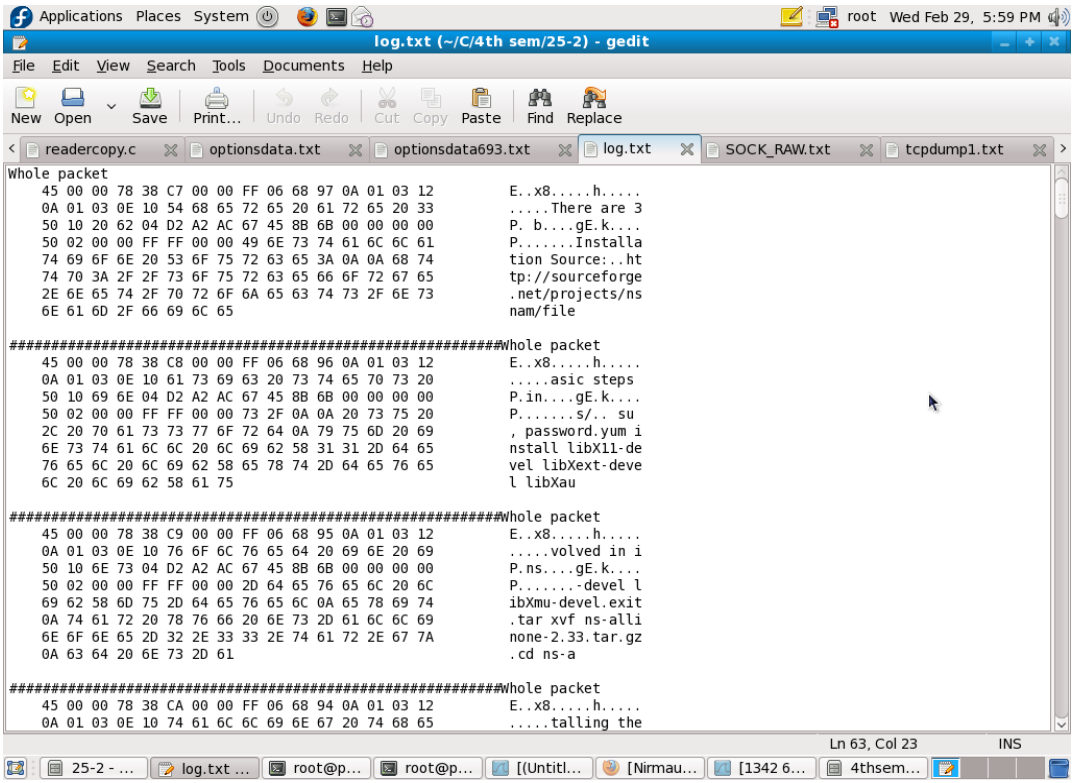

Figure 5.6: Output of sniffer program

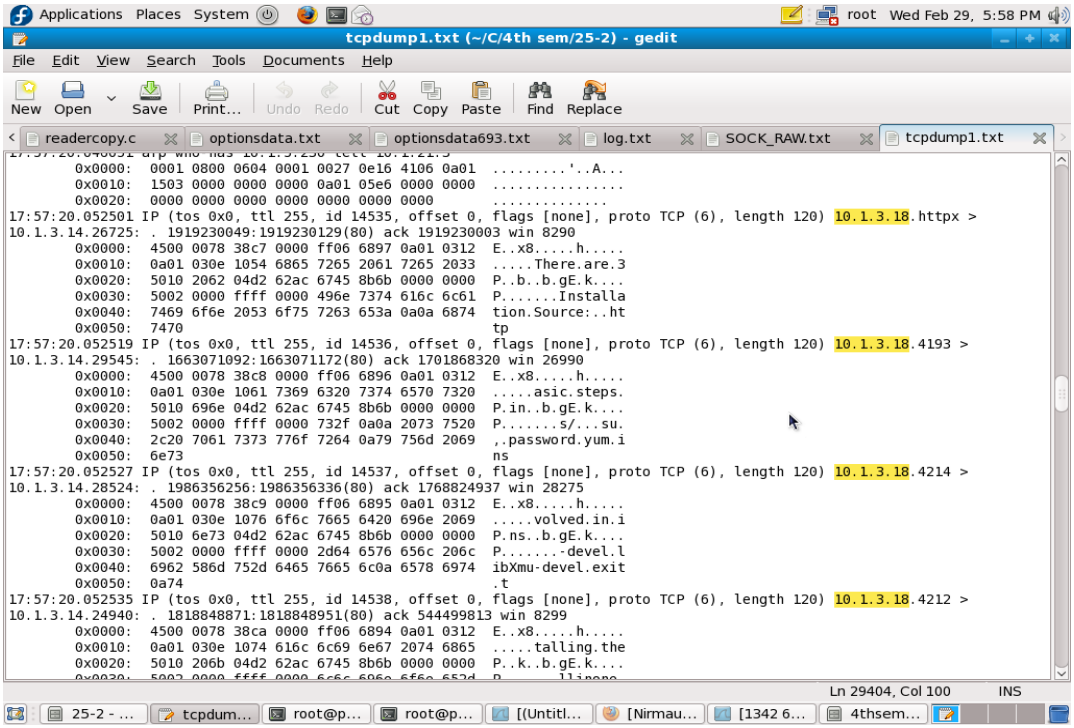

Figure 5.7: Output of tcpdump

- Figure 5.8 shows output of wireshark when sender is 10.1.3.18 which sends hidden data in IP header using approach-V. Figure 5.9 shows output of wireshark when sender sends normal packet which does not contain hidden data. From these two figures we can say that wireshark decodes our packet which contains hidden data as normal packet which does not contain hidden data. It means that wireshark is not able to detect that our packet contains hidden data.

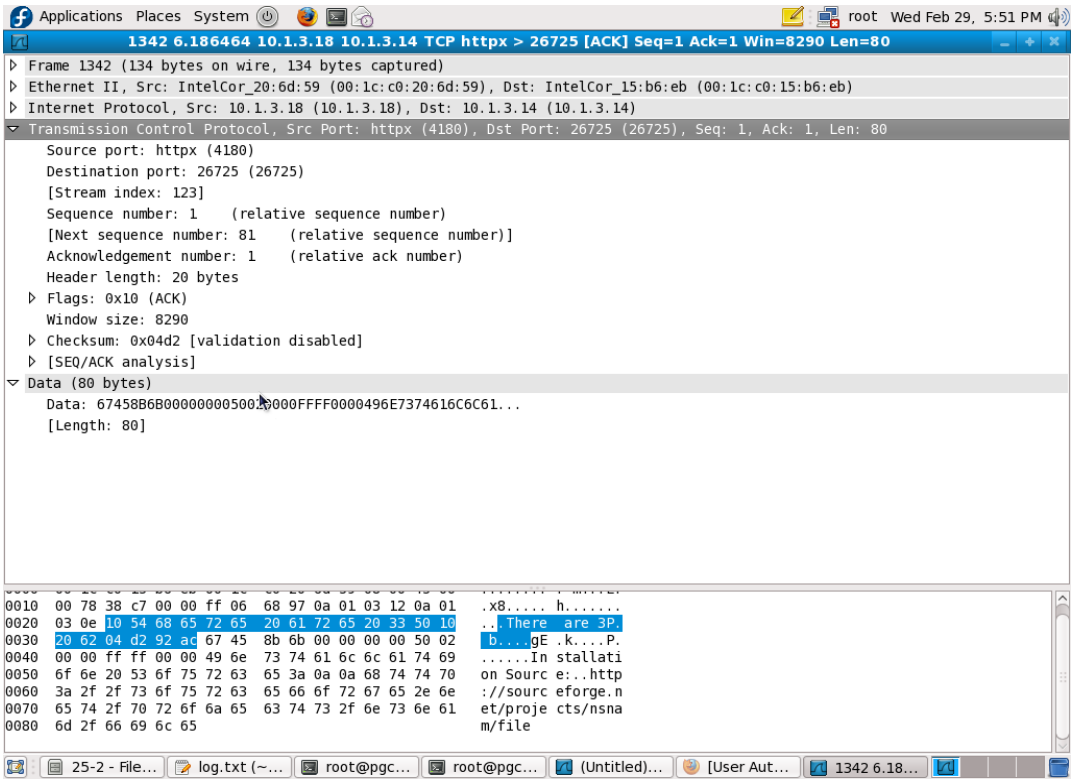

Figure 5.8: Output of wireshark

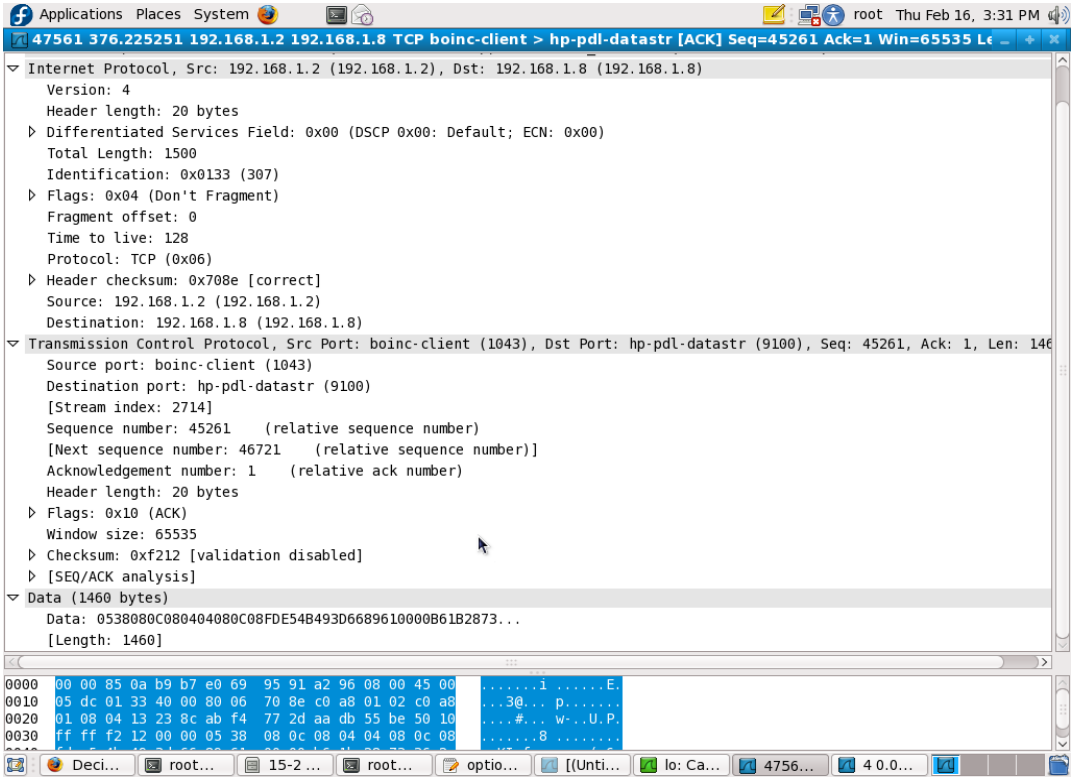

Figure 5.9: Output of wireshark

Figure 5.10 and 5.11 shows the output of tcpdump and wireshrk for encrypted data.

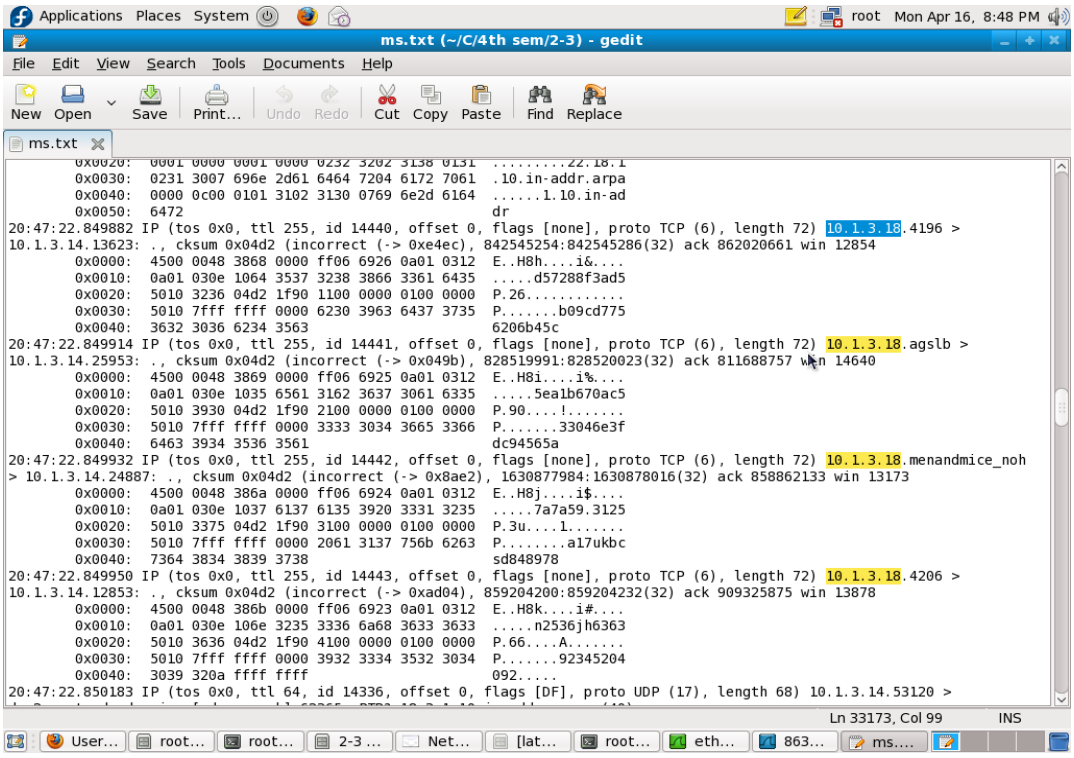

Figure 5.10: Output of tcpdump for encrypted data

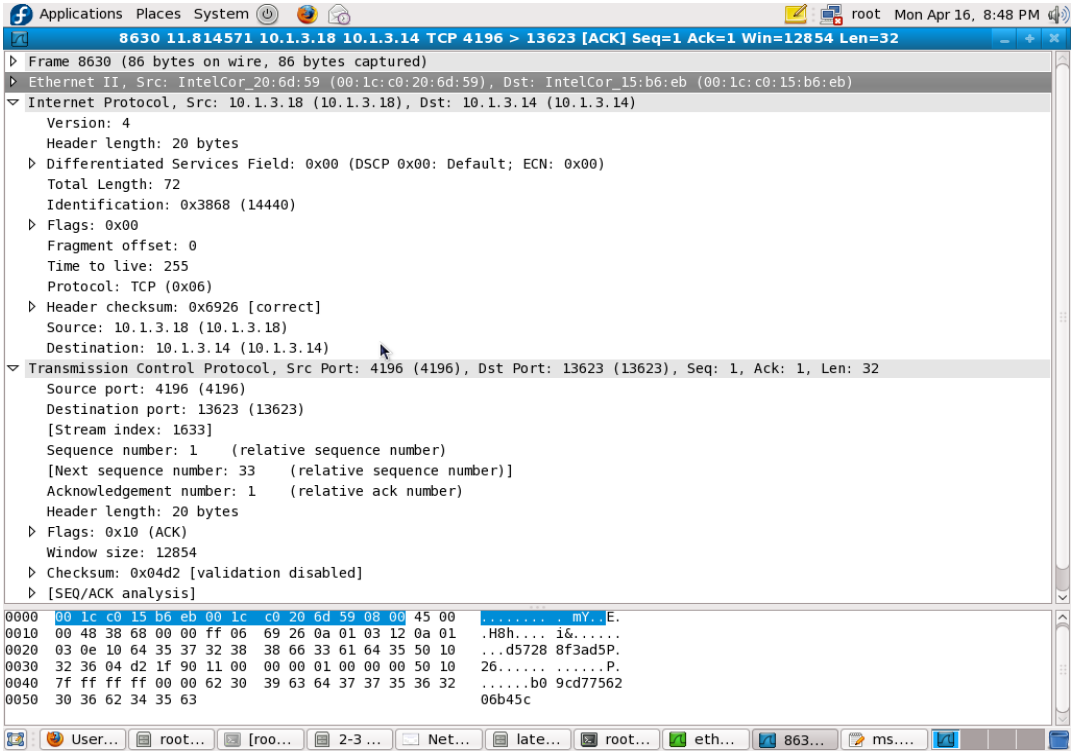

Figure 5.11: Output of wireshark for encrypted data

- Figure 5.12 shows the output of receiver program when sender sends packets using Timestamp options. Sender has used 24 bytes for timestamp option. So value of ihl field is 11(20+24=44/4=11) because total size of IP header is 44 in which 20 bytes of standard header and 24 bytes for timestamp.

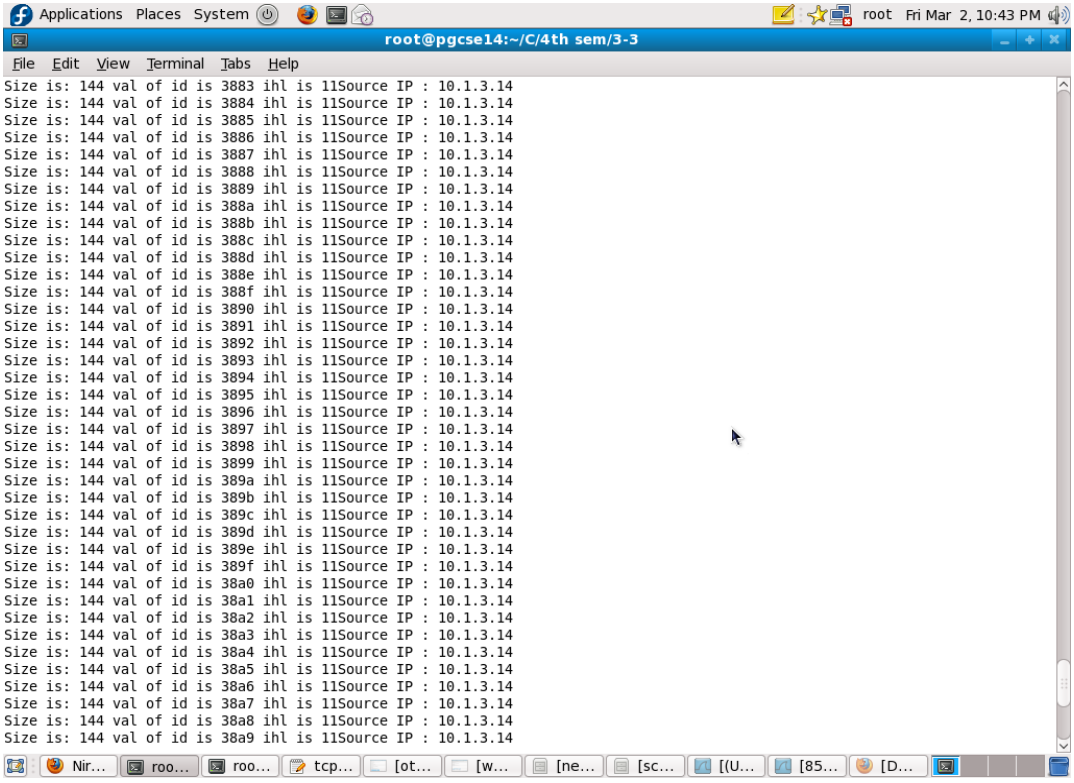

Figure 5.12: Output of program at receiver side

- Figure 5.13 and 5.14 shows the output of tcpdump and wireshark respectively when sender sends hidden data using approach-V with timestamp option.

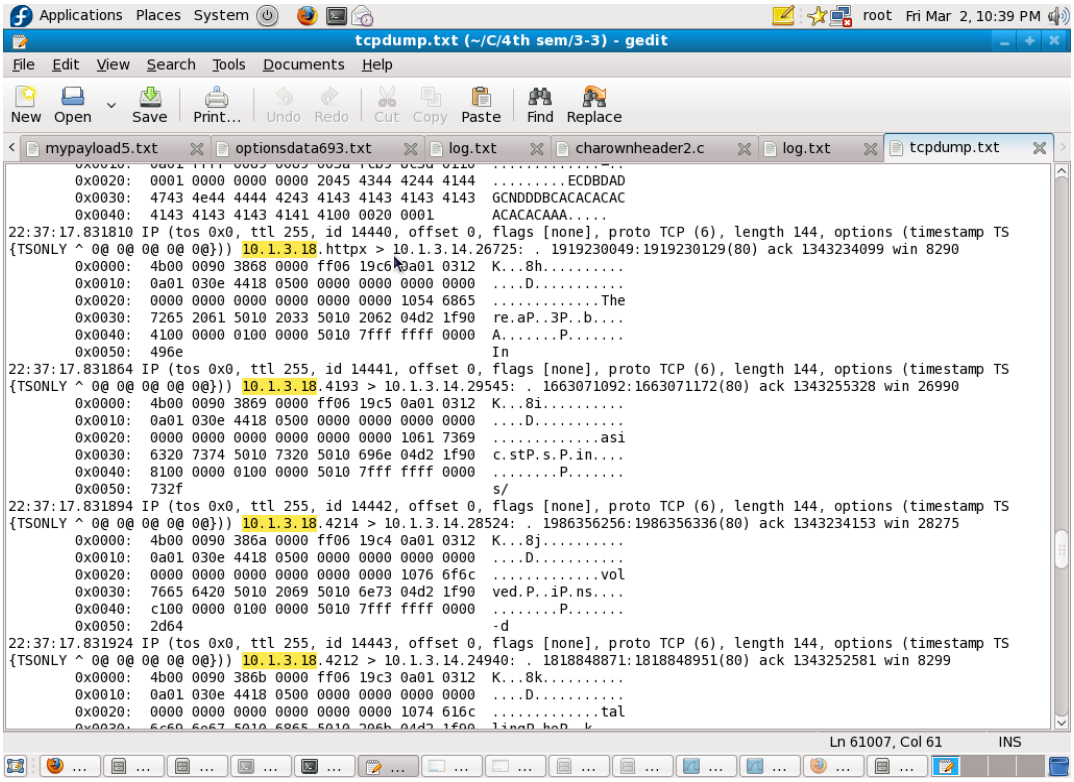

Figure 5.13: Output of tcpdump using Timestamp

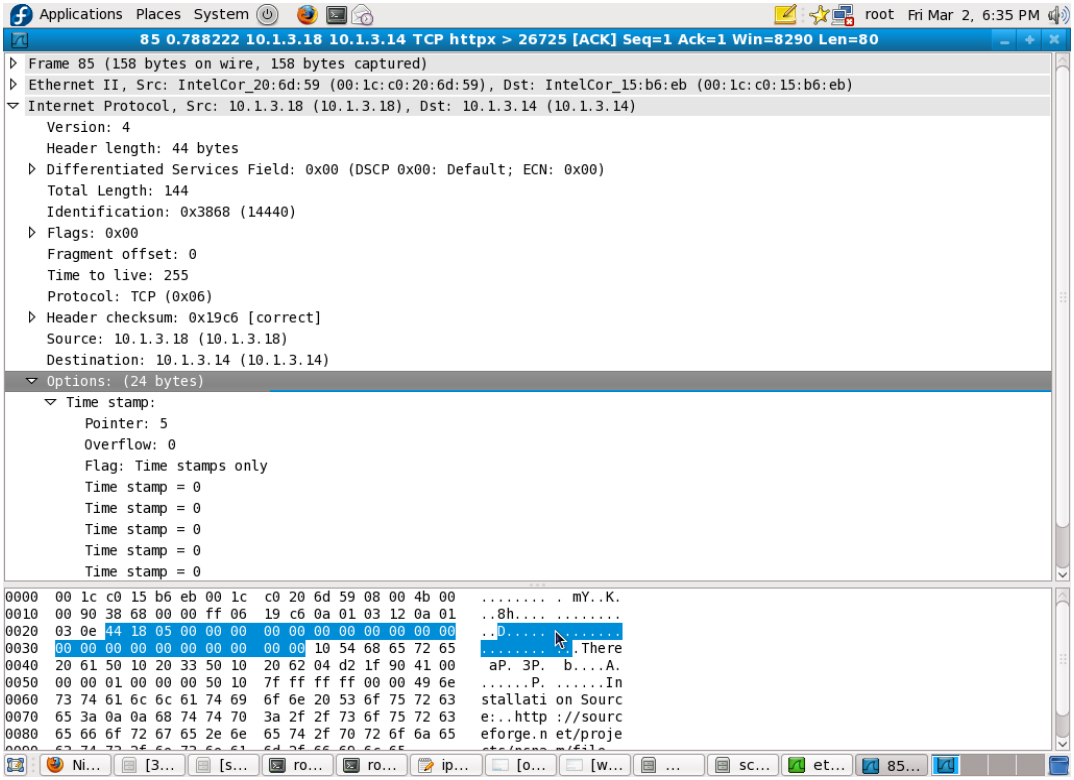

Figure 5.14: Output of wireshark using Timestamp

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this dissertation I have proposed an approach to watermark TCP/IP packets. I have used options field of IP header for that. After implementation and analysis of result we can say that wireshark and tcpdump is not able to identify that our packet contains hidden data. Most of the intruders uses wireshark and tcpdump sniffing tools. So we can say that our approach is secure to transmit hidden data in the IP header from sender to receiver. So Watermarking in network packets can be used to send hidden information in the free space of the packet header.

6.2 Future Scope

In this thesis I have proposed method to send text files in the options field of IP header. Different compression and authentication algorithms can also be integrated with the proposed scheme. Proposed approach can be extended to send any kind of multimedia data like images, audio, video.

Appendix A

List of Publication

Maitrik Shah, Prof. Samir B. Patel, **"Network based packet watermarking using TCP/IP protocol suite"**, of the 2^{nd} International Conference on Current Trends in Technology - 'NUiCONE', organized by Institute of Technology, Nirma University, Ahmedabad, India, 8-10 December 2011.

References

[1] Craig H. Rowland *"Covert Channels in the TCP/IP protocol suite"*, Techniques for Data Hiding IBM Systems Journal Vol 35, 2003.

[2] Enrique Cauich, Roberto Gmez, Ryouske Watanabe, *"Data Hiding in Identification and Offset IP fields"*, Proc. 5th Int'l. School and Symp. Advanced Distributed Systems (ISSADS),Jan.2005 .

[3] Mrs. S.S.Sherekar, Dr. V.M.Thakare, Dr.Sanjeev Jain, *"Role of Digital Watermark in e-governance and e-commerce"*, IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.1, January 2008.

[4] Theodore G. Handel, Maxwell T Sanford, *"Data hiding in the OSI Network model"*,First International workshop on Information Hiding, May-June 1996..

[5] Wojciech Mazurczyk and Krzysztof Szczypiorski, *"Steganography in Handling Oversized IP Packets"*, Proc. Int. Conf. Multime. Inf. Netwo. Security MINS-2009.

[6] http://www.g-loaded.eu/2005/12/14/the-complete-fedora-kernel-headers/

[7] http://forums.fedoraforum.org/archive/index.php/t-101436.html.

[8] http://www.fedoraproject.org/wiki/Building-a-custom-kernel

[9] http://www.6test.edu.cn/~lujx/linux-networking/

[10] K. Ahsan and D. Kundur, *"Practical data hiding in TCP/IP"*, Proc. ACM Workshop on Multimedia Security, 2002.

[11] Zander S., Armitage G., Branch P., *"A Survey of Covert Channels and Countermeasures in Computer Network Protocols"*, IEEE Communications Surveys Tutorials, 3rd Quarter 2007, Volume: 9, Issue: 3, ISSN: 1553-877X.

[12] http://www.cs.unh.edu/cnrg/people/gherrin/linux-net.html

[13] R M Goudar, S J Wagh, M D Goudar, *"Secure Data Transmission using Steganography Based Data Hiding in TCP/IP"*, International Conference and Workshop on Emerging Trends in Technology (ICWET) TCET, Mumbai, India, 2011.

[14] http://www.linuxhowtos.org/C_C++/socket.htm

[15] Peter Jay Salzman, Ori Pomerantz, *"The Linux Kernel Module Programming Guide"*, 2004.

[16] Brian Beej Jorgensen Hall, *"Beej's Guide to Network Programming Using Internet Sockets"*, 2009.

[17] Behrouz A. Forouzan, *"TCP/IP Protocol Suite"*, 3rd edition, TMH publications, May-2005 .

[18] W. Richard Stevens, Gary R. Wright, *"TCP/IP Illustrated: The Protocols"*, volume-1, Addison-Wesley Professional

[18] W. Richard Stevens, Gary R. Wright, *"TCP/IP Illustrated: Implementation"*, volume-2, Addison-Wesley Professional

[20] http://www.ietf.org/rfc/rfc791.txt

[21] Mansfied, K Ohta, Y. Takei, N. Kato and Y. Nemoto *"Towards trapping wily intruders in large computer networks"* ,In proceedings of the second annual workshop in recent advances in intrusion detection(RAID) west lafayette, IN, sept-1999

[22] http://moss.csc.ncsu.edu/~mueller/cluster/ps3/doc/LinuxKernelOverview.html

Index