

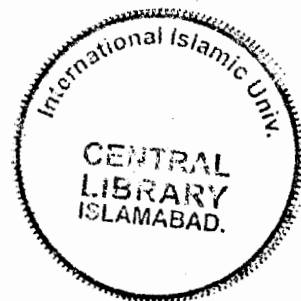
# **IPV4 / IPv6 Translation Gateway**

7-4973



**Developed By**  
**Haroon Mushtaq**  
**Umar Khan**

**Supervised By**  
**Dr. Muhammad Sher**



**Department of Computer Science**  
**Faculty of Basic and Applied Sciences**  
**International Islamic University,**  
**Islamabad**  
**(2008)**

**WITH THE NAME OF  
ALMIGHTY ALLAH,  
THE MOST BENEFICIENT,  
THE MOST MERCIFUL**

**Department of Computer Science**  
**International Islamic University Islamabad**

Date: \_\_\_\_\_

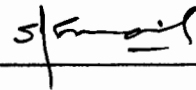
**Final Approval**

This is to certify that we have read the thesis submitted by **Haroon Mushtaq** 71-CS/MS/02 and **Umer Khan** 76-CS/MS/02. It is our judgment that this thesis is of sufficient standard to warrant its acceptance by International Islamic University, Islamabad for the degree of MS in Computer Science.


Committee:

**External Examiner**

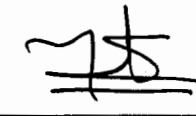
Dr. Ismail Shah  
Professor  
Department of Computer Science,  
Iqra University,  
Islamabad.

  
\_\_\_\_\_**Internal Examiner**

Mr Qaisar Javed  
Assistant Professor  
Department of Computer Science,  
Faculty of Basic and Applied Sciences,  
International Islamic University,  
Islamabad.

  
\_\_\_\_\_**Supervisor**

Dr. Muhammad Sher  
Chairman  
Department of Computer Science,  
Faculty of Basic and Applied Sciences,  
International Islamic University,  
Islamabad.

  
\_\_\_\_\_

**A dissertation Submitted To  
Department of Computer Science,  
International Islamic University, Islamabad  
As a Partial Fulfillment of the Requirement for the Award of the  
Degree of MS in Computer Science.**

**Dedicated To**  
**The Most Beloved Hazrat Muhammad (SAW),**  
**Our Parents,**  
**Our Motherland**  
**And Our Family**

## **Declaration**

We hereby declare that this Research "*IPv4 / IPv6 Translation Gateway*" neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this research with the accompanied report entirely on the basis of our personal efforts, under the proficient guidance of our teachers especially our supervisor **DR. Muhammad Sher**. If any part of the system is proved to be copied out from any source or found to be reproduction of any project from any of the training institute or educational institutions, we shall stand by the consequences.

**Haroon Mushtaq**

**71-CS/MS/02**

**Umer Khan**

**76-CS/MS/02**

## Acknowledgement

We are grateful to Allah Almighty, the most Merciful and Beneficent, for blessing us with the knowledge, courage and determination to complete this research project. We are also whole heartedly thankful to **Dr Muhammad Sher** for his priceless guidance and knowledge sharing. His continuous efforts to help us complete this project are valuable to us and we will always be thankful to him.

We want to thank all the respected faculty members for teaching us in a very professional way.

We want to show our regards and greetings to all of our class fellows and friends with whom we spent a very nice and memorable time.

Above all we owe every thing to our beloved parents and our loving families for their love, guidance and their moral and financial support.

**Haroon Mushtaq**

**71-CS/MS/02**

**Umer Khan**

**76-CS/MS/02**

## **Project In Brief**

<b>Project Title:</b>	IPv4 / IPv6 Translation Gateway
<b>Undertaken By:</b>	Haroon Mushtaq Umar Khan
<b>Supervised By:</b>	Dr. Muhammad Sher
<b>Start Date:</b>	September 2003
<b>Completion Date:</b>	March 2008
<b>Tools &amp; Technologies</b>	Visual Basic 6.0 (To Develop Simulation Software)
<b>Documentation Tools</b>	Microsoft Word XP Microsoft Visio XP Microsoft Power Point Microsoft Project 2000 Rational Rose 98
<b>Operating System:</b>	Windows XP Professional
<b>Systems Used:</b>	Pentium III (Celeron) 700 MHz Pentium II (Celeron) 333 MHz. Pentium 4 (Centrino) 1.6 GHz



## **Abstract**

The basic domain of the research conducted is the network communication between nodes based on different protocols. With the advent of the new protocol, IPv6, issues were raised as to how communication between the two protocols (IPv4 and IPV6) will be made until all the devices have shifted to IPv6. All proposed methods have their pros and cons, with respect to performance, efficiency, compatibility, scalability. There is need to develop a translation gateway that is compatible with most (if not all) network devices, which is efficient, scalable and does not require major hardware up gradations to be implemented.

We have proposed an efficient technique to calculate the TCP Checksum, by avoiding recalculating same values over and over again. This is done by maintaining Partial Checksum Values in local memory. This technique reduces the work load and CPU utilization or the Translation Gateway (Router), thus increasing its performance and efficiency.

# Table of contents

<b>1. Introduction</b>	1
1.1 IPv4	1
1.1.1 Historic Background	1
1.1.2 Growth of Internet	2
1.1.3 Technical Specifications	3
1.1.3.1 The IPv4 Header	3
1.1.4 Drawbacks and Problems in IPv4	5
1.1.4.1 Address Space Exhaustion	5
1.1.4.2 Increment of Routing Information	7
1.1.4.3 Cost of Configuration	7
1.1.4.4 Security Issues	7
1.1.4.5 Quality of Service (QoS)	8
1.2 IPv6	9
1.2.1 Introduction	9
1.2.2 Historic Background	10
1.2.3 Technical Specifications	10
1.2.3.1 The IPv6 Header	10
1.2.3.2 IPv4 and IPv6 Header Differences	11
1.2.3.2.1 IPv6 Address Representation	14
1.2.4 Improvements / Benefits of IPv6	14
1.3 Moving from IPv4 to IPv6	15
1.4 Outline of the Thesis	16
<b>2. State of the Art Review</b>	18
2.1 Transition Definitions	18
2.2 Transition Mechanisms	19
2.2.1 Dual – Stack	19
2.2.2 Tunnelling	19
2.2.3 Translation	21
2.2.3 Translation	21
2.3 Enhancements Required	26
<b>3. Requirements Analysis</b>	27
3.1 Problem Analysis	27
3.2 Use Case Analysis	31
3.2.1 Translation from IPv4 to IPv6	32
3.2.2 Translation from IPv6 to IPv4	35
<b>4. Proposed Architecture</b>	38
4.1 Overview	38
4.2 Components of the Proposed System	38
4.3 Data Flow across the Translation Gateway	40
4.4 Our Approach	41
4.4.1 Translation from IPv6 to IPv4	41
4.4.2 TCP Checksum Recalculation	42
4.4.3 TCP Checksum Calculation	42
4.4.4 Partial Source Address Calculation for TCP Checksum Calculation	43

4.4.5	Partial Updated TCP Checksum Calculation Formula .....	43
4.4.6	TCP Checksum Calculation for Individual Packets .....	44
4.4.7	TCP Checksum Calculation for Fragmented Packets .....	45
4.5	Conclusion .....	48
<b>5.</b>	<b>Implementation .....</b>	<b>50</b>
5.1	Pseudo Code of Packet Translation Functions .....	50
5.1.1	Convert IPv4 Packets to IPv6.....	50
5.1.2	Calculate TCP Checksum of the converted IPv6 Packe .....	51
5.1.3	Calculate Partial Checksum .....	52
5.2	Class Modules .....	52
5.2.1	clsIPV4.....	52
5.2.2	clsIPv6.....	53
5.2.3	clsTCP .....	53
5.2.4	clsTCP4Pseudo.....	54
5.2.5	clsTCP6Pseudo.....	54
5.3	Utility / Helping Functions .....	54
5.3.1	IntToBin .....	54
5.3.2	BinToInt .....	55
5.3.3	Dec2Bin .....	55
5.3.4	Bin2Hex .....	57
5.3.5	AddBinaryNumbers .....	60
5.3.6	CalculateOnesComplement.....	61
5.3.7	ConvertTo16bitWords .....	61
5.3.8	PadZerosToLeft .....	62
5.4	Main Coding Loop .....	63
5.4.1	ReadAndLoadInput .....	64
5.4.2	Fill_IPV4_Data .....	64
5.4.3	ConvertIPV4ToIPv6 .....	64
5.4.4	CalculateTCPChecksum .....	64
5.4.5	CalculatePartialCheckSum.....	65
5.4.6	CalculateSumOfTCPPseudoHeader .....	66
5.4.7	AddPartialCheckSumEntry .....	67
<b>6.</b>	<b>Results .....</b>	<b>68</b>
6.1	Main Screen .....	68
6.2	Result Screen .....	69
6.3	Comparison based on Number of Packet (No Fragmentation) .....	72
6.3.1	Sample Size for Test .....	72
6.3.2	Comparison Results .....	72
6.3.3	Comparison Graph .....	74
6.4	Comparison based on Number of Fragments Per Packet (Fixed No. of Packets) .....	75
6.4.1	Sample Size for Test .....	75
6.4.2	Comparison Results.....	75
6.4.3	Comparison Graph .....	76
6.5	Comparison based on Number of Packet (Fixed No. of Fragments) .....	76
6.5.1	Sample Size for Test .....	76
6.5.2	Comparison Results.....	77
6.5.3	Comparison Graph .....	77

6.6 Result Analysis and Conclusion .....	78
<b>7. Conclusion and Future Enhancements .....</b>	<b>79</b>
7.1 Conclusion .....	79
7.2 Future Enhancements .....	79
<b>Appendix A References.....</b>	<b>80</b>
<b>Appendix B Source Code of TCP Checksum Calculation in KAME Project.....</b>	<b>82</b>

# **CHAPTER 1**

## **INTRODUCTION**

## 1. Introduction

Internet Protocol (IP) defines the set of rules which enables computers and other network devices to communicate with each other over the internet. Currently IPV4 is being used, which provides an address space that contains only 4 billion unique addresses. Thus, the number of devices that can be assigned globally unique addresses is limited. The usage of internet has been growing exponentially ever since its advent and thus more and more new devices are connecting to the internet, thus requiring globally unique IP addresses. The IT Community has come up with tweaks and workarounds to temporarily overcome the shortage of globally unique IP addresses, but IPv6 provides the ideal and long term solution to the problem, by offering an address space that is virtually unlimited.

But everyone cannot shift from IPv4 to IPv6 right away. It is an enormous task to move all the network devices in the world from one base structure to the other. This process may take decades to accomplish. Therefore, until all the network devices of the world have been shifted to IPv6, there needs to be a process or technique that allows network devices based on IPv4 to communicate with the ones based on IPv6 and vice versa. For this purpose, a number of methods like NAT have been tested and used but each one has its limitations and thus there is still need to develop that perfect technique which is transparent, scalable, compatible, efficient and cost effective.

### 1.1 IPv4

This section discusses IPv4 in terms of its historic background as well as its technical aspects. An overview of the problems in IPv4 have also been highlighted.

#### 1.1.1 Historic Background

The Internet Protocol was invented by Robert Kahn and Vinton Cerf and was originally known as Kahn-Cerf protocol, named after its inventors. Kahn was hired by the Information Processing Techniques Office (IPTO) in 1972, to work in the field of networks, where he invented a technique that made possible for 40 different computers to connect to the Advanced Research Projects Agency Network (ARPANET). This success made his work and the concept of networks known to people all over the world and also encouraged him to start working on an open architecture network model that would allow any computer to connect to any other computer, irrespective of its hardware and software specifications. In 1973, Kahn

was joined by Vinton Cerf, who was a former scientist at the Defense Advanced Research Projects Agency (DARPA). Their combined studies led to the development of what we call TCP/IP.

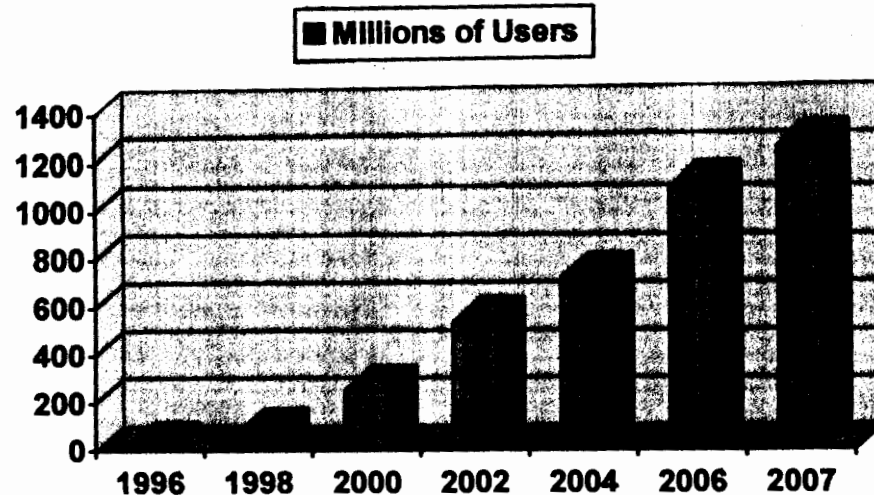
Their first draft, "A protocol for Packet Networking Interconnection" [1], was presented in 1974 at the IEEE Transactions of Communications Technology. The first technical specification of TCP/IP was presented in December 1974 by Kahn and Cerf.. They kept on working on TCP/IP and designed a few versions, amongst which the fourth version (IPv4) is what has been used for decades and will be used in years to come. The beauty of IPv4 is that it has managed to connect systems based on a wide range of network technologies. Most of the technologies being widely used today were not invented when IP was developed, but still the design of IPv4 is so strong that there is not even a single technology that IPv4 fails to support [2].

### 1.1.2 Growth of Internet

Internet has made real what in the 1970's that visionary of the communications Marshall McLuhan (1911-1980) called the "Global Village". In a matter of on a few years, the Internet has consolidated itself as a very powerful platform that has changed the way we do business, and the way we communicate with each other. The Internet, as no other medium, has given a "Globalized" dimension to the world. It is the Universal source of information.

Internet is actually the most democratic of all the mass media. With a very low investment, anyone can have a web page in Internet. This way, almost any business can reach a very large market, directly, fast and economically, no matter the size or its location of your business. With a very low investment almost anybody that can read and write can have access to the World Wide Web. This is a big achievement and we owe it to the efforts of Robert Kahn and Vinton Cerf. Internet has been growing ever since it was introduced to the world.

The following graph shows the statistics of the growth of the internet, in terms of the number of users using it:



**Fig 1-1 Stats of the growth of Internet**

The computer market has been the driver of the Internet growth. It comprises the Internet and countless smaller intranets that are not connected to the Internet. The main goal is to connect together the computers in government, business, universities, and schools. The growth of the computer market has been exponential. The future growth of the computer market is not expected to be exponential; instead other markets are expected to represent the largest growth of the Internet.

Since the Internet suite of protocols was developed, a lot of advancements have been made in the communication field. High-speed backbones using ATN and SONET technologies, real time voice and video communication over the web, ecommerce, virtual LANs using switching technologies, are only a few examples.

More and more devices are connecting to the internet and using it as a base to communicate. Cellular and multimedia devices are more and more getting closer to being computers themselves. The inventions of high definition televisions and nomadic personal computers are only a few examples of technologies that require a standard protocol, for them to work.

To cope up with the fast track growth of the internet and the usage of the new technologies that are being enhanced rapidly, there needs to be a robust, efficient and simple solution that brings out the best of all the latest technologies.



### 1.1.3 Technical Specifications

In this section, we will discuss the various technical aspects of IPv4.

#### 1.1.3.1 The IPv4 Header

The IPv4 packet header consists of 20 bytes of data. An option exists within the header that allows further optional bytes to be added, but this is not normally used (with the occasional exception of something called "Router Alert"). The full header is shown below:

Version	HL	Type of Service	Total Length	
Identification			Flags	Fragment Offset
Time to Live	Protocol		Header Checksum	
Source Address				
Destination Address				
Options			Padding	

**Fig 1-2 The IPv4 Header**

The header fields [3] are discussed below:

- **Version** (always set to the value 4 in the current version of IP)
- **IP Header Length** (number of 32-bit words forming the header, usually five)
- **Type of Service (ToS)**, now known as **Differentiated Services Code Point (DSCP)** (usually set to 0, but may indicate particular Quality of Service needs from the network, the DSCP defines the way routers should queue packets while they are waiting to be forwarded).
- **Size of Datagram** (in bytes, this is the combined length of the header and the data)
- **Identification** ( 16-bit number which together with the source address uniquely identifies this packet - used during reassembly of fragmented datagrams)
- **Flags** (a sequence of three flags (one of the 4 bits is unused) used to control whether routers are allowed to fragment a packet (i.e. the Don't Fragment, DF, flag), and to indicate the parts of a packet to the receiver)
- **Fragmentation Offset** (a byte count from the start of the original sent packet, set by any router which performs IP router fragmentation)

- **Time To Live** (Number of hops /links which the packet may be routed over, decremented by most routers - used to prevent accidental routing loops)
- **Protocol** (Service Access Point (SAP) which indicates the type of transport packet being carried (e.g. 1 = ICMP; 2= IGMP; 6 = TCP; 17= UDP).
- **Header Checksum** (A 1's complement checksum inserted by the sender and updated whenever the packet header is modified by a router - Used to detect processing errors introduced into the packet inside a router or bridge where the packet is not protected by a link layer cyclic redundancy check. Packets with an invalid checksum are discarded by all nodes in an IP network)
- **Source Address** (the IP address of the original sender of the packet)
- **Destination Address** (the IP address of the final destination of the packet)
- **Options** (not normally used, but, when used, the IP header length will be greater than five 32-bit words to indicate the size of the options field)
- **Pad** is used to ensure that the datagram header ends on a 32-bit boundary

#### 1.1.4 Drawbacks and Problems in IPv4

IPV4 has been used for a long time as the standard Internet Protocol, but over the period of time, its limitations and problems were noticed and faced, which gave rise to the need of a new set of standards that solves these issues. Following are the major problems that invented the need of a new internet protocol:

##### 1.1.4.1 Address Space Exhaustion

The IPv4 address structure is based on a 32-bit address length. It can manage about 4 billion addresses, but cannot be assigned to everyone living in the world, which contains about 6.3 billion people. As the Internet made the transition from a government-sponsored to a commercially driven communications environment, some of the original operational characteristics required reexamination. For example, prior to commercial use of the Internet, widespread security of end-user communication was not required. Now, as many end users conduct business on the Internet and routinely submit their credit card information for purchases, the need for private communication sessions is enhanced. But the most dramatic issue related to growth became the huge numbers of hosts that were connecting to the

Internet, and the associated IP addresses that were being consumed by those hosts.

The need for more and more unique IP addresses will continue to grow in the future as well, since massive work is being done in the field of information technology and telecommunication. And especially with the world moving very quickly towards the wireless technology, the urge for new unique IP addresses is bound to increase over the next few years. And the current version of internet protocol (IPv4) is not capable of fulfilling the requirements of the future.

The IP address shortage is also expected to become a major issue for wireline networks, particularly in Asia, as countries such as China and Japan jump aboard the Internet bandwagon and service providers need to provision more users than ever before. The need to extend the addressing capabilities of IPv4, providing "have not" countries with more address space, is the key driver for adopting a new IP protocol. Other reasons for designing a new protocol include the need to provide

It is not very far that we will run out of IPv4 addresses. An expert in this field is Geoff Huston. He knows more about this, than 99.99% of people. Huston has a formulation, which analyses current allocations and predicts three dates.

The first is the date IANA will run out of IPv4 blocks to allocate. And that is quite soon - 16 February 2010. That's when the global unallocated stock will run out.

The second is the date the Regional Internet Registries will run out of their local stocks, seeing they can get no more than IANA, and that is only a few months later on 21 September 2010.

Now it's not quite the end of the IPv4 world at that stage, because there are allocated blocks which could be freed up. For example Ford Motor Company (and Haliburton!) have a Class A or /8 block of 16,777,216 addresses, and they certainly don't use them all. The estimate is these could keep IPv4 allocations going until March 2017.

But as the IANA looks to run out in early 2010, we should expect to see this being the

catalyst for large scale transitions to IPv6. Address Space Exhaustion is the largest and the most crucial problem, among all the problems posed by IPv4 that needs to be addressed before it is too late.

#### **1.1.4.2 Increment of Routing Information**

Routing tables are large in size and complex, due to the amount and structure of data stored in them. For better utilization of network devices, and thus the network itself, the routing tables should be less complex, light weighted and well organized. As the internet grows, so does the routing tables, as more data is required to be stored in them. The system with class A, B and C addresses of such different sizes together with the rationing of IPv4 addresses, Internet addressing and routing is complex.

The large and complex routing tables put huge burden on the backbone routers, thus making them inefficient. The use of CIDR is supposed to make routing more efficient, but it does not guarantee an efficient and scalable hierarchy [4].

#### **1.1.4.3 Cost of Configuration**

A manual configuration or use of a stateful address configuration such as DHCP is required in IPv4. This is complicated, especially in cases where a company needs to reconfigure the entire network. It causes much downtime, which can lead to great costs in big setups specially. The configuration cause more administrative problems as the Internet and other markets that require an IP-address grow. And specially considering the alarmingly growing usage of the internet, having the network down every time it needs to be configured, is very critical indeed.

#### **1.1.4.4 Security Issues**

IPv4 was designed with a network of relatively trusted users in mind where the network infrastructure was likely to be relatively secure and the information that was being transmitted was relatively public. It was never thought that it would be used so extensively in so many areas of life. Consequently, it did not seem important at the time to include security features, like non-repudiation or authentication, directly into the protocol.

As time passed and newer technologies started to develop their dependencies on IPv4 and the way the IPv4 Internet has been used has changed radically. The huge number of Internet users means that trusting them all is simply not an option. The network infrastructure itself now involves cooperation between a large number of public and private organizations, with wildly differing agendas. Most significantly, the data that is being transmitted on the Internet now is often commercially, financially or personally sensitive. Packets sent at IP-level need encryption to protect the private data from being viewed or modified. From a security perspective, IPv4 is way out of its depth. There needs to be a much stronger level of security implemented in the protocol that is supposed to be used all over the world for even the most sensitive data of all.

#### 1.1.4.5 Quality of Service (QoS)

The ability to guarantee that the network traffic you send gets there on time is known as Quality of service (QoS). One of the earliest approaches to guaranteeing a certain level of QoS in IPv4 networks was a field called "Type of Service" in the IP header. The IPv4 header itself contains fields that are set to particular values depending on what kind of treatment the packets "want" from routers; the idea being that packets proclaiming themselves to be worthy of immediate forwarding will be plucked out of queues by routers and preferentially dealt with. In other words, the concept of prioritizing packets or chunks of data according to their nature was adopted. But this was a rather crude approach, not widely implemented, and in its first revision died a death. Perhaps the two biggest problems with it were that it provided no mechanism for authenticating the request for a particular QoS, and that there was no flexible way to assign priorities within a particular set of flows, such that certain ones could be designated lower priority. In essence, we have the rather contradictory result that a mechanism introduced to allow for more appropriate and fairer treatment of packets leads to unfairness!

The "Type of Service" field in the IP header is limited and has had a number of definitions during the years [5]. There have been various efforts to retrofit more complete QoS features to IPv4, especially now that some people consider IP networks 'mission critical' and others want to run their telephones over their IP infrastructure. There has been a significant amount

of work designing frameworks such as DiffServ (RFC 2475) and IntServ (RFC 1633), and protocols such as RSVP (RFC 2205) to deal with QoS. It's unclear whether we will ever see wide ranging public deployment of these mechanisms, as they require significant cooperation between networks. Some potential adopters discover that, rather than invest time and equipment into deciding which packets to drop; it is cheaper to buy more bandwidth so that all the traffic gets through! There is no question, however, that a well-implemented mechanism to offer QoS guarantees would be of immense value to IP users in the future. And as the time passes by, advantage will become a necessity and will have to be achieved one way or the other.

A lot has changed since the Internet suite of protocols was developed in the early 1970s. Among these changes are virtual LANs using switching technologies, high-speed backbones using ATM and SONET technologies; real-time traffic, such as voice and video over the Internet; the World Wide Web; and electronic commerce. And many of these technologies either directly or indirectly impact the worldwide Internet. As a result, the Internet must continue to be enhanced. IPv4 has problems. Some of them can be worked around, some not; but as time goes on, new applications will pile increasing amounts of crust on top of the venerable protocol, making a new start even more attractive.

In short, the protocols and processes of a generation ago may not be optimal for the networks of the twenty-first century.

## **1.2 IPv6**

This section discusses IPv6 in terms of its historic background as well as its technical aspects. An overview of the problems in IPv4, solved in IPv6 has also been highlighted.

### **1.2.1 Introduction**

The new version of IP protocol, IP version 6, seems to be a satisfactory solution to the above mentioned limitations of IPv4. It is thought and believed to be the best possible answer to the limitations of IPv4 as well as to the future requirements of the telecommunication world. It is foreseen that the deployment of IPv6 is probably inevitable and it is only a matter of time to see exactly when ipv6 will become the basic Internet-working protocol [6]. Since the number

of network applications that IPv4 currently supports is enormous, and the porting procedure will cost much in terms of money and time, the only applicable solution that will lead to a global dominance of IPv6 is the coexistence of IPv4 and IPv6 for a reasonable period of time. In a mixed situation, where both protocols co-exist, communication between IPv4 hosts over an IPv6 network, IPv6 hosts over an IPv4 network and IPv6 host and an IPv4 host must be achievable.

Since Internet Protocol version 6 (IPv6) not only provides rich IP addresses but also support better mobility, security and Quality of Service (QoS) than IPv4, IPv6 has become one of the most important protocol in the next generation Internet.

### **1.2.2 Historic Background**

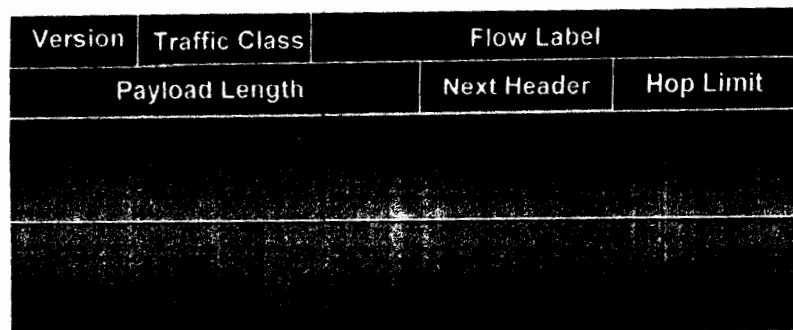
The new version of internet protocol, IPv6, was recommended by the IPng Area Directors of the IETF at the Toronto IETF meeting on July 25, 1994 in RFC 1752. The recommendation was approved by the Internet Engineering Steering Group and made a Proposed Standard on November 17, 1994. The core set of IPv6 protocols were made an IETF Draft Standard on August 10, 1998[2]. The initiative to make a new version of the Internet protocol was mainly caused by the shortage in address space, as described in chapter 2. In addition there seemed to be possibilities to improve areas of IPv4 in the new version. The new version is supposed to allow new features in the Internet in the future to be added in a less complex way than today. And most importantly, the new version is supposed to be practical and compatible to future needs for a very long period of time.

### **1.2.3 Technical Specifications**

In this section, we will discuss the various technical aspects of IPv6.

#### **1.2.3.1 The IPv6 Header**

An IPv6 packet is a block of data that contains a header and a payload. The header is the information necessary to deliver the packet to a destination address; the payload is the data that you want to deliver. IPv6 packets can use a standard or an extended format. The IPv4 packet header consists of 40 bytes of data. IPv6 packet headers contain many of the fields found in IPv4 packet headers. The full header is shown below:

**Fig 1-3 The IPv6 Header**

The header fields [7] are discussed below:

- **Version (4 bits)** - Indicates the version of the Internet Protocol.
- **Traffic class (8 bits)** - Previously the type-of-service (ToS) field in IPv4, the traffic class field defines the class-of-service (CoS) priority of the packet. However, the semantics for this field (for example, diffserv code points) are identical to IPv4.
- **Flow label (20 bits)** - The flow label identifies all packets belonging to a specific flow (that is, packet flows requiring a specific class of service [CoS]); routers can identify these packets and handle them in a similar fashion.
- **Payload length (16 bits)** - Previously the total length field in IPv4, the payload length field specifies the length of the IPv6 payload.
- **Next header (8 bits)** - Previously the protocol field in IPv4, the Next Header field indicates the next extension header to examine.
- **Hop limit (8 bits)** - Previously the time-to-live (TTL) field in IPv4, the hop limit indicates the maximum number of hops allowed.
- **Source address (128 bits)** - Identifies the address of the source node sending the packet.



- **Destination address (128 bits)** - Identifies the final destination node address for the packet.
- **Extension Headers** - In IPv6, extension headers are used to encode optional Internet-layer information. Extension headers are placed between the IPv6 header and the upper-layer header in a packet. IPv6 allows you to chain extension headers together by using the next header field. The next header field, located in the IPv6 header, indicates to the router which extension header to expect next. If there are no more extension headers, the next header field indicates the upper-layer header (TCP header, UDP header, ICMPv6 header, an encapsulated IP packet, or other items). The current IPv6 specification defines six extension headers:

- Hop-by-hop options header
- Routing header
- Fragment header
- Destination options header
- Authentication header
- Encrypted security payload header

Options in IPv6 are handled in an extension header. The good thing about extension headers is that they can be inserted if they are needed, but not inserted if they're not needed. Usually extension headers are examined by the destination node only. The hop-by-hop options header is an exception to this rule and carries optional information that must be examined by every node along the path of the packet. It has a Next Header value of zero.

### 1.2.3.2 IPv4 and IPv6 Header Differences

Following is a visual comparison between the headers of IPv4 and IPv6, followed by the textual explanation of the differences:

IPv4 Header				
Version	IHL	Type of Service	Total Length	
Identification			Flags	Fragment Offset
Time to Live	Protocol		Header Checksum	
Source Address				
Destination Address				
Options			Padding	

IPv6 Header			
Version	Traffic Class	Flow Label	
Payload Length		Next Header	Hop Limit

**Fig 1-4 Comparison of IPv4 and IPv6 Headers**

Five fields have been removed from the IPv4 header: [11]

1. Header Length (fixed length header doesn't need it)
2. Identification
3. Flags
4. Fragment Offset
5. Header Checksum (checked at the transport layer)

Three fields have been renamed and modified from the IPv4 header:

1. Traffic Class (type of service in IPv4)
2. Next Header (protocol type in IPv4)
3. Hop Limit (Time to live in IPv4)

One field has been added:

1. Flow Label

### 1.2.3.2.1 IPv6 Address Representation

IPv6 Addresses are represented as 16-bit fields in case insensitive colon hexadecimal

For Example:

2031:0000:130F:0000:0000:09C0:876A:130B

The leading zeros in a field are optional:

2031:0:130F:0:0:9C0:876A:130B

Successive fields of 0 are represented as :: but only once in an address:

2031:0:130F::9C0:876A:130B

### 1.2.4 Improvements / Benefits of IPv6

The new header structure of IPv6 itself has made improvements to the older version of internet protocol. Following:

- **Extended address space:** The most obvious change is the newly extended address space, which was the driving reason to develop IPv6 in the first place. IPv6 increases the IP address size from 32 bits to 128 bits. The new address has 128 bits, which gives us space that can be used to create the hierarchical addressing system and provide plenty of IP addresses for everyone and all devices that need an IP address in the future.
- **Expanded auto configuration mechanisms:** IPv6 has auto configuration mechanisms that will make our work as network engineers a lot easier. With auto configuration you can install plug and play machines without needing to configure IP addresses. It's now also much easier to re-number the networks.
- **Simplification of the header format:** It has a simplified header format, which has fixed length of 40 bytes [8]. Options are now inserted as extension headers only if needed. Extension headers are placed after the IPv6 header. Some of the IPv4 header fields have been dropped or made optional. This reduces the routine processing cost of packet handling. It decreases the bandwidth cost of the IPv6 header.
- **Improved Support for Extensions and Options (security)**  
In IPv4, the options were integrated into the basic IPv4 header. In IPv6 the options are handled as Extension headers. Extension headers are optional and only inserted between the IPv6 header and the payload, whenever necessary. In this way the IPv6 packets are believed

to be more flexible and streamlined. In addition new options that may be defined in the future can be integrated more easily than for IPv4.

- **Flow Labeling Capability (QoS – Quality of Service)**

IPv6 adds labeling on packets which enable packets of a certain type to get special handling on sender's request. This is for packets with non-default quality of service e.g. real-time service.

- **Scalability of multicast routing** – IPv6 provides a much larger pool of multicast addresses with multiple scoping options.

IPv6 addresses are being assigned to new machines every day and all operating systems have already upgraded their systems to support both IPv4 and IPv6 (e.g. Windows XP). IANA has already started to assign blocks of addresses to several ISPs.

### 1.3 Moving from IPv4 to IPv6

The deployment of IPv6 is a long and complicated process, which has only just started. It is not possible to shift all the IPv4 networks to the new IPv6 architecture all at once. This process will take some time, as it has to be an orderly transition. The migration will happen gradually, and for many years IPv4 and IPv6 will have to exist together [9]. Parts of the Internet will migrate at different times, and at different speed. It is not possible to design one standard solution for how to migrate; the mechanism to be used depends on the situation. Different networks need different mechanisms, and different mechanisms are needed at different stages of the migration progress.

If we can design a translation mechanism, that can translate traffic of IPv4 to IPv6 and vice versa, we will be able to go through this transition phase, without any interruptions in the communication department. There are quite a few transition techniques used to move traffic from IPv4 to IPv6 and vice versa. We will discuss the current techniques in the next chapter.

## 1.4 Outline of the thesis

In this thesis we are going to propose a new algorithm to calculate TCP Checksum which will be used at the border NAT-PT router. The proposed algorithm is supposed to improve the efficiency of the checksum calculation process thus decreasing the time consumed to process each packet traversing the NAT-PT router. The thesis consists of the following chapters:

**Chapter 1** contains the introduction to the internet, the protocol used for communication over the internet, this chapter also contains the two commonly used versions of the protocol: IPv4 and IPv6 in details, explaining all the fields. A brief about migration from IPv4 to IPv6 is also discussed.

**Chapter 2** contains the state of the art review describing the transition mechanisms between the two Internet protocol versions; these are mainly Dual Stack, Tunneling and translation, which further branches out in different types of translation techniques – NATPT being the most reliable technique amongst all.

**Chapter 3** contains the requirement analysis which in turn includes the problem analysis and the use case analysis.

**Chapter 4** describes the proposed architecture's overview, its components including the checksum calculation usual method and our proposed method and the comparison of the algorithm of the two showing the increase in efficiency with the use of the proposed checksum calculation algorithm.

**Chapter 5** describes the implementation details in the form of pseudo code explaining the functions used for simulating the conversion of packets from IPv4 to IPv6 and the proposed algorithm for checksum calculation.

**Chapter 6** shows the results obtained while running the simulator that uses our proposed technique. Simulator's different components are explained by the help of figures. Later in the chapter are the comparison charts and graphs showing the efficiency of the proposed

algorithm over the current one. The results are obtained for both fragmented and non-fragmented packets and with fixed number of packets and variable number of packets.

**Chapter 7** concludes our research and also specifies some areas where further research can be done for the improvement.

## **CHAPTER 2**

### **STATE OF THE ART REVIEW**

## 2. State of the Art Review

A lot of transition mechanisms have been developed over the years, each having its advantages and drawbacks. Thus, the communication world still seeks a perfect solution for translation of data from one protocol to the other. There are tens of mechanisms for transition. Only a few are widely used for general situations. These are presented in this chapter. The mechanisms not mentioned in this chapter are mechanisms for very specific scenarios, which may only differ slightly from one to the other.

To make IPv4 and IPv6 coexist, transition mechanisms have been designed. The mechanisms can be divided into three groups:

- Tunneling techniques, used when IPv6 packets traverse the IPv4 infrastructure.
- Dual-stack techniques, allowing IPv4 and IPv6 to coexist in the same devices and networks
- Translation techniques, making IPv6-only nodes able to communicate with IPv4-only nodes.

Even though the techniques are presented separately, they can and likely will be used in combination with one another.

### 2.1. TRANSITION DEFINITIONS

- **IPv4-only node:** A node or a router that recognizes and implements only IPv4.
- **IPv6/IPv4 node:** A node or a router that recognizes and implements both IPv4 and IPv6
- **IPv6-only node:** A node or a router that recognizes and implements only IPv4.
- **IPv6 node:** A node that recognizes and implements IPv4.
- **IPv4 node:** A node that recognizes and implements IPv6.
- **IPv4-compatible IPv6 address:** An IPv6 address that has been assigned to an IPv6/IPv4



host. It was converted from an IPv4 address to an IPv6 one, by adding 96-bit prefix 0:0:0:0:0:0 in the higher order bits. Thus the actual IPv4 address is placed in the lower 32 bits.

- **IPv6-only address:** An IPv6 address that has a prefix other than 0:0:0:0:0:0.

## 2.2. TRANSITION MECHANISMS

This section will discuss the different transition mechanisms developed over the years for better transition from IPv4 to IPv6, keeping the cost as low as possible.

### 2.2.1 Dual – Stack

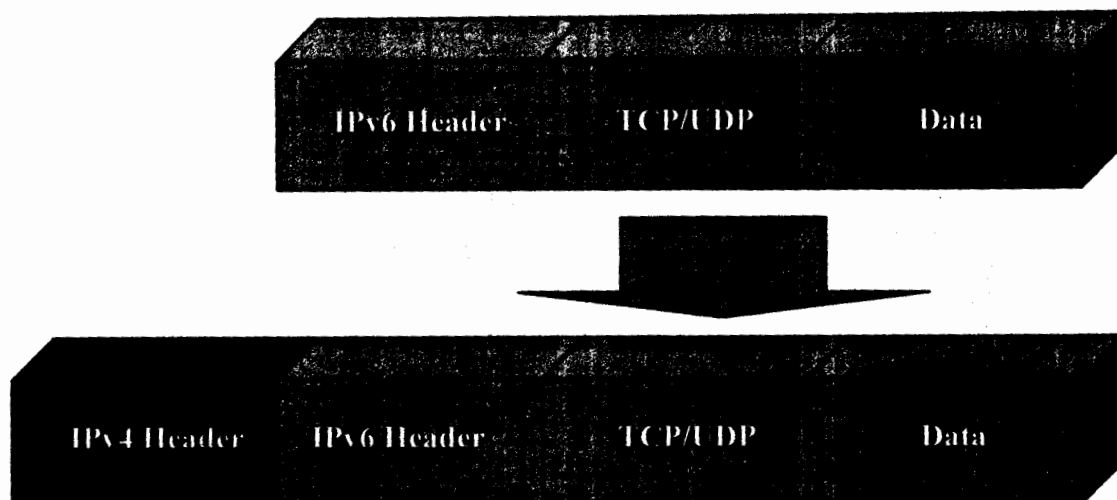
The Dual Stack [10] mechanism is based on two protocol stack architecture. In this technique, a host is equipped with two separate protocol stacks in the operating system, one for each version of the Internet Protocol. Both, IPv4 and IPv6 are handled by separate stacks. A host with such two-stack architecture is called an “IPv4/IPv6 node” [11]. An IPv4/IPv6 node is capable of communicating with both IPv4 and IPv6 nodes, as it can send and receive data to and from either protocol. The protocol version indicator in the physical frame decides which stack to use to handle an incoming datagram, whereas the sending application decides which stack to use to send the data. The Dual Stack mechanism is a rather simple technique to provide compatibility with IPv4 as well as IPv6. But the limitation of this technique is that, it cannot translate packets of IPv4 to IPv6 and vice versa. Thus only those nodes can communicate with each other, which are based on the same IP version.

### 2.2.2 TUNNELLING

Tunneling (or encapsulation) is used when two nodes of the same protocol, have the need to communicate over a network of the other protocol. Tunneling allows the encapsulation of packet of one protocol into the packet of the other protocol. There are four possible tunnel configurations that could be established between routers and hosts [12]:

- Router to Router
- Host to Router

- Host to Host
- Router to Host



**Fig 2-1 Tunneling Basic Technique**

There are two different types of tunnels used between dual-stack nodes:

#### **Configured Tunnels**

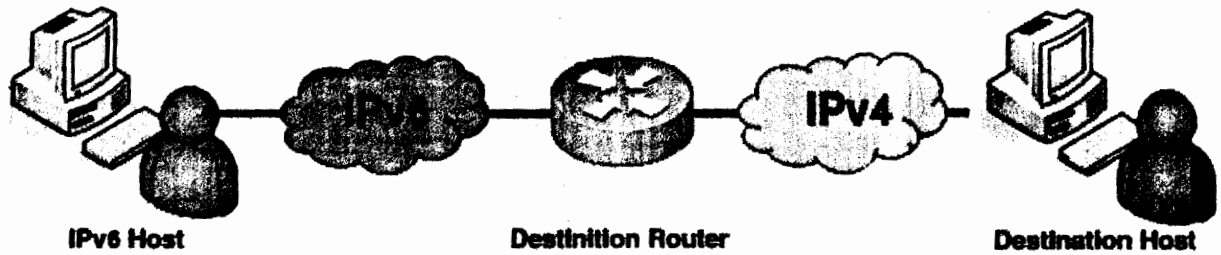
In this case the address of the tunnel end point is pre-defined and thus the encapsulating node should have the information about the end points of the tunnel. The tunneling node provides configuration information which is used to determine the end point address of the tunnel.



**Fig 2-2 Configured Tunneling**

#### **Automatic Tunnels**

In automatic tunnels, as the name suggest, the tunnel's endpoint is not required to be configured manually. Instead, the address of the end point of the tunnel is derived from the embedded IPv4 address of the packet's IPv6 address.



**Fig 2-3 Automatic Tunneling**

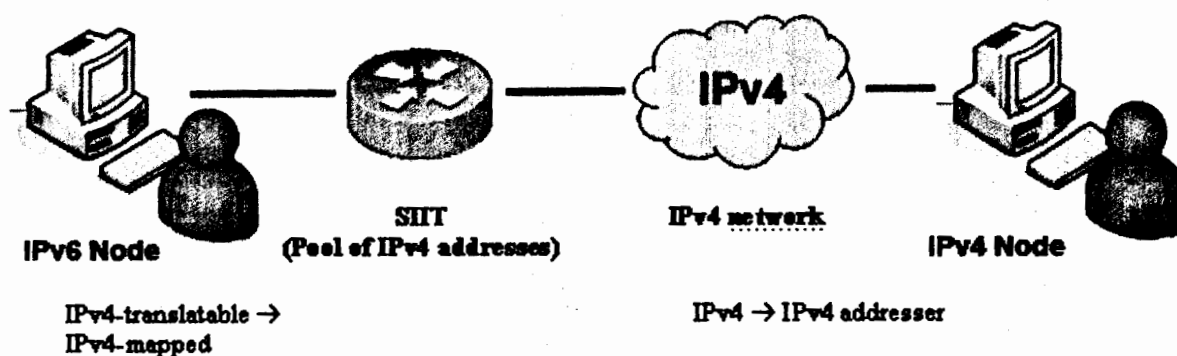
The drawback of tunneling is that, it can slow down throughput and (in case of configured tunnel) requires network technicians to configure the tunnel endpoints information into the encapsulation node, which takes time and of course costs. Furthermore, it's not possible for an IPv6-Only node to communicate with an IPv4-Only node using tunneling.

### **2.2.3 TRANSLATION**

Translation is the process of converting the datagram of one protocol, into the datagram of the other protocol. It makes it possible for IPv6 nodes to communicate with IPv4 nodes and vice versa. There are a number of different translation methods that have been developed to get closer to the perfect solution. Following is the brief description of a few of them:

#### **SIIT (Stateless IP/ICMP Translation)**

Stateless IP/ICMP Translation [13], a mechanism to support communication between IPv4 and IPv6 nodes, was first introduced in Feb 2000. SIIT provides a method, through which a router read an IPv4 header, parses it and creates a corresponding IPv6 header with the same information. Similarly an IPv4 header is created, corresponding to an IPv6 header, as per need. The actual algorithm and technique may vary, to convert one header from the other, but the basic idea is the same.



**Fig 2-4 Stateless IP/ICMP Translation**

The SIIT is a protocol translation mechanism that allows communication between IPv6-only and IPv4-only nodes via protocol independent translation of IPv4 and IPv6 datagrams, requiring no state information for the session.

### **NAT-PT**

Network Address Translation - Protocol Translation (NAT-PT) [14] is one of the very famous transition methods introduced and used to provide communication between IPv4 and IPv6 nodes. It is an extension of NAT (Network Address Translation) [15]. NAT-PT may be implemented using a router or any other suitable network device that is located at the boundary of an IPv4 network connecting to an IPv6 network, and vice versa. The NAT-PT Translator keeps a pool of globally unique IPv4 addresses that are assigned to IPv6 addresses, as per requirement. These addresses are assigned dynamically, when communication is required between v4 and v6 nodes. This method does not require any changes to the network components already installed, thus reducing the cost of implementation, exponentially, as compared to some other methods. The communication between the two address realms is absolutely transparent.

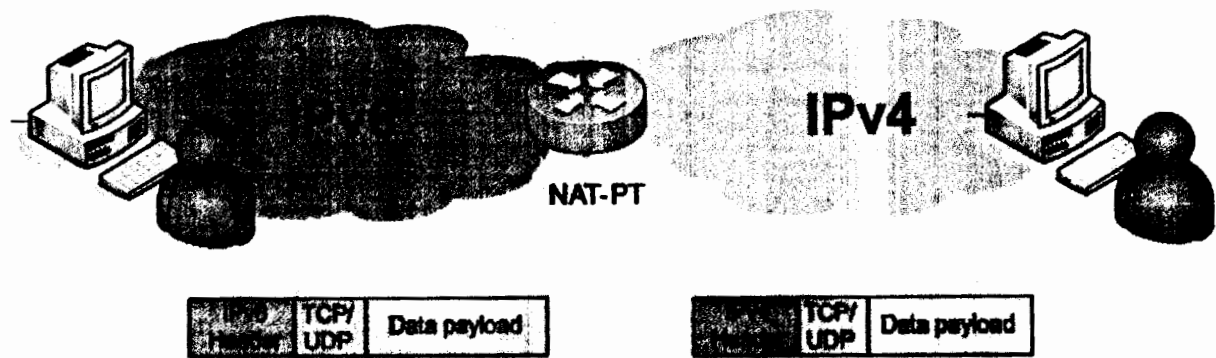


Fig 2-5 NAT-PT Setup Example

NAT-PT is probably the cheapest and easiest method to implement, amongst all the other transition methods. But it also has its limitations. It has to maintain a log of all the sessions initiated and make sure that all the traffic belonging to a particular session, traverses the same router. Due to the address translation performed by NAT-PT, the applications that transfer the IP address in the upper layers, do not work [16]. They require Application Layer Gateways (ALG) to be functional. NAT-PT does not provide end-to-end network, application and transport layer security. This is a major drawback.

In addition to the IP Header translation, the upper layer protocols like TCP/UDP uses Pseudo Headers (which contains field values from the IP Header) to compute their checksum. The checksum value needs to be adjusted with the difference between the original IP addresses and the translated IP addresses.

#### Header checksum

TCP header's checksum is computed by creating a momentary TCP Pseudo Header [17] which inherits its fields from the IP header which includes the source and destination addresses and the protocol fields, remaining two fields are unique to the pseudo header (96 bits), one is the reserved field which comprises of all zeros and the last one is the Header length field.

This Pseudo header prepends itself to the actual TCP header [18] and then performs the checksum by setting the checksum fields to zero for the calculation, if there is a mismatch between the calculation and the value the source device put in the Checksum field, this indicates that an error of some sort occurred and the segment is normally discarded.

### **NAPT-PT**

Network Address Port Translation – Protocol Translation (NAPT-PT) is an extension of NAT-PT. It allows the v6 nodes, to share traffic with ipv4 nodes, by using only a single IPv4 address. Further more, NAPT-PT also translates the transport identifiers. It maintains a list of port numbers that are assigned dynamically, to sockets on the recipient side of the NAPT-PT node.

### **Transport Relay Translator (TRT)**

A Transport Relay Translator [19] is a translator that is placed in the transport layer. It is located between the two nodes which are communicating with each other TRT allows IPv4-only nodes to communicate with IPv6-only nodes. TRT can use different protocols while communication with the client and the application server. TCP connections are established between the clients and the hosts, on IPv4 as well as IPv6, and then TRT acts as the terminator of transport connection for each node. TRT is a state full system that requires no modification to the concerned IPv4-only and IPv6-only nodes, to be implemented. It has no fragmentation or path MTU issues associated with it. However, TRT is unidirectional (from IPv6 to IPv4). The address mapping from IPv4 to IPv6 is very difficult. TRT has a single point of failure, as all the transport layer connections have to go through the same TRT.

### **Application Level Gateway**

There are a lot of applications that send host names and IP addresses within the payload of the IP packet. NAT-PT does not read through the packet payload, therefore an Application Level Gateway (ALG) is required to read that those addresses from the payload and replace them with corresponding IPv4 or IPv6 addresses. The use of ALG's, enhances the use and applicability of NAT-PT to a great extent. FTP-ALG and SIP-ALG are two of the more common ALG's. ALG has the limitation that it can support only one application protocol. Therefore, in the situations where more than one application protocols is required to be handled, a separate ALG has to be implemented for each protocol. Since a single protocol is handled by an ALG, it will get most of the request that are similar. Therefore it can cache those request and reduce the need to contact the application server to a great extent.

## **SOCKS**

SOCKS [20] are usually referred to as “proxy protocol for client/server environments”, although it’s an example of transport relay. A SOCKS proxy differs from the traditional transport relay, very slightly. Whenever a node/client wants to connect to the application server, it first establishes a connection with the preconfigured proxy server, providing it the IP address and the port number of the application server. The connection with the proxy server is established using a special proxy protocol. Once this is done, it is the responsibility of the proxy server to create a connection with the application server requested, and to allow transfer of packets between the client and the application server transparently.

SOCKS consist of two major components: SOCKS Server and the Client SOCKS Library. The Clients Socks Library creates an authenticated TCP connection with the SOCKS server. A fake IPv4 address is returned to the client; for the client, that’s the IP address of the application server. The SOCKS server establishes the connection with the requested node and from then onwards, it plays the role of a relay between the client and the requested node. SOCKS provide better security; its installation is very easy for the networks that have adopted SOCKS as their fire walling mechanism. The drawback of SOCKS is that, the Client library has to be installed / compiled in all the clients that wish to establish a connection with the SOCKS Server. Furthermore, SOCKS supports only those connections that are initiated by the clients.

## **Related Research Work**

The migration from IPv4 to IPv6 is unavoidable but this migration will be a relatively slow process and thus some transition mechanism shall be required for IPv4 and IPv6 to coexist and communicate with each other [9]. A lot of work has been done on the different transition techniques available, with a lot of focus on Translation Mechanism, specially on NAT / NAT-PT because it is probably the most cheapest and quickest solution available to the co-existence problems, as it does not require any changes to the end nodes and packet routing is totally transparent [14]. NAT-PT is totally interoperable and does not require any

modifications / installations such as dual stacks neither on the IPv4 nodes, nor on the IPv6 ones.

One of the biggest Limitations of NAT-PT is that all the packets belonging to a particular session need to traverse through the same NAT-PT router [14]. The NAT-PT router is placed at the border of networks, thus all the traffic in and out of the network needs to traverse through that NAT-PT router. Each time a packet reaches the NAT-PT router, the router has to perform the following main tasks:

- Address Translation
- Protocol Translation

These two tasks are hectic ones, with respect to processing and CPU Utilization. Therefore a lot of processing is required on each packet, which slows down the overall traffic processing speed. If the processing required for each packet is reduced, the overall traffic processing speed shall be increased, which is one of the basic measures of performance of network devices.

Work has been done and is continued to be done on removing the limitations of NAT / NAT-PT and providing ALG's that enhance the support provided by NAT/NAT-PT but currently no work is being done in the area of improving the performance by improving the checksum recalculation process in NAT routers. So our research is unique and one of its kind.

### **2.3. ENHANCEMENTS REQUIRED**

Problems exist in the current techniques in the domains of compatibility, security and specially efficiency. The need exists for enhancements and improvement in terms of cost and efficiency. Even the smallest of enhancement in these two departments will make an exceptional difference, especially due to the increased usage of networks and the increased size of data that is regularly sent over the networks.



**CHAPTER 3**

**REQUIREMENTS ANALYSIS**

### 3. Requirements Analysis

To overcome the problems existing in the current techniques, in our proposed system we have made an effort to decrease the processing required on packets, in the scenario where a packet is traversing through the NAT-PT router. We have proposed a technique that may be used, to decrease the repeated processing required to calculate checksum on every packet passing through the translation gateway.

The requirement analysis is the first step towards developing software. Analysis must be performed in a systematic and correct manner so as to have as few mistakes as possible in the software and to have an end product completely fulfilling the expectations of the client. The reliability and the robustness of the software are highly dependent on the fact that the analysis is carried out properly. The main objective of this phase is to identify all possible requirements. Problems are identified and then a possible solution is proposed.

#### 3.1 Problem Analysis

There are a number of considerations that come into play, while planning for a new transition mechanism but the most important ones are as follows:

- **Transparent**

All the end-points should be oblivious of the translation. For any node, it should not make a difference whether the received packet of data is coming from an IPv4 node or an IPv6 node. The Translator / Gateway should make sure that it has handled all the related issues specific to each protocol, so that when a packet reached an IPv4 node from an IPv6 node, the IPv4 node should be able to process it as it was received from another IPv4 node. Transparency holds a key position when it comes to the basic requirements of transition methods and techniques.

- **Scalable**

It should scale with the size of the network that it serves. As we can see even today that there are hundreds and thousands of networks that are set up for one reason or the other. Each network is composed of as many network devices as its requirements. Therefore there is a lot of variation in the size of networks. The ideal translation

gateway should be capable and customizable enough to be easily implemented to any network in the world, regardless of its size. No extra configuration should be required to support the size of any particular network. And at the same time, there should not be any compromise on performance, consistency and security, due to the size of the network.

- **Failure Resilient**

There should not be a single point of failure. Because the translator / transition gateway serves as a relay between an IPv4 network and IPv6 network, traffic may be concentrated on the translator. As a result, communication may be stopped in case of traffic congestion or any other error, making services unavailable to users. Since more and more new services are being moved on the global networks, especially critical services used by the defense authorities, having the complete system down even for a single minute could be highly costly and may not be tolerated at any level. Therefore, the translation gateway should not have a single point of failure. If there is a problem with a device in the network, the network should continue to serve its users, using alternate devices and paths.

- **Performance**

The processing of IP packets should be very efficient. Today, the telecommunication world is moving towards more and more fast transmission of data. A very simple proof of this is the fact that in the early days, 33.6 kbps data transfer rate was considered to be a big achievement. But today, with the advent of fiber optics and broadband networks, even 2 MB data transfer rate is considered to be just a reasonable one. Moreover, due to the advent of more and more real time applications like video conferencing, data streaming, GPS applications like Google maps, and tons of other such applications, the need for very high data processing has become one of the biggest challenge and consideration for networks. Therefore the translation gateway should be a catalyst for high speed data processing and should make the required processing effort of network devices minimal. This will certainly make the network traffic communication a lot faster.

- **Easy to deploy**

The mechanism should be very easy to deploy, and an effort should be made avoid any changes (hardware and software) in order to implement the mechanism. Due to the large number of networks all around the world and the exponentially growing count of network devices installed in these networks, it would be near to impossible to make hardware changes on all the network devices of the world. And for that matter, even making a software change or upgrade would not be very easy or appreciated by all the stakeholders. Therefore a solution should be found such that minimal changes in the existing network are required to implement any new translation gateway. Otherwise, the time required to move from IPv4 to IPv6 will increase exponentially and we would not see that happening in the near future.

- **Seamless Communication**

To perform seamless communication between an IPv4 network and an IPv6 network, communication must be conducted according to the differences between the IP headers and address space structures of IPv4 and IPv6. This is because the translator acts as a host node for each protocol. The translator should be capable enough to handle each protocol according to its structure and deal with its constraints and limitations explicitly.

- **Load Handling**

Since a translator is installed between an IPv4 network and an IPv6 network, packets may be concentrated and the communication load increased. For this reason, we require a structure that can sufficiently support a system according to increases in the communication load without needing to stop the network. While designing a new transition mechanism, it should be tried to achieve such a technique, that helps maintain a manageable load on all network points. The communication load should be divided throughout the network, thus avoiding any network failures due to over burden.

- **End to End Security**

End to End network layer security should be allowed and should be possible to be implemented without any problems. Similarly Transport and application layer security are also as important and an effort should be made to achieve end to end security on all levels. Network services used by the defense section and other highly critical departments would never compromise their privacy and security for anything. Such departments spend billions of dollars to achieve a level of security that keeps them going. Therefore the translation gateway should be definitely capable of keeping their security and should not provide a security block hole, that could be exploited to interfere with their data communication, or even just to keep a check on the data being transferred.

- **Low Cost**

The cost of implementation should be minimal. Due to the large number of networks all around the world and the exponentially growing count of network devices installed in these networks, it would be near to impossible to make hardware changes on all the network devices of the world. It would just not be practically achievable due to the cost involved in it. And for that matter, even making a software change or upgrade would not be very easy or appreciated by all the stakeholders, as it would cost millions, may be billions of dollars to implement even the smallest of upgrades. Therefore a solution should be found such that minimal changes in the existing network are required, both in hardware as well as software, to implement any new translation gateway. Otherwise, the cost to move from IPv4 to IPv6 will increase exponentially and we would not see that happening in the near future.

The ideal transition mechanism would fit perfectly into this criterion. There are many possible combinations of technical IPv6 transition strategies. There are also a number of transition mechanisms (e.g. dual-stack, tunneling, and translation) which agencies can choose

from with more emerging from the technical community. The introduction of a new translation gateway on an enterprise scale will have to face a number of challenges including scalability, integration, and security. In the near term, there is concern about creating vulnerabilities in existing IPv4 networks by deploying a new translation gateway. Therefore it should be taken care that, all such challenges are met and all considerations are dealt with, while designing and implementing a new translation gate , otherwise it will not be of any good if only small networks are able to implement it, while large network stakeholders remain reluctant to use it and depend on it.

### **3.2 Use Case Analysis**

Analysis of the project is presented in terms of use case diagrams indicating the actors and use cases in expanded format. This helps visualizing the work and indicating the system boundaries while presenting the functionalities. The Use Case Model describes the proposed functionality of the new system.

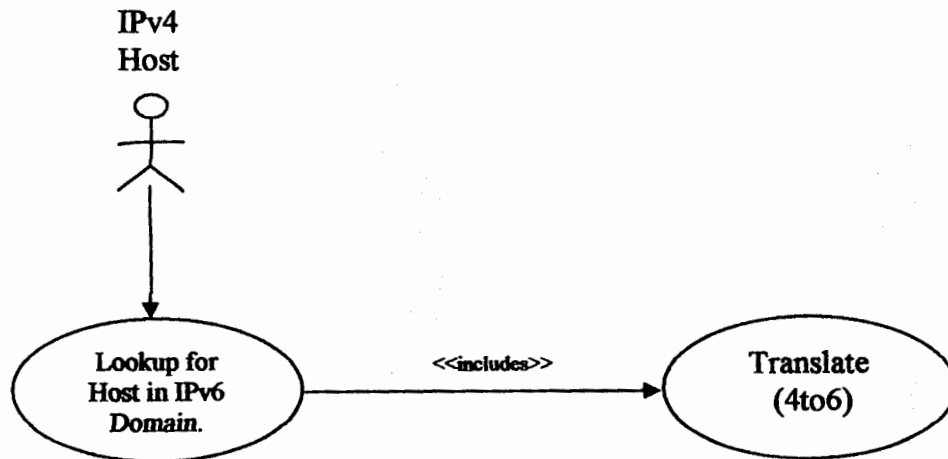
Use case depicts a set of scenarios that describing an interaction between a user and a system. Use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.

The basic functionality of our proposed system consists of two major parts which are as follows:

- Translation from IPv4 to IPv6
- Translation from IPv6 to IPv4

We will describe each one visually in the form of a use case diagram and follow it with the detailed specifications of the respective use case, for better understanding.

### 3.2.1 Translation from IPv4 to IPv6



**Fig 3-1 Use Case Diagram of IPv4 to IPv6 Translation**

#### Use Cases in Expanded Format

**Name:** Lookup for Host in IPv6 Domain.

**Purpose:** To lookup for host in the different domain for communication.

**Actors:** Node A, Node C.

**Precondition:** Session not established.

**Post Condition:** Session is established.

**Basic Path:**

Actor Action	System Response
1. Node C's name resolver sends a name look up request for Node A.	1)A The local DNS is checked for the name that is looked up. (NAT-PT is residing on the border router between V4 and V6 networks), this request datagram would traverse through the NAT-PT router.

	1)B Calls: Translate (4to6).
--	------------------------------

**Alternate Path:**

Actor Action	System Response
1. Node C's name resolver sends a name look up request for Node A.	1)A If a V4 address is not previously assigned to this V6 node 1)B Calls: Translate (4to6).

**Name:** Translate (4to6).

**Purpose:** To translate the IP address to the type it is destined to.

**Actors:**

**Precondition:** The address should be translatable.

**Post Condition:** The address is translated.

**Basic Path:**

Actor Action	System Response
	1)A The DNS-ALG on the NAT-PT device would modify DNS Queries for A records going into the V6 domain as follows:  a) For Node Name to Node Address Query requests: Change the Query type from "A" to "AAAA" or "A6".  b) For Node address to Node name query requests: Replace the string "IN-ADDR.ARPA" with the string



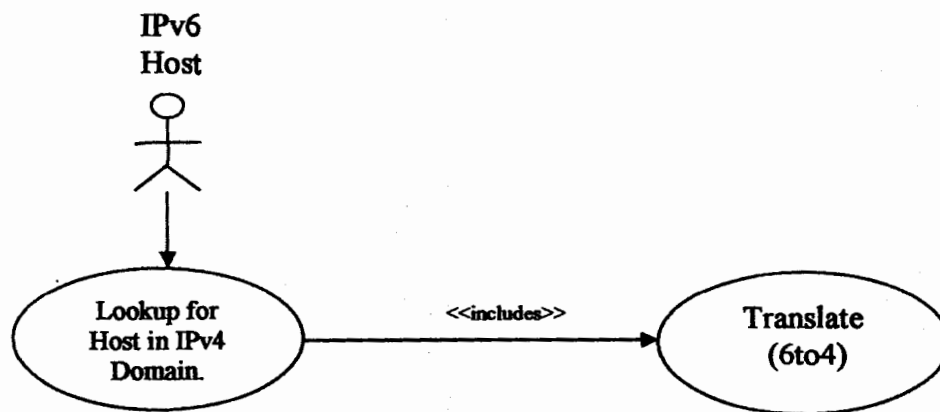
	<p>"IP6.INT". Replace the V4 address octets (in reverse order) preceding the string "IN-ARPA" with the corresponding V6 address (if there exists a map) octets in reverse order.</p> <p>1)B In the opposite direction, when a DNS response traverses from the DNS server on the V6 network to the V4 node, the DNS-ALG once again intercepts the DNS packet and would:</p> <p>a) Translate DNS responses for "AAAA" or "A6" records into "A" records, (only translate "A6" records when the name has completely been resolved)</p> <p>b) Replace the V6 address resolved by the V6 DNS with the V4 address internally assigned by the NAT-PT router.</p>
--	--

**Alternate Path:**

Action	Reason
	<p>1) If a V4 address is not previously assigned to this V6 node, NAT-PT would assign one at this time from the</p>

	IPv4 Address pool depending upon availability.
--	--

### 3.2.2 Translation from IPv6 to IPv4



**Fig 3-2 Use Case Diagram of IPv6 to IPv4 Translation**

#### Use Cases in Expanded Format

**Name:** Lookup for Host in IPv4 Domain.

**Purpose:** To lookup for host in the different domain for communication.

**Actors:** Node A, Node C.

**Precondition:** Session not established.

**Post Condition:** Session is established.

**Basic Path:**

Actor Action	System Response
1. Node A's name resolver sends a name look up request for Node C.	1)A V6 nodes learn the address of V4 nodes from the DNS server internal to the V6 network. 1)B In the case where the DNS server

	<p>in the v6 domain contains the mapping for external V4 nodes, the DNS queries will not cross the V6 domain and that would obviate the need for DNS-ALG intervention.</p> <p>1)C Calls: <b>Translate (6to4).</b></p>
--	---

**Alternate Path:**

Actor/Action	System/Response
1. Node A's name resolver sends a name look up request for Node C.	<p>1)A V6 nodes alternatively learn the address of V4 nodes from the DNS server in the V4 domain in case it is not found in the DNS server internal to the V6 network.</p> <p>1)B Calls: <b>Translate (6to4).</b></p>

**Name:** Translate (6to4).

**Purpose:** To translate the IP address to the type it is destined to.

**Actors:**

**Precondition:** The address should be translatable.

**Post Condition:** The address is translated.

**Basic Path:**

Actor/Action	System/Response
	1) V6 node that needs to communicate with a V4 node needs to use a specific prefix (PREFIX::/96) in front of the

	IPv4 address of the V4 node. This 96-bit PREFIX is added by the Translator.
--	---

**Alternate Path:**

Action/Action	Expected Response
	1) The queries will cross the V6 domain and are subject to DNS-ALG intervention, which actually forwards this query to the external DNS server for the name look up.

In this chapter we have discussed in detail the basic requirements for a system, better than the current systems. In the next chapter, we shall describe the complete proposed architecture and detailed algorithm enhancements that we worked upon during the period of our work.

**CHAPTER 4**

**PROPOSED ARCHITECTURE**

## 4. Proposed Architecture

The basic area of our study and interest has been the NAT-PT. We studied its various aspects in detail and tried to figure out all its limitations. Amongst its limitations, one of the most costly ones is the fact that NAT-PT has to process on each and every packet that is traversed through it, thus increasing the CPU utilization and the overall processing cost of the system. This also adversely affects the performance and the efficiency of the system.

### 4.1 Overview

In our proposed system, we have made an effort to decrease the processing required on packets, in the scenario where a packet is traversing through the NAT-PT router. We have proposed a technique that may be used, to decrease the repeated processing required to calculate checksum on every packet passing through the translation gateway.

### 4.2 Components of the Proposed System

The architecture proposed by us, is shown in figure 4-1. It consists of the following three main components:

- **Header Analyzer**

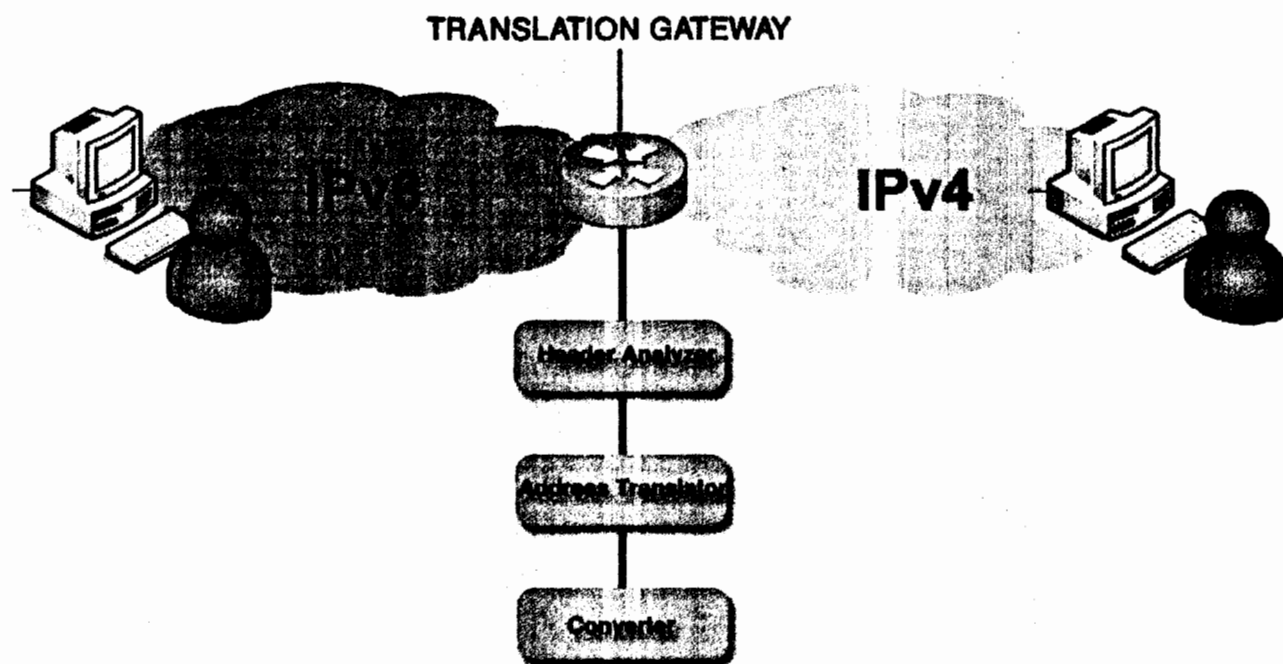
The Header Analyzer is the first component in the hierarchy. It reads through the header of the incoming IP packet to gather all the necessary information about the source and destination nodes and about the data that is being sent between the source and the destination. The Header Analyzer determines whether the IP packet being analyzed, belongs to an IPv4 node or an IPv6 node. It determines the type of the IP address of the source as well as the destination node.

- **Address Translator**

The Address Translator is responsible to translate between different types of addresses. By the time the Address Translator comes into action, it has already been established that the source node belonging to which protocol wants to communicate with the destination node belonging to which protocol. The Address Translator then translates the IPv4 address to an IPv6 address and vice versa.

- **Converter**

This is where the most practical work is performed. The IPv4 header is based on 32-bit architecture while the IPv6 header is based on 128-bit architecture. Due to this difference, it is not very simple to map one type of header to the other. A lot of considerations come into play. Each tiny bit of information is to be read from one header and fitted into the other, in the correct place. This task is done by the Converter. For example if a V4 header is to be converted to a V6 header, the converter will carry all the information from the V4 header to the V6 header and place it into the proper places so that there are no problems while reading this header later on. Secondly, the converter will add padding where ever required adjusting the V4 header into the 128-bit V6 header.



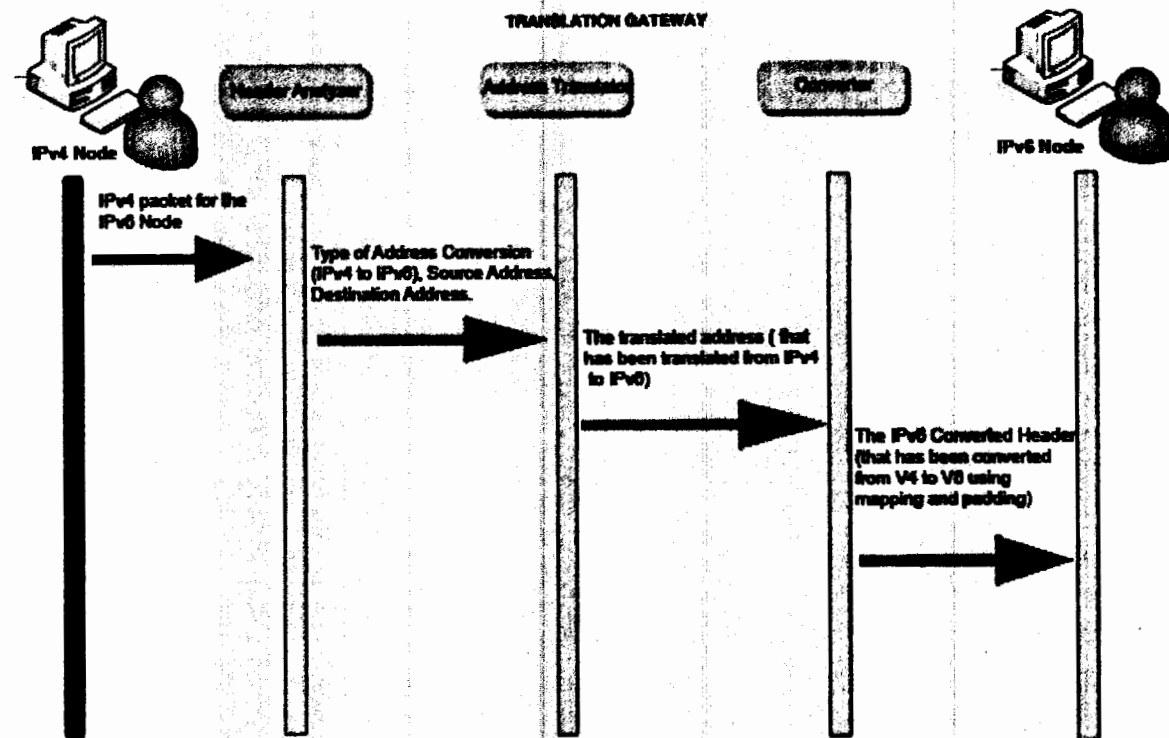
**Figure 4-1: Block diagram of the proposed architecture**

As shown in Figure 4-1, first the Header Analyzer comes into play, after its work is completed, the control is passed on to the Address Translator, which then passes all the gathered information about the header to the converter which actually performs the conversion of one header to the other. This transition/conversion is absolutely transparent to

the source as well as the destination node because each node will receive IP packets of the same type of protocol they belong to. For example the IPv6 nodes will always get IP packets with 128-bit addresses while the IPv4 nodes will always get IP packets with 32-bit addresses.

### 4.3 Data Flow across the Translation Gateway

Following is the graphical view of the interaction between the different components of the proposed system, when an IPv4 node sends a request to communicate with an IPv6 node:



**Figure 4-2: Sequence diagram of the proposed architecture**

Each component plays its part and moves the control onto the next component. The Header Analyzer and the Address Translator are in fact, the components that fulfill the pre-requisites of the header conversion process. In other words, the Header Analyzer and the Address Translator are the facilitators of the Converter. Once the converter has converted the header into an IPv6 header, for the IPv6 node or any other node in the IPv6 cloud, it is as if that packet was sent by another IPv6 node, and not by an IPv4 node. Thus, the conversion and in



general, the translation is transparent to the source and the destination node. Same procedure would be followed while converting from IPv6 header to IPv4 header.

#### 4.4 Our Approach

When ever an IPv4 packet is converted to IPv6, it is basically translated, thus the 32-bit IPv4 source address has to be converted to an IPv6 compatible address, so that it is compliant with the IPv6 nodes. An IPv6 compatible IPv4 address is created by adding a 96-bit prefix to the 32-bit IPv4 address, to make it 128-bit long [21], as shown in Figure 4-3.

Prefix	IPv4 Address
0000:0000:0000:0000:0000:0000 (96 Bits)	FE6A:412C (32 Bits)

Figure 4-3: IPv6 compatible IPv4 address

So the IPv6 compatible IPv4 address in the above example would be:

IPv6 Compatible IPv4 Address = 0000:0000:0000:0000:0000:0000: FE6A:412C

In the case of NAT-PT, each NAT enabled router is assigned a NAT prefix, discussed in the next section.

##### 4.4.1 NAT-PT Prefix

An IPv6 prefix with a length of 96bits must be specified for NAT-PT to use. The IPv6 prefix can be a unique local unicast prefix, a subnet of your allocated IPv6 prefix, or even an extra prefix obtained from your Internet service provider (ISP). The NAT-PT prefix is used to match a destination address of an IPv6 packet. If the match is successful, NAT-PT will use the configured address mapping rules to translate the IPv6 packet to an IPv4 packet. The NAT-PT prefix can be configured globally or with different IPv6 prefixes on individual interfaces. Using a different NAT-PT prefix on several interfaces allows the NAT-PT router to support an IPv6 network with multiple exit points to IPv4 networks.

Let us consider an example:

NAT Prefix: 2010:0000:0000:0000:0000:0000

IPv4 Address: FE6A:412C

IPv6 Compatible IPv4 Address = 2010:0000:0000:0000:0000:0000:FE6A:412C

All the IPv4 addresses, which are traversed through this NAT-PT router, and are translated to IPv6 will have the same NAT-Prefix, appended to them, to create their IPv6 compatible IPv4 addresses.

#### 4.4.2 TCP Checksum Recalculation

When ever the header of a packet is altered, its checksum is supposed to be recalculated.

*“Any transport or other upper-layer protocol that includes the addresses from the IP header in its checksum computation must be modified for use over IPv6, to include the 128-bit IPv6 addresses instead of 32-bit IPv4 addresses.”[22]*

Therefore we have to recalculate the TCP Checksum, every time we translate an IPv4 packet to IPv6.

#### 4.4.3 TCP Checksum Calculation

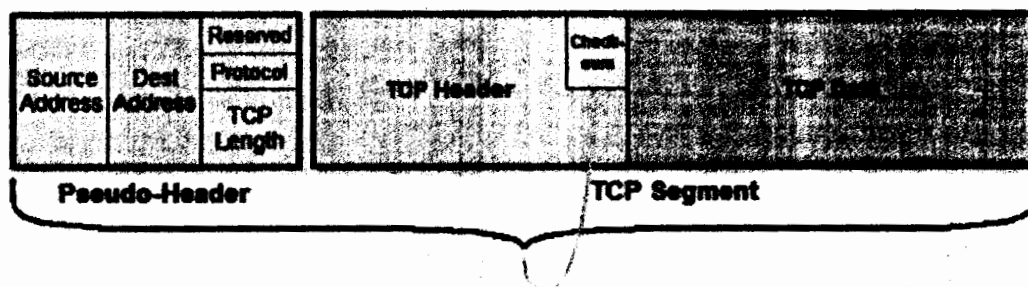
In current practice the TCP/UDP checksum is computed for each packet of data using the formula defined in RFC 2460, and that computation requires the following fields from the IP Header along with the TCP Header itself and its Data:

Octet Number	Len (Octets)	Field Name	Remarks / Description
0-3	4	Source	Source IP address
4-7	4	Destination	Destination IP address
8	1	Zero	Always zero
9	1	Protocol	Always 6 for TCP
10-11	2	Length	Length of TCP packet (excluding this pseudo header)

**Table 4-1: TCP Pseudo Header Format**

In standard practice, the TCP checksum is computed by including the above pseudo header plus the total TCP packet including the real TCP header. Following is the formula/algorithm (used in KAME Project, a very famous open source implementation of NAT-PT. Source Code is attached in Appendix B) to calculate the TCP Checksum:

$$\begin{aligned}
 &\textbf{TCP Checksum} \\
 &= \\
 &\quad \textbf{Ones Complement Sum of Pseudo Header} \\
 &\quad (\text{Source Address} + \text{Destination Address} + \text{Protocol} + \text{Header Length}) \\
 &\quad + \\
 &\quad \textbf{Ones Complement Sum of TCP Header} \\
 &\quad + \\
 &\quad \textbf{Ones Complement Sum of TCP Data}
 \end{aligned}$$



Checksum Calculated Over Pseudo Header and TCP Segment

**Figure 4-4: TCP Checksum Calculation Formula**

#### 4.4.4 Partial Source Address Calculation for TCP Checksum Calculation

As explained in section 4.4.1, each IPv4 packet that is translated to IPv6, will have the same prefix in its first 96 bits, i.e the NAT Prefix. Therefore, the first 96 bit calculation for the TCP Pseudo header will always be the same. We suggest that instead of processing these (first 96bits) for TECP Checksum calculation for every packet, we can save the partial calculation in the NAT-PT router and use it ever time a packet needs to be translated. Thus, the Partial Source Address calculation for TCP Checksum calculation would be as follows:

### Partial Source Address Calculation

=

*Ones Complement Sum of the 96 bits NAT Prefix*

#### 4.4.5 Partial Updated TCP Checksum Calculation Formula

So now, after implementing the partial calculation mentioned in the previous section, the TCP Checksum Calculation formula defined in 4.4.3 may be updated to the following formula:

*TCP Checksum*

=

**Partial Source Address Calculation**

*(Ones Complement Sum of the 96 bits NAT Prefix)*

+

*Ones Complement Sum of Pseudo Header (excluding first 96 bits)*

*(Last 32 bits of Source Address + Destination Address + Protocol + Header Length)*

+

*Ones Complement Sum of TCP Header*

+

*Ones Complement Sum of TCP Data*

This is the formula that we will use to calculate TCP Checksum. This formula is much more efficient than the one defined in section 4.4.3. The Results in Chapter 6 prove it as well.

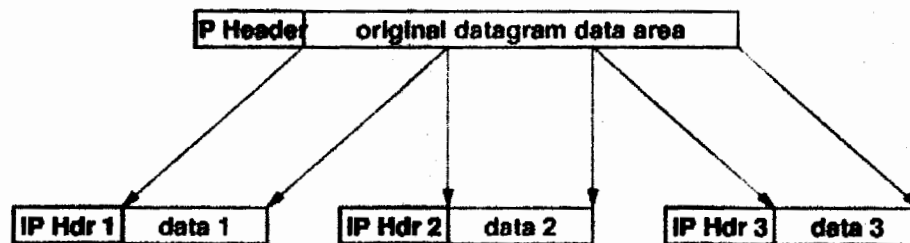
#### 4.4.6 TCP Checksum Calculation for Individual Packets

When an individual packet (with no fragments), destined for an IPv6 node, reaches our translation gateway, it will be translated normally and its TCP Checksum will be calculated according to the updated formula defined in section 4.4.5. The use of the updated TCP

Checksum Formula will enhance the processing rate of the translation gateway, as it will now use, the already calculated and saved partial calculation, to calculate the complete TCP Checksum.

#### 4.4.7 TCP Checksum Calculation for Fragmented Packets

Since the data whose size exceeds the size of path MTU, it has to be fragmented [Figure 4-3] into chunks or fragments [23]. We will first understand how fragmentation works. Here is how the fragments would be structured:



**Figure 4-5: Datagram fragmentation**

In this illustration, a large packet is broken down into three fragments. The sizes of the fields are shown to scale. The Unfragmentable Part (IP Header), shown in blue, begins each fragment, followed by the divided Data Area part of the original datagram data area. Consider the following example:

Original Datagram Data Area = 4000 Bytes.

Then following will be the structure of the fragments [24]:

1. **First Fragment:** The first fragment would consist of the 20-bytes Unfragmentable Part (IP Header), followed by 24-bytes of TCP Header, followed by 1456 bytes of data. This makes 1500 bytes of first fragment.
2. **Second Fragment:** The second fragment would consist of the 20-bytes Unfragmentable Part (IP Header), followed by 24-bytes of TCP Header, followed by 1456 bytes of data. This makes 1500 bytes of second fragment.

3. **Third Fragment** The third fragment would consist of the 20-bytes Unfragmentable Part (IP Header), followed by 24-bytes of TCP Header, followed by 956 bytes of data. This makes 1000 bytes of third fragment.

The "M" (More Fragments flag would be set to one in the first two fragments and zero in the third, and the Fragment Offset values would be set appropriately. The receiving device reassembles by taking the Unfragmentable Part from the first fragment and then assembling the Fragment data from each fragment in sequence.

For each fragment the TCP/IP headers are sent along with the payload and their checksum needs to be recalculated using the translated IP Header fields borrowed and placed in the prepended Pseudo Header.

The Table 4-1 clearly states that only the "Length" field would require re-computation, so what we can do here is that we can calculate the one's complement sum for the three afore mentioned unchanged fields and keep them in a table associated with the unique id assigned to every datagram that is transmitted in fragments. We will use this one time computed partial value of the Pseudo Header for the entire transmission of fragmented packet and that record would be washed out after the successful transfer of the entire fragmented packet. What we achieve by doing this is that we decrease the cost of processing involved by avoiding the occurrence of identical computation in replication.

To understand this process in detail we take an example of IPv4 fragmented packet which is received by the NAT-PT border router, once it is identified that the packet is fragmented, we assign a unique id to that datagram and calculate the partial checksum for it which is the sum of one's complement of the set of fields described earlier (fields that stay unchanged throughout this session) and we save this value in our "Checksum list for fragmented datagram", now for each consecutive fragment that is received the pre computed partial sum of Pseudo Header is used. On the completion of the fragments transfer (which is identified when the M flag in the fragment header is set to zero) the checksum information is deleted

since its TTL is only equal to the number of fragments or the session time set for the transfer of the fragmented datagram.

Datagram ID (8 bit)	Partial TCP Checksum (16 bit)
0000001	1000110101000100
0000002	1100001110001011
0000003	1100010010001011
.....	.....

**Table 4-2: Partial TCP Checksum Table**

Following is the flow of TCP Header Checksum Calculation, according to our proposed architecture:

**Step-1:** When a packet reaches the Translation Gateway, first it will check the Flags field. When a packet is fragmented all fragments have the MF flag set except the last fragment, which does not have the MF flag set. The MF flag is also not set on packets that are not fragmented.

**Step-2:** If the flag is not set to MF, then the translator will carry on with the way it normally works. But if the MF is set, i.e. the packet is fragmented and there are more fragments to come, it will read the identification field which is primarily used for uniquely identifying fragments of an original IP datagram.

**Step-3:** Search for the Identification Number of this packet (fragment), in the local Partial TCP Checksum Table (Table 4-3).

**Step-4:** If the Identification does not match, then the Partial Checksum is calculated (from the TCP Pseudo Header and saved in the Partial TCP Checksum Table against the packet identification. The Partial TCP Checksum is calculated using the following formula (which is infact a break-up of the original formula to calculate the complete TCP Checksum):

***Partial TCP Checksum***

$$\begin{aligned}
 &= \\
 &\quad \textbf{Ones Complement Sum of Source Address Words} \\
 &\quad + \\
 &\quad \textbf{Ones Complement Sum of Destination Address Words} \\
 &\quad + \\
 &\quad \textbf{Ones Complement Sum of Protocol Field Value}
 \end{aligned}$$

This partial checksum is used to calculate the checksum of the TCP Header also.

**Step-5:** If the Identification matches, then the corresponding Partial Checksum is read and used to calculate the TCP Header Checksum; thus avoiding the need to calculate the Ones Complement Sum of Source Address, Destination Address and the Protocol field value.

**Step-6:** If a packet with a matching Identification is traversed from the translator, which has the Flag field set to DF, which means that this is the last fragment of that packet, we will process it and delete the matching identification row from the Checksum list for fragmented datagram, so that more entries can be made in the table and only a reasonable amount of memory is required to maintain it. If the last fragment of a packet is not received in a while, the matching entry will automatically be deleted, after the timeout time is achieved.

## **4.5 Conclusion**

This chapter describes the proposed solution for computing the checksum for the TCP header of the fragmented datagrams. Using our methodology for calculating the TCP Checksum, the CPU Utilization and thus the processing time is minimized, that enforced great cost at the NAT-PT router. We have proposed a technique that reduces the need for a lot of processing, in order to calculate the TCP Header Checksum of fragmented and unfragmented packets. It is evident from the results in Chapter 6, that the updated TCP Checksum Calculation Formula proposed by us is logically and practically more efficient, as it requires lesser processing on each packet that traverses through the translation gateway. We used the



algorithm of the very famous open source implementation of NAT-PT, KAME Project, as a sample. We used the algorithm implemented by them, as the “Current Technique”, and modified that algorithm to form the “Proposed Technique”, which clearly proves to be an improved version of the Current Technique.

algorithm of the very famous open source implementation of NAT-PT, KAME Project, as a sample. We used the algorithm implemented by them, as the “Current Technique”, and modified that algorithm to form the “Proposed Technique”, which clearly proves to be an improved version of the Current Technique.

**CHAPTER 5**  
**IMPLEMENTATION**

## 5. Implementation

The simulation software, that tests and verifies the correctness of our technique and the improvement made by our proposed method, has been developed using Microsoft Visual Basic 6.0. The application has been made using object oriented implementation. Following are the major components of the simulation software:

### 5.1 Pseudo Code of Packet Translation Functions

#### 5.1.1 Convert IPv4 Packets to IPv6

Get Current Tick Count and Save in Variable as Start Tick Count

Display Current Tick Count on screen, in the "Start Tick Count" Section

Set Status to "Processing..."

For 1 to Number of Packets

    Read and Load IPv4Packet

    Fill IPv4 Packet data into clsIPv4 Class Object

    For 1 to Number of Fragments per Packet

        Convert IPv4 Packet To IPv6

        Calculate TCP Checksum of the converted IPv6 Packet

    Loop

Loop

Fill IPv6 Packet Data on Screen

Set Status to " DONE..."

Get Current Tick Count and Save in Variable as End Tick Count

Display Current Tick Count on screen, in the "End Tick Count" Section

Calculate and Display Processing Time (End Tick Count – Start Tick Count)

Append to the "Conversion Summary" Section

### **5.1.2 Calculate TCP Checksum of the converted IPv6 Packet**

Convert TCP Header (Hexa Decimal Format) to words of 16-bits each

Calculate Ones Complement Sum of All 16-bits words

Calculate Sum of all Ones Complemented 16-bits Words and save in variable.

If Current Technique is being used to Process the Packets then

    Calculate Partial Check Sum

Else if Proposed Technique is being used to process the packets then

    Check if the packet has fragments using the MF field in the header

        If more fragments follow this packet then

            Get Partial Check Sum for this packet from Partial Checksum Table

            If there is no entry for this packet in the Partial Checksum Table Then

                Save in Partial Checksum Table

            End If

        Else

            Calculate Partial Checksum for packet

        End If

End If

Calculate Sum as follows:

Sum = Sum of TCP Header + Partial TCP Checksum + TCP Length

Calculate Ones Complement of the Total Sum

Convert the Ones Complementd Total Sum to Hexa Decimal Format  
This Hexa Decimal Value is the TCP Checksum.

### 5.1.3 Calculate Partial Checksum

If Packet is Fragmented Then

Convert Source Address to 16-bit Words

Calculate Ones Complement of all 16-bit Source Address Words

Calculate Sum of all Ones Complementd Source Address Words

Else

Source Address = Last 32 Bits of Translated Source Address

Convert Source Address to 16-bit Words

Calculate Ones Complement of all 16-bit Source Address Words

Calculate Sum of all Ones Complementd Source Address Words

Add NATPrefixChecksum to the Sum Calculated so far

End If

Convert Destination Address to 16-bit Words

Calculate Ones Complement of all 16-bit Destination Address Words

Calculate Sum of all Ones Complementd Destination Address Words

Calculate Partial TCP Checksum using the following formula:

$\text{PartialTCPChecksum} = \text{Sum\_SourceAddress} + \text{Sum\_DestinationAddress} + \text{Protocol}$

## 5.2 Class Modules

Following are the class modules used by us:

### 5.2.1 clsIPV4

This class is used to hold the fields of the IPv4 Packets. Following are its variables.

#### Variables

Public Version

Public IHL

Public TypeOfService

Public TotalLength

Public Identification

Public Flags

Public FragmentOffset

Public TimeToLive  
Public Protocol  
Public HeaderChecksum  
Public SourceAddress  
Public DestinationAddress  
Public HeaderHexa  
Public Data

### 5.2.2 **clsIPv6**

This class is used to hold the fields of the IPv6 Packets. Following are its variables.

#### **Variables**

Public Version  
Public Class  
Public FlowLabel  
Public PayLoadLength  
Public NextHeader  
Public HopLimit  
Public SourceAddress  
Public DestinationAddress  
Public Data

### 5.2.3 **clsTCP**

This class is used to hold the fields of the TCP Header and Segment fields. Following are its variables.

#### **Variables**

Public SourcePort  
Public DestinationPort  
Public SequenceNumber  
Public AcknowledgeNumber  
Public HLEN  
Public Reserved  
Public CodeBits  
Public Window  
Public CheckSum  
Public UrgentPointer  
Public TCPOptions  
Public TCPData

### 5.2.4 clsTCP4Pseudo

This class is used to hold the fields of the TCP Pseudo Header for IPv4.

Following are its variables.

#### Variables

Public SourceAddress  
Public DestinationAddress  
Public Zeros  
Public Protocol  
Public TCPLength

### 5.2.5 clsTCP6Pseudo

This class is used to hold the fields of the TCP Pseudo Header for IPv6.

Following are its variables.

#### Variables

Public SourceAddress  
Public HexaSourceAddress  
Public DestinationAddress  
Public HexaDestinationAddress  
Public TCPLength  
Public HexaTCPLength  
Public Zeros  
Public HexaZeros  
Public NextHeader  
Public HexaNextHeader

## 5.3 Utility / Helping Functions

Following are the utility function used by us:

### 5.3.1 IntToBin

This function converts integer value to its equivalent binary.

*Function IntToBin(ByVal IntegerNumber As Long)*

*IntNum = IntegerNumber*

*Do*

*'Use the Mod operator to get the current binary digit from the  
'Integer number*

*TempValue = IntNum Mod 2*

*BinValue = CStr(TempValue) + BinValue*

*'Divide the current number by 2 and get the integer result*



```

        IntNum = IntNum \ 2
    Loop Until IntNum = 0
    IntToBin = BinValue
End Function

```

### 5.3.2 BinToInt

This function converts Binary value to its equivalent Integer

*Function BinToInt(ByVal BinaryNumber As String)*

```

    'Get the length of the binary string
    Length = Len(BinaryNumber)

    'Convert each binary digit to its corresponding integer value
    'and add the value to the previous sum
    'The string is parsed from the right (LSB - Least Significant Bit)
    'to the left (MSB - Most Significant Bit)
    For x = 1 To Length
        TempValue = TempValue + Val(Mid(BinaryNumber, Length - x + 1, 1))
        * 2 ^ (x - 1)
    Next
    BinToInt = TempValue
End Function

```

### 5.3.3 Dec2Bin

This function converts a Decimal value to its equivalent Binary

*Function Dec2Bin(InputData As Double) As String*

```

" Converts Decimal to Binary
" This uses the Quotient Remainder method

```

```

Dim Quot As Double
Dim Remainder As Double
Dim BinOut As String
Dim i As Integer
Dim NewVal As Double
Dim TempString As String
Dim TempVal As Double
Dim BinTemp As String
Dim BinTemp1 As String
Dim PosDot As Integer
Dim Temp2 As String

```

```

" Check to see if there is a decimal point or not
"

```

```

If InStr(1, CStr(InputData), ".") Then

```

```
MsgBox "Only Whole Numbers can be converted", vbCritical
GoTo eds
End If

BinOut = ""
NewVal = InputData

DoAgain:

" Start the Calculations off
NewVal = (NewVal / 2)

" If we have a remainder
If InStr(1, CStr(NewVal), ".") Then
    BinOut = BinOut + "1"

" Get rid of the Remainder
NewVal = Format(NewVal, "#0")
NewVal = (NewVal - 1)

If NewVal < 1 Then
    GoTo DoneIt
End If
Else
    BinOut = BinOut + "0"
    If NewVal < 1 Then
        GoTo DoneIt
    End If
End If
GoTo DoAgain
DoneIt:
BinTemp = ""
" Reverse the Result
For i = Len(BinOut) To 1 Step -1
    BinTemp1 = Mid(BinOut, i, 1)
    BinTemp = BinTemp + BinTemp1
Next i
BinOut = BinTemp
" Output the Result
Dec2Bin = BinOut
eds:
End Function
```

**5.3.4 Bin2Hex**

This function converts a Binary value to its equivalent Hexadecimal format.

```

Function Bin2Hex(InputData As String) As String
    " Converts Binary to hex
    Dim i As Integer
    Dim LenBin As Integer
    Dim JOne As String
    Dim NumBlocks As Integer
    Dim FullBin As String
    Dim HexOut As String
    Dim TempBinBlock As String
    Dim TempHex As String

    LenBin = Len(InputData)

    "
    " Make sure that it is a Binary Number
    "
    For i = 1 To LenBin
        JOne = Mid(InputData, i, 1)
        If JOne <> "0" And JOne <> "1" Then
            MsgBox "NOT A BINARY NUMBER", vbCritical
            Exit Function
        End If
    Next i

    " Set the Variable to the Binary
    "
    FullBin = InputData

    "
    " If the value is less than 4 in length, build it up.
    "
    If LenBin < 4 Then
        If LenBin = 3 Then
            FullBin = "0" + FullBin
        ElseIf LenBin = 2 Then
            FullBin = "00" + FullBin
        ElseIf LenBin = 1 Then
            FullBin = "000" + FullBin
        ElseIf LenBin = 0 Then
            MsgBox "Nothing Given..", vbCritical
            Exit Function
        End If
        NumBlocks = 1
    
```

```
GoTo DoBlocks
End If

If LenBin = 4 Then
    NumBlocks = 1
    GoTo DoBlocks
End If

If LenBin > 4 Then

    Dim TempHold As Currency
    Dim TempDiv As Currency
    Dim AfterDot As Integer
    Dim Pos As Integer

    TempHold = Len(InputData)
    TempDiv = (TempHold / 4)

    "
    " Works by seeing whats after the deciomal place
    "
    Pos = InStr(1, CStr(TempDiv), ".")

    If Pos = 0 Then
        " Divided by 4 perfectly
        NumBlocks = TempDiv
        GoTo DoBlocks
    End If

    AfterDot = Mid(CStr(TempDiv), (Pos + 1))

    If AfterDot = 25 Then
        FullBin = "000" + FullBin
        NumBlocks = (Len(FullBin) / 4)
    ElseIf AfterDot = 5 Then
        FullBin = "00" + FullBin
        NumBlocks = (Len(FullBin) / 4)
    ElseIf AfterDot = 75 Then
        FullBin = "0" + FullBin
        NumBlocks = (Len(FullBin) / 4)
    Else
        MsgBox "Big Time Screw up happened, WAHHHHHHHHHHHH", vbInformation
        Exit Function
    End If
```

*GoTo DoBlocks*  
*End If*

*"*

*" The rest will process the now built up number*

*"*

*DoBlocks:*

*HexOut = ""*

*For i = 1 To Len(FullBin) Step 4*  
*TempBinBlock = Mid(FullBin, i, 4)*

*If TempBinBlock = "0000" Then*  
*HexOut = HexOut + "0"*  
*ElseIf TempBinBlock = "0001" Then*  
*HexOut = HexOut + "1"*  
*ElseIf TempBinBlock = "0010" Then*  
*HexOut = HexOut + "2"*  
*ElseIf TempBinBlock = "0011" Then*  
*HexOut = HexOut + "3"*  
*ElseIf TempBinBlock = "0100" Then*  
*HexOut = HexOut + "4"*  
*ElseIf TempBinBlock = "0101" Then*  
*HexOut = HexOut + "5"*  
*ElseIf TempBinBlock = "0110" Then*  
*HexOut = HexOut + "6"*  
*ElseIf TempBinBlock = "0111" Then*  
*HexOut = HexOut + "7"*  
*ElseIf TempBinBlock = "1000" Then*  
*HexOut = HexOut + "8"*  
*ElseIf TempBinBlock = "1001" Then*  
*HexOut = HexOut + "9"*  
*ElseIf TempBinBlock = "1010" Then*  
*HexOut = HexOut + "A"*  
*ElseIf TempBinBlock = "1011" Then*  
*HexOut = HexOut + "B"*  
*ElseIf TempBinBlock = "1100" Then*  
*HexOut = HexOut + "C"*  
*ElseIf TempBinBlock = "1101" Then*  
*HexOut = HexOut + "D"*  
*ElseIf TempBinBlock = "1110" Then*  
*HexOut = HexOut + "E"*  
*ElseIf TempBinBlock = "1111" Then*

```

    HexOut = HexOut + "F"
End If

Next i

```

```

Bin2Hex = HexOut

```

```

eds:
End Function

```

### 5.3.5 AddBinaryNumbers

This function is used to add two binary numbers.

```

Function AddBinaryNumbers(strOne As String, strTwo As String) As String

```

```

    Dim strSum, ch1, ch2, chR, Carry As String

```

```

    Carry = "0"

```

```

    For i = 0 To Len(strOne) - 1

```

```

        ch1 = Mid(strOne, Len(strOne) - i, 1)
        ch2 = Mid(strTwo, Len(strTwo) - i, 1)

```

```

        If ch1 = "0" And ch2 = "0" Then

```

```

            If Carry = "1" Then

```

```

                chR = "1"

```

```

                Carry = "0"

```

```

            Else

```

```

                chR = "0"

```

```

                Carry = "0"

```

```

            End If

```

```

        End If

```

```

        If (ch1 = "1" And ch2 = "0") Or (ch1 = "0" And ch2 = "1") Then

```

```

            If Carry = "1" Then

```

```

                chR = "0"

```

```

                Carry = "1"

```

```

            Else

```

```

                chR = "1"

```

```

                Carry = "0"

```

```

            End If

```

```

        End If

```

```

        If ch1 = "1" And ch2 = "1" Then

```

```

    If Carry = "1" Then
        chR = "1"
        Carry = "1"
    Else
        chR = "0"
        Carry = "1"
    End If
End If

```

```
strSum = strSum + chR
```

```
Next i
```

```
AddBinaryNumbers = StrReverse(strSum)
```

```
End Function
```

### 5.3.6 CalculateOnesComplement

This function calculates the One's Complement.

```
Function CalculateOnesComplement(Value As Double)
```

```
    Dim lngComp As Long
```

```
    lngComp = Value Xor &H7FFFFFFF
```

```
    Dim strComplement As String
```

```
    strComplement = IntToBin(lngComp)
```

```
    'CalculateOnesComplement = strComplement
```

```
    CalculateOnesComplement = Mid(strComplement, Len(strComplement) - 16, 16)
```

```
End Function
```

### 5.3.7 ConvertTo16bitWords

This function converts the input data into 16bit words array.

```
Function ConvertTo16bitWords(ByRef Words() As String, data)
```

```
    Dim str As String
```

```
    str = data
```

```
    Dim word, bytecount, WordCount
```

```
    word = "0x"
```

```
    bytecount = 0
```

```
    WordCount = 0
```

```
    Dim ch
```

```
    ReDim Words((Len(str) / 4) + 1)
```

```
    For i = 1 To Len(str)
```

```

    ch = Mid(str, i, 1)

    If ch = "\n" Then
        ch = " "
    End If

    If ch = "\t" Then
        ch = " "
    End If

    If ch = "," Then
        ch = " "
    End If

    If ((ch >= "0" And ch <= "9") Or (ch >= "a" And ch <= "f") Or (ch >= "A"
And ch <= "F")) Then

        word = word + ch
        bytcount = bytcount + 1

    ElseIf ch <> " " Then
        MsgBox (ch + "is not a valid hex character")
    End If

    If bytcount = 4 Or (ch = " " And bytcount > 0) Then

        Words(WordCount) = word
        WordCount = WordCount + 1
        word = "0x"
        bytcount = 0

    End If
Next i

If bytcount > 0 Then
    Words(WordCount) = word
End If

```

*End Function*

### 5.3.8 PadZerosToLeft

This function pads Zero's to the left.

*Function PadZerosToLeft(str As String, TotalLength As Integer)*

*For i = 1 To TotalLength - Len(str)*



*strLeft = strLeft + "0"*

*Next i*

*PadZerosToLeft = strLeft + str*

*End Function*

## 5.4 Main Coding Loop

When the "Convert" button is clicked, that is when the applications starts its actual processing, and the packets are converted from one format to the other. Following is the make loop of code executed:

*bln = ReadAndLoadInput("Data.DAT", 4, objIPV4, objTCP4Pseudo, objIPV6, objTCP6Pseudo, objTCP)*

*Call Fill\_IPV4\_Data*

*For i = 1 To cmbNumberOfPackets*

*objIPV4.Identification = IntToBin(BinToInt(objIPV4.Identification) + 1)*

*For j = 1 To cmbNumberOfFragments*

*Call ConvertIPV4ToIPV6(objIPV4, objIPV6)*

*Call CalculateTCPChecksum*

*Next j*

*Next i*

*Call Fill\_IPV6\_Data*

*Call DisplayPartialCheckSumTable*

*lblAction.Caption = "DONE..."*

*lngEndTicker = GetTickCount*

*EndTime = DateTime.Time*

*lblEndTime.Caption = EndTime*

*lblProcessingTime.Caption = CStr(lngEndTicker - lngStartTicker) + " msec"*

*Call UpdateSummary*

### 5.4.1 ReadAndLoadInput

This function reads the data and loads it into the respective objects.

### 5.4.2 Fill\_IPV4\_Data

This function will fill the clsIPv4 object with the received IPv4 Data.

### 5.4.3 ConvertIPV4ToIPV6

The actual header conversion from IPv4 to IPv6 is done here.

*Public Sub ConvertIPV4ToIPV6(objIPV4 As clsIPv4, objIPV6 As clsIPv6)*

*Dim NATPrefix As String*

*With objIPV6*

*.Version = "0110"*

*.Class = objIPV4.TypeOfService*

*.FlowLabel = "00000000000000000000"*

*.PayloadLength = objIPV4.TotalLength*

*.NextHeader = objIPV4.Protocol*

*.HopLimit = objIPV4.TimeToLive*

*.SourceAddress = NATPrefix + objIPV4.SourceAddress*

*.DestinationAddress = NATPrefix + objIPV4.DestinationAddress*

*.data = objIPV4.data*

*End With*

*End Sub*

### 5.4.4 CalculateTCPChecksum

This is the function that actually calculates the TCP Checksum of all incoming data:

*Public Sub CalculateTCPChecksum()*

*Dim Words() As String*

*Dim Sum, Sum\_TCP, Sum\_TCPLength As Long*

*Dim CheckSum As String*

*txtChecksum.Text = "caculating ..."*

*Call ConvertTo16bitWords(Words, objTCP.HeaderHexa)*

*Call CalculateOnesComplementOfAllWords(Words)*

*Sum\_TCP = CalculateSumOfTCP(Words)*

*If rdbCurrent.Value = True Then*

*PartialTCPChecksum = CalculatePartialCheckSum()*

*Else*

*'Check if the packet has fragments using the MF field in the header*

*If objIPV4.Flags = "001" Then 'It means more fragments to follow*

*PartialTCPChecksum = GetPartialCheckSum(objIPV4.Identification)*

*If PartialTCPChecksum = -1 Then 'No Entry was found*

*PartialTCPChecksum = CalculatePartialCheckSum()*

*Call AddPartialCheckSumEntry(objIPV4.Identification,*

*PartialTCPChecksum)*

*End If*

*End If*

*End If*

*Sum\_TCPLength = Hex2Dec(objTCP6Pseudo.HexaTCPLength)*

*Sum = Sum\_TCP + PartialTCPChecksum + Sum\_TCPLength*

*'Ones Complement of the Total Sum*

*Checksum = Sum Xor &H7FFFF*

*txtChecksum.Text = IntToBin(CheckSum)*

*txtChecksumHex.Text = Bin2Hex(IntToBin(Sum))*

*'wordsToHec (Words)*

*End Sub*

#### **5.4.5 CalculatePartialCheckSum**

This function calculates the Partial Check Sum for our technique..

*Function CalculatePartialCheckSum() As Long*

```

    Dim TCPPpseudoWords() As String
    Dim TCPPpseudoWords_SA() As String
    Dim TCPPpseudoWords_DA() As String
    Dim TCPPpseudoWords_Pr() As String
    Dim TCPPpseudoWords_TL() As String
    Dim Sum, Sum_SourceAddress, Sum_DestinationAddress, Sum_Protocol,
    Sum_TCPLength As Long

    Call ConvertTo16bitWords(TCPPpseudoWords_SA,
    objTCP6Pseudo.HexaSourceAddress)

    Call CalculateOnesComplementOfAllWords(TCPPpseudoWords_SA)

    Sum_SourceAddress = CalculateSumOfTCPPpseudoHeader(TCPPpseudoWords_SA)

    Call ConvertTo16bitWords(TCPPpseudoWords_DA,
    objTCP6Pseudo.HexaDestinationAddress)

    Call CalculateOnesComplementOfAllWords(TCPPpseudoWords_DA)

    Sum_DestinationAddress =
    CalculateSumOfTCPPpseudoHeader(TCPPpseudoWords_DA)

    Sum_Protocol = Hex2Dec(objTCP6Pseudo.HexaNextHeader)

    PartialTCPChecksum = Sum_SourceAddress + Sum_DestinationAddress +
    Sum_Protocol

    CalculatePartialChecksum = PartialTCPChecksum

End Function

```

#### 5.4.6 CalculateSumOfTCPPpseudoHeader

This function calculates the Partial Sum of TCP Pseudo Header..

*Function CalculateSumOfTCPPpseudoHeader(ByRef PseudoWords() As String)*

```

    Dim Sum As Long
    Sum = 0

    For i = 0 To UBound(PseudoWords) - 1

```

```
If PseudoWords(i) <> "" And PseudoWords(i) <> " " Then  
    Sum = Sum + CLng(BinToInt(PseudoWords(i)))  
End If
```

```
Next i
```

```
CalculateSumOfTCPPseudoHeader = Sum
```

```
End Function
```

#### 5.4.7 AddPartialChecksumEntry

This function calculates the Partial Sum of TCP Pseudo Header..

```
Public Sub AddPartialChecksumEntry(strIdentification As String, PartialChecksum As  
Long)
```

```
    arrPartialChecksums(arrEntryCount, 0) = Bin2Hex(strIdentification)  
    arrPartialChecksums(arrEntryCount, 1) = CStr(PartialChecksum)  
    arrEntryCount = arrEntryCount + 1
```

```
End Sub
```

## **CHAPTER 6**

### **RESULTS**

## 6. Results

After importing the data stream in our Header Checksum Query Analyzer and executing the current algorithm for the checksum calculation and our proposed method for the same we obtained the following results.

The results are displayed on the screen along with the calculated headers and the checksum value.

### 6.1 Main Screen

TCP Header Checksum Analyzer.exe Application is executed from the Application folder and following screen Fig 6.1 is displayed.

**IPv4 / IPv6 Translation Gateway** Gateway Profile = 2082-8000-8000-8000-8000-8001 (2082: 1/36) Exit

**Current Technique**

IPv4 Packet  
 Ver(4) IHL(4) 108 (8)  
 0100 0101 00000000  
 Total Length (16)  
 0000000000010101  
 Identification (16)  
 0101011101001001  
 Flags(3) Fragment Offset(16)  
 001 000000000000  
 TTL (8) Protocol (8) Header Checksum (16)  
 11111010 00001110 100010101110111  
 Source Address (32)  
 11000111101101100111100000001110  
 Destination Address (32)  
 1100111011010110101010101010000  
 Data  
 0010101101001110101011001000000100000110000  
 101110011011100110110111011011101110010011001  
 00011000000111001001100101011100010110101101

**Proposed Technique**

IPv6 Packet  
 Ver(4) Class (8)  
 0110 00000000  
 Flow Label (20)  
 00000000000000000000  
 Payload Length (16) Next Header(8) Hop Limit (8)  
 000000000101011 0000110 11111010  
 Source Address (128)  
 001000  
 00  
 00000111010110110110110110110100000001110  
 Destination Address (128)  
 001000  
 00  
 000001110101101101101101101101010101010000  
 Data  
 0010101101011110101010101010000000010100001100  
 0010111001101100110110011011011011011101100101  
 1001000010000001100100110010101110001011101010

**Convert from IPv4 to IPv6**

Start Tick Count: 1000000000  
 End Tick Count: 1000000000  
 Processing Time: 1.140 msec

**DONE...**

**Conversion Summary**

Time: 5.474544  
 Packets processed = 10  
 Fragments per Packet = 10  
 Total Fragments = 100  
 Technique Used = Current  
 Processing Time = 140 msec

Time: 5.474544  
 Packets processed = 1  
 Fragments per Packet = 10  
 Total Fragments = 10  
 Technique Used = Current

**TCP Checksum**

1's Compl Sum of Pseudo Hdr:  
 1's Compl Sum of TCP Hdr:  
 1's Compl Sum of TCP Data:

**TCP Pseudo Header (IPv6)**

Source Address (128)  
 001000  
 00  
 00000111010110110110110110110100000001110  
 Destination Address (128)  
 001000  
 00  
 000001110101101101101101101101010101010000  
 Zero (8) Protocol (8) TCP Length (16)  
 00000000 0000110 0000000000010010

**TCP CHECKSUM** 1011110010110010101  
 21A6A

Show Addresses in Hexadecimal

Fig 6.1 Main Screen of Simulation Software

## 6.2 Result Screen

After Start Process is clicked the process will start running and after completion of the process the following Fig 6.2 screen will be displayed.

DONE...		TCP Pseudo Header (IPv6)		
Start Tick Count	00000000000000000000000000000000	Source Address (128)		
End Tick Count	00000000000000000000000000000000	00000000000000000000000000000000 00000000000000000000000000000000 000000000011000111101101100111100000001110		
Processing Time	00000000000000000000000000000000	Destination Address (128)		
		00000000000000000000000000000000 00000000000000000000000000000000 0000000000110011101101101001010101010000		
		Zeros (8)	Protocol (8)	TCP Length (16)
		00000000	00000110	00000000000010010
		CHECKSUM 110101110011111100		

**Fig 6-2 Result Screen**

The 'Result Screen' above shows the TCP Pseudo Header along with the calculated checksum value and processing details elaborated below. All the data is displayed in binary format by default.

Start Tick Count	00000000000000000000000000000000
End Tick Count	00000000000000000000000000000000
Processing Time	00000000000000000000000000000000

**Fig 6-3 Time Panel**

The 'Time Panel' shows the start time (tick count), end time (tick count) and the Processing time (milliseconds). This is the actual time taken to process the selected number of packets.



[illegible]

### Fig 6-4 IPv4 Packet Panel

**'IPv4 Packet Panel' shows the IPv4 header and data created from the input data stream which is also displayed in binary format.**

[illegible]

### Fig 6-4-a IPv6 Packet Panel

**‘IPv6 Packet Panel’ shows IPv6 header and data translated from the IPv4 packet which is also displayed in binary format. This is done by using the algorithm defined in RFC 2766 [14]**

**TCP Pseudo Header [IPv6]**

Source Address (128)		
9000		
99		
A99900		

Destination Address (128)		
00		
9000		
999900		

Zero (8)	Protocol (8)	TCP Length (16)
00000000	00000010	999000000000000000

### Fig 6-4-b TCP Pseudo Header Panel

**'TCP Pseudo Header Panel'** shows the TCP Pseudo Header with fields inherited from the IP Header.

TCP CHECKSUM	1011110010110010101
	21A6A

### Fig 6-4-c Checksum Panel

**'Checksum Panel'** shows the checksum calculated using the Checksum algorithm. [17]

☐ Current Technique   ☒ Proposed Technique

### Fig 6-5 Algorithm Options Panel

**‘Algorithm Options Panel’ allow user to select between the existing technique for checksum calculation and the proposed technique.**

Partial CheckSum Table		
Identification	.	Partial Checksum
574A	.	89112
574B	.	89113
574C	.	89114
574D	.	89115
574E	.	89116
574F	.	89117
5750	.	89118
5751	.	89119
5752	.	89120
5753	.	89121

### Fig 6-6 Processing Status Panel

'Processing Status Panel' displays the 'Partial Checksum Table' maintained by the Translation Gateway, Identification is inherited from the packet and the partial checksum is calculated using the proposed technique.

Conversion Summary
Time :2:17:06 AM
Packets Processed = 10
Fragments per Packet = 10
Total Fragments = 100
Technique Used = Proposed
Processing Time = 156 msec
.....

**Fig 6-7 Conversion Summary Panel**

'Conversion Summary Panel' shows the completed summary of the conversion process displaying the processing time, number of packets processed, fragments per packet(in case of fragmented packets) and the technique used.

### 6.3 Comparison based on Number of Packets (No Fragmentation)

#### 6.3.1 Sample Size for Test

	Number of Packets
Case 1	1000
Case 2	5,000
Case 3	10,000
Case 4	20,000
Case 5	100,000

**Fig 6-8 Sample Size for Test**

The table above shows the number of packets we took for each sample of data, for the exercise of performance testing, in the case where no fragmentation occurs.

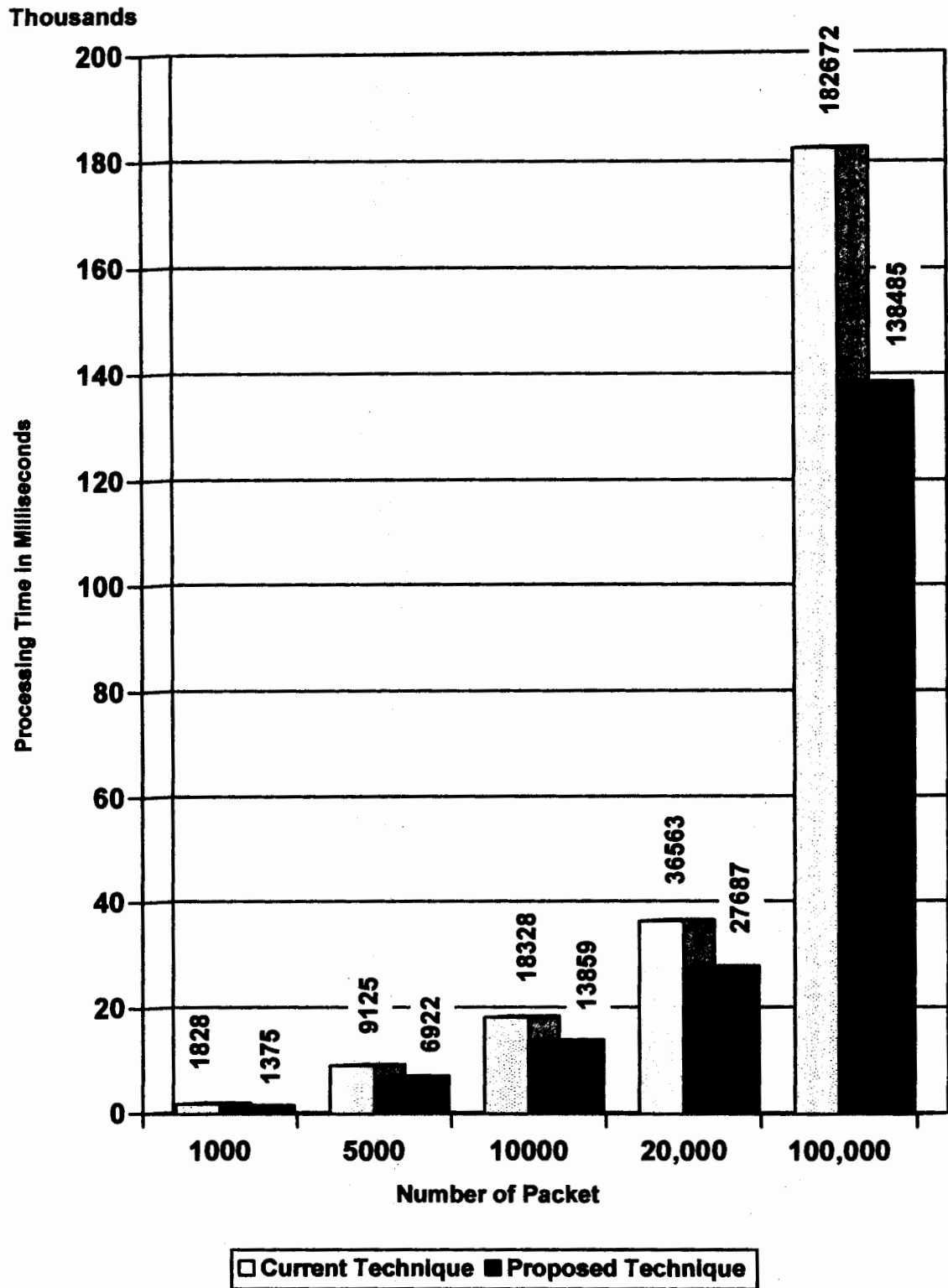
### 6.3.2 Comparison Results

	Processing Time	
	Current Technique	Proposed Technique
Case 1	1828 msec	1375 msec
Case 2	9125 msec	6922 msec
Case 3	18328 msec	13859 msec
Case 4	36563 msec	27687 msec
Case 5	303000 msec	3000031 msec

**Fig 6-9 Comparison Results**

The table above shows the processing time taken by our testing systems, to process the respective number of packets for each sample of data, for the exercise of performance testing, in the case where no fragmentation occurs. It is clearly seen from the results that the proposed technique results in lesser time required to process the same number of packets, as compared to the current technique.

## 6.3.3 Comparison Graph



## 6.4 Comparison based on Number of Fragments Per Packet (Fixed No. of Packets)

### 6.4.1 Sample Size for Test

	Number of Packets	Fragments per Packet
Case 1	1000	5
Case 2	1000	10
Case 3	1000	20
Case 4	1000	50

Fig 6-10 Sample Size for Test

The table above shows the number of packets we took for each sample of data, for the exercise of performance testing, in the case where the packets were fragmented.

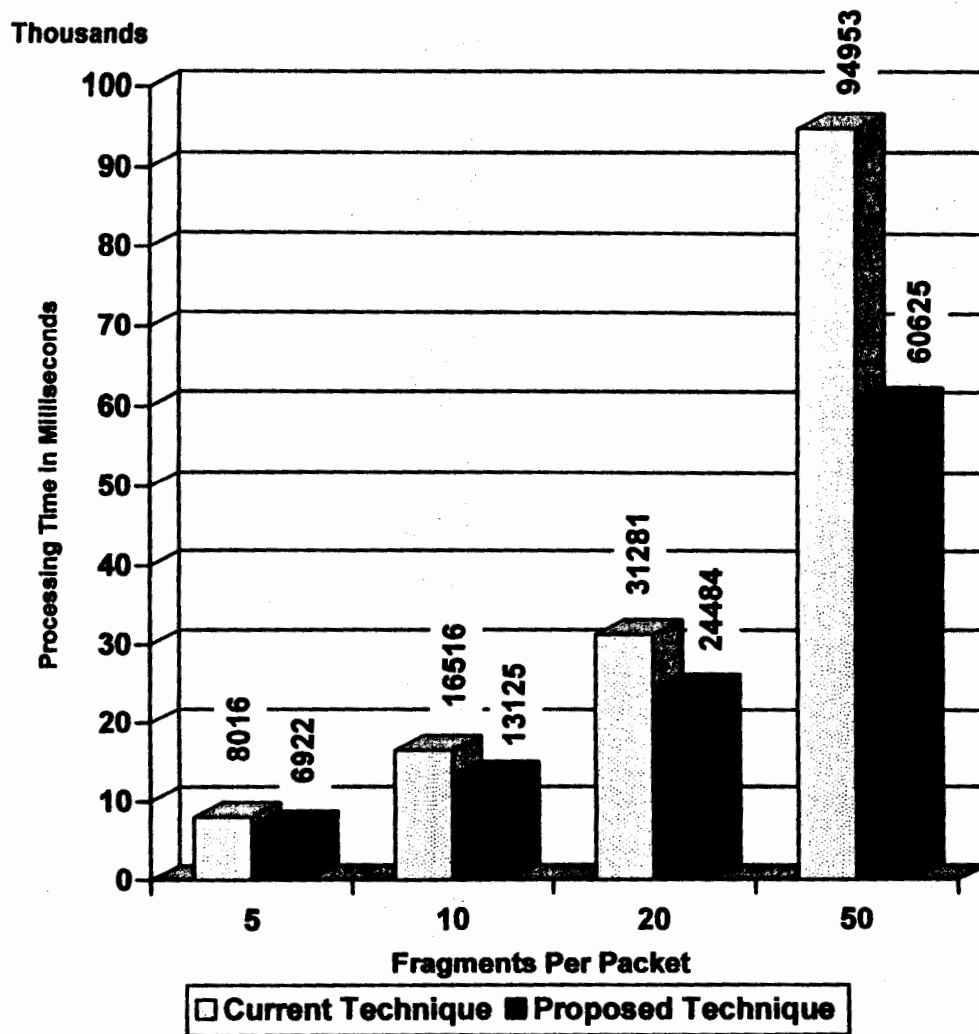
### 6.4.2 Comparison Results

	Processing Time	
	Current Technique	Proposed Technique
Case 1	8016 msec	6922 msec
Case 2	16516 msec	13125 msec
Case 3	31281 msec	24484 msec
Case 4	94953 msec	60625 msec

Fig 6-11 Comparison Results

The table above shows the processing time taken by our testing systems, to process the respective number of packets for each sample of data, for the exercise of performance testing, in the case where the packets were fragmented. It is clearly seen from the results that the proposed technique results in lesser time required to process the same number of packets, as compared to the current technique.

### 6.4.3 Comparison Graph



### 6.5 Comparison based on Number of Packet (Fixed No. of Fragments)

#### 6.5.1 Sample Size for Test

	Number of Packets	Fragments per Packet
Case 1	1	5
Case 2	10	5
Case 3	100	5
Case 4	1000	5

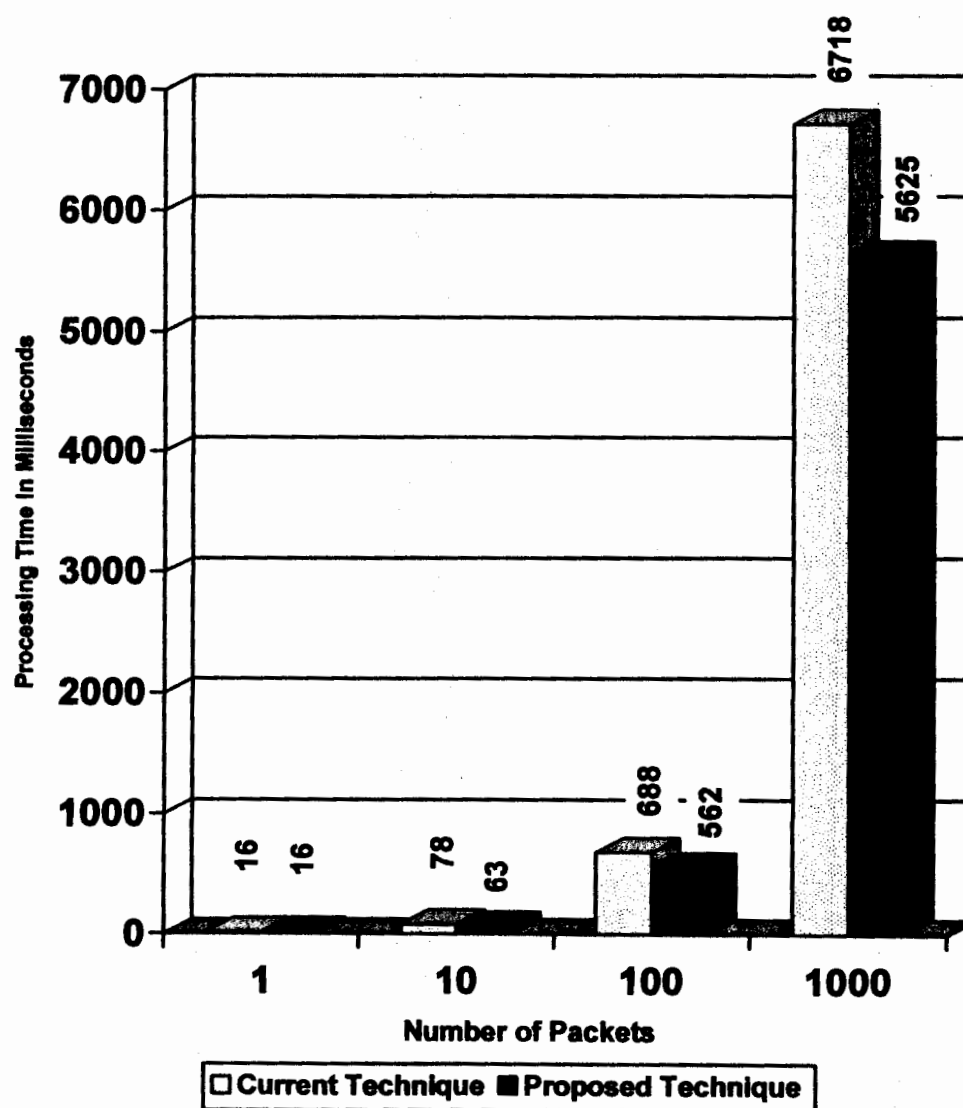
Fig 6-12 Sample Size for Test

## 6.5.2 Comparison Results

	Processing Time	
	Current Technique	Proposed Technique
Case 1	16 msec	16 msec
Case 2	78 msec	63 msec
Case 3	688 msec	562 msec
Case 4	6718 msec	5625 msec

Fig 6-13 Comparison Results

## 6.5.3 Comparison Graph





The graph convincingly displays the better performance of the proposed technique, as compared to the current one.

### **6.6 Result Analysis and Conclusion**

From all the results given above, and from other testing scenarios, it was observed that if the packets are fragmented, then under all circumstances, the proposed system will take less processing time, as compared to the time taken by the current technique. The proposed technique is always more efficient than the current one.

It was also observed that if the number of fragments per packet is increased, the performance of the proposed architecture increases even more, and it will save even more time. The increase in the number of fragments per packet has a greater effect than the effect caused by increasing the number of packets.

In the case where the packets are not fragmented, there is an efficiency increase, caused by our technique.

Thus, it is concluded that the proposed system has proven to make the translation process more efficient in case of fragmented, as well as unfragmented traffic.

## **CHAPTER 7**

# **CONCLUSION AND FUTURE ENHANCEMENT**

## **7. Conclusion and Future Enhancements**

The simulation software show the result that the processing time has decreased, in case of fragmented packets, thus decreasing the processing cost at the NAT-PT router. Processing speed and CPU Utilization are vital aspects of networks. We have managed to make a contribution to the cause of increasing processing speed and reducing CPU Utilization.

### **7.1 Conclusion**

Our enhancement to the TCP checksum calculation for both fragmented and unfragmented data has resulted in lesser computation required for creating the checksum of the TCP/UDP headers. This will definitely reduce the work load on the router containing the NAT-PT and thus its cost. The lesser the load on the router, the better the efficiency and speed experienced by the traffic on the network.

### **7.2 Future Enhancements**

Research is open for enhancements or improvement in term of cost or storage efficiency, and even the tiniest of enhancement in these two departments will make a exceptionally big difference, specially due to the increased usage of networks and the increased size of data that is regularly sent over the networks. Algorithms other than the checksum calculation algorithm should also be analyzed and exceptions should be detected and intelligent algorithms should be designed, even for scenario based working.

**APPENDIX A**  
**REFERENCES**

## References

- [1] **"A Protocol for Packet Network Interconnection"**, V. Cerf and R. Kahn, IEEE Transport of Communications Com-22 (5) 637-648, May 1974
- [2] **"Introduction to IPv6"**,  
<http://www.microsoft.com/windowsserver2003/technologies/ipv6/introipv6.mspix>,  
27.03.2003
- [3] **"Internet Protocol"**, Information Sciences Institute, IETF RFC 791, September 1981.
- [4] **"The case for IPv6"**, S. King, R. Fax, D. Haskin, W. Ling, T. Meehan, R. Fink, C.E. Perkins, Internet draft, version 6, <http://www.6bone.net/misc/case-for-ipv6.html>,  
25.12.2002
- [5] **"Introduction to IPv6"**,  
<http://www.microsoft.com/windowsserver2003/technologies/ipv6/introipv6.mspix>, ",  
March 2003.
- [6] **"Tech Info - Ipv6"** <http://www.zytrax.com/tech/protocols/ipv6.html>
- [7] **"The ABCs of IP Version 6"**,  
[http://www.sixxs.net/archive/docs/Cisco IPv6 ABC.pdf](http://www.sixxs.net/archive/docs/Cisco%20IPv6%20ABC.pdf)
- [8] **"NAT-PT Documentation"**, <http://tomicki.net/naptldocs.php>
- [9] **"IPv6 Transition and Co-existence"**,  
<http://www.6diss.org/workshops/ca/integration.pdf>
- [10] **"Transition Mechanisms for IPv6 Hosts and Routers"**, R. Gilligan, E. Nordmark, IETF RFC 2893, August 2000.
- [11] **IPv4 to IPv6 scoping report for end site networks/universities**, Tim Chown (University of Southampton), D2.3.1, 6NET.
- [12] **"Transition Mechanisms for IPv6 Hosts and Routers"**, R. Gilligan, E. Nordmark, IETF RFC 1933, April 1996.
- [13] **"Stateless IP/ICMP Translation Algorithm (SIIT)"**, E. Nordmark, IETF RFC 2765, Feb 2000
- [14] **"Network Address Translation - Protocol Translation (NAT-PT)"**, G. Tsirtsis, P. Srisuresh, IETF RFC 2766, Feb 2000

- [15] **"The IP Network Address Translator (NAT)"**, Egevang, K. and P. Francis ,RFC 1631, May 1994.
- [16] **"Computer Networking: A Top Down Approach Featuring the Internet"**, 2nd edition, Jim Kurose, Keith Ross, Addison-Wesley, July 2002.
- [17] **"TCP Checksum Calculation and the TCP Pseudo Header"**,  
[http://www.tcpipguide.com/free/t\\_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm](http://www.tcpipguide.com/free/t_TCPChecksumCalculationandtheTCPPseudoHeader-2.htm)
- [18] **"Transmission Control Protocol"**, Information Sciences Institute, IETF RFC 793, September 1981.
- [19] **"An IPv6-to-IPv4 Transport Relay Translator"**, J. Hagino et al. RFC 3142, June 2001
- [20] **"SOCKS Protocol Version 5"**, M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones, RFC 1928, March 1996
- [21] **"SIIT and NAT-PT"**, [http://www.ipv6-es.com/02/docs/alberto\\_garcia.pdf](http://www.ipv6-es.com/02/docs/alberto_garcia.pdf)
- [22] **"Internet Protocol, Version 6 (IPv6) Specification"**, S. Deering, R. Hinden, IETF RFC 2460, December 1998.
- [23] **"IP Message Fragmentation Process"**,  
[http://www.tcpipguide.com/free/t\\_IPMessageFragmentationProcess-2.htm](http://www.tcpipguide.com/free/t_IPMessageFragmentationProcess-2.htm)
- [24] **"What is Packet Fragmentation"**,  
<http://www.tech-faq.com/packet-fragmentation.shtml>
- [25] **"IPv6 in Free BSD"**, *Joseph Khoshy*,  
<http://people.freebsd.org/~jkoshy/images/ipv6.pdf>,
- [26] **"Whats new in Net BSD in 2006?"**, Emanuel Dreyfus, October 2006

## **APPENDIX B**

### **Source Code of TCP Checksum Calculation in KAME Project**

**Source Code of TCP Header Checksum Calculation in KAME Project [25] [26]**

```
/*      $KAME: in6_cksum.c,v 1.18 2005/07/15 15:13:57 jinmei Exp $ */

/*
 * Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
 * 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. Neither the name of the project nor the names of its contributors
 *    may be used to endorse or promote products derived from this
 *    software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE PROJECT 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 PROJECT 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.
 */

/*
 * Copyright (c) 1988, 1992, 1993
 * The 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 University of
* California, Berkeley and its contributors.
* 4. Neither the name of the University nor the names of its
contributors
* 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.
*
* @(#)in_cksum.c 8.1 (Berkeley) 6/10/93
*/

#include <sys/param.h>
#include <sys/mbuf.h>
#include <sys/system.h>
#include <netinet/in.h>
#include <netinet/ip6.h>
#include <netinet6/scope6_var.h>

#include <net/net_osdep.h>

/*
* Checksum routine for Internet Protocol family headers (Portable
Version).
*
* This routine is very heavily used in the network
* code and should be modified for each CPU to be as fast as possible.
*/

```

```

#define ADDCARRY(x)  (x > 65535 ? x -= 65535 : x)
#define REDUCE(l_util.l = sum; sum = l_util.s[0] + l_util.s[1];
ADDCARRY(sum);)

/*
 * m MUST contain a continuous IP6 header.
 * off is an offset where TCP/UDP/ICMP6 header starts.
 * len is a total length of a transport segment.
 * (e.g. TCP header + TCP payload)
 */

int
in6_cksum(m, nxt, off, len)
    struct mbuf *m;
    u_int8_t nxt;
    u_int32_t off, len;
{
    u_int16_t *w;
    int sum = 0;
    int mlen = 0;
    int byte_swapped = 0;
    struct ip6_hdr *ip6;
    struct in6_addr in6;
    union {
        u_int16_t phs[4];
        struct {
            u_int32_t   ph_len;
            u_int8_t    ph_zero[3];
            u_int8_t    ph_nxt;
        } ph __attribute__((packed));
    } uph;
    union {
        u_int8_t    c[2];
        u_int16_t    s;
    } s_util;
    union {
        u_int16_t s[2];
        u_int32_t l;
    } l_util;

    /* sanity check */
    if (m->m_pkthdr.len < off + len) {
        panic("in6_cksum: mbuf len (%d) < off+len (%d+%d)",
            m->m_pkthdr.len, off, len);
    }

    /* Skip pseudo-header if nxt == 0. */
    if (nxt == 0)
        goto skip_phdr;

    bzero(&uph, sizeof(uph));

    /*
     * First create IP6 pseudo header and calculate a summary.
     */
    ip6 = mtod(m, struct ip6_hdr *);

```

```

uph.ph.ph_len = htonl(len);
uph.ph.ph_nxt = nxt;

/*
 * IPv6 source address.
 */
in6 = ip6->ip6_src;
in6_clearscope(&in6);
w = (u_int16_t *)&in6;
sum += w[0]; sum += w[1]; sum += w[2]; sum += w[3];
sum += w[4]; sum += w[5]; sum += w[6]; sum += w[7];

/* IPv6 destination address */
in6 = ip6->ip6_dst;
in6_clearscope(&in6);
w = (u_int16_t *)&in6;
sum += w[0]; sum += w[1]; sum += w[2]; sum += w[3];
sum += w[4]; sum += w[5]; sum += w[6]; sum += w[7];

/* Payload length and upper layer identifier */
sum += uph.phs[0]; sum += uph.phs[1];
sum += uph.phs[2]; sum += uph.phs[3];

skip_phdr:
/*
 * Secondly calculate a summary of the first mbuf excluding
offset.
 */
while (m != NULL && off > 0) {
    if (m->m_len <= off)
        off -= m->m_len;
    else
        break;
    m = m->m_next;
}
w = (u_int16_t *) (mtod(m, u_char *) + off);
m_len = m->m_len - off;
if (len < m_len)
    m_len = len;
len -= m_len;
/*
 * Force to even boundary.
 */
if ((1 & (long) w) && (m_len > 0)) {
    REDUCE;
    sum <= 8;
    s_util.c[0] = *(u_char *)w;
    w = (u_int16_t *)((char *)w + 1);
    m_len--;
    byte_swapped = 1;
}
/*
 * Unroll the loop to make overhead from
 * branches &c small.
 */
while ((m_len -= 32) >= 0) {
    sum += w[0]; sum += w[1]; sum += w[2]; sum += w[3];

```

```

        sum += w[4]; sum += w[5]; sum += w[6]; sum += w[7];
        sum += w[8]; sum += w[9]; sum += w[10]; sum += w[11];
        sum += w[12]; sum += w[13]; sum += w[14]; sum += w[15];
        w += 16;
    }
    mlen += 32;
    while ((mlen -= 8) >= 0) {
        sum += w[0]; sum += w[1]; sum += w[2]; sum += w[3];
        w += 4;
    }
    mlen += 8;
    if (mlen == 0 && byte_swapped == 0)
        goto next;
    REDUCE;
    while ((mlen -= 2) >= 0) {
        sum += *w++;
    }
    if (byte_swapped) {
        REDUCE;
        sum <= 8;
        byte_swapped = 0;
        if (mlen == -1) {
            s_util.c[1] = *(char *)w;
            sum += s_util.s;
            mlen = 0;
        } else
            mlen = -1;
    } else if (mlen == -1)
        s_util.c[0] = *(char *)w;
next:
    m = m->m_next;

    /*
     * Lastly calculate a summary of the rest of mbufs.
     */

    for (; m && len; m = m->m_next) {
        if (m->m_len == 0)
            continue;
        w = mtod(m, u_int16_t *);
        if (mlen == -1) {
            /*
             * The first byte of this mbuf is the continuation
             * of a word spanning between this mbuf and the
             * last mbuf.
             *
             * s_util.c[0] is already saved when scanning
previous
             * mbuf.
             */
            s_util.c[1] = *(char *)w;
            sum += s_util.s;
            w = (u_int16_t *)((char *)w + 1);
            mlen = m->m_len - 1;
            len--;
        } else
            mlen = m->m_len;
    }

```

```

    if (len < mlen)
        mlen = len;
    len -= mlen;
    /*
     * Force to even boundary.
     */
    if ((1 & (long) w) && (mlen > 0)) {
        REDUCE;
        sum <= 8;
        s_util.c[0] = *(u_char *)w;
        w = (u_int16_t *)((char *)w + 1);
        mlen--;
        byte_swapped = 1;
    }
    /*
     * Unroll the loop to make overhead from
     * branches &c small.
     */
    while ((mlen -= 32) >= 0) {
        sum += w[0]; sum += w[1]; sum += w[2]; sum += w[3];
        sum += w[4]; sum += w[5]; sum += w[6]; sum += w[7];
        sum += w[8]; sum += w[9]; sum += w[10]; sum += w[11];
        sum += w[12]; sum += w[13]; sum += w[14]; sum +=
w[15];
        w += 16;
    }
    mlen += 32;
    while ((mlen -= 8) >= 0) {
        sum += w[0]; sum += w[1]; sum += w[2]; sum += w[3];
        w += 4;
    }
    mlen += 8;
    if (mlen == 0 && byte_swapped == 0)
        continue;
    REDUCE;
    while ((mlen -= 2) >= 0) {
        sum += *w++;
    }
    if (byte_swapped) {
        REDUCE;
        sum <= 8;
        byte_swapped = 0;
        if (mlen == -1) {
            s_util.c[1] = *(char *)w;
            sum += s_util.s;
            mlen = 0;
        } else
            mlen = -1;
    } else if (mlen == -1)
        s_util.c[0] = *(char *)w;
    }
    if (len)
        panic("in6_cksum: out of data");
    if (mlen == -1) {
        /* The last mbuf has odd # of bytes. Follow the
         * standard (the odd byte may be shifted left by 8 bits
         * or not as determined by endian-ness of the machine) */

```

```
        s_util.c[1] = 0;  
        sum += s_util.s;  
    }  
    REDUCE;  
    return (~sum & 0xffff);  
}
```

