UNIVERSITY OF TECHNOLOGY, MAURITIUS

SCHOOL OF INNOVATIVE TECHNOLOGIES AND ENGINEERING

DEPARTMENT OF INDUSTRIAL SYSTEM ENGINEERING

Development of a packet translation Algorithm for IPv4-v6 networks.

By

Loky Jagveer (140952)

This dissertation is submitted in partial fulfillment for the requirements of Bachelor's Degree in Computer Science with Network Security.

May 2017

List of Abbreviations

IP Internet Protocol

IPng IP the next generation

IPv4 IP version 4

IPv6 IP version 6

ISP Internet Service Provider

QOS Quality of Service

NIC Network Interface Card

TCP Transmission Control Protocol

ISO International Organization for Standardization

OSI Open Systems Interconnection

UDP User Datagram Protocol

HTTP Hypertext Transfer Protocol

TTL Time To Live

IETF Internet Engineering Task Force

NAT Network Address Translation

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

IGMP Internet Group Management Protocol

ICMP Internet Control Message Protocol

SNMP Simple Network Management Protocol

VPN Virtual Private Network

RFC Request for Comments

ICP Internet Content Provider

CP Content Provider

SP Service Provider

HA High Availability

UDP User Datagram Protocol

URL Uniform Resource Locater

VPN Virtual Private Network

VoIP Voice over IP

WAN Wide Area Network

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Dr. Mohamudally, Nawaz Ali for the continuous support of my Dissertation, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this Dissertation. Also I thank my friends for their supportive advice and helpful information. I would like to thank my family for supporting me spiritually throughout my life.

Abstract

Currently, the IPv4 protocol is heavily used by institutions, companies and individuals, but everyday there is a higher number of devices connected to the network such as home, appliances, mobile phones or tables. Each machine or devices needs to have its own IP address to communicated with other machines connected to Internet. This implies the need for multiple IP addresses for a single user and the current protocol begins to show some deficiencies due to IPv4 address space exhaustion.

Therefore, for several years experts have been working on an IP protocol update: the IPv6 128-bit version can address up to about 340 quadrillion system devices concurrently. With IPv6, today, every person on the planet could have millions of devices simultaneously connected to the Internet.

When changing a network from IPv4 to IPv6, Internet networks will be hybrid by using both IPv4 networks and IPv6 networks. This thesis defines the essential information about compatibility between Ipv4-Ipv6 mechanisms. Dual Stack is one of the IPv4-IPv6 compatible mechanism by running both IPv4 stack and Ipv6 stack in a single node. 6 to 4 tunneling mechanism encrypts IPv6 packets in IPv4 packets to make communications possible, from IPv6 network over IPv4 network. Dual Stack & Tunneling mechanisms were completely implemented later in this thesis work. This thesis examine transmission latency, throughput, jitter and delay from end to end, through empirical observations of both Dual Stack and tunneling mechanisms by using TCP/UDP as transport protocols in different scenarios. This thesis work contains some useful strategic point of view before trying to deploy IPv6 in a network.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Introduction

The internet protocol (IP) is the primary protocol used for relaying the network packets from source to destination thought the Internet. It was initially design in 1974 by Vint Cerf and Bob Kahn to connect systems that are in different geographical locations. The term ‘Internet’, simply means Internetwork, which is a connection between multiple networks. During the early stages of development, this protocol was used only by the military and research universities but gradually.

Internet Protocol is the backbone of the Internet. It specifies how independent networks can work together to form a global network. Each of the hosts connected to the Internet has associated IP address. A source and a destination IP address are assigned with in a packet and forwarded into the network. When packets are sent to a host which is not located within the same network as the source host, networking devices such as routers, are used to receive packets from the source host and forward it one step closer to the location of the network where the destination host resides.

The concept of forwarding packets arrived as a new idea to early research. The telephone network was a global network which did not use the concept of packet forwarding. Telephone networks are used a circuit switch implementation, consisting of a fixed circuit source and destination. All the traffic was sent over this connection. With the advent of IP, packets do not travel along established fixed circuits. The packets are addressed with a source and destination address, then forwarded through the network.

IP allows a packet to be forwarded through different networks in order to reach the destination. These individual networks may define their own rules of sending data through their routing devices with each network adhering to their respective rules. IP allows a packet to adapt to each of the individual networks as it traverses. The size of the packet may vary from network to network. The IP allows a packet to be fragmented and then reassembled at the destination.

Addressing and fragmentation are the two basic functions implemented by the IP. Network modules within the Internet use included addresses within a data packet to transmit that data to tis destination. The field option in the packet header contains information necessary to perform fragmentation and reassembly of the data packet. This is useful in the transmission of a packet through small packet networks.

1.2 Problem Statement

There is a clear sign that IPv4 will come to be an extinct, as stated by the authorities that are responsible for provision of address allocation. This has also highlighted the drawbacks that are causing the delay and lack of full acceptance of IPv6 at the present time. Some of these drawbacks are attributed to technical and political issues guiding the distribution of IPv4 addresses.

1.3 Objectives

The objectives of this project is to implement the transition mechanisms are one of the best solutions to make Ipv6 and Ipv4 Networks run in the same infrastructure. Ipv4 and Ipv6 several transition mechanism have been developed for according to different organization needs. Each mechanism have their own advantages and disadvantages in different infrastructure. All of these mechanism can be applied to different situations.

Transition Mechanism are such as:

- Dual Stack
- Translations
- Tunneling

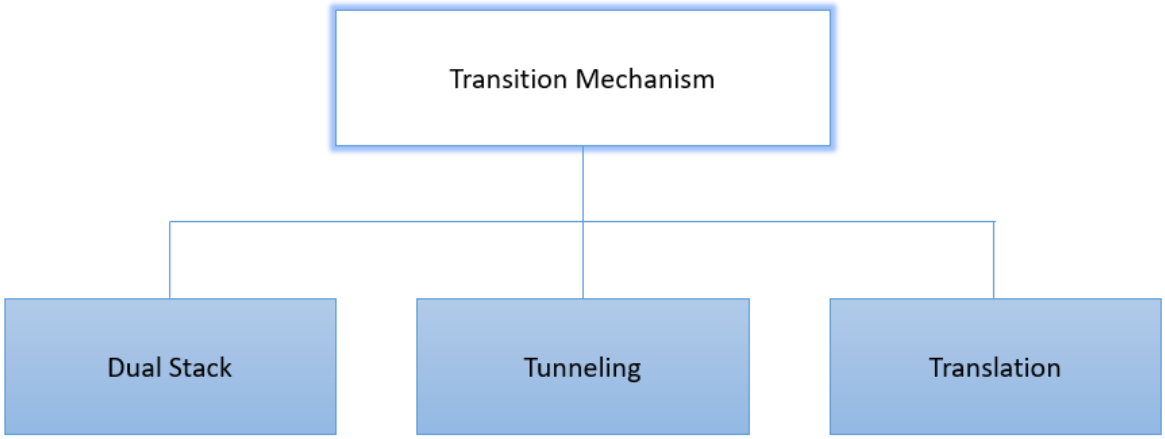


Figure 1: Transition Mechanism

1.4 Aim of the project

The rapid growth of the Internet needs better IP address solutions. This headed to deployment of IPv6 in the Internet. As a first step towards deploy IPv6 across the world, is Co-existence of IPv4 and IPv6. The complete transition of the Internet will be a huge task. The transition is expected to take several years. The best choice will be to use co-existence mechanisms such as DualStack, Tunneling and Translation. This thesis motivation is to make survey on different co-existence mechanisms to find better and cost-effective IPv6 deployment.

1.5 Problem analyze and approaching

Problems analyze and approaching is focused on both IPv4 and IPv6 implementation and routing functionalities.

Below are stated some of the general enquires & problems.

- Could a node support Ipv6
- Could a node support both IPv4 and Ipv6
- Is co-existence a good mechanism
- How could we achieve routing between nodes with both IPv4 and Ipv6?
- What are the smooth transition mechanisms?

1.6 Methodology

The goal is to gain the theoretical knowledge and to do the research and help the individuals and organizations in moving from IPv4 to IPv6. The aim is to identify the challenges and problems and provide the easier solutions to help individuals not to depend on IPv4 but moving to IPv6. As this project require the technical knowledge to provide the possible solutions from moving IPv4 to IPv6. Using the available resources which will more focused on IPv4 and IPv6, will clarify the problems, benefits and challenges that we are facing nowadays. In the simulation part, experimental work using a Graphical Network Simulator (GNS) network emulator will be carried out, this tool provides friendly user interface rather than other existing emulators. Later tests and configuration will be done and based on that results will be provided. This thesis will be based extensively on literature study and lab tests.

Chapter 2: Internet Protocol

2.1 Overview

In this literature review, prior research and studies had been explored regarding the domain. This chapter consists of a brief description of the two main INTERNET protocols, which are Ipv4 and Ipv6. This is important as it provides a logical understanding of why these two protocols are included in this study. The problems and opportunities offered by these protocols are also presented. Discussion and explanation of why the performance of a network is important to be monitored at all times is also given. In this part a discussion of the use of the two main transmission protocols (TCP and UDP) their features and the differences between these will two will also be presented. This study has been carried out using the standard Ethernet packet sizes and as well as the header. This chapter will also discuss the features, differences and importance of implementing Ipv4 of Ipv6.

2.2 Internet Protocol

On the Internet, every computer has a unique address. This address is called IP, which stands for Internet Protocol. It defines the format of packets and provides an addressing system, which has two functions: identifying hosts and providing a logical location service. The dominant version of Internet Protocol is IPv4 and the next generation is Internet Protocol Version 6 (IPv6). Now, one may wonder where the IPv5 is. After the IPv4, IPv5 was introduced to overcome the obstacles or problems of IPv4. Mainly, it was designed to provide Quality of Service (Qos) for streaming services. It was envisioned to be the connection-oriented complement to IPv4 but was never introduced for public use. The next generation of Internet Protocol is IPv6, which is also called IPng or IP next generation. The features of IPv4 and IPv6 will be discussed later. Before we go further, it is important to discuss what a network is. A network is the group of two or more computers connected with each other to exchange data using cable or wireless. When connected to the network, a computer is on-line and when disconnected, it is off-line. In a network we can exchange different resources like data, application, hardware and other information. For this exchange, we need the following hardware: cables, switch, routers, network interface card (NIC) etc. Other than hardware,

We need a server computer and a client computer. A server computers shares scanners, printers and other network services that have Internet access. Client computers are all the other computers, which are in same network with the Server computer. They can access all the resources provided by the server. To connect to the network, the computer needs a NIC. The NIC in the computers physically connects to the network with the help of an Ethernet cable. The Ethernet cable does not connect to network but it only helps to connect to the switch while the switch connects to the network.

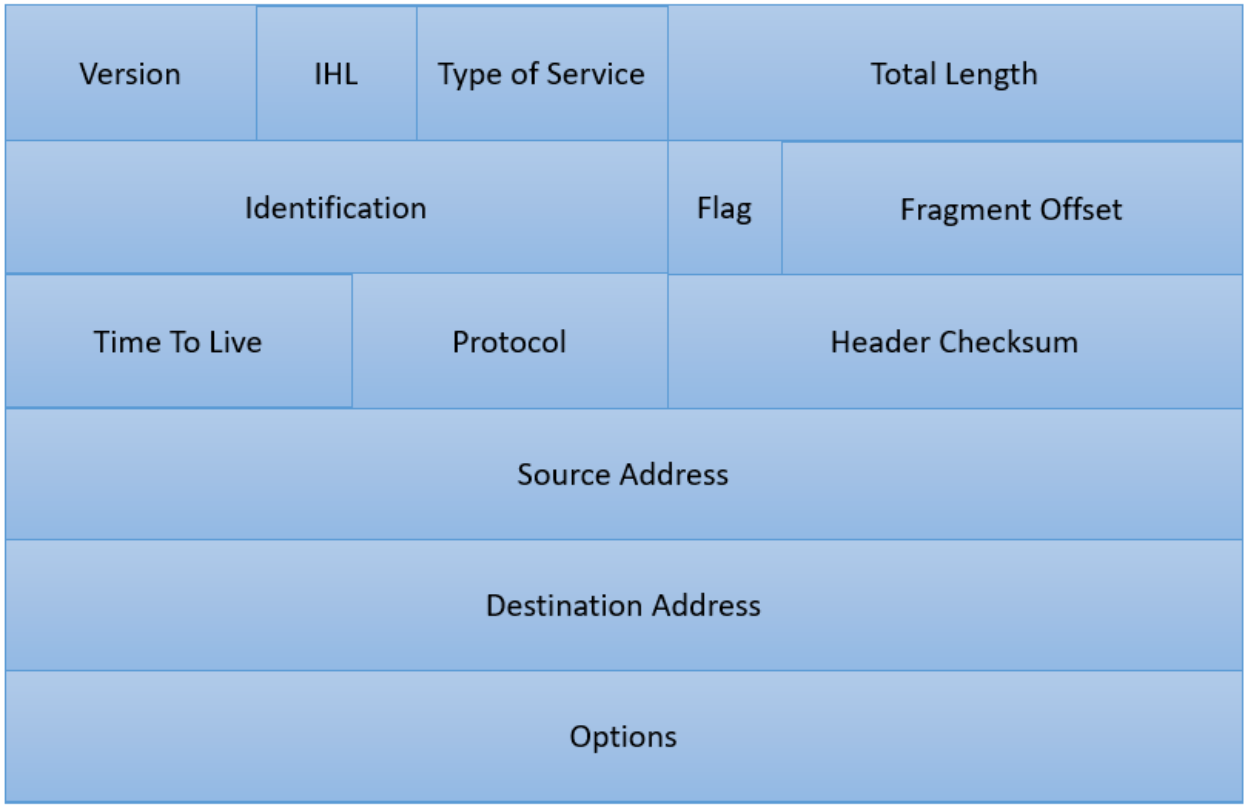


Figure 2: IP header format

An IP header consists of a 4-bit protocol version number, length of header (IHL) and a total length of 16-bit, along with some control fields and source and destination IP addresses (32-bits each), as shown in Figure 3. As stated in the previous sentence, IP addresses are 32-bits long and are classified into 4 octets, separated by periods [5] – for instance, *192.168.100.1.*

2.3 OSI Model

OSI Model the Open Systems Interconnection (OSI) model defines how the communication between two users in the network happens. The OSI model is composed of seven layers, each one having their own functions. Now, every network is built on this OSI model. The main purpose of building this model is to understand the networks and how it works. Having different layers it makes easy to troubleshoot if we have any problems in the future. The different layers of the OSI model are independent of each other but provide service to the other layers that are connected. Again, these seven layers are divided into two parts: the four upper layers and lower three layers. The four upper layers are used when a message passes from or to a user. The three lower layers are used when a message passes through the host computer.

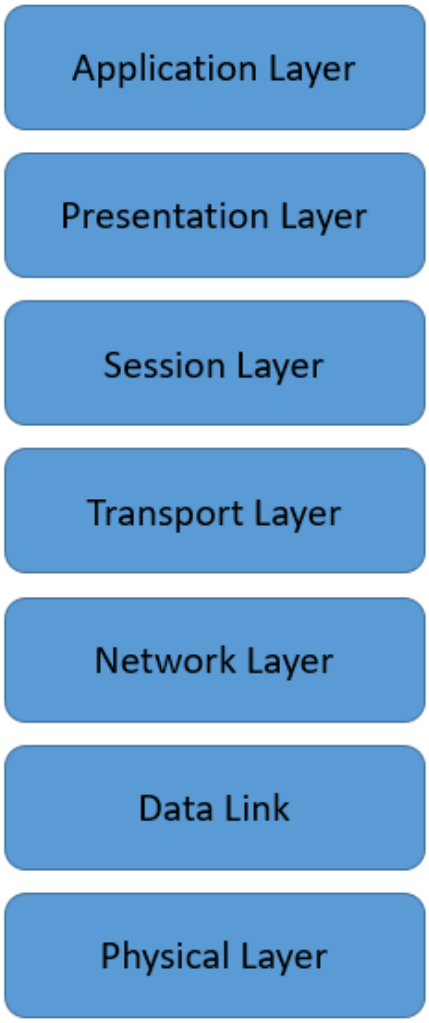


Figure 3: OSI Model

There are seven layers in OSI model, which is shown in the OSI Models.

- **Physical layer**

  It defines the physical equipment used for transferring the data across the network I.e wires, computers, network-cards, cables etc.

- **Data link layer**

  This layer is used to control the signal that enter and leae the network cable. It deals with framing and encapsulation

- **Network Layer**

  It is concerned with the process of packet forwarding including routing through intermediate routers. It handles the addressing and routing of data based on logical addressing.

- **Transport layer**

  This layer is responsible for transferring the data from one point to point another without errors. Examples of the Transport layer are Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Sequenced Packet Exchange.

Session layer

It manages, establishes, and terminates the connection between applications. It also deals with the session and connection coordination.

- **Presentation layer**

  This layer converts data so that systems, which use different data formats, can exchange information. It is also called the syntax layer.

- **Application layer**

  It represents the services that directly support applications such as software for file transfers, database access, emails and network games. It contains web-browser, FTP clients, emails clients but does not include computer application software. Examples are WWW browsers, Telnet, HTTP, FTP, etc.

2.4 TCP/IP

TCP/IP is the network standard, which defines the INTERNET. The INTERNET Protocol (IP) standard explains how packets of information are exchanged over a set of networks. It has a packet addressing method that lets any computer forward a packet over the network to other computer, which is closer to the packet's recipient. TCP checks the packets that are sent through the network and requests for transmission if errors are found. TCP/IP was designed to solve the problems before the OSI model was introduced as shown in Figure 2.

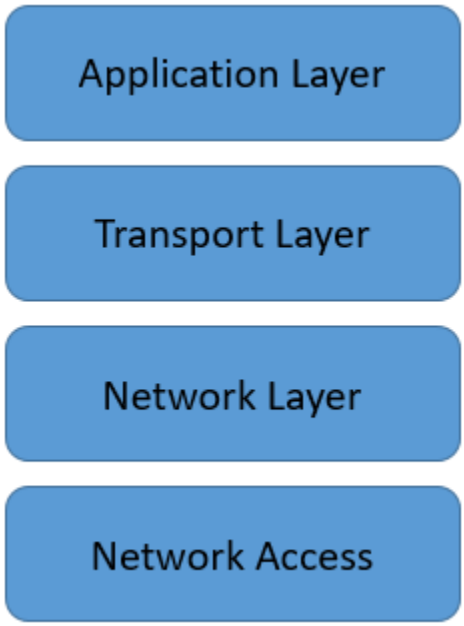


Figure 4: TCP/IP

Application Layer

It is the top most layer of TCP/IP model. The application layer is where the network and its application-layer protocol reside. Using the application protocol layers, packets of information are exchanged between hosts and remote users can communicate. The application layer includes many protocols like DNS (Domain Naming System), HTTP protocol (which provides web document request and transfer), SMTP (which provides transfer of email message) and FTP (which provides transfer of files between two end systems).

Transport Layer

It is the third layer of TCP/IP model, which resides in between the Transport Layer and the Application Layer. The Internet's transport layer transports application layer message between application endpoints. The Transport Layer includes protocols like TCP (which provides connection oriented services to its application) and UDP (which provides connectionless service to its application).

Internet layer

The Internet layer is the second layer of the TCP/IP model. The Internet layer is between the Network Access layer and the Transport Layer. The Internet layer is responsible for moving data packets known as IP datagrams from one host to another. IP datagrams contain source and destination addresses, which are used to forward the datagrams between hosts and across network. The Internet layer includes protocols like IP, ICMP, ARP, RARP and IGMP.

Network

The Network Access Layer is the first or lower layer of the TCP/IP model. The Network Access Layer explains how data is sent physically through the network including individual bits. The Network Access Layer includes protocols like FDDI, X.25, Frame Relay, Ethernet, and Token Ring etc.

2.4.1 TCP (Transmission Control Protocol)

Transmission Control Protocol is a *transport layer* protocol used by most web applications, e.g. HTTP and FTP. TCP is a connection-oriented protocol, designed to provide reliability in the communication between two systems. The mechanism involved in the reliability of this protocol is the use of checksums on both headers and data. Whenever data is received, an *acknowledgement* (ACK) is sent back to the server within a specific timeframe. If the acknowledgement is not received, the data is then resent. TCP makes use of IP (Internet Protocol) to send data in blocks, called segments. Each segment contains 20 bytes of header information with IP header [5]. The TCP header starts with a 16-bit source and destination port fields which specify the *application layers* that have sent and are to receive data in a particular communication as shown in figure 2 below.

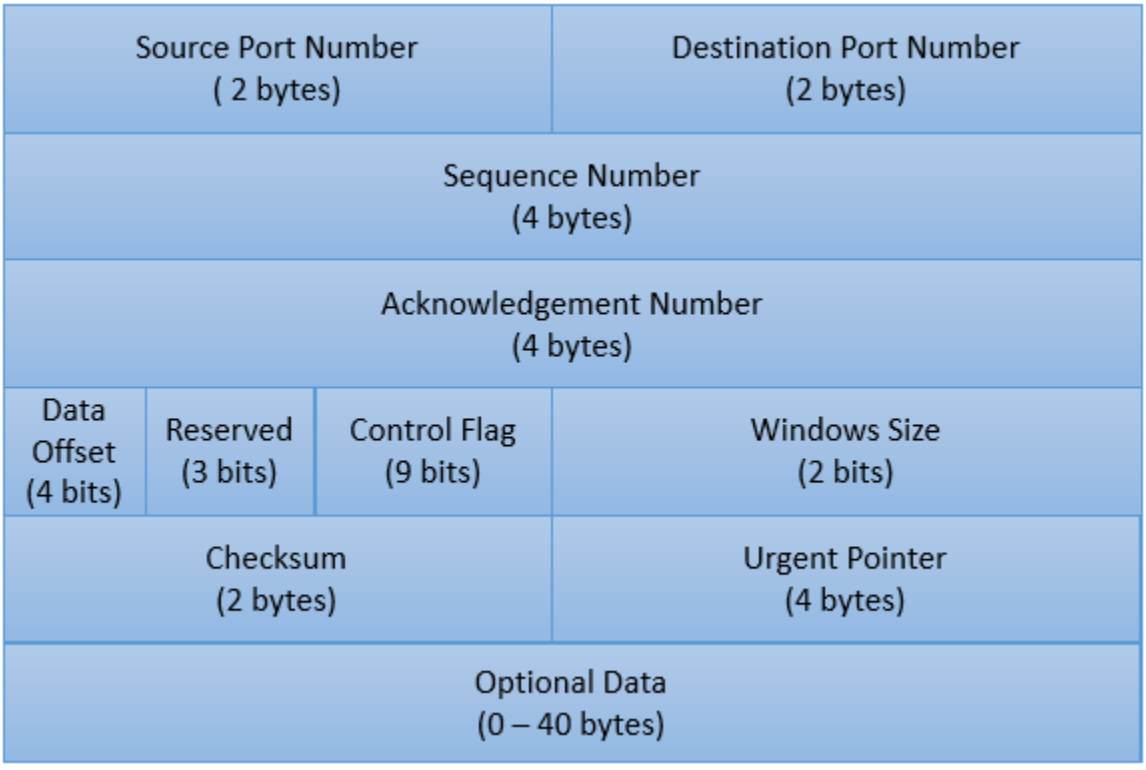


Figure 5: TCP header format

TCP Packet Field Descriptions:

The following descriptions summarize the TCP packet fields illustrated in Figure. 6

Source Port and Destination Port: Identifies points at which upper-layer source and destination processes receive TCP services.

Sequence Number: usually specifies the number assigned to the first byte of data in the current message. In the connection-establishment phase, this field also can be used to identify an initial sequence number to be used in an upcoming transmission.

Acknowledgment Number: contains the sequence number of the next byte of data the sender of the packet expects to receive.

Data Offset: indicates the number of 32-bit words in the TCP header.

Reserved: Remains reserved for future use.

Control Flags: carries a variety of control information, including the SYN and ACK bits used for connection establishment, and the FIN bit used for connection termination.

Window size: Specifies the size of the sender's receive window (that is, the buffer space available for incoming data).

Checksum: indicates whether the header was damaged in transit.

Urgent Pointer: Points to the first urgent data byte in the packet.

Options and Data: Specifies various TCP options and Contains upper-layer information.

TCP Connection Establishment:

To use reliable transport services, TCP hosts must establish a connection-oriented session with one another. Connection establishment is performed by using a "three-way handshake" mechanism.

A three-way handshake synchronizes both ends of a connection by allowing both sides to agree upon initial sequence numbers. This mechanism also guarantees that both sides are ready to transmit data and know that the other side is ready to transmit as well. This is necessary so that packets are not transmitted or retransmitted during session establishment or after session termination.

Each host randomly chooses a sequence number used to track bytes within the stream it is sending and receiving. Then, the three-way handshake proceeds in the following manner:

The first host (Host A) initiates a connection by sending a packet with the initial sequence number (X) and SYN bit set to indicate a connection request. The second host (Host B) receives the SYN, records the sequence number X, and replies by acknowledging the SYN (with an ACK = X + 1). Host B includes its own initial sequence number (SEQ = Y). An ACK = 20 means the host has received bytes 0 through 19 and expects byte 20 next. This technique is called *forward acknowledgment*. Host A then acknowledges all bytes Host B sent with a forward acknowledgment indicating the next byte Host A expects to receive (ACK = Y + 1). Data transfer then can begin.

Positive Acknowledgment and Retransmission (PAR):

A simple transport protocol might implement a reliability-and-flow-control technique where the source sends one packet, starts a timer, and waits for an acknowledgment before sending a new packet. If the acknowledgment is not received before the timer expires, the source retransmits the packet. Such a technique is called *positive acknowledgment and retransmission* (PAR).

By assigning each packet a sequence number, PAR enables hosts to track lost or duplicate packets caused by network delays that result in premature retransmission. The sequence numbers are sent back in the acknowledgments so that the acknowledgments can be tracked.

PAR is an inefficient use of bandwidth, however, because a host must wait for an acknowledgment before sending a new packet, and only one packet can be sent at a time.

TCP Sliding Window:

A *TCP* sliding window provides more efficient use of network bandwidth than PAR because it enables hosts to send multiple bytes or packets before waiting for an acknowledgment.

In TCP, the receiver specifies the current window size in every packet. Because TCP provides a byte-stream connection, window sizes are expressed in bytes. This means that a window is the number of data bytes that the sender is allowed to send before waiting for an acknowledgment. Initial window sizes are indicated at connection setup, but might vary throughout the data transfer to provide flow control. A window size of zero, for instance, means "Send no data."

In a TCP sliding-window operation, for example, the sender might have a sequence of bytes to send (numbered 1 to 10) to a receiver who has a window size of five. The sender then would place a window around the first five bytes and transmit them together. It would then wait for an acknowledgment.

The receiver would respond with an ACK = 6, indicating that it has received bytes 1 to 5 and is expecting byte 6 next. In the same packet, the receiver would indicate that its window size is 5. The sender then would move the sliding window five bytes to the right and transmit bytes 6 to 10. The receiver would respond with an ACK = 11, indicating that it is expecting sequenced byte 11 next. In this packet, the receiver might indicate that its window size is 0 (because, for example, its internal buffers are full). At this point, the sender cannot send any more bytes until the receiver sends another packet with a window size greater than 0.

2.4.2 UDP (User Datagram Protocol)

The User Datagram Protocol (UDP) is a connectionless transport-layer protocol (Layer 4) that belongs to the Internet protocol family. UDP is basically an interface between IP and upper-layer processes. UDP protocol ports distinguish multiple applications running on a single device from one another.

Unlike the TCP, UDP adds no reliability, flow-control, or error-recovery functions to IP. Because of UDP's simplicity, UDP headers contain fewer bytes and consume less network overhead than TCP.

UDP is useful in situations where the reliability mechanisms of TCP are not necessary, such as in cases where a higher-layer protocol might provide error and flow control.

UDP is the transport protocol for several well-known application-layer protocols, including Network File System (NFS), Simple Network Management Protocol (SNMP), Domain Name System (DNS), and Trivial File Transfer Protocol (TFTP).

The UDP packet format contains four fields, as shown in Figure 6. These include source and destination ports, length, and checksum fields.

UDP packets consist of four main 16-bit fields – source and destination port numbers, checksum fields for error checking and a UDP length field. The usage of port numbers in UDP coincides the mechanism that TCP makes use of, except that UDP makes use of software ports only.

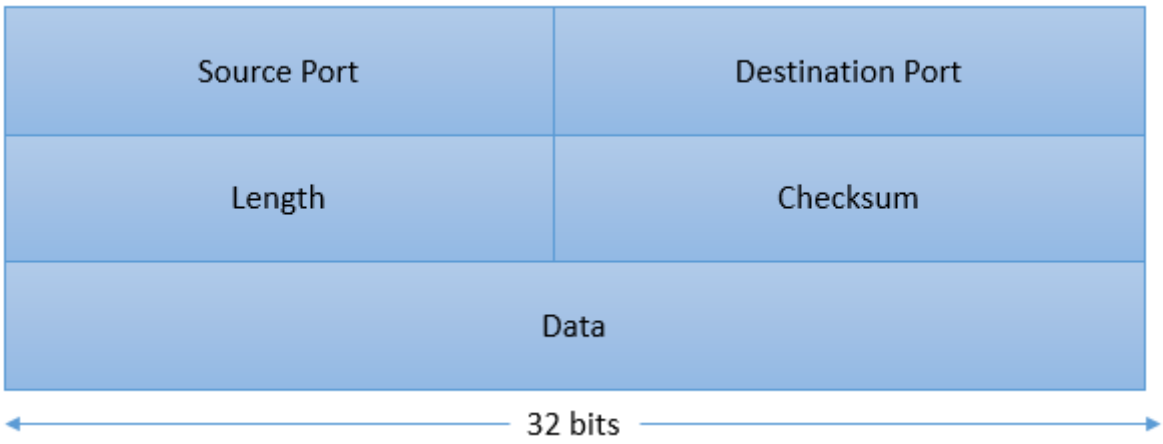


Figure 5: UDP header format

Source and destination ports contain the 16-bit UDP protocol port numbers used to de-multiplex datagrams for receiving application-layer processes. A length field specifies the length of the UDP header and data. Checksum provides an (optional) integrity check on the UDP header and data.

2.4.3 Comparison of OSI Model and TCP/IP

Listed below are some of the major differences between the OSI Model and the TCP/IP model with a diagrammatic comparison as shown in figure 3 below.

OSI (Open System InterConnection)	TCP/IP (Transmission Control Protocol/Internet Protocol
It has seven layers	It has four layers
The transport layer guarantees delivery of packets	The transport layer does not guarantee the delivery of packets
OSI defines functions of all the layer and provides layer functioning	TCP/IP is not flexible with other layers as it is more based on protocols
It has separate session and presentation layer	It does not have separate session and presentation layer
The network layer provides both connection-oriented and connectionless service	The network layer provides connectionless service
The OSI model is generic, protocol-independent standard	TCP/IP protocols are considered to be standards
Protocols do not fit well into this model	Protocols fit well in this model

Figure 6: OSI vs TCP/IP

2.5 IPv4

It was soon evident that implementation of Ipv4 was not possible with the rapid growth of the INTERNET. Quantity of IP addresses in Ipv4 was not sufficient to keep up with the proliferation of devices on the INTERNET. A 32-bit address length gives us 4.2 billion IP addresses. When Ipv4 written, it appeared to be a sufficient amount of IP addresses. However, as time progressed, the Internet grew with the advent of new networking devices such as phones, television and gaming consoles, which were IP-capable. This lead to the exhaustion of IP address spaces.

Temporary solutions were found to overcome the exhaustion of Ipv4 address spaces. The first solution was Classless InterDomain Routing (CITR) which is the method for allocating IP addresses the growth of routing tables within the Internet in order to restrict the rapid exhaustion of Ipv4 addresses. The second solution was a technique termed Network Address Translation

(NAT) in which one IP address could be translated to multiple hosts within the NAT network. The Third solution is termed Dynamic Host Configuration Protocols (DHCP) which is used on IP networks as the automatic configuration protocol. However, in CIDR, the need for larger routing tables in routers became evident, which resulted in routing difficulties. The NAT solution breaks the Principle of the Internet and is not supported by some applications. These three technologies were designed as solutions but made the networks much slower and complex. In addition, these three technologies did not overcome the problem of Ipv4 address exhaustion, but only delayed it.

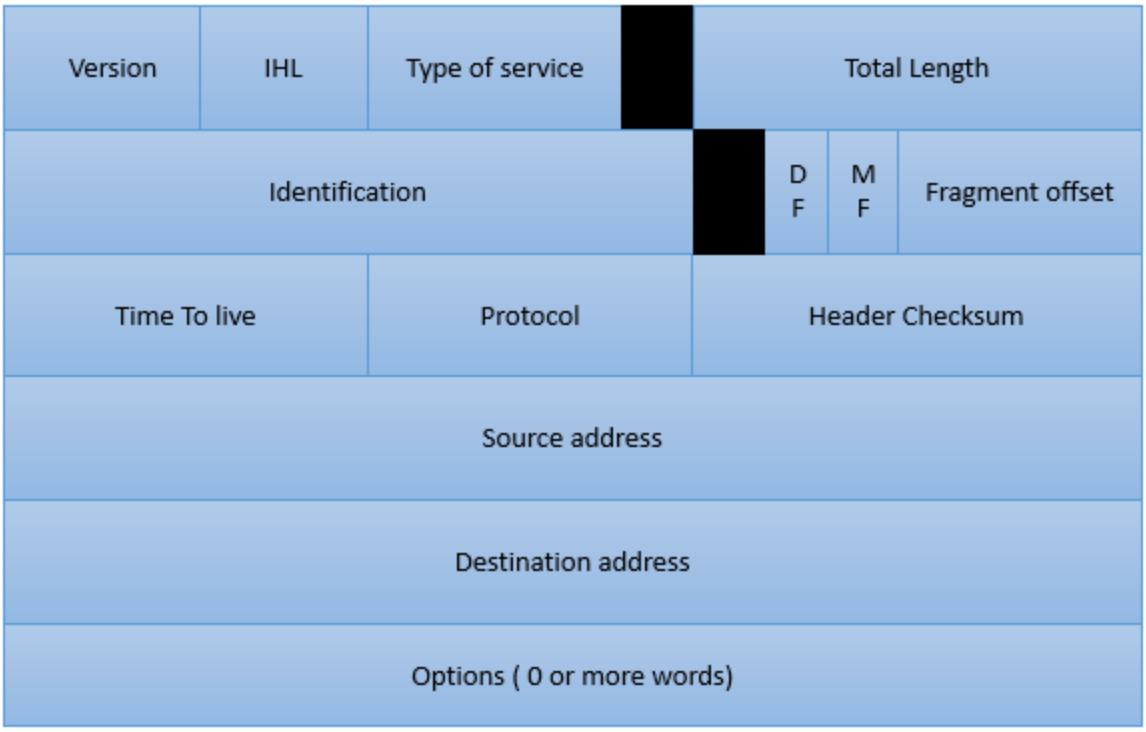


Figure 7:Ipv4 Diagram

- The **version** field is 4 bits long. This field specifies the format of the internet header. It it set in binary format. The value would be 4 in case of Ipv4 and 6 in case of Ipv6.
- **Internet Header Length** (IHL) specifies the length of Internet Header in32 bit words and is pointing to the beginning of the word. Minimum value for a correct header is 5.
- The **Type of Service** (TOS) field specifies the type of service desired and has an 8-bit length. TOS specifies the IP priority. Several networks have service precedence in which high precedence traffic is considered more important. Sometime during high load, routers

accept traffic above defined precedence. Delay, throughput, and reliability are others parameters available to define the precedence.

- The **Total length** is a 16-bit field. Total length is the length of the packet including INTERNET header and data. It is measured in octets. Packets up to 65,535 octets in length are allowed in this field.
- The **identification** field has a 16-bit length. This aids in assembling the fragments of packets at the destination.
- The **flags** field has a 32-bit length. The first bit is reserved and must be set to zero.
- The **fragment offset** field has a 13-bit length. It is measured in units of 8 octets. The first fragment has offset zero. It specifies where this fragment fits in the packet.
- **Time to Live field** has a 8-bit length. It indicates the maximum time a packet is allowed to stay alie in the INTERNET. Whenever a packet arrives at the router this field value is reduced by 1. When the value become zero packet is discarded.
- The **protocol** field has an 8-bit length. This field specifies the protocol used in the data portion of the Internet packet and the values are maintained by the Internet Assigned Numbers Authority.
- **Header Checksum** is 16-bit field. Checksum is performed on the header only, since the data field remains the same while traveling towards the destination.
- The **Source Address** is a 32-bit field. It specifies the source address of the packet. This value does not change during fragmentation.
- The **Destination Address** is a 32-bit field. It specifies the final destination address of the packet. This value does not change during transmission along the route.
- The **Option field** is variable in length. There may be zero or more options. This field is not mandatory for every IP packet.
- The **padding** field like option is of variable length. It ensures the INTERNET header ends on a 32-bit boundary

.

2.6 IP addressing

IP address is an address which is unique and is used to identify the device in the network. It is composed of 32 binary bits. These 32 binary bits are broken into four octets, i.e., 8 bits each. Each octet is converted to a decimal and is separated by a dot. This is why we have IP addresses in dotted format, for example 82.50.69.83. The IP address Huber has two parts: a network number and a host number. The network number determines which network the host computer is located. The host number determines the exact host computer in that network. Diagram is shown below for IP addressing.

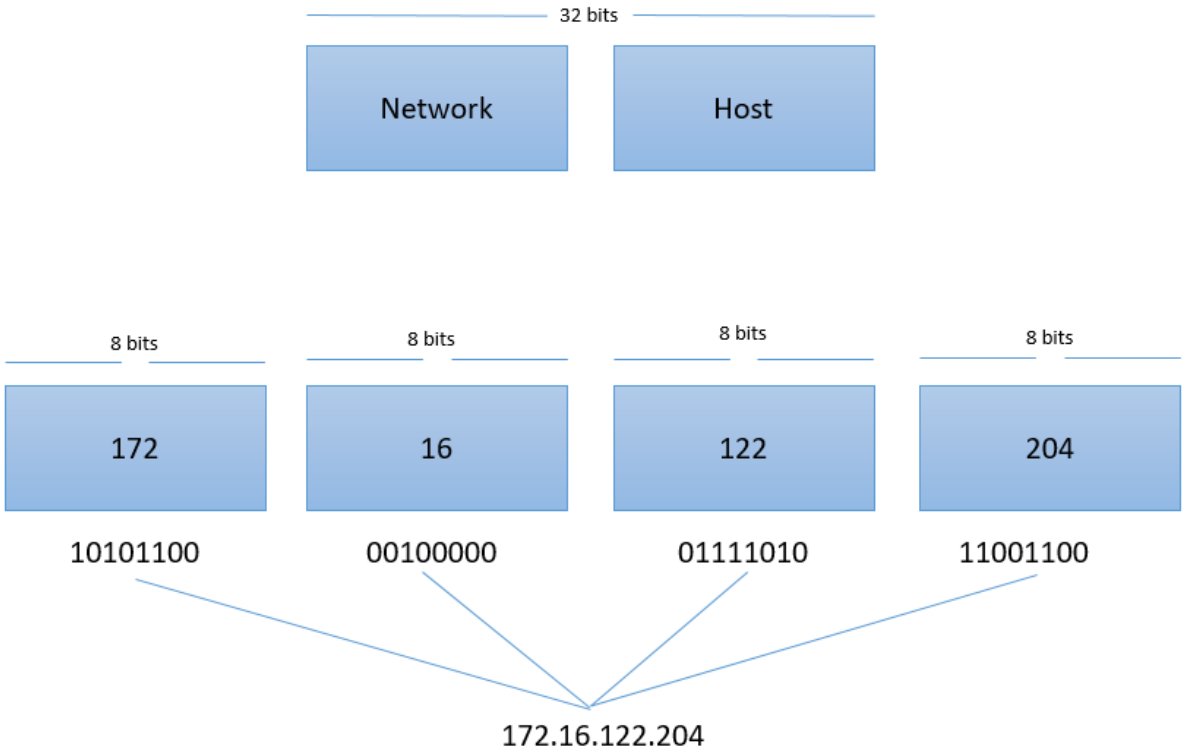


Figure 8: IP addressing

2.6.1 Ipv4 Address format

The Structural format of IPv4 in figure 6 where different octets are segmented to describe the binary content of each octet and its composited bit capacity. Moreover, the decimal and binary composition helps to observe each octet.

It is not easy to remember an address in binary form like 10101100.10010110.1101100.1100110 for 172.16.122.204. So to make it easy, a unique name is assigned by the Domain Name System (DNC).

2.6.2 IP address Classes

There are several classes of IP address as the requirement of host per network. IPv4 is divided into 5 classes (A,B,C,D and E) and they are determined by the first octet of the IP address.

2.6.3 Describing 5 classes of IPv4 and their area of purposes

Address Class	Bit Pattern of first Byte	First Byte Decimal Range	Host Assignment Range in Dotted Decimal
A	0XXXXXX	1 - 127	1.0.0.1 to 126.255.255.254
B	10XXXXXX	128 - 191	128.0.0.1 to 191.255.255.254
C	110XXXXX	192 - 223	192.0.0.1 to 223.225.255.254
D	1110XXXX	224 - 239	224.0.0.1 to 239.255.255.254
E	11110XXX	240 - 255	240.0.0.1 to 255.255.255.255

Figure 9: IP address classes

The different classes of IP addresses of version 4 are shown based on format and range of IP addresses. It also adds the subnet masks applied to each class along with application of the respective classes. Basically, IPv4 it has been categorized in four classes which are detailed in above figure [10].

2.6.3 IP subnet Addressing

IP networks can be divided into smaller networks called sub-networks (or subnets). Sub-netting provides the network administrator with several benefits, including extra flexibility, more efficient use of network addresses, and the capability to contain broadcast traffic (a broadcast will not cross a router).

Subnets are under local administration. As such, the outside world sees an organization as a single network and has no detailed knowledge of the organization's internal structure.

A given network address can be broken up into many sub networks. For example, 172.16.1.0, 172.16.2.0, 172.16.3.0, and 172.16.4.0 are all subnets within network 171.16.0.0. (All 0s in the host portion of an address specifies the entire network.)

2.6.4 Address Resolution Protocol (ARP) Overview:

For two machines on a given network to communicate, they must know the other machine's physical (or MAC) addresses. By broadcasting Address Resolution Protocols (ARPs), a host can dynamically discover the MAC-layer address corresponding to a particular IP network-layer address.

After receiving a MAC-layer address, IP devices create an ARP cache to store the recently acquired IP-to-MAC address mapping, thus avoiding having to broadcast ARPS when they want to re-contact a device. If the device does not respond within a specified time frame, the cache entry is flushed.

2.6.5 Internet Routing:

Internet routing devices traditionally have been called gateways. In today's terminology, however, the term gateway refers specifically to a device that performs application-layer protocol translation between devices. Interior gateways refer to devices that perform these protocol functions between machines or networks under the same administrative control or authority, such as a corporation's internal network. These are known as autonomous systems. Exterior gateways perform protocol functions between independent networks.

Routers within the Internet are organized hierarchically. Routers used for information exchange within autonomous systems are called interior routers, which use a variety of Interior Gateway Protocols (IGPs) to accomplish this purpose. The Routing Information Protocol (RIP) is an example of an IGP.

2.6.6 IP routing

IP routing protocols are dynamic. Dynamic routing calls for routes to be calculated automatically at regular intervals by software in routing devices. This contrasts with static routing, where routers are established by the network administrator and do not change until the network administrator changes them.

An IP routing table, which consists of destination address/next hop pairs, is used to enable dynamic routing. An entry in this table, for example, would be interpreted as follows: to get to network 172.31.0.0, send the packet out Ethernet interface 0 (E0).

IP routing specifies that IP datagrams travel through internetworks one hop at a time. The entire route is not known at the onset of the journey, however. Instead, at each stop, the next destination is calculated by matching the destination address within the datagram with an entry in the current node's routing table.

Each node's involvement in the routing process is limited to forwarding packets based on internal information. The nodes do not monitor whether the packets get to their final destination, nor does IP provide for error reporting back to the source when routing anomalies occur. This task is left to another Internet protocol, the Internet Control-Message Protocol (ICMP), which is discussed in the following section

Routers that move information between autonomous systems are called exterior routers. These routers use an exterior gateway protocol to exchange information between autonomous systems. The Border Gateway Protocol (BGP) is an example of an exterior gateway protocol.

Internet Control Message Protocol (ICMP):

The *Internet Control Message Protocol (ICMP)* is a network-layer Internet protocol that provides message packets to report errors and other information regarding IP packet processing back to the source. ICMP is documented in RFC 792.

ICMP Messages:

ICMPs generate several kinds of useful messages, including Destination Unreachable, Echo Request and Reply, Redirect, Time Exceeded, and Router Advertisement and Router Solicitation. If an ICMP message cannot be delivered, no second one is generated. This is to avoid an endless flood of ICMP messages.

When an ICMP destination-unreachable message is sent by a router, it means that the router is unable to send the package to its final destination. The router then discards the original packet. Two reasons exist for why a destination might be unreachable. Most commonly, the source host has specified a nonexistent address. Less frequently, the router does not have a route to the destination.

Destination-unreachable messages include four basic types: network unreachable, host unreachable, protocol unreachable and port unreachable. *Network-unreachable messages* usually mean that a failure has occurred in the routing or addressing of a packet. Host-unreachable messages usually indicate delivery failure, such as a wrong subnet mask. Protocol-unreachable messages generally mean that the destination does not support the upper-layer protocol specified in the packet. Port-unreachable messages imply that the TCP socket or port is not available.

An ICMP echo-request message, which is generated by the ping command, is sent by any host to test node reachability across an inter network. The ICMP echo-reply message indicates that the node can be successfully reached.

An ICMP Redirect message is sent by the router to the source host to stimulate more efficient routing. The router still forwards the original packet to the destination. ICMP redirects allow host routing tables to remain small because it is necessary to know the address of only one router, even if that router does not provide the best path. Even after receiving an ICMP Redirect message, some devices might continue using the less-efficient route.

The router sends an ICMP Time-exceeded message if an IP packet's Time-to-Live field (expressed in hops or seconds) reaches zero. The Time-to-Live field prevents packets from continuously

circulating the internetwork if the internetwork contains a routing loop. The router then discards the original packet.

2.6.7 Problem with IPv4

The Internet has lost a lot of the functionality it had in the early days because of the address conservation;

- Loss of transparency, due to the use of mechanisms such as NAT (Network Address Translator).
- Loss of robustness because of the implemented topology that has little room for redundancy.
- Loss of unique addresses.
- Loss of stable addresses, i.e. the address of a node changes each time it is connected to the Internet.
- Loss of connection-less services.
- Loss of "always-on" services.
- Loss of end-to-end communication model.
- Loss of application- independence. An example is that many systems are developed with functionality to avoid problems created by NAT.

2.6.8 Address space

The address space in IPv4 is expected to be exhausted within the next two to three years [12]. The exhaustion is a result of a growing number of hosts and networks connected to the Internet, and also because of an inefficient assignment of IPv4 addresses. This inefficiency arises because of the structure of the IP address space, divided in class A, B and C addresses. This structure forces network address space to be handed out in fixed size chunks of three very different sizes. This leads to a bad exploitation of the address space, especially of the class B addresses. Any network with more than 255 hosts would want one class B network prefix instead of several class C network prefixes. A class B network consisting of for only 256 hosts represents an efficiency of 255/65535

= 0, 39% [3]. To solve this, CIDR (Classless Inter-Domain Routing) has been developed. In addition to saving the address space, CIDR also slows the growth of backbone routing tables.

NAT is a method for mapping multiple private addresses to a single public address. There is a lot of skepticism towards NAT as it may be appropriate to some businesses that do not need full connectivity to the outside world, but for others, who require constant and robust contact with the Internet, NAT will not fulfil the requirements. It creates a bottleneck between the business and the Internet; it does not support end-to-end security and breaks the peer-to-peer model [13]. Another problem is when applications embed IP-addresses in the packet payload, above the network layer; these can be applications like FTP programs and mobile IP. Most likely NAT will fail in translating some embedded addresses and lead to application failure [14].

2.6.9 Routing Tables

Routing tables are large and complex. As the Internet grows, so do the routing tables. In order to achieve efficient routing the address hierarchy must be well organized. The system with Masters Thesis 2003 IPv6 – Prospects and problems Mette Olsen & Siw Ånonsen - 17 - class A, B and C addresses of such different sizes together with the rationing of IPv4 addresses, Internet addressing and routing is complex. The use of CIDR is supposed to make routing more efficient, but it does not guarantee an efficient and scalable hierarchy [14].

2.6.10 Configuration

In IPv4 one must either do a manual configuration or use a stateful address configuration such as DHCP. 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. The configuration cause more administrative problems as the Internet and other markets that require an IPaddress grow.

2.6.11 Security

Packets sent at IP-level needs encryption to protect the private data from being viewed or modified.

2.6.12 Quality of Service

In IPv4 this depends on the TOS field in the header. The field is limited and has had a number of definitions during the years [1].

2.7 IPv6

Finally, the solution for the IP address exhaustion was resolved. The Internet Engineering Task Force (IETF) came up with a resolution called IPng. The aIPv6 address size is 128 bits compared to the 32 bit address in Ipv4. The 128 -bit size gives approximately 1500 addresses per square foot of the earth's surface. Internet registries are making the block of Ipv6 available and are present in most of the recent operating systems. The INTERNET community is taking time to adapt to Ipv6. The main reason would be that it is difficult for Ipv4 and Ipv6 to coexist. Whenever an Ipv6 host wants to communicate with an Ipv4 host, it has to use tunneling mechanisms where an Ipv6 packet is encapsulated within an Ipv4 packet. It may be predicted that when Ipv4 addresses are exhausted the Internet community will be forced to adopt Ipv6 conversion at a faster rate. Datagram is shown below for IPv6.

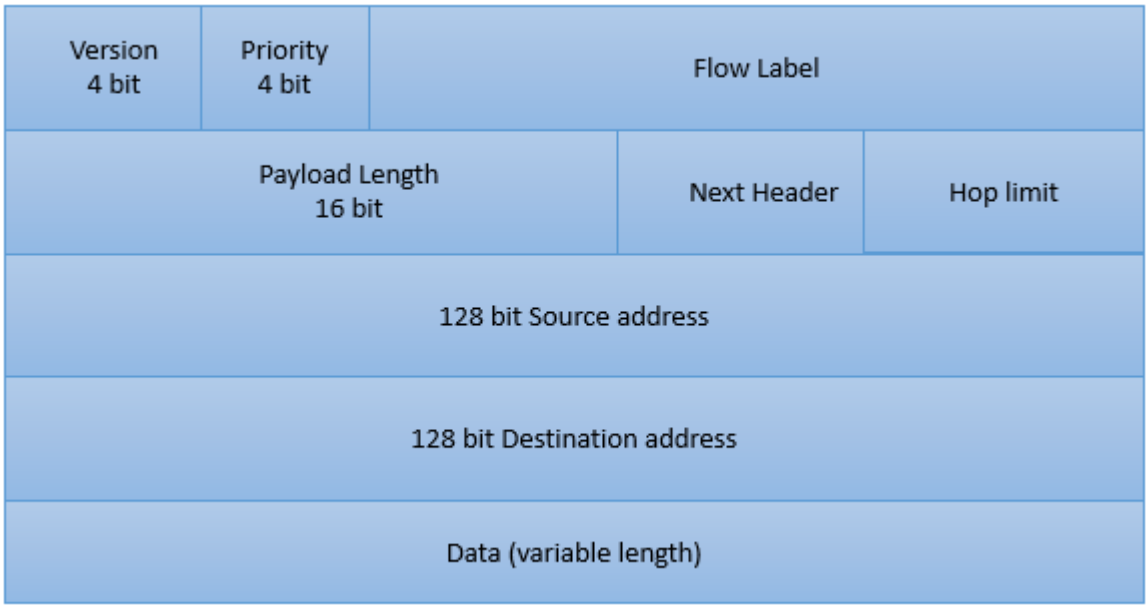


Figure 10: IPv6 packet

2.7.1 Description of Ipv6 packet

- The Version has a 4-bit length. This field specifies the format of the Internet Header. It is set in binary the format of the Internet Header. It is et in binary format. The number would be 4 in case of Ipv4 packet and 6 for an Ipv6 packet.

- The traffic class has a 8-bit length and has replaced “Type of Service” in the Ipv4 packet. It is used by source nodes and routers on the path to the destination in order to distinguish between different priority packets. General requirement that apply to the Traffic Class field are as follows. Interface to Ipv6 service must allow the upper layer protocol to provide the values to the Traffic Class bits of a packet, which is originated at the upper layer protocol.

- The Flow label has a 20-bit length. This field is used by the source to label the sequence of packets in order to get special handling from routers. The flow Label is checked at the router, values are changed based on the next interface and then packets are forwarded. If routers do not support this field then it would just forward the packet and the field value is left unchanged. This saves unnecessary delays such as routing table look up.

- The payload field has a 16-bit length. This field has replaced the Total Length Field in Ipv4. This specifies the length of the payload following the Ipv6 header. It is expressed in octets.

- The Next Header field has a 8-bit length. It uses the same values that are used in the Ipv4 field and is used to identify the next header to follow.

- The Hop limit is similar to “Time To Live” field in the Ipv4 header. It does not perform the calculation of the time interval. Whenever a packet passes through a node the value in hop limit is decreased by 1. When the value in this field is zero, the packet gets discarded.

- The Source Address has a 128-bit length. This field contains the address of the source from where this packet has originated.

- The destination Address has a 128-bit length. This field contains the Address of the destination where this packet will finally reach.

2.7.2 Address categories (Ipv6 Address)

An IPv6 address can be classified into one of three categories:

- Unicast

- Multicast

- Any cast

IPv6 addresses are assigned to interfaces, as IPv4 addresses are. Each interface of a node needs at least one unicast address. A single interface can be assigned multiple IPv6 addresses of any type (unicast, multicast and any cast). A node can therefore be identified by any of its interfaces.

It is also possible to assign one unicast address to multiple interfaces for load sharing reasons, if the hardware and drivers support it.

The **unicast** address uniquely identifies an interface of an IPv6 node. An object sent to a unicast address is delivered to the interface identified by that address. This type of address will be described in more detail later in this chapter.

The **multicast** address identifies a group of IPv6 interfaces. A packet sent to a multicast address is processed by all members of the multicast group.

The **anycast** address is assigned to multiple interfaces (usually multiple nodes). A packet sent to an anycast address is delivered to only one of these interfaces, usually the nearest one.

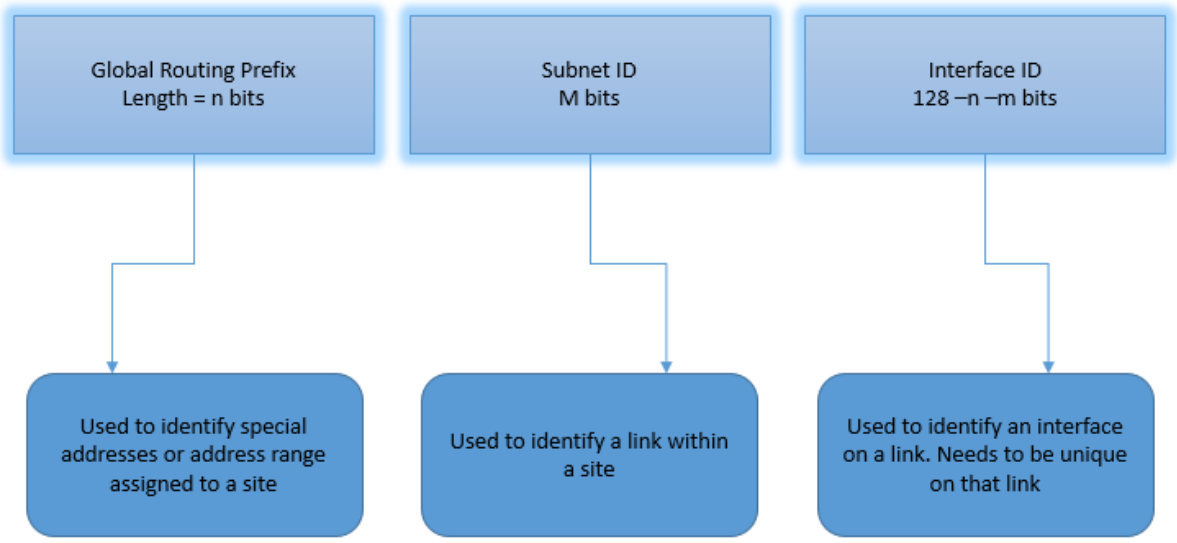


Figure 11: Ipv6 general address format

The global routing prefix is used to identify a special address, such as multicast, or an address range assigned to a site. A subnet ID (also referred to as “subnet prefix” or just “subnet”) is used to identify a link within a site. An interface ID is used to identify an interface on a link and needs to be unique on that link.

2.7.2.1 Address notation

Before any further description of the IPv6 address it is important to give a basic introduction on the IPv6 address notation.

An IPv6 address has 128 bits (16 bytes). The address is divided into eight, 16 bit hexadecimal blocks, separated by colons. For example:

FE80:0000:0000:0000:0202:B3FF:FE1E:8329

There are several ways to write these addresses, many zeros can be avoided, as there will be a lot of these in the address field. The first example shows the above address:

FE80:0:0:0:202:B3FF:FE1E:8329

To make the address even shorter a double colon can replace consecutive zeros, or leading or trailing zeros, within the address. If this rule applies the above address will look like this:

FE80:: 202:B3FF:FE1E:8329

The double colon can only appear once in an address, as the computer always uses the full 128 bits. Where the double colon is present the computer fills the address with zeros so that the address reaches the full length of 128 bits.

In an environment where both IPv4 and IPv6 nodes are mixed, there is another convenient form of IPv6 address notation. The IPv4 address can be inserted in the end of the IPv6 address in its original form:

IPv4 address example: 128.39.0.2 IPv6 address for above example: 0:0:0:0:0:0:128.39.0.2 or ::128.39.0.2 If preferred it also can be written in hexadecimals: ::8027:2

2.7.2.2 Prefix notation

The prefix notation is an important part of understanding the more complex hierarchy of IPv6 addresses than used in IPv4. IPv4 mainly divided the addresses into A-, B- and C-class addresses. The IPv6 prefix structure allows a larger range of network/subnetwork splits in the address.

A format prefix (also referred to as global routing prefix) is the high-order bits of an IPv6 address used to identify the subnet or a specific type of address. The notation appends the prefix length, written as a number of bits with a slash:

IPv6 address/prefix length

This is very similar to the CIDR notation for IPv4 or for subnetted IPv4 addresses. The prefix length indicates how many of the left most bits are a part of the prefix. The prefix is used by routers to identify which subnet the address belongs to. The packet is then forwarded using the value of the prefix only.

Example prefix notation: 2E78:DA53:12::/40

Figure 13 show this more clearly with the hexadecimal digits converted into binary

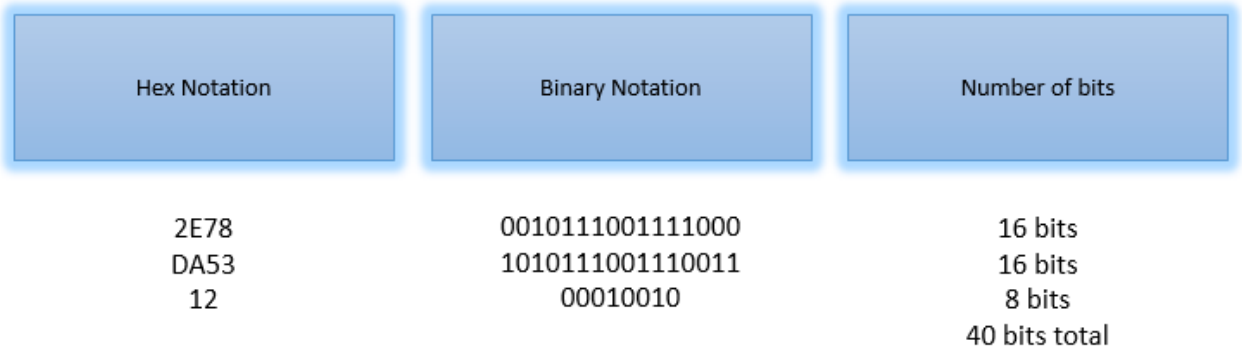


Figure 12: Hexadecimal Notation

Note that in the address notation the address would have to be written 2E78:DA53:1200::, but as it is only the 40 left most bits that are of interest, the double colon (::) will replace the remaining bits with zeros until the address reaches 128 bits.

The format prefixes that are used to identify special addresses, such as link-local addresses or multicast addresses are reserved prefixes, as shown in table 2.7.2.3 below.

Allocation	Prefix Binary	Prefix Hex	Fraction of address space
Reserved	0000 0000	::0/8	1/256
Reserved for NSAP Allocation	0000 001	::2/7	1/128
Reserved for IPX Allocation	0000 010	::4/7	1/128
Aggregatable global Unicast Addresses	001	::20/3	1/8
Link-local unicast Addresses	1111 1110 10	FE80::/10	
Site-local unicast addresses	1111 1110 11	FECO::/10	
	1111 1111	FF00::/8	

Figure 13: list of assigned prefixes

Some of the special addresses are assigned out of the reserved address space with the binary prefix 0000 0000. These include the unspecified address, the loop back address and the IPv6 addresses with the embedded IPv4 addresses.

Unicast addresses can be distinguished from multicast addresses by their prefix. Globally unique unicast addresses have a high-order byte starting at 001. An IPv6 address with a high order byte of 1111 1111 (FF in hex) is always a multicast address.

Anycast addresses are taken from the unicast address space, so it is not possible to identify these only by looking at the prefix. If a unicast address is assigned to multiple interfaces, which makes it an anycast address, the interfaces need to be configured to let them all know that the address is an anycast address.

Addresses in the prefix range 001 to 111 should use the 64-bit interface identifier that follows the EUI-64 (Extended Unique Identifier) format. The EUI-64 is a unique identifier defined by the IEEE.

2.8 A brief explanation of the fields Ipv4 packets.

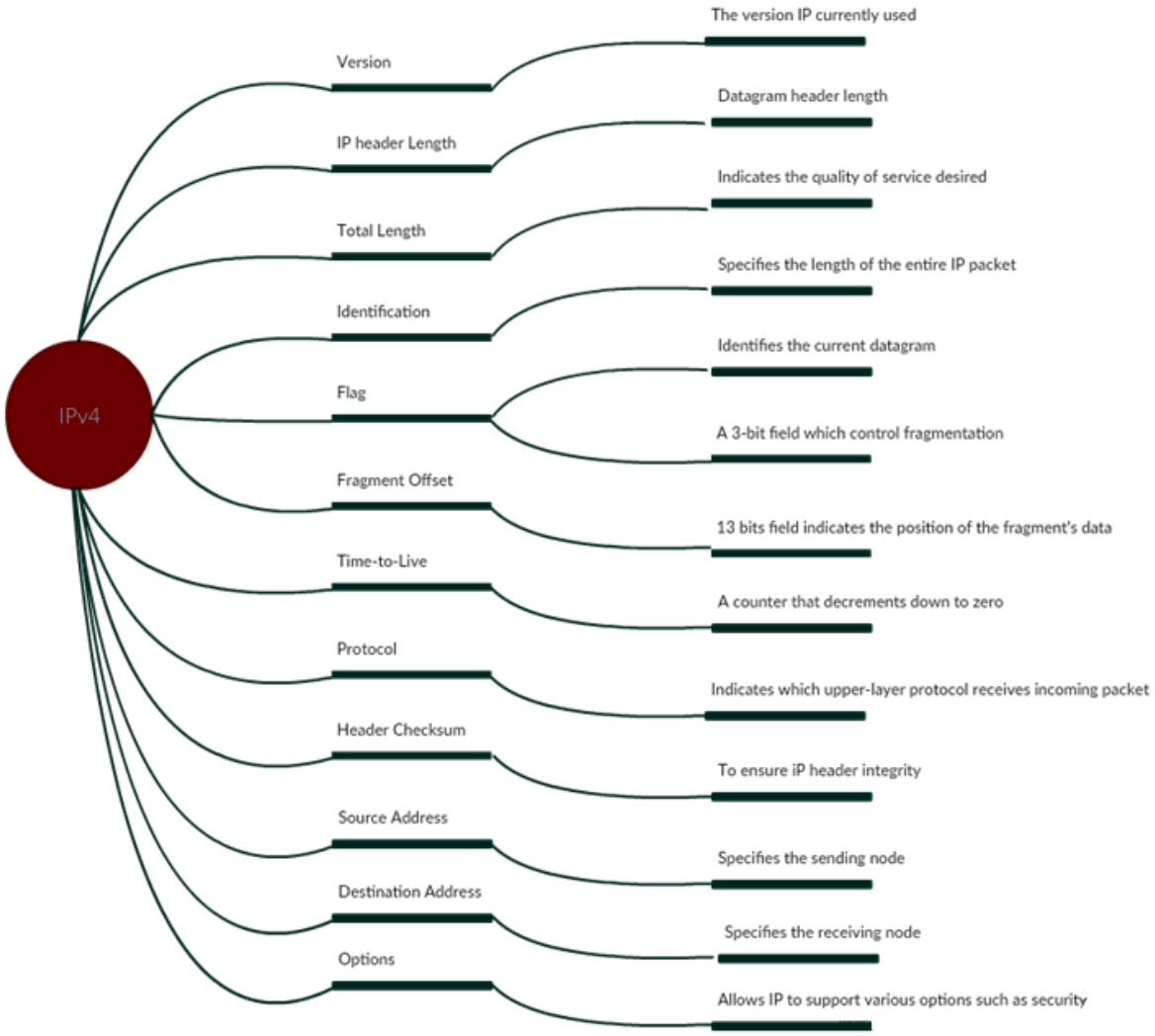


Figure 14: IPvv4 packet fields

2.9 A brief explanation of the field of IPv6 packets.

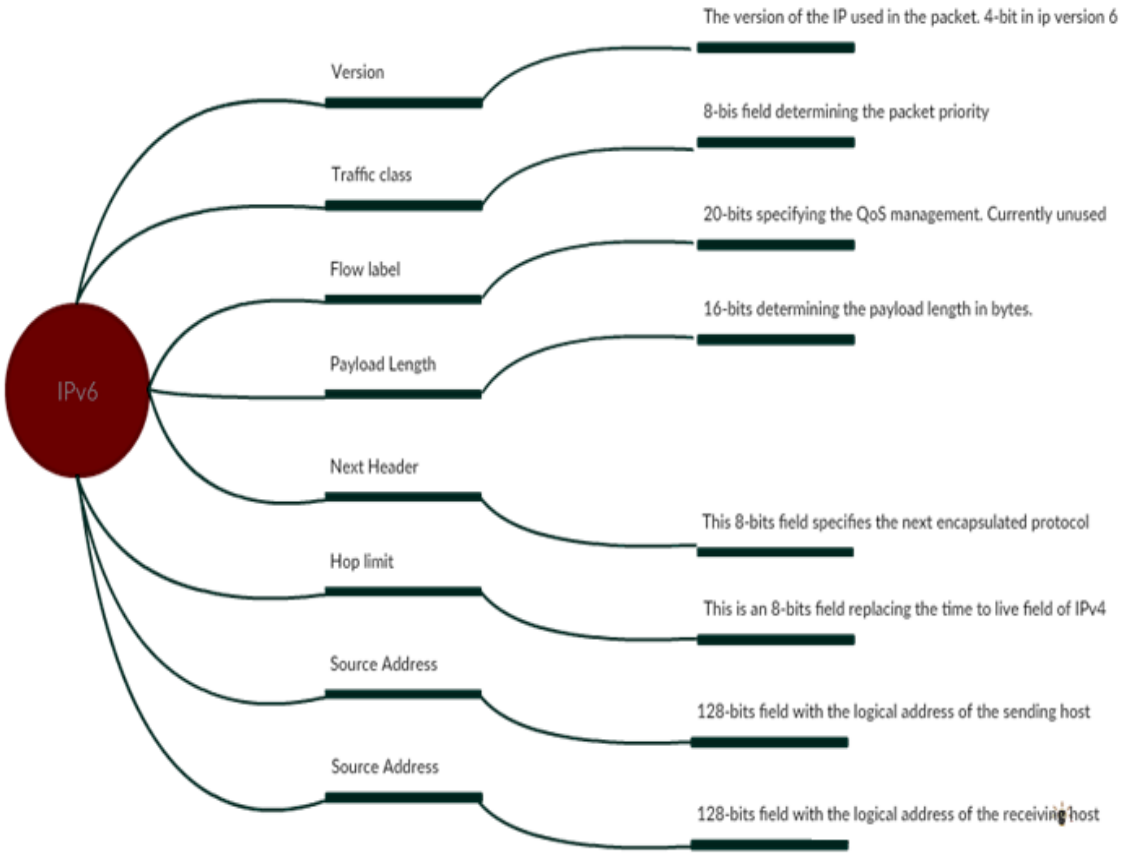


Figure 15:Ipv6 packet fields

Options header for IPv6 packets are:

- Hop-by-Hop Options header
- Routing header
- Fragment header
- Destination Options header

- Authentication header
- Encrypted Security payload header

There is not always an Extension header with every header. There may be just one or there may be more than one between the IPv6 header and the Upper-Layer Protocol header, which is always the last header in an IP packet. It all depends on the requirements of the processing of the payload of the packet. Each Extension header is identified in the Next header field of the preceding header. Only the destination node and none of the other nodes between the source and the destination process the Extension headers. The Extension headers must be processed in the order they are arranged in, in the packet header. The only exception to the rule about the destination node being the only one to process the Extension headers is when the Hop-by-hop Options header is in use. All nodes in the route towards the destination node must process this. The Hop-by-hop header must therefore always immediately follow the IPv6 header.

2.10 Header differences between Ipv4 and Ipv6

The innovation of IPv6 lies in its header. It is two times larger than IPv4 header and it is formed of a Fixed Header and zero or more Extensions (optional headers). All the essential information for a router is kept in the fixed header. The Extension contains optional information that helps routers to understand how to handle a packet. The IPv6 header has lost some fields that were used in the IPv4 header as you can see in Figure 8, thus saving time processing the packets. IPv6 fixed header is 40 bytes long while IPv4 is 20 bytes. The version field represents the version of Internet protocol.

2.11 Benefits of Ipv6 over Ipv4

Ipv6 has been inundated with a wide variety of features that were not available in Ipv4. This section will put light on those features and how they are responsible for increasing the efficiency of the Ipv6.

- **Address length**

  The increase in the address length from 32-bit to 128 bit resulted in a large quantity of available addresses. Even if a single utilizes thousands of IP capable devices, the IP addresses would not get exhausted. With the increase in the quantity of IP addresses the

requirement for NAT was eliminated. Availability of IP addresses resulted in a more efficient assignment of addresses to the networks and as well as a more simplistic routing procedure. The routing table in Ipv6 have fewer entries compared to Ipv4, thereby enabling quick look-ups. With the increase in the number of IP addresses more peer to peer applications were designed with improved speed and reliability.

- **Address Auto Configuration**

DCHP was used to obtain Ipv4 addresses and the domain name server. DCHPv6 was published for Ipv6. The most important feature ipv6 is to automatically configure itself. The neighbor Discovery feature in Ipv6 enables the hosts to know the availability of routers on the network, whereas in ipv4, hosts must wait until a router advertises its address. In Ipv6 the host broadcasts the router solicitation message and waits for a response from a router indicating its presence. Through the router solicitation message a host gets the network prefix from a router on the

- **Simplified Header**

  IEFT designed a simplified header format for ipv6. This header was stripped of non-essential fields that were available in ipv4. This format increased the performance and efficiency at the nodes. The INTERNET header length field was removed. Likewise Total Length is replaced by payload Length, which refers to the size of the payload after header. Further fields removed are Fragmentation Offset and Flag, since ipv6 packets do not undergo fragmentation. The time To live filed was replaced by Hop limit. This avoided the calculation of the time interval. Flow Label field is used to label the packets belonging to the same data flow. Unlike Ipv4, Ipv6 does not contain IP checksum and error handling capabilities.

- **Flow Label**

  This is one of the fields present in the Ipv6 header, which indicates the routers that will handle the packets traveling from source to destination in a special manner. The flow on the network is identified by Flow Label and the source address of the packet. Specific flow indicates that the packets originated from the same source. The packets originating from the same source belonging to a flow will have the same kind of requirement. The requirements of the packets are decided before they are transmitted onto the network. Once it is on the network, routers check the flow label to identify the special handling requirements of the packet. Packets belonging to the same flow will have the same source, destination address and routing options. This enables the router to process at a faster rate.

- **Extension Headers**

Most of the extension headers are processed by destination nodes, so routers have less processing to be done. The Hop-by-Hop options header is the only header that has to be processed by all nodes. It immediately follows the Ipv6 header. Each of these extension header may occur only once, but Destination options headers may occur only once, but Destination options header may occur twice in different positions.

- **Hop-by-Hop**

  It is used to carry optional information that has to be processed by each and every node. The Next Header filed is set to zero in the Ipv6 header to indicate that presence of the Hop-by-Hop field. It always appears after the Ipv6 header.

- **Destination Options Header**

  This header holds the content that has to be processed by destination node. This header appears after the Hop-By-Hop and before the Upper Layer headers.

- **Routing Header**

  This header holds that information specified by the Ipv6 source. Ipv6 source specifies the intermediate nodes that the packet will have to travel in order to reach its destination. The Routing Header helps in getting the maximum transmission unit of the next link known. If the link maximum transmission unit is smaller than the packet size then the router sends the ICMP message to the source indicating the packet is too big and has to be limited to a smaller size.

- **Fragment Header**

  Fragmentation procedure implemented in Ipv6 is different from that of Ipv4. In Ipv4 packets are fragmented by the routers if the maximum transmission unit is too large compared to the maximum transmission unit of the link on which it has to traverse. In Ipv6 when the packets whose size is bigger than the maximum transmission unit of the link, then, that packet is discarded and ICMP message is sent to the source node indicating the maximum transmission unit of a packet that will be accepted. The source will then fragment the packet accordingly and transmit it.

- **Authentication Header**

  This header helps in checking the packets authenticity and integrity when it arrives at the destination. MDA5 and SHA algorithms are used to protect the packets against deliberate modifications.

- **Encrypted Security Payload**

When this header is used it has to be the last one on the header chain. This header hides the entire upper level payload and the Next Headers. The default encryption used in Ipv6 is DES-CBC, which is a data encryption standard algorithm in cipher block chaining mode.

Chapter 3: Packet Sniffer

3.1 Overview to Packet Sniffer

A Packet Sniffer is a program that can see all of the information passing over the network it is connected to. A Packet Sniffer is a Wire-tapping device that plugs into computer Networks and eavesdrop on the network traffic.

A packet sniffer (also known as a network analyzer or protocol analyzer or, for particular types of networks, an Ethernet sniffer or wireless sniffer) is computer software that can intercept and log traffic passing over a digital network or part of a network. As data streams flow across the network, the sniffer captures each packet and eventually decodes and analyzes its content.

Most Ethernet networks use to be of a common bus topology, using either coax cable or twisted pair wire and a hub. All of the nodes (computers and other devices) on the network could communicate over the same wires and take turns sending data using a scheme known as carrier sense multiple access with collision detection (CSMA/CD). Think of CSMA/CD as being like a conversation at a loud party, you may have to wait for quite a spell for your chance to get your words in during a lull in everybody else's conversation. All of the nodes on the network have their own unique MAC (media access control) address that they use to send packets of information to each other. Normally a node would only look at the packets that are destined for its MAC address. However, if the network card is put into what is known as "promiscuous mode" it will look at all of the packets on the wires it is hooked to?

3.1.1 Importance of Packet Sniffers:

Computer administrators need Packet Sniffer to monitor their networks and perform diagnostic tests or trouble shoot problems. It can be used to analyze the load on the network and according to that administrator can redirect the traffic so that packets flow across networks without leading to congestion. It is also used for debugging by Programmers developing network related software. Anyone interested in having a full picture of the traffic going through one's LAN segment can use

the Packet Sniffer. Packet Sniffer converts the data flowing in the network into human readable format, so that people can read the traffic and understand it.

On wired broadcast LANs, depending on the network structure (hub or switch), one can capture traffic on all or just parts of the traffic from a single machine within the network; however, there are some methods to avoid traffic narrowing by switches to gain access to traffic from other systems on the network (e.g. ARP spoofing). For network monitoring purposes it may also be desirable to monitor all data packets in a LAN by using a network switch with a so-called monitoring port, whose purpose is to mirror all packets passing through all ports of the switch. When systems (computers) are connected to a switch port rather than a hub the analyzer will be unable to read the data due to the intrinsic nature of switched networks. In this case a shadow port must be created in order for the sniffer to capture the data.

On wireless LANs, one can capture traffic on a particular channel.

On wired broadcast and wireless LANs, in order to capture traffic other than unicast traffic sent to the machine running the sniffer software, multicast traffic sent to a multicast group to which that machine is listening, and broadcast traffic, the network adapter being used to capture the traffic must be put into promiscuous mode; some sniffers support this, others don't. On wireless LANs, even if the adapter is in promiscuous mode, packets not for the service set for which the adapter is configured will usually be ignored; in order to see those packets, the adapter must be put into monitor mode.

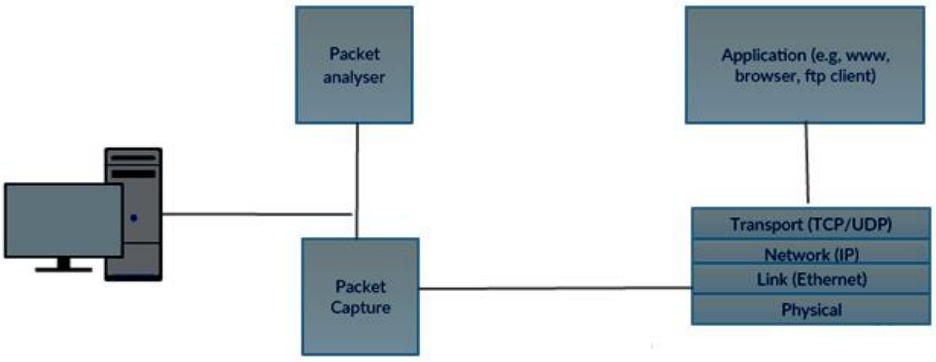


Figure 16: Packet Sniffer Structure

3.1.2 Uses of Sniffers

Sniffing programs have been around for a long time in two forms. Commercial packet sniffers are used to help maintain networks.

Typical uses of such programs include:

- Automatic sniffing of clear-text passwords and usernames from the network.
- Conversion of data to human readable formats so that people can read the traffic.
- Fault analysis to discover problems in the network, such as why computer A can't talk to computer B.
- Performance analysis to discover network bottlenecks.
- Network intrusion detection in order to discover hackers/crackers.
- Network traffic logging, to create logs that hackers can't break into and erase.
- Monitor network usage.
- Debug client/server communications.
- Debug network protocol implementations.

Example Uses:

- A packet sniffer for a token ring network could detect that the token has been lost or the presence of too many tokens (verifying the protocol).
- A packet sniffer could detect that messages are being sent to a network adapter; if the network adapter did not report receiving the messages then this would localize the failure to the adapter.
- A packet sniffer could detect excessive messages being sent by a port, detecting an error in the implementation.
- A packet sniffer could collect statistics on the amount of traffic (number of messages) from a process detecting the need for more bandwidth or a better method.
- A packet sniffer could be used to extract messages and reassemble into a complete form the traffic from a process, allowing it to be reverse engineered.
- A packet sniffer could be used to diagnose operating system connectivity issues like web, ftp, sql, active directory, etc.
- A packet sniffer could be used to analyze data sent to and from secure systems in order to understand and circumvent security measures, for the purposes of penetration testing or illegal activities.
- A packet sniffer can passively capture data going between a web visitor and the web servers decode it at the HTTP and HTML level and create web log files as a substitute for server logs and page tagging for web analytics.

3.2 Network Testing Tools

Tcpdump / Wireshark

As you go about developing your firewall you might find it useful to observe the packets arriving at the network interface. Amongst other things observing packet data is helpful for debugging (you can try and determine properties of packets not being processed correctly), making sure that your firewall is actually being tested and just determining the kinds of packets generated by a variety of applications.

Packet sniffers are commonly used to accomplish these tasks, your VM has two of these installed: Wireshark and tcpdump. Wireshark is graphical, while tcpdump is a command line tool. Both are capable of filtering packets and are almost equally powerful. We briefly describe both below and point to a few sources of information online. We strongly encourage you to look through tutorial and other documentation.

Normally both tools require root privileges to run, since packets contain security critical information. However, in the provided VM, you don't need to do "sudo" every time to run them (we did something special for you).

When you run tcpdump/Wireshark on your own machine, you will see a lot of packets, since there are many background applications that connect to the Internet. In the VM, we disabled most such background applications, so you will see much fewer "noise" packets.

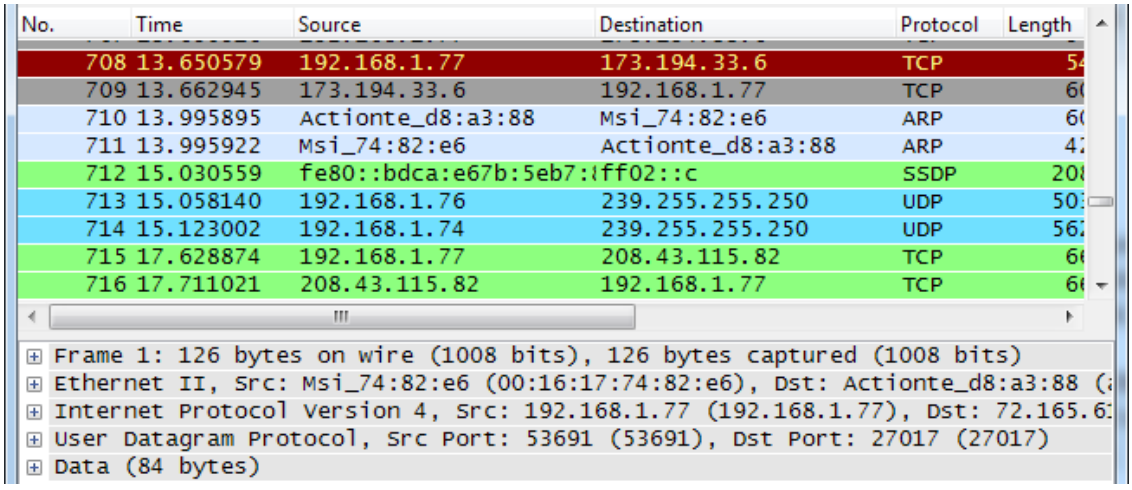


Figure 17: Capture Packet

3.3 Wireshark:

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color-coding and other features that let you dig deep into network traffic and inspect individual packets.

This tutorial will get you up to speed with the basics of capturing packets, filtering them, and inspecting them. You can use Wireshark to inspect a suspicious program's network traffic, analyze the traffic flow on your network, or troubleshoot network problems.

3.3.1 Getting Wireshark

You can download Wireshark for Windows, Linux or Mac OS X. If you're using Linux or another UNIX-like system, you'll probably find Wireshark in its package repositories. For example, if you're using Ubuntu, you'll find Wireshark in the Ubuntu Software Center.

Just a quick warning: Many organizations don't allow Wireshark and similar tools on their networks. Don't use this tool at work unless you have permission.

3.3.2 Capturing Packets

After downloading and installing Wireshark, you can launch it and click the name of an interface under Interface List to start capturing packets on that interface. For example, if you want to capture traffic on the wireless network, click your wireless interface. You can configure advanced features by clicking Capture Options, but this isn't necessary for now.

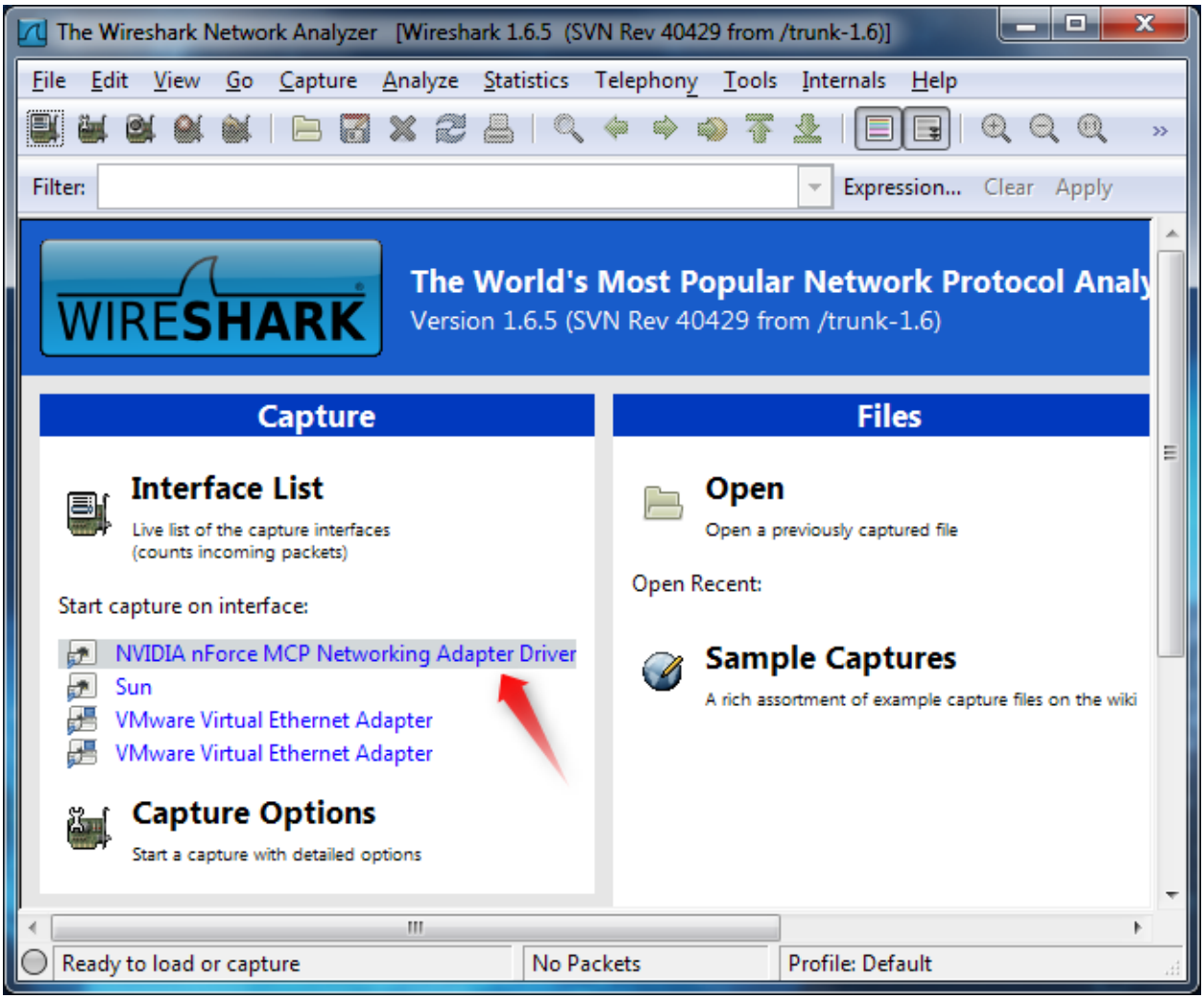


Figure 18: Wireshark

As soon as you click the interface's name, you'll see the packets start to appear in real time. Wireshark captures each packet sent to or from your system. If you're capturing on a wireless interface and have promiscuous mode enabled in your capture options, you'll also see other the other packets on the network.

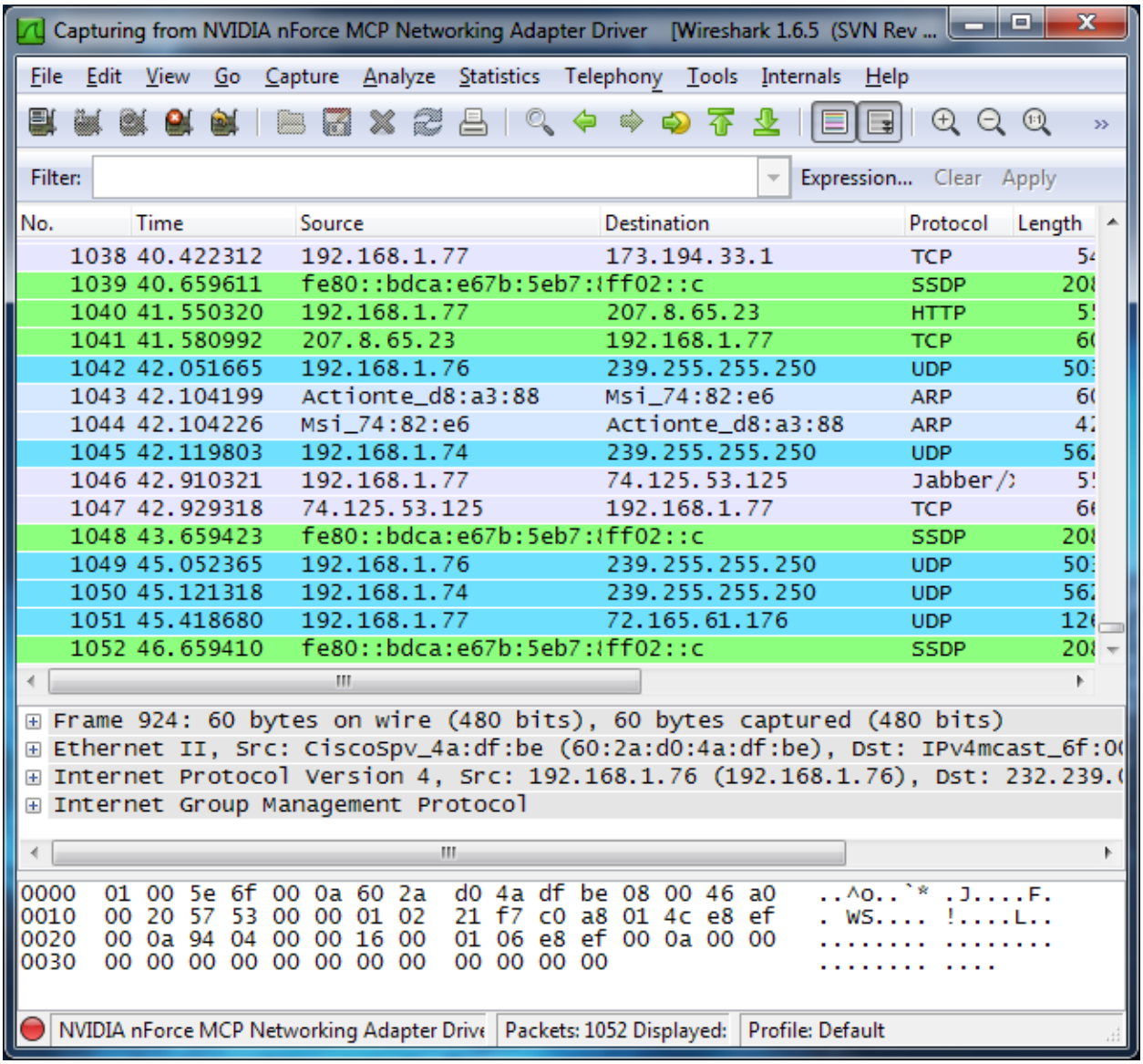


Figure 19: Start sniffing packet

Click the stop capture button near the top left corner of the window when you want to stop capturing traffic.

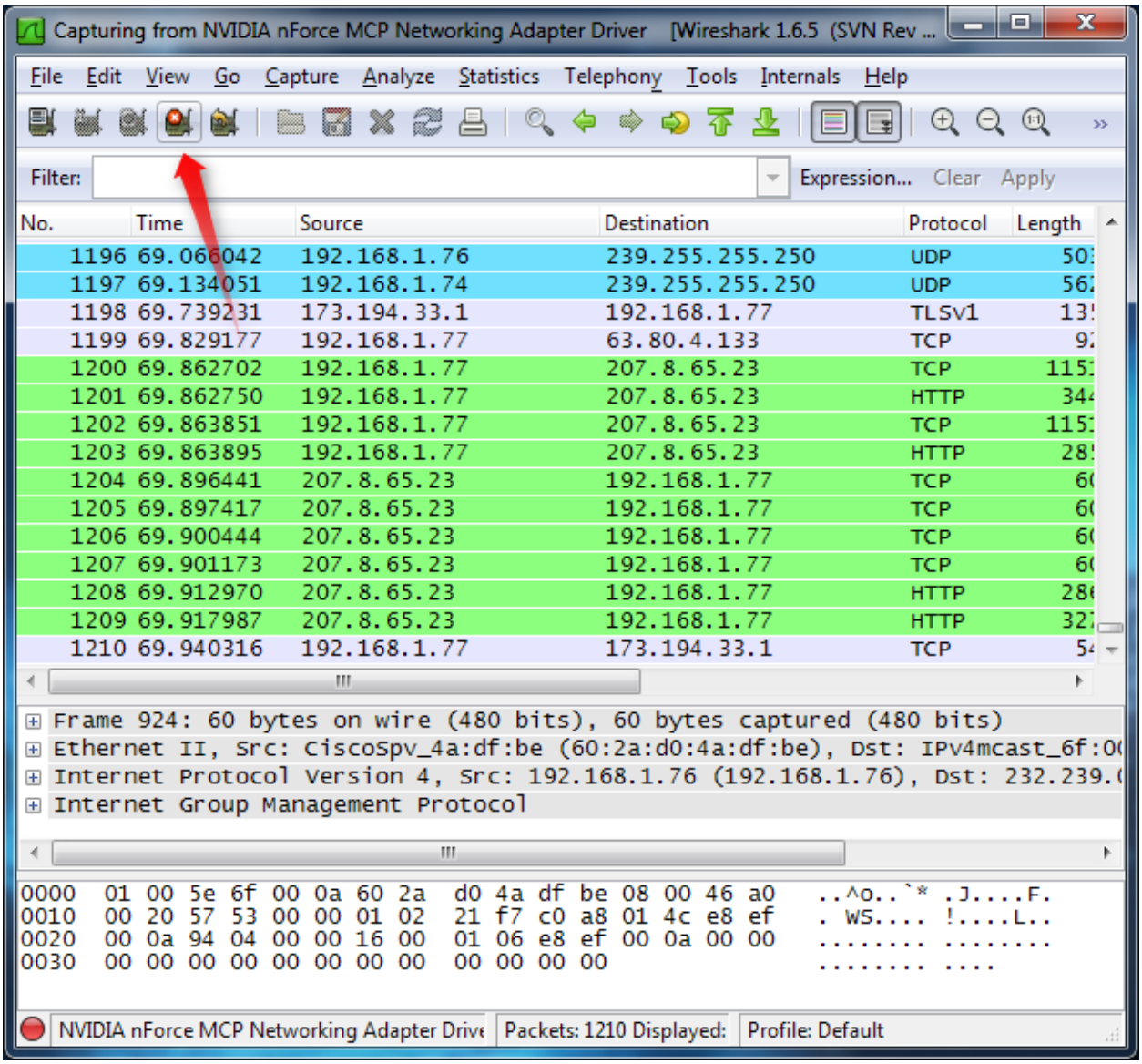


Figure 20: Stop Capturing Packet

3.3.3 Color Coding

You'll probably see packets highlighted in green, blue, and black. Wireshark uses colors to help you identify the types of traffic at a glance. By default, green is TCP traffic, dark blue is DNS traffic, light blue is UDP traffic, and black identifies TCP packets with problems — for example, they could have been delivered out-of-order.

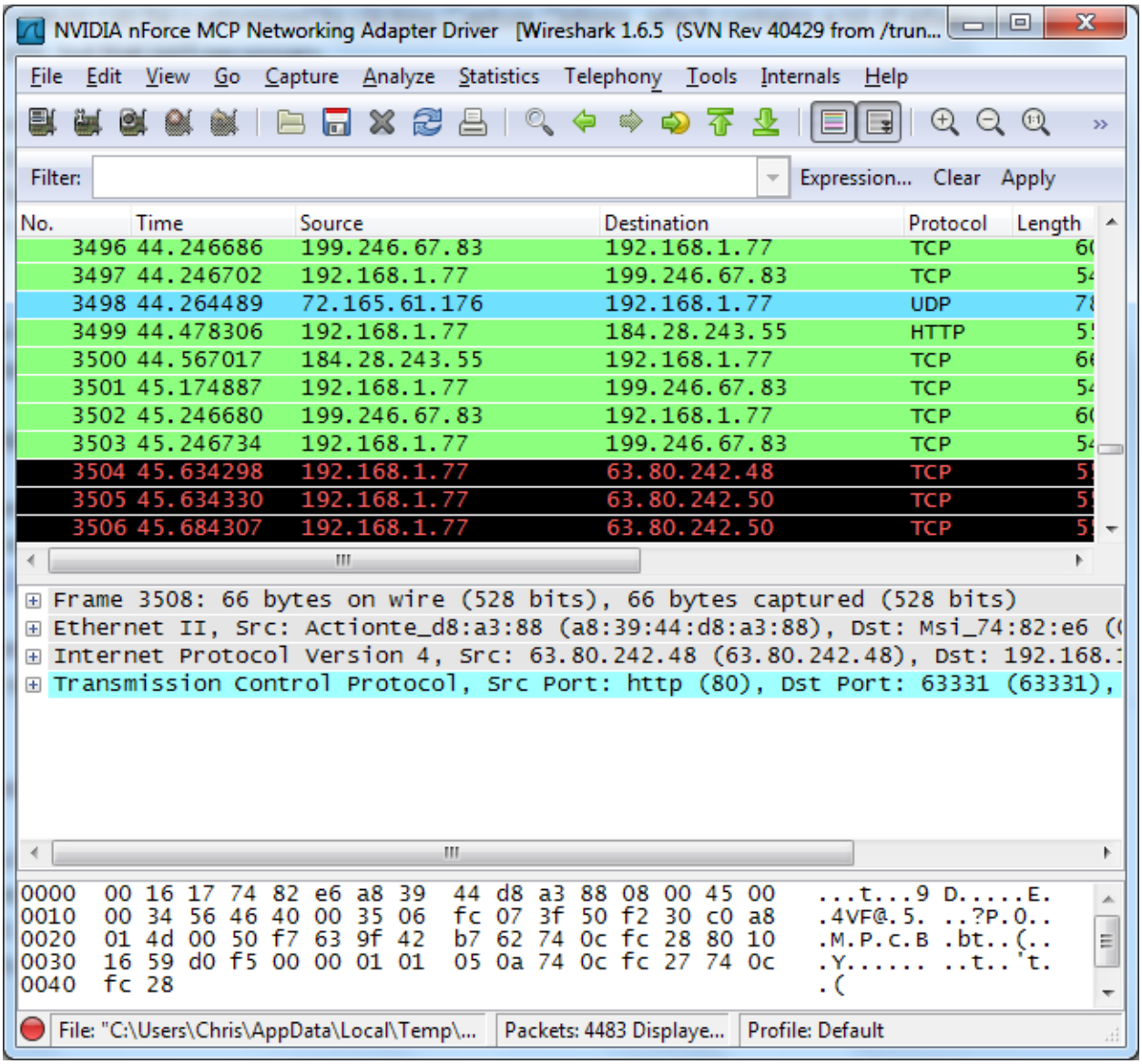


Figure 21: Color code

3.3.4 Filtering Packets

If you're trying to inspect something specific, such as the traffic a program sends when phoning home, it helps to close down all other applications using the network so you can narrow down the traffic. Still, you'll likely have a large amount of packets to sift through. That's where Wireshark's filters come in.

The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type “dns” and you’ll see only DNS packets. When you start typing, Wireshark will help you autocomplete your filter.

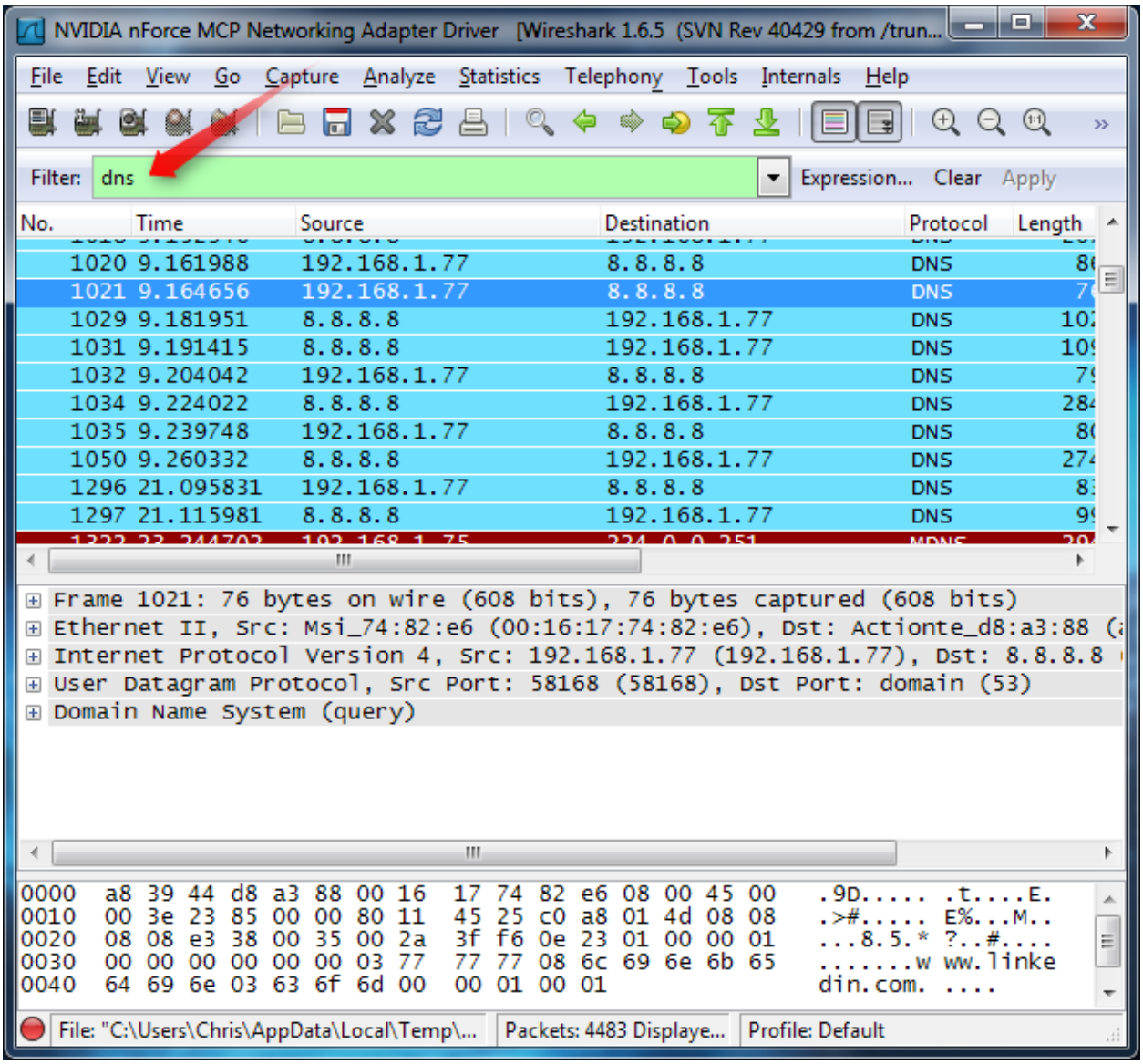
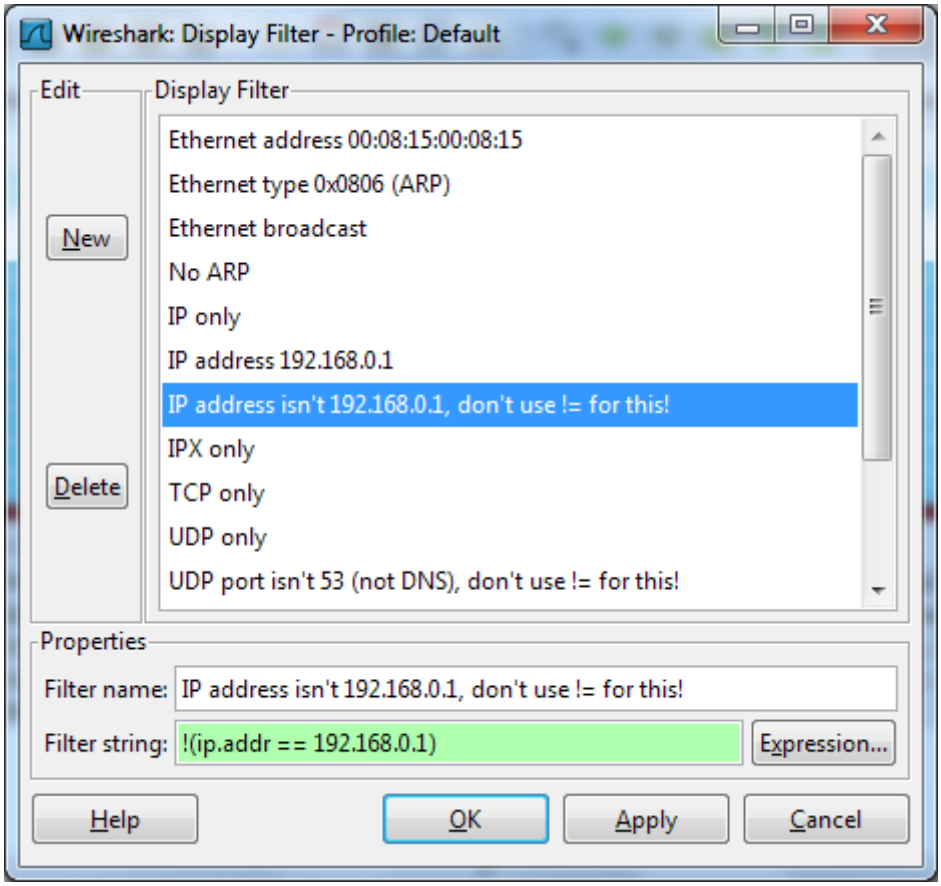


Figure 22: Filtering Packets

You can also click the Analyze menu and select Display Filters to create a new filter.



Another interesting thing you can do is right-click a packet and select Follow TCP Stream.

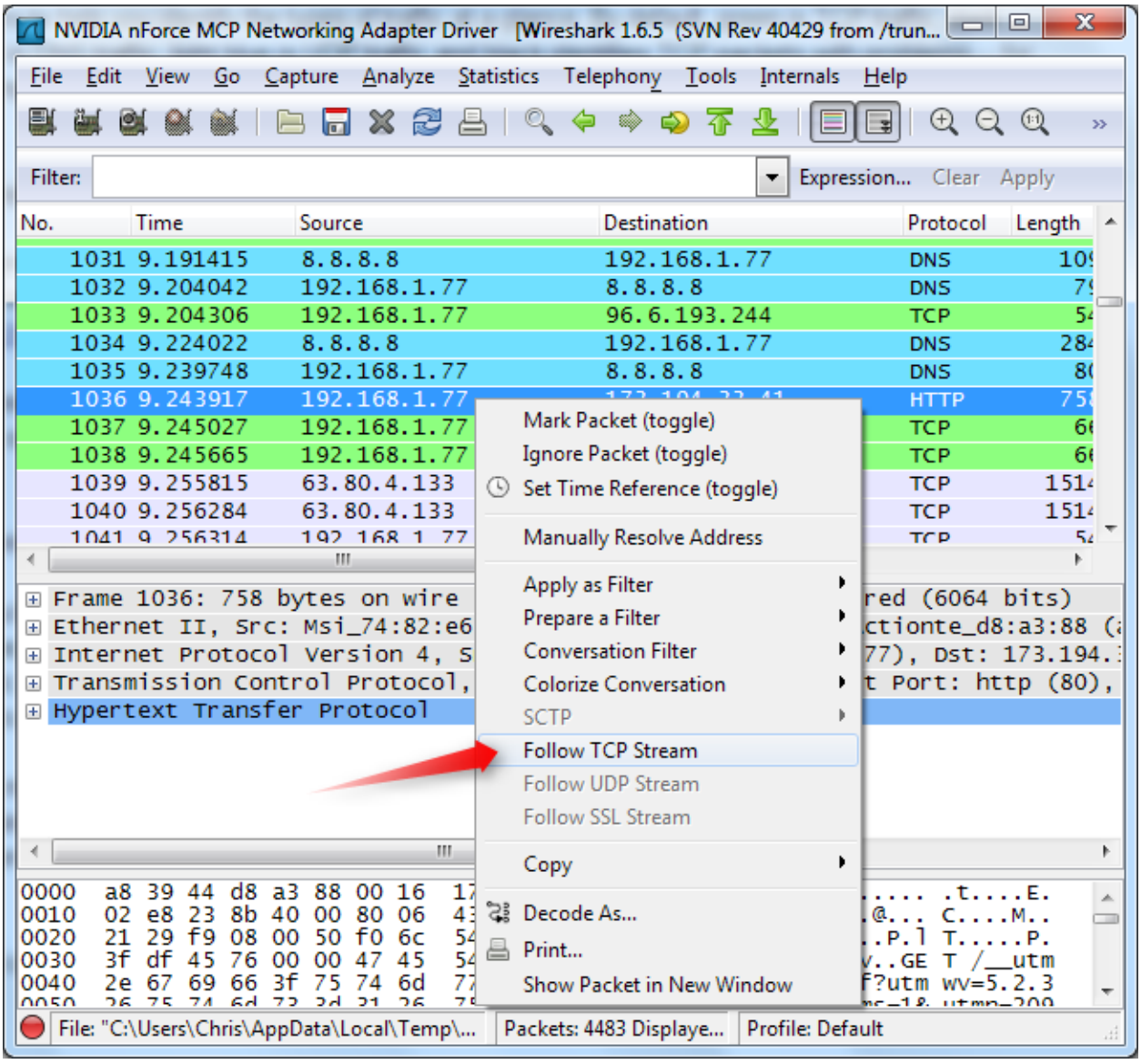
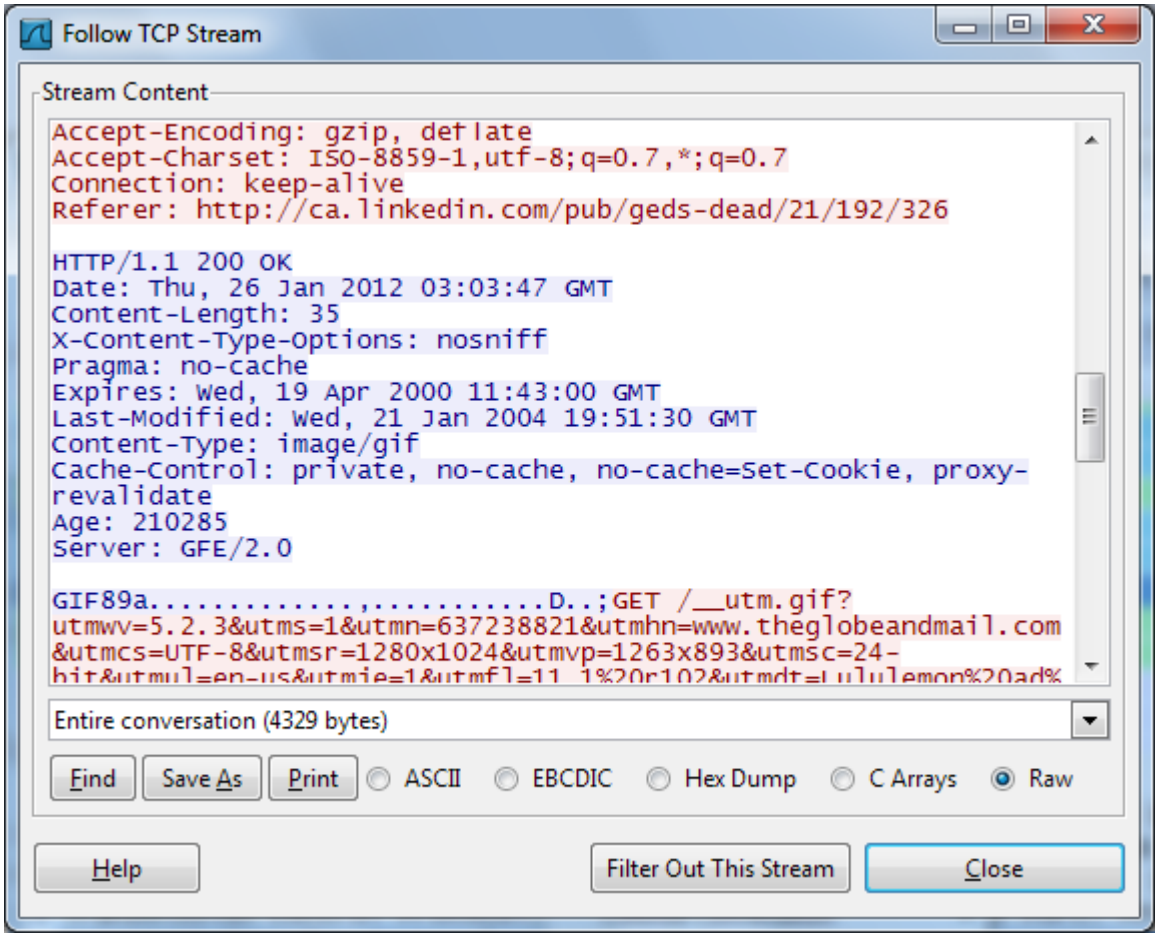
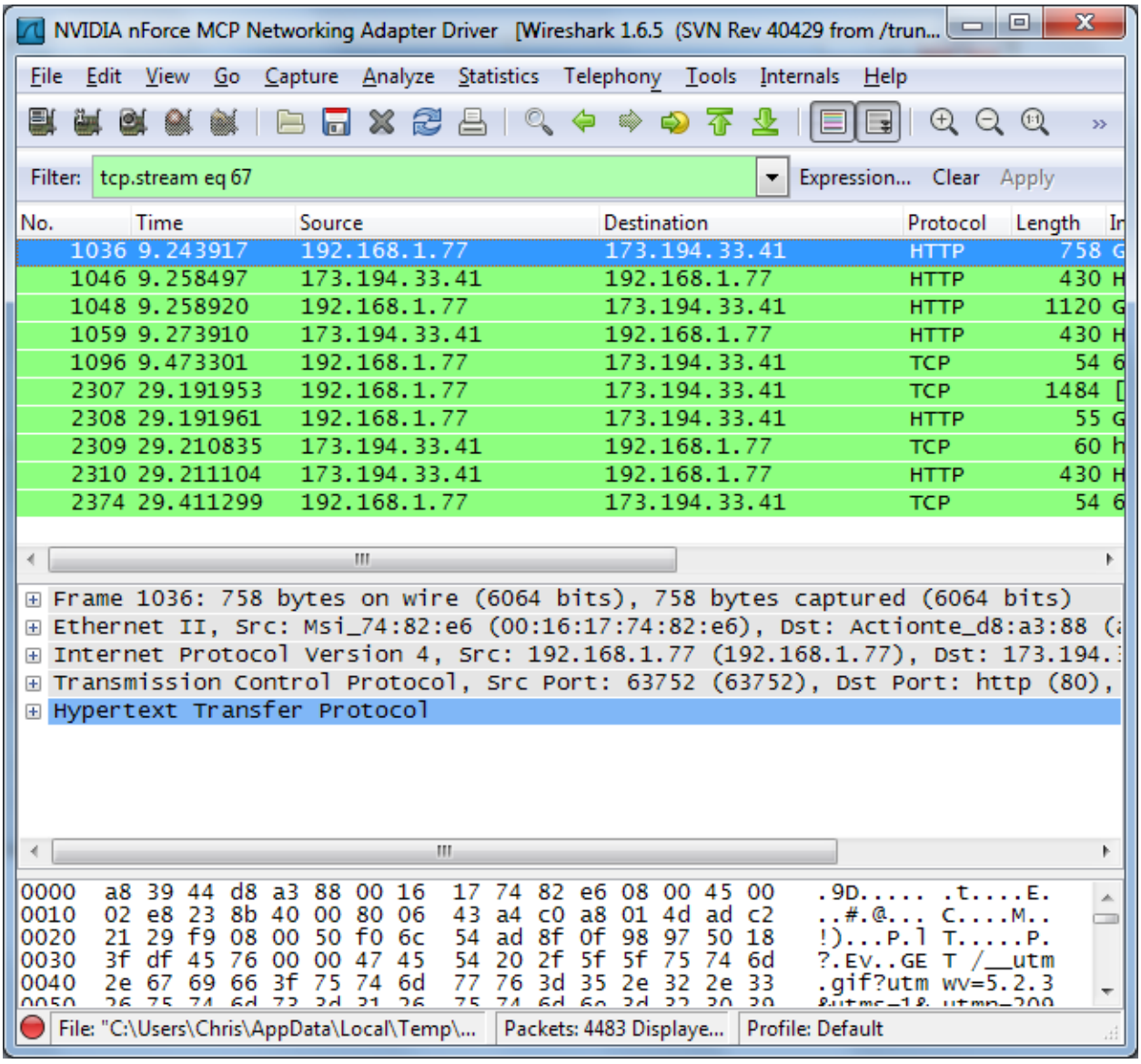


Figure 23: Follow TCP Stream

You'll see the full conversation between the client and the server.



Close the window and you'll find a filter has been applied automatically. Wireshark is showing you the packets that make up the conversation.



3.3.5 Inspecting Packets

Click a packet to select it and you can dig down to view its details.

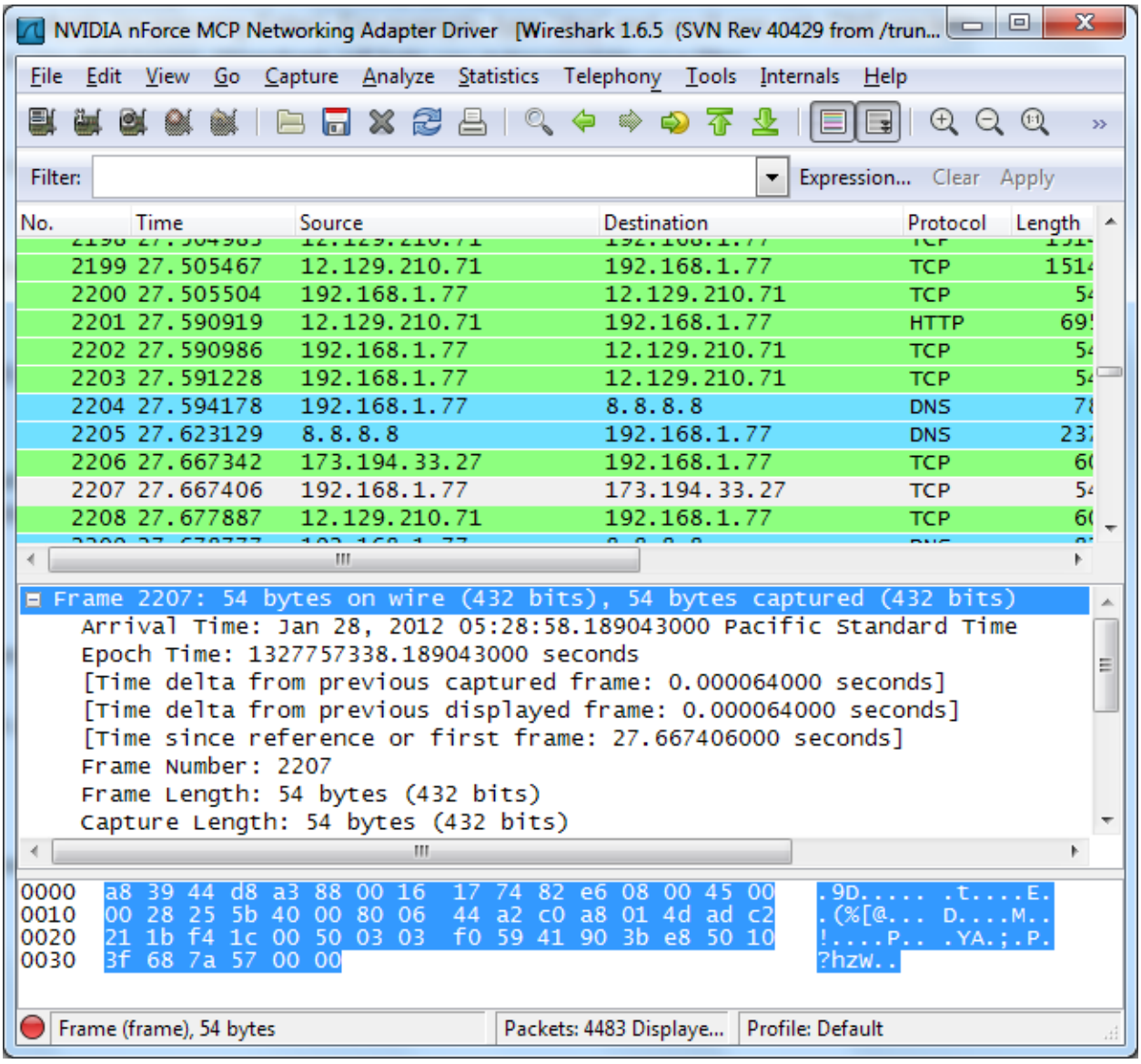
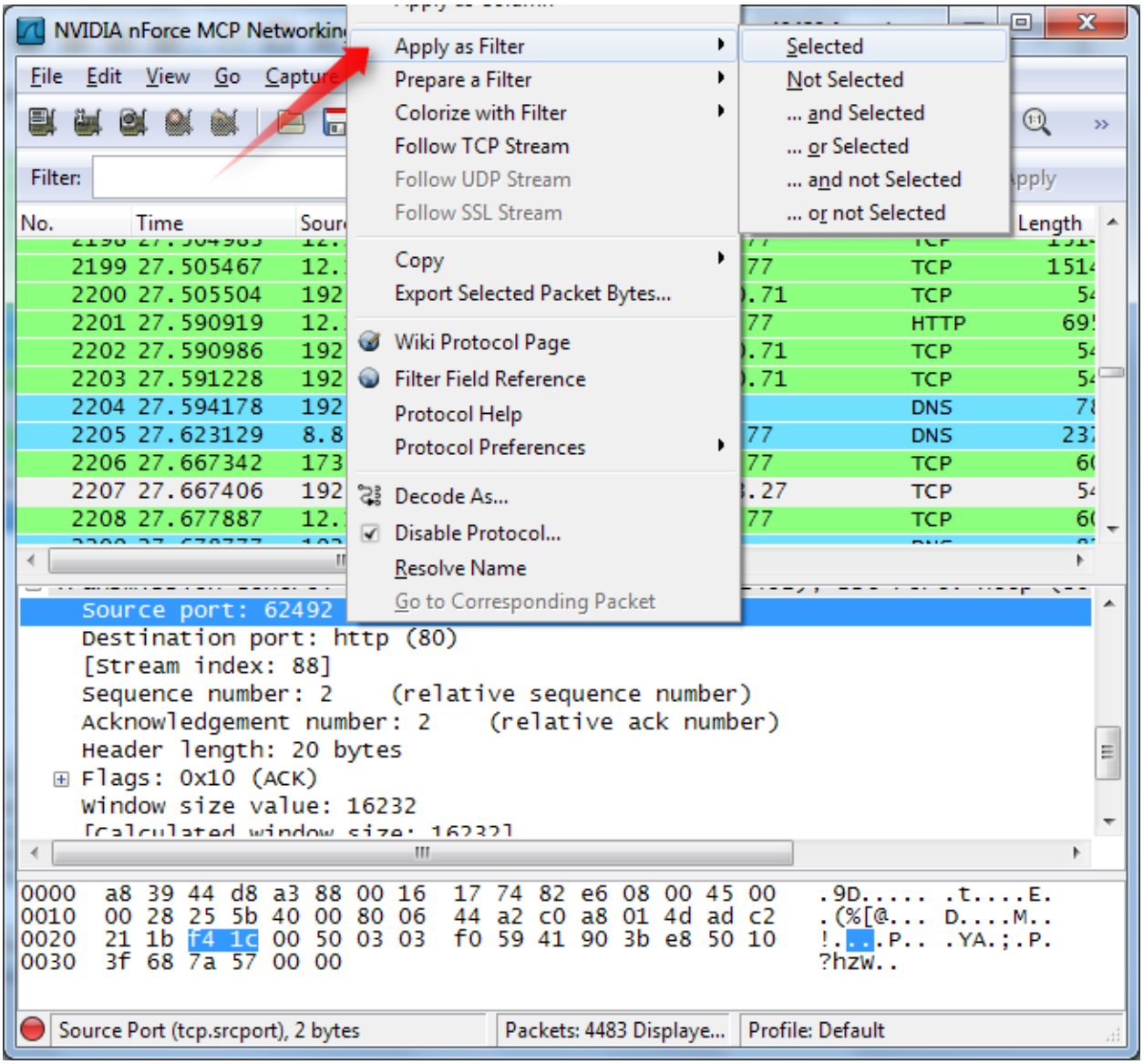


Figure 24: Inspecting Packets

You can also create filters from here, just right-click one of the details and use the Apply as Filter submenu to create a filter based on it.


NVIDIA nForce MCP Networking
File Edit View Go Capture
Filter:
No. Time Sour Length
2199 27.505467 12. 77 TCP 1514
2200 27.505504 192 .71 TCP 54
2201 27.590919 12. 77 HTTP 69
2202 27.590986 192 .71 TCP 54
2203 27.591228 192 .71 TCP 54
2204 27.594178 192 DNS 78
2205 27.623129 8.8 77 DNS 237
2206 27.667342 173 77 TCP 60
2207 27.667406 192 .27 TCP 54
2208 27.677887 12. 77 TCP 60
Apply as Filter
Prepare a Filter
Colorize with Filter
Follow TCP Stream
Follow UDP Stream
Follow SSL Stream
Copy
Export Selected Packet Bytes...
Wiki Protocol Page
Filter Field Reference
Protocol Help
Protocol Preferences
Decode As...
Disable Protocol...
Resolve Name
Go to Corresponding Packet
Selected
Not Selected
... and Selected
... or Selected
... and not Selected
... or not Selected
Source port: 62492
Destination port: http (80)
[Stream index: 88]
Sequence number: 2 (relative sequence number)
Acknowledgement number: 2 (relative ack number)
Header length: 20 bytes
Flags: 0x10 (ACK)
Window size value: 16232
0000 a8 39 44 d8 a3 88 00 16 17 74 82 e6 08 00 45 00 .9D..... .t....E.
0010 00 28 25 5b 40 00 80 06 44 a2 c0 a8 01 4d ad c2 .(%[@... D....M..
0020 21 1b f4 1c 00 50 03 03 f0 59 41 90 3b e8 50 10 !....P.. .YA.;.P.
0030 3f 68 7a 57 00 00 ?hzW..
Source Port (tcp.srcport), 2 bytes
Packets: 4483 Displaye...
Profile: Default

3.4 Conclusion

Wireshark is an extremely powerful tool, and it is just scratching the surface of what you can do with it. Professionals use it to debug network protocol implementations, examine security problems and inspect network protocol internals.

Chapter 4: Transition Mechanism

4.1 Overview:

Transition mechanisms help in the transition from one protocol to another. In the perspective of IPv6, transition basically means moving from IPv4 to IPv6. One day, IPv6 networks will completely replace today's IPv4 networks. For the near term, a number of transition mechanisms are required to enable both protocols to operate simultaneously. Some of the most widely used transition mechanisms are discussed in the following sections.

4.2 Introduction

Transition methods support gradually moving from Ipv4 to Ipv6 without major interruptions of services and networks. It is expected that most networks will have to support both Ipv4 and Ipv6 in parallel for a long time because of legacy equipment and the dependency on others to completely switch to the new protocol. Transition methods can be categorized can be categorized into tunneling and translation methods.

4.3 Transition mechanisms

The transition from Ipv4 to Ipv6 is expected to take years, and in the meantime, both protocols will have to coexist and interoperate. For this to happen IEFT has developed various tools that come to help the network administrator's transition to Ipv6. There are three caterogies of migration techniques:

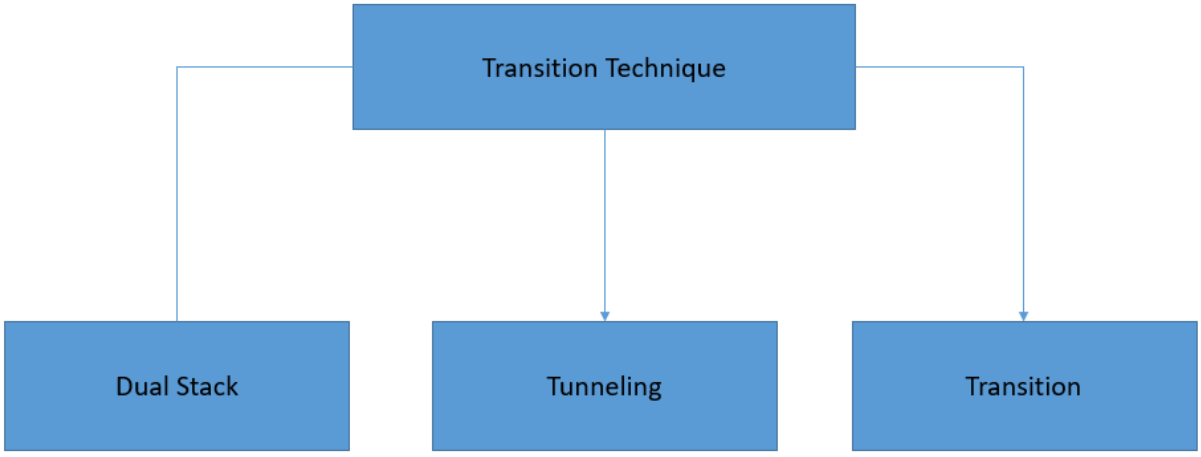


Figure 25: Transition Mechanism

- **Dual Stack**

  Both Ipv4 and Ipv6 will run simultaneously on devices in the network, allowing them to coexist in the ISP network.

- **Tunneling**

  An IPv6 packet is encapsulated in IPv4 packet and send over an IPv4 network.

- **Translation**

  A similar technique to NAT for IPv4 is used. Using NAT (Network Address Translation), the IPv6 packet is translated to IPv4 packet.

4.3.1 Dual Stack

A dual stack network is a network in which all of the nodes are both IPv4 and IPv6 enabled. This is especially important at the router, as the router is typically the first node on a given network to receive traffic from outside of the network.

Many experts believe that network infrastructure will shift from IPv4 to IPv6 in order to provide more address space and serve growing global connectivity. Dual stack networks are one of the many IPv4 to IPv6 migration strategies that have been presented in recent years.

The dual-stack mechanism has been used in the past. Previous examples of this mechanism include IPv4 with IPX and/or AppleTalk coexisting on the same node. As with other uses of dual-stack, the IPv4 and IPv6 protocols are not compatible with each other.

The dual-stack model enables the smoothest transitioning from IPv4 to IPv6 environments with minimal service disruptions. This model works by enabling IPv6 in the existing IPv4 environments along with the associated features required to make IPv6 routable, highly available, and secure.

The primary advantage of the dual-stack mechanism is that it does not require tunneling within the network. The dual-stack runs the two protocols as "ships-in-the-night," meaning that IPv4 and IPv6 run alongside one another and have no dependency on each other to function, except that they share network resources. Both IPv4 and IPv6 have independent routing, high availability (HA), quality of service (QoS), security, and multicast policies. Dual-stack also offers forwarding performance advantages because packets are natively forwarded without requiring additional encapsulation and lookup overhead.

The nodes in dual-stack support both protocol stacks (IPv4 and IPv6), enabling them to be configured with both IPv4 and IPv6 addresses. The dual-stack nodes use IPv4 and IPv6 mechanisms such as Dynamic Host Configuration Protocol (DHCP) to acquire their respective configurations like addresses.

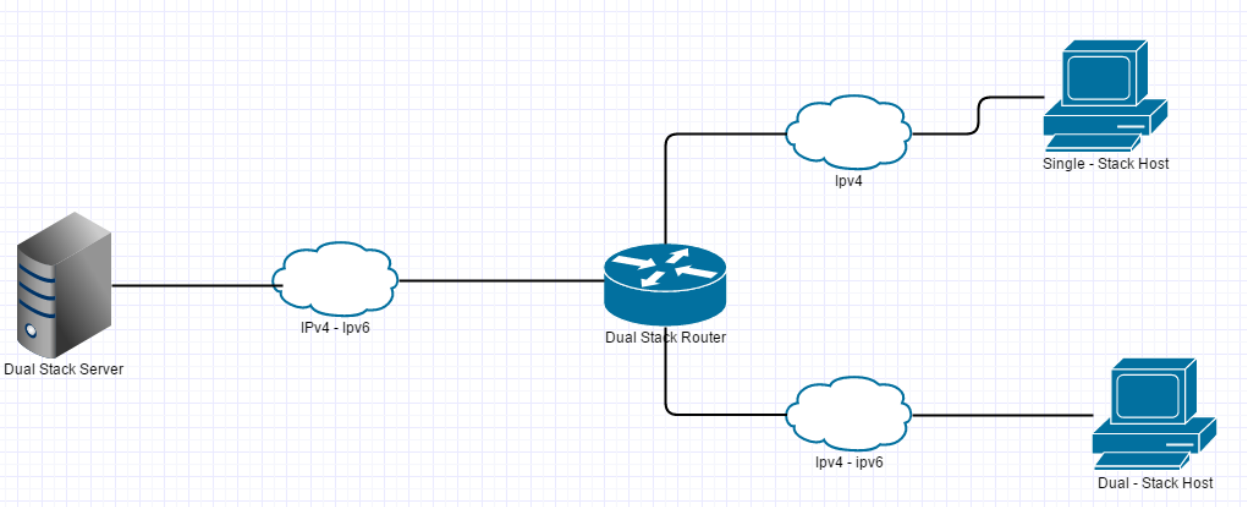


Figure 26: Dual Stack

4.3.2 Tunneling

IPv4 is the dominant IP protocol deployed in enterprise networks**.** As the adoption and deployment of IPv6 grow, IPv6 hosts or entire sections of the network might need to communicate over IPv6 end to end, but IPv4-only portions of the network are in the way. This is common in WAN/branch deployments, where the branch and head-end sites are IPv6-enabled, but the WAN in between supports only IPv4. IPv6-over-IPv4 tunnels encapsulate IPv6 datagrams inside of IPv4, enabling end-to-end communication.

Traditionally point-to-point or point-to-multipoint tunnels have been used to "carry" or transport one protocol, in this case IPv6, over another protocol (IPv4). Tunnels have been used for many years to support Novell IPX/SPX, AppleTalk, SNA, and others.
Figure 3-3 shows the basic framework for encapsulating IPv6 inside the IPv4 tunnel header.

There are a variety of tunnel types, and the different tunnel types are used for different purposes. Some of the uses of the tunnel types are based on the termination points (host or router), number of termination points, and in some cases, the operating system version. The tunnel types include

■ **Router-to-router:** In router-to-router tunneling, routers connected over IPv4 network infrastructure can transport IPv6 packets by encapsulating these IPv6 packets inside the IPv4 header.

■ **Host-to-router:** In host-to-router tunneling, the IPv4/IPv6 hosts can tunnel the IPv6 packets to an IPv4/IPv6 border router. This tunnel terminates at the border router and from there on is sent natively over IPv6 to the end host.

■ **Host-to-host:** In host-to-host tunneling, the tunnel exists between the two or more hosts. The IPv6/IPv4 hosts use the tunnel to communicate between themselves by tunneling IPv6 packets within the IPv4 header.

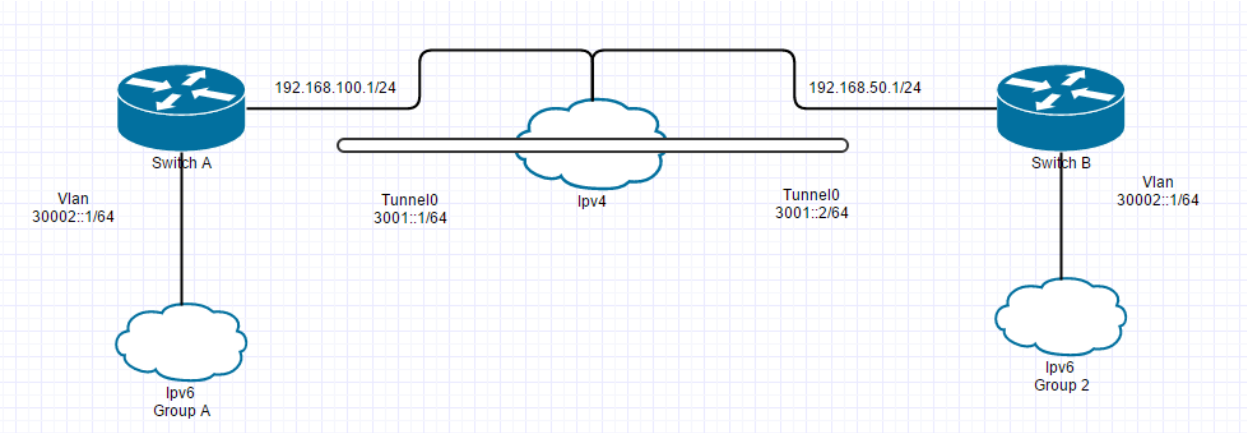


Figure 27: Tunneling

The last category for IPv6 transition process is tunneling as presented in Figure 21. This is used to transfer data between compatible networking nodes over incompatible networks. There are two ordinary scenarios to apply tunneling: the allowance of end systems to apply off link transition devices in a distributed network and the act of enabling edge devices in networks to inter-connect over 66 incompatible networks. Technically speaking, the tunneling technique utilizes a protocol whose function is to encapsulate the payload between two nodes or end systems. This encapsulation is carried out at the tunnel entrance and the payload will be de-capsulated at the tunnel exit. This process is known as the definition of tunnel. Therefore, the main issue in deploying tunnel is to configure tunnel endpoints, determine positions for applying encapsulation. Based on our research, this mechanism are generally attained via manual or tool-based parameter entry, existing services like DNS or DHCP, or by taking into use the embedment of information into IP addresses or applying an IPv6 anycast address.

4.3.3 Translation

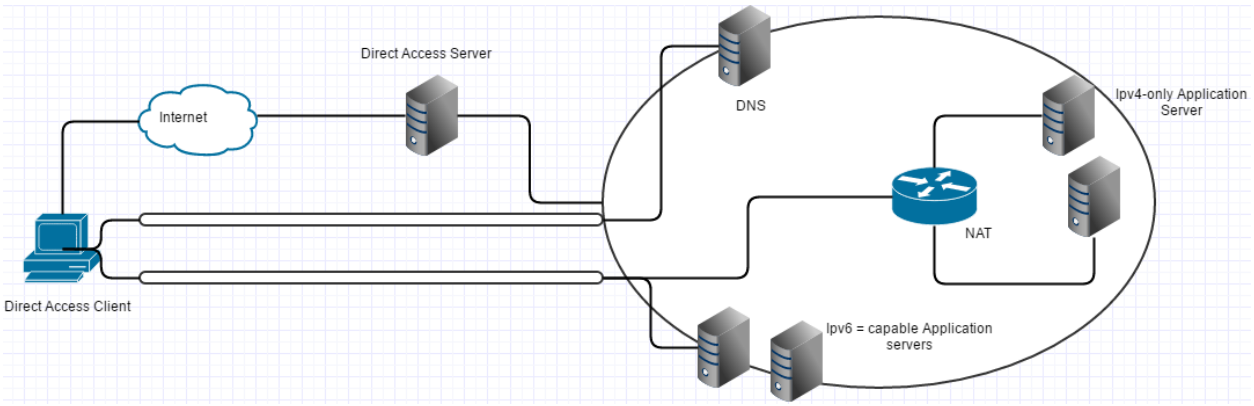


Figure 28: Translation

Together with IPv6 deployment strategies into large enterprise networks, the figure above indicates that there are various translation mechanisms (NAT, application level gateways (ALG)) also applied to enable the communication between IPv4-only applications and IPv6-only applications. The meaning of translation is to convert directly protocols from IPv4 to IPv6 or vice versa, which might result in transforming those two protocol headers and payload. This mechanism can be established at layers in protocol stack, consisting of network, transport, and application layers. The translation method has many mechanisms, which can be either stateless or stateful. While stateless means that the translator can perform every conversion separately with no reference to previous packets, stateful is the vice versa, which maintains some form of state in regard to previous packets.

The fundamental part of translation mechanism in transition process is the conversion of IP and ICMP packets. All translation methods, which are used to establish communication between IPv6-only and IPv4-only hosts, for instance, NAT-PT or BIS, apply an algorithm known as Stateless IP/ICMP Translator (SIIT). The function of this algorithm is to translate packet-by-packet the headers in the IP packet between IPv4 and IPv6, and also addresses in the headers among IPv4, IPv4-translated or IPv4-mapped IPv6 addresses.

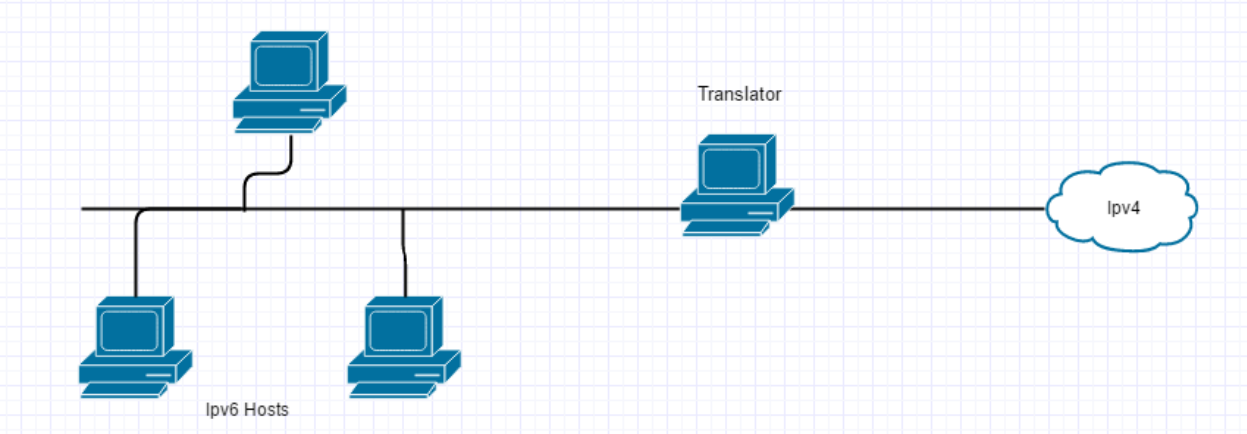


Figure 29: Translation

The above figure indicates an algorithm that designates a two-way translation between IPv4 and IPv6 packet headers or between ICMPv4 and ICMPv6 messages. The interpretation has been arranged so that UDP and TCP header checksums are not influenced during the process.

4.3.4 NAT Protocol Translation

This is another important method of transition to IPv6 by means of a NAT-PT (Network Address Translation – Protocol Translation) enabled device. With the help of a NAT-PT device, actual can take place happens between IPv4 and IPv6 packets and vice versa. See the diagram below:

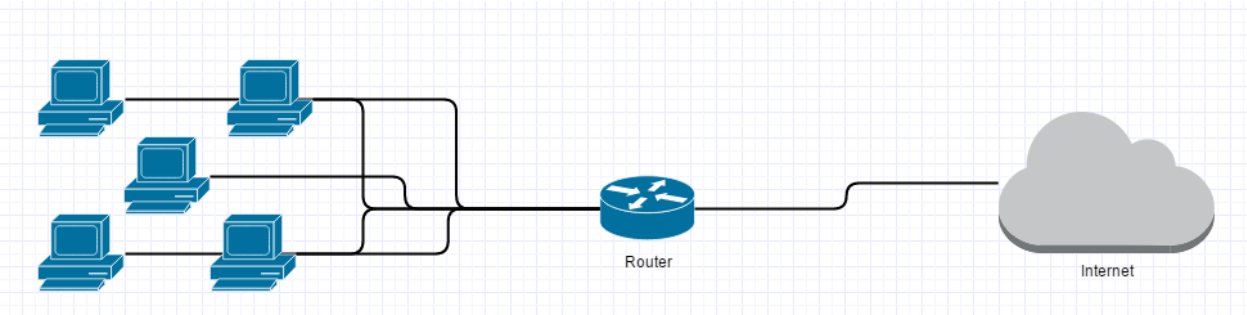


Figure 30: NAT

A host with IPv4 address sends a request to an IPv6 enabled server on Internet that does not understand IPv4 address. In this scenario, the NAT device can help them communicate. When the IPv4 host sends a request packet to the IPv6 server, the NAT device/router strips down the IPv4 packet, removes IPv4 header, and adds IPv6 header and passes it through the Internet. When a response from the IPv6 server comes for the IPv4 host, the router does vice versa.

4.4 Conclusion

In general, it is necessary for enterprises to thoroughly analyze and implement an IPv6 transition with clear instructions to serve expectations. However, because of the specific expectations may change from time to time, and they can be different by various enterprises, a complete approach with careful planning and preparation as listed in this part, accompanied by the details for each phase will allow the IPv6 implementation project to be achieved successfully, which will open a new path for each enterprise to be ready for the next generation of communication networks.

Chapter 5: Implementation and Testing

5.1 Capturing Packets

Start Wireshark by clicking on the Wireshark icon or type Wireshark in the command line. When Wireshark starts it launches the following screen and provides the following ways to capture packets:

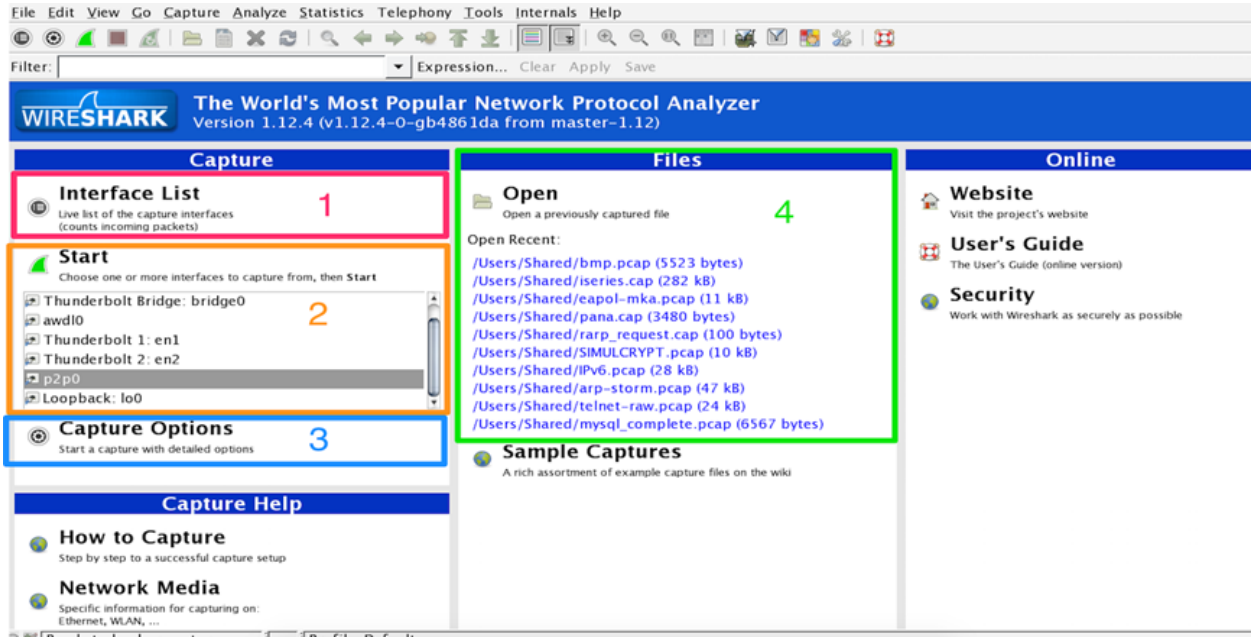


Figure 31: Capturing packets

The following table explains the various options that we have in the Start screen:

Table 1: Interfaces

Sr no	Wireshark capture options	What is this?
1	Interface List	Opens up a live list of capture interfaces, and counts the incoming/outgoing packets
2	Start	You can choose an interface from the list and start capturing packets
3	Capture Options	Provides various options for capturing and displaying packets
4	Open Recent	Wireshark displays recently used packets

5.2 Capturing Packets with interface list.

Click on **Interface List**; Wireshark will show a list of available network interfaces in the system and which one is active, by showing packets going in and out of the Interface, as shown in the following screenshot:

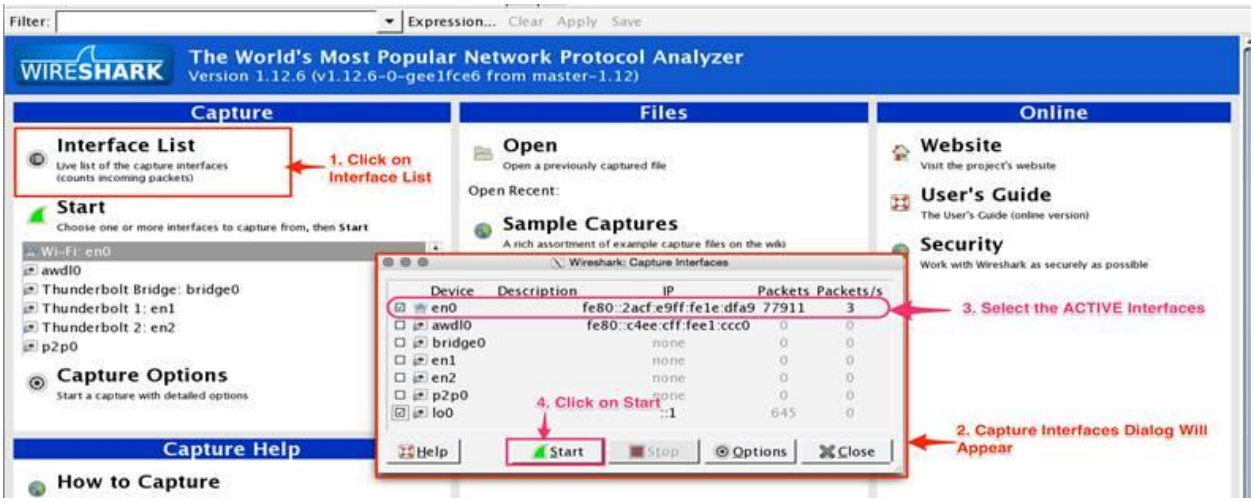


Figure 32: Selecting Interface to be captured

Choose the right (live) interfaces and click on the **Start** button to start capturing packets. If you want to capture packets on loopback (*127.0.0.1*), select the interface **lo0**.

5.3 Common interface names

The Interface name tells the network type; by looking at the name of interface the user should understand what network the capture setup is associated with—for example, *eth0* stands for *Ethernet*. A few of them are shown in the following diagram:

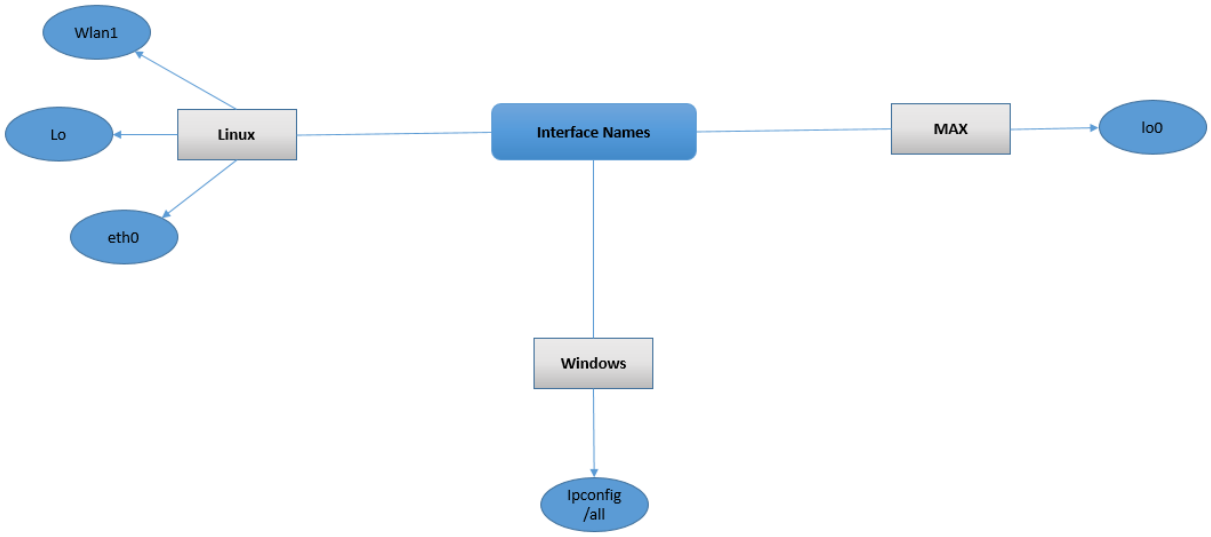
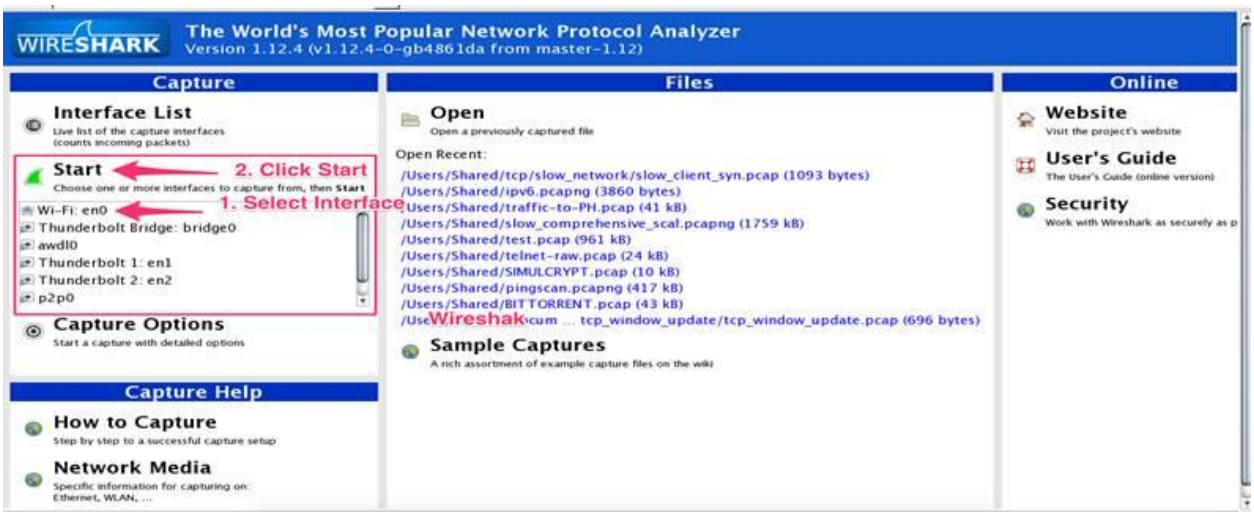


Figure 33: Interface Name on different OS

5.4 Capturing packets with Start options

In **Start** options, users can multiselect or select the interface displayed in the list and then click on **Start**. This doesn't give the flexibility to see on which interface the packets are active users can configure the capture options by double clicking on the interface or by clicking on **Capture Options**:



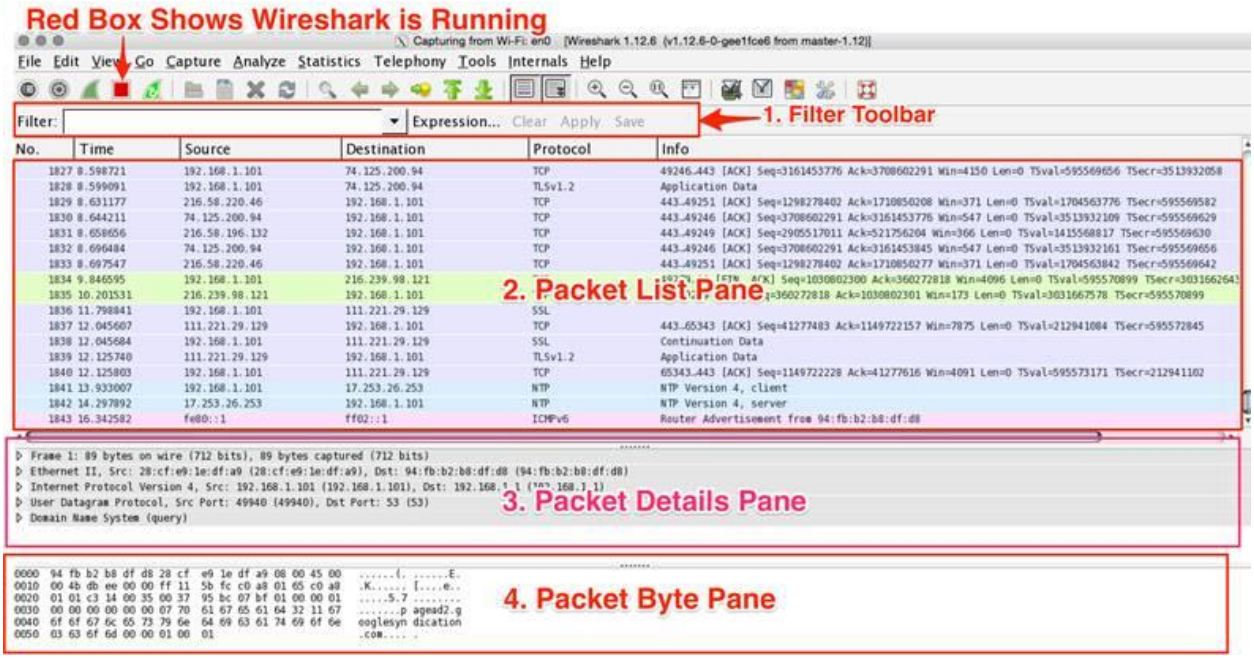
Wireshark provides the flexibility to configure packet that need to be captured with various capture options. To begin, try these basic settings:

1. Choose the live interface, where packets are going in and out.
2. Click on **Capture Options**, Wireshark will open the Capture Options dialog box.
3. Enable promiscuous mode, which will allow the network interface to receive all packets.
4. Check the snaplength size. This option will tell you the size of data for each frame that should be captured by Wireshark; this is useful when capturing the header frame or to keep the packet size small.
5. Name resolution tries to resolve the numerical address (for example, the MAC address, the IP address, and port) to its corresponding name, under the category where these options are defined.
6. Resolve MAC address by converting the MAC address to a human-readable format; for example *28:cf:e9:1e:df:a9* will translate to *192.168.1.101*.
7. Resolve network-layer names (IP name resolution) to convert the IP address to its corresponding hostname (for example, *216.58.220.46* will translate to *google.com*).
8. Resolve transport-layer names (TCP/UDP port name resolution) to convert well-known ports to human-readable format (for example, *443* will translate to *https*).

9. Use external the network name resolver to perform a reverse DNS lookup for each unique IP address (for example *216.58.196.14* will translate to *ns4.google.com*) also referred as reverse DNS lookup.

5.5 Wireshark user interface

Wireshark main window appears when Wireshark starts capturing a packet, or when a *.pcap* file is open for offline viewing. It looks similar to the following screenshot:



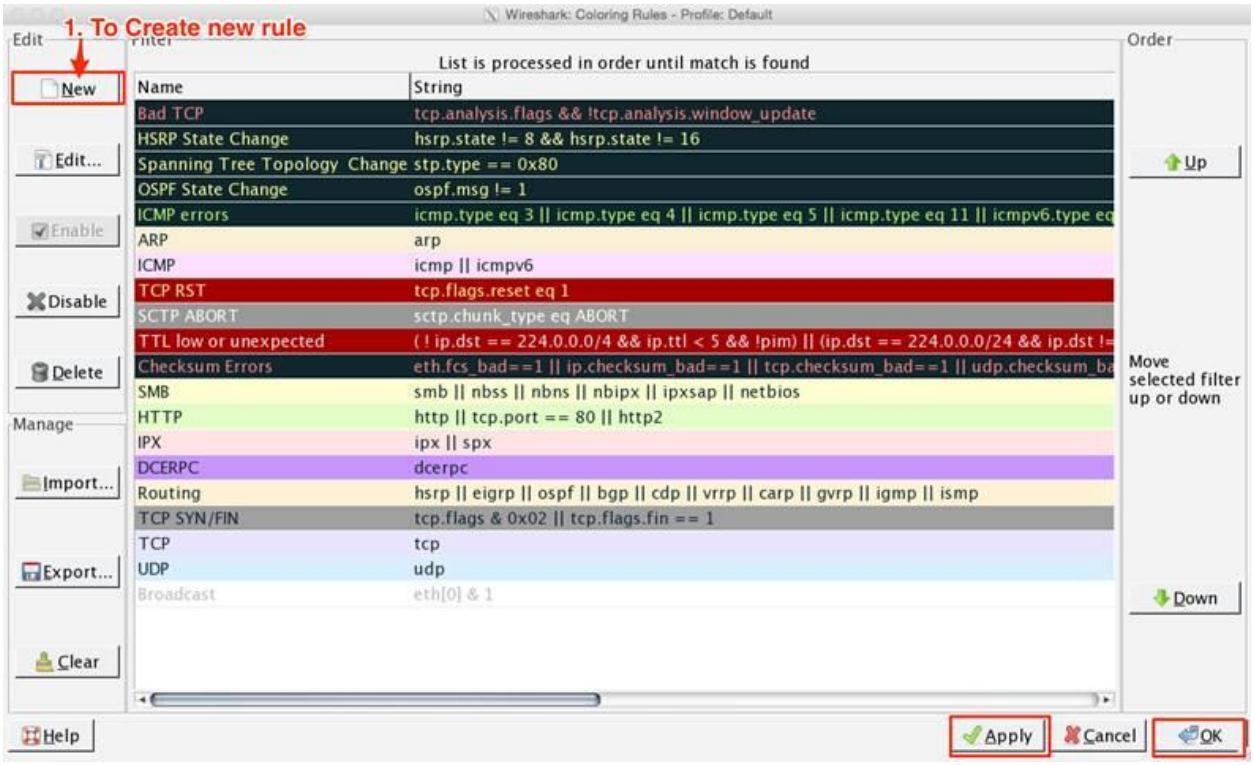
Wireshark UI interface consists of different panes and provides various options to the user for customizing it. In this article, we will cover these panes in detail:

Table 2: Colors

Item	What is it?
The red box	This shows that Wireshark is running and capturing a packet
1	This is the **Filter** toolbar, used for filtering packets based on the applied filter
2	This is the Packet List pane, which displays all the captured packets
3	This is the Packet Details pane, which shows the selected packet in a verbose form
4	This is the Packet Byte pane, which shows selected packet in a hex dump format

First, just observe pane 2 in the screen; the displayed packets appear with different colors. This is one of Wireshark best features; it colorizes packets according to the set filter and helps you visualize the packet looked for.

To manage (view, edit, or create) a coloring rule, go to View Coloring Rules. Wireshark will display the Coloring Rules dialog box as shown in the screenshot:



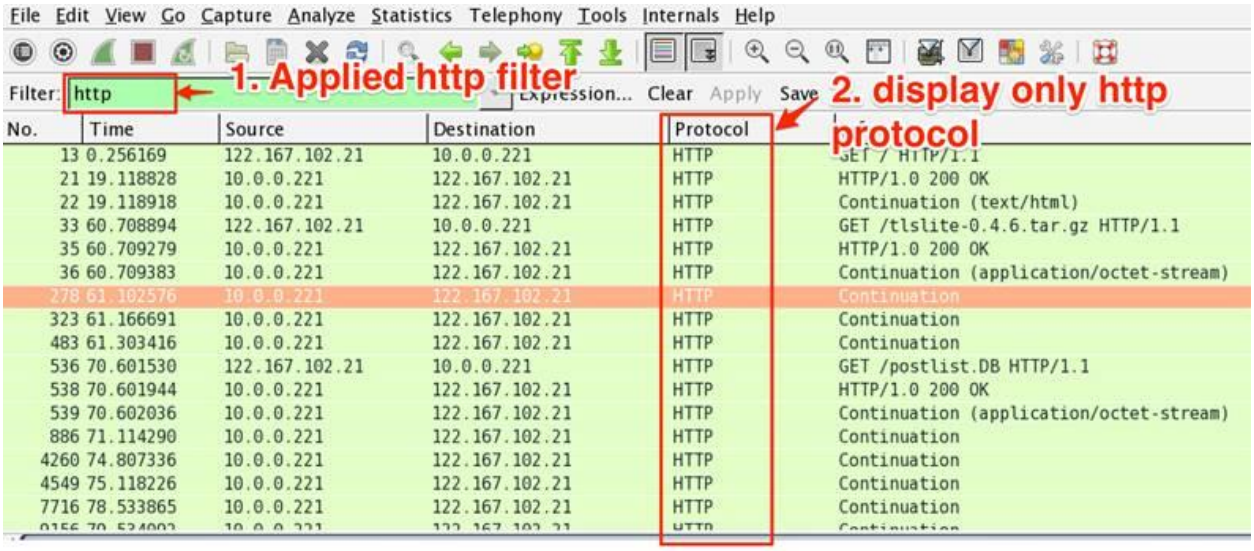
Users can create a new rule by clicking on the **New** button, choosing the filter name and filter string, and then applying a foreground and background color to it, to customize the packet with a specific color.

The Wireshark display filter displays packets with its available coloring options. Wireshark display filters are used to change the view of a capture file by providing the full dissection of all packets, which helps analyzing a network tracefile efficiently. For example, if a user is interested in only HTTP packets, the user can set the display filter to **http**, as shown in the following screenshot.

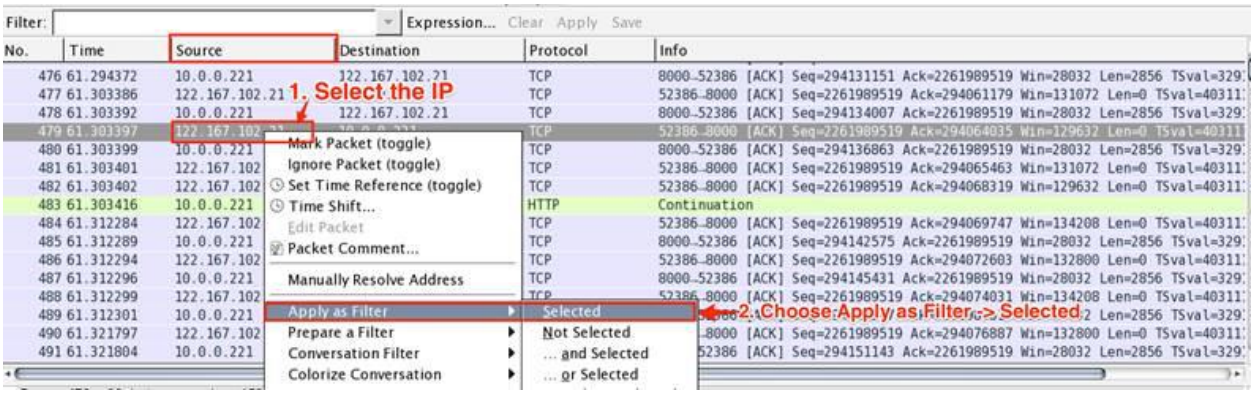
The steps to apply display filters are as follows:

1. Open the *http_01.pcap* file.
2. Type the *http* protocol in the filter area and click on **Apply**.

Once the filter is applied, the Packet List pane will display only HTTP protocol-related packets:



Wireshark display filter can be applied or prepared from the column displayed in the Packet List pane by selecting the column, then right-clicking and going to **Apply as Filter** | **Selected** (as shown in the following screenshot) to create the filter from the source IP address *122.167.102.21*:

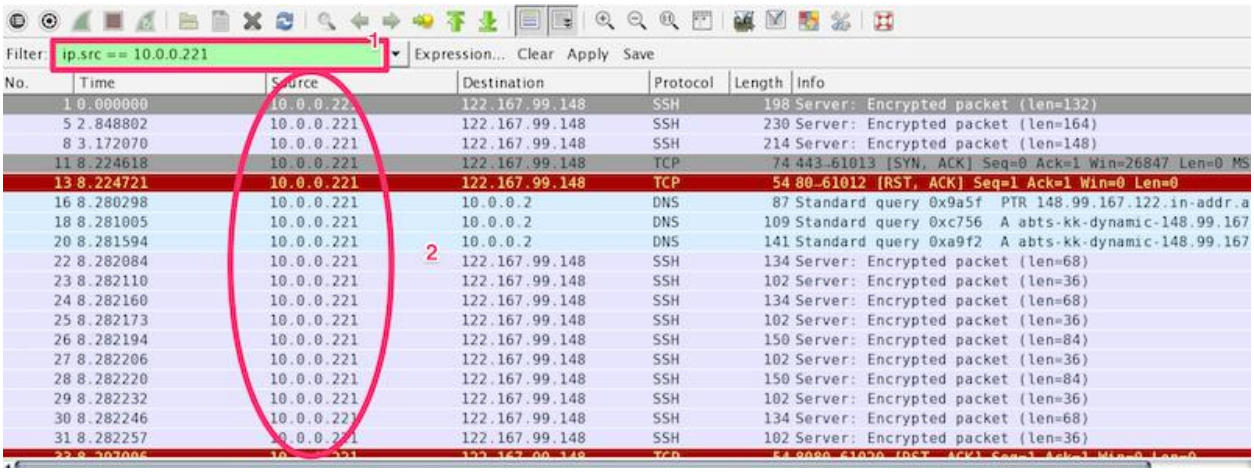


Wireshark gives the flexibility to apply filters from the Details pane; the steps remain the same.

Wireshark also provide the option to clear the filter. To do this click on **Clear** (available in the **Filter** toolbar) to display the entire captured packet.

5.6 Filtering techniques

Capturing/displaying packets properly will help you with packet captures. For example, to track a packet exchanged between two hosts: *HOSTA* (*10.0.0.221*) and *HOSTB* (*122.167.99.148*), open the *SampleCapture01.pcap* file and apply the filter *ip.src == 10.0.0.221* as shown:

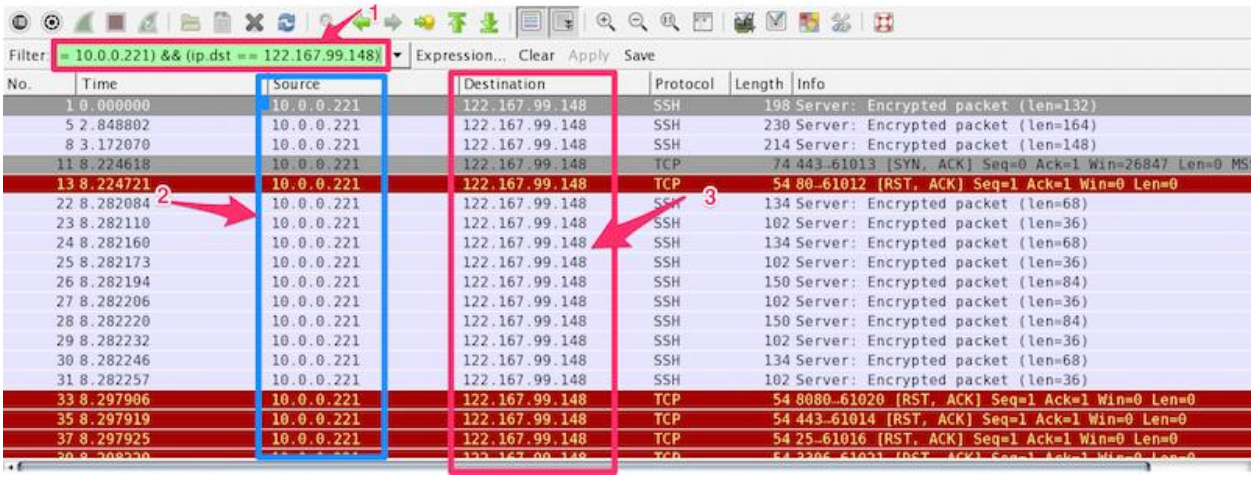


Highlighted sections:

Table 3: Highlighted Section 1

Item	Description
1	Apply filter *ip.src == 10.0.0.221.*
2	The Packet List pane displays the traffic from source to destination. The source shows the constant IP address *10.0.0.221.* There is no evidence as to which packet is sent from host *122.167.99.148* to host *10.0.0.221.*

Now modify the filter *(ip.src == 10.0.0.221) && (ip.dst == 122.167.99.148)* to *(ip.src == 10.0.0.221) or (ip.dst == 122.167.99.148)*. This will give the result shown in the following screenshot:



Highlighted sections:

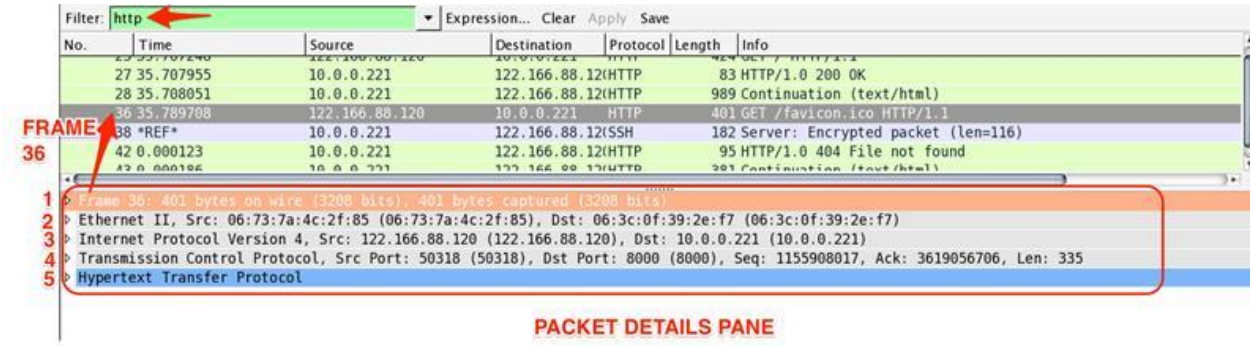
Table 4: Highlighted Sections 2

Item	Description
1	Applied filter *(ip.src == 10.0.0.221) && (ip.dst == 122.167.99.148)*
2	The source IP address (*10.0.0.221*) is not changed
3	The destination IP address (*122.167.99.148*) is not changed

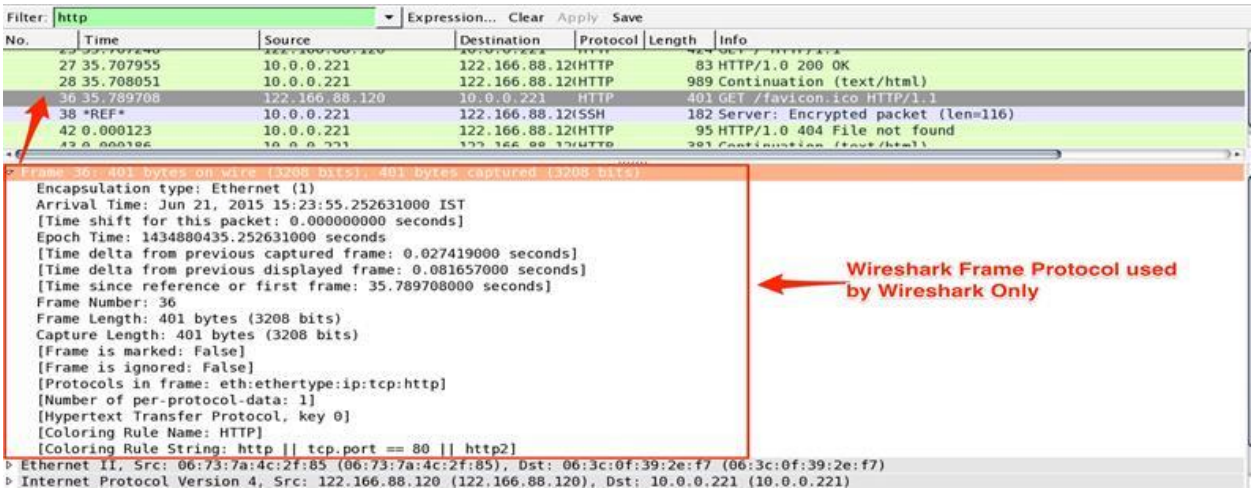
The same conversation captured by choosing the destination MAC address using the display filter *eth.addr == 06:73:7a:4c:2f:85*.

5.7 The Packet Details pane

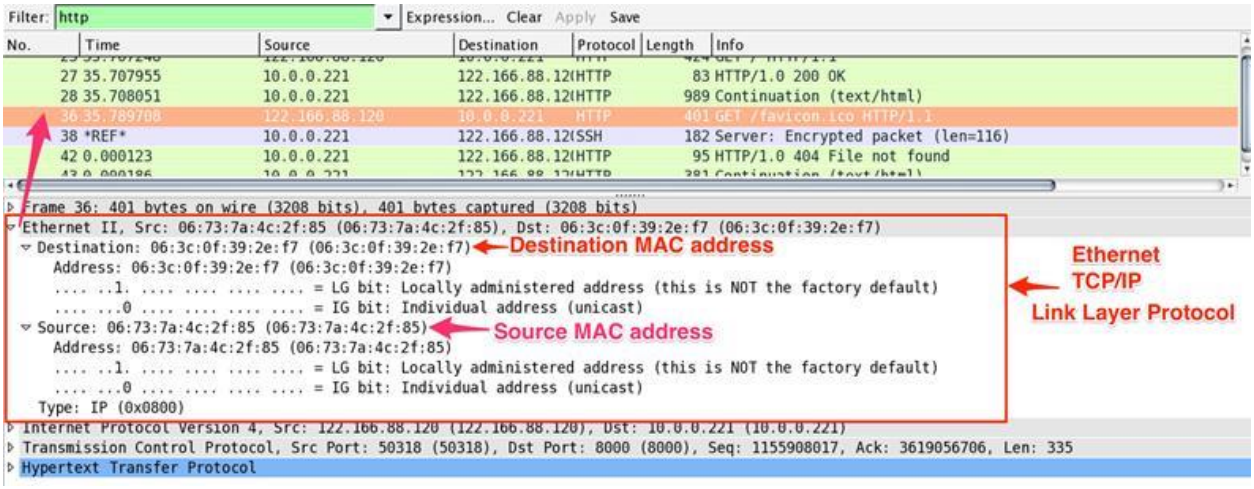
The Packet Details pane will show the currently selected packet in a more detailed form. It'll show the selected protocol in a more verbose form. In the following screenshot, an HTTP packet is selected and its details are shown in the detailed information labeled with number **1** to **5**. Let's see what these are:



The frame protocol is only used by Wireshark. All the TCP/IP protocols sits on top of this. The frame shows at what time the packet was captured, as shown in the following screenshot:



Ethernet is the link layer protocol, in the TCP/IP stack. It sends network packets from the sending host to one (Unicast) or more (Multicast/Broadcast) receiving hosts, as shown:



Useful filters in Ethernet are:

- *eth.dst == 06:3c:0f:39:2e:f7*: This shows packets to this MAC address only
- *eth.dst==ff:ff:ff:ff:ff:ff*: This shows broadcast traffic only

The packet structure of Ethernet frames is described in the following table:

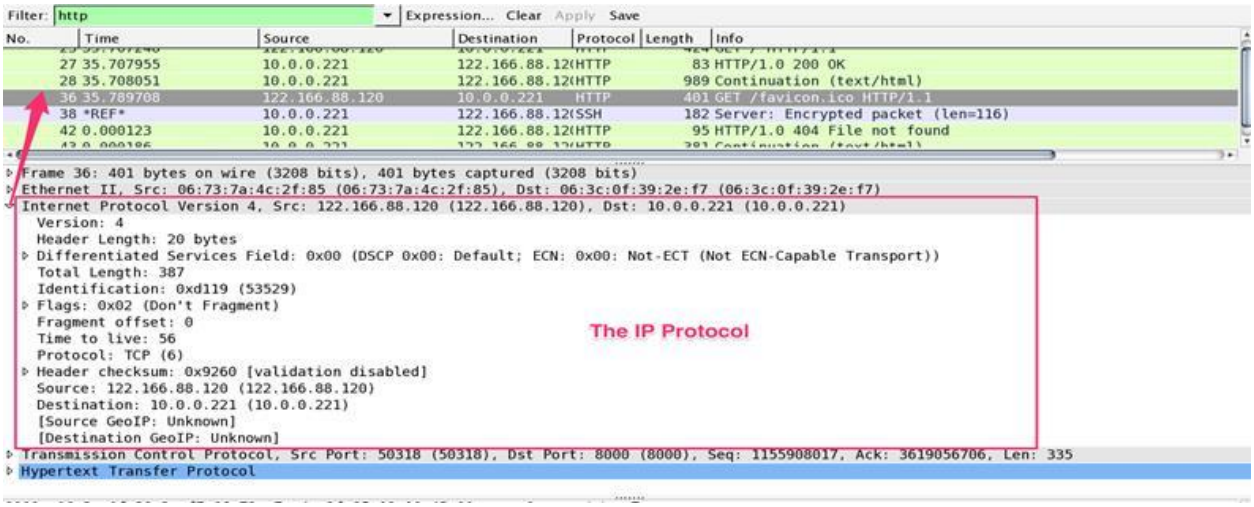
Table 5: Packet Structure of Ethernet Frames

Preamble	Destination MAC address	Source MAC address	Type/length	User-data	Frame check sequence (FCS)
8	6	6	2 0800 for IPv4 86DD for IPv6 0806 for ARP	46-1500	4

The preamble (8 bytes) and FCS (4 bytes) are not part of the frame and Wireshark will not capture this field.

So the total Ethernet header is 14 bytes—6 byte for the destination address, 6 byte for the source address, and 2 byte for the Ether Type.

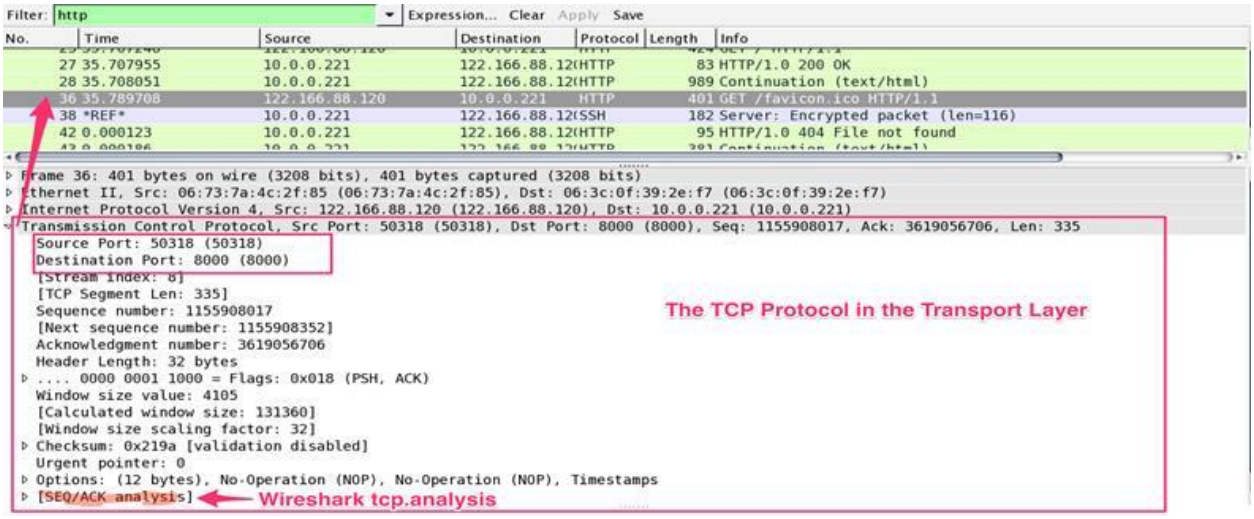
The Internet Protocol information relates to how the IP packet is delivered and whether it has used IPv4 or IPv6 to deliver the datagram packets.



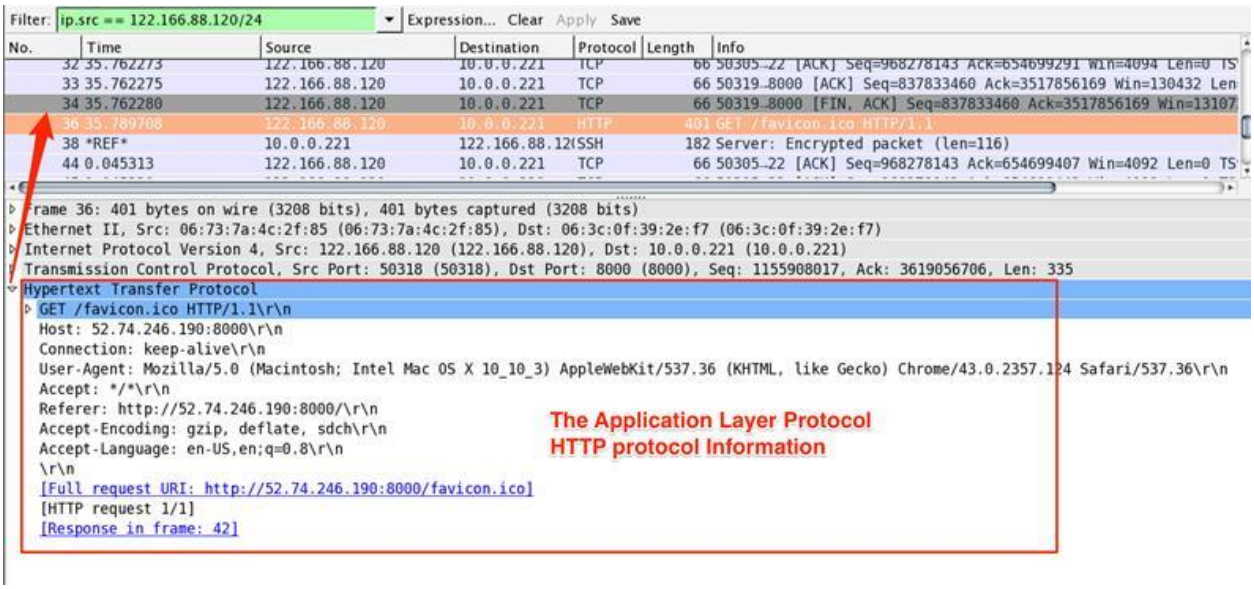
The preceding screenshots show that a IPv4 protocol is used to deliver the datagram packet. Useful display filters in the IP protocol are:

- *ip.src == 122.166.88.120/24* shows traffic from the subnet
- *ip.addr==122.166.88.120* shows traffic to or from the given host
- Host *122.166.88.120* captures/filters traffic from the host

The TCP protocol packet contains all TCP-related protocol data. If the communication is over UDP, the TCP will be replaced by the UDP, as shown in the following screenshot. The SEQ/ACK analysis will done by Wireshark based on the sequence number and expert info will be provided:



The APPLICATION-LAYER protocol is shown if the packet contains any application protocol. As shown in the following screenshot, the selected packet 36 has HTTP protocol data. Wireshark has the ability to decode the protocol based on the standard port and present this information in the Packet Details pane in a readable (RFC-defined) format.



5.8 The Packet Bytes pane

The Packet Bytes pane displays the bytes contained in the frame, with the highlighted area being set to the node selected in the Packet Details pane.

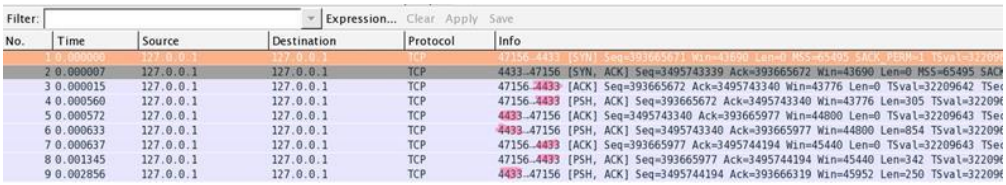
Wireshark features:

Wireshark is loaded with some awesome features. Let's go few with them not limited:

Decode-as

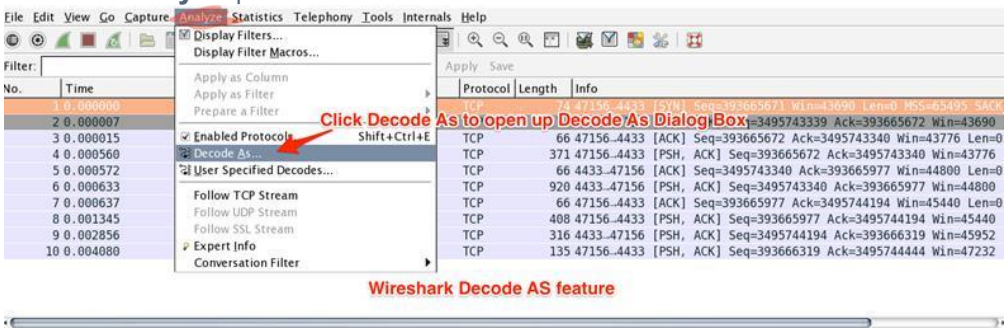
The decode-as feature allows Wireshark to decode the packet based on the selected protocol. Usually Wireshark will automatically identify and decode incoming packets based on the standard port—for example, port *443* will be decoded as SSL. For non-standard ports, the decode-as feature will decode the packet based on the protocol selected. This is a very useful feature. For example, if the SSL server runs on non-standard port *4433*.

Open the sample *https.pcap* file. HTTPS traffic is captured when the file is opened in Wireshark. It doesn't show SSL-related data; instead it just shows all TCP communications:

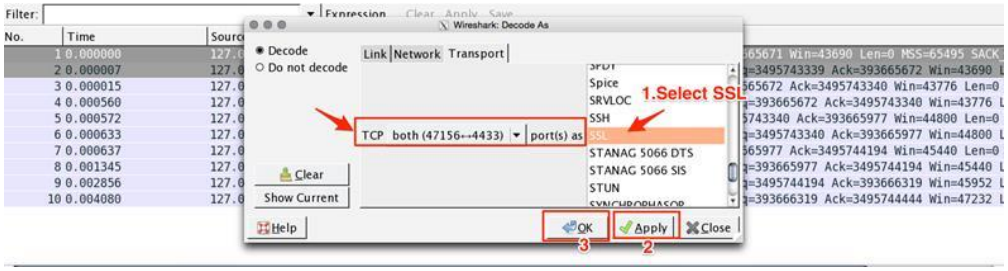


To decode this traffic as SSL, follow these steps:

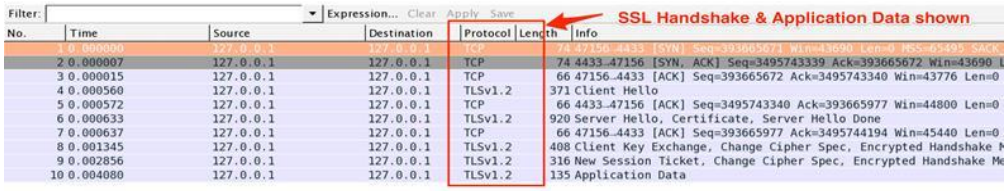
1. Click on **Analyze** | **Decode As**:



2. The **Decode As** popup will appear as shown in following screenshot. Choose the protocol (**SSL** in this example) that is required for decoding for the given traffic:



3. The SSL traffic protocol is shown in Wireshark:



5.9 Translation Ipv4 to Ipv6

This is part is very important where the translation mechanisms take place. The user can translate *Ipv4 to Ipv6 packets or Vice Versa* by using *.cap file*. I used the powerful c programming language the translation mechanism.

Here is the Code:

```
#include<netinet/in.h>

#include<errno.h>

#include<netdb.h>

#include<stdio.h> //For standard things

#include<stdlib.h>    //malloc

#include<string.h>    //strlen



#include<netinet/ip_icmp.h>   //Provides declarations for icmp header

#include<netinet/udp.h>   //Provides declarations for udp header

#include<netinet/tcp.h>   //Provides declarations for tcp header

#include<netinet/ip.h>    //Provides declarations for ip header

#include<netinet/if_ether.h>  //For ETH_P_ALL

#include<net/ethernet.h>  //For ether_header

#include<sys/socket.h>

#include<arpa/inet.h>

#include<sys/ioctl.h>

#include<sys/time.h>
```

```
#include<sys/types.h>

#include<unistd.h>



void ProcessPacket(unsigned char* , int);

void print_ip_header(unsigned char* , int);

void print_tcp_packet(unsigned char * , int );

void print_udp_packet(unsigned char * , int );

void print_icmp_packet(unsigned char* , int );

void PrintData (unsigned char* , int);

FILE *logfile;

struct sockaddr_in source,dest;

int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;



int main()

{

    int saddr_size , data_size;

    struct sockaddr saddr;
```

```
unsigned char *buffer = (unsigned char *) malloc(65536); //Its Big!


logfile=fopen("log.txt","w");
if(logfile==NULL)
{
   printf("Unable to create log.txt file.");
}
printf("Starting...\n");


int sock_raw = socket( AF_PACKET , SOCK_RAW , htons(ETH_P_ALL)) ;
//setsockopt(sock_raw , SOL_SOCKET , SO_BINDTODEVICE , "eth0" , strlen("eth0")+ 1 );


if(sock_raw < 0)
{
   //Print the error with proper message
   perror("Socket Error");
   return 1;
}
```

```
    while(1)
    {
        saddr_size = sizeof saddr;
        //Receive a packet
        data_size = recvfrom(sock_raw , buffer , 65536 , 0 , &saddr , (socklen_t*)&saddr_size);
        if(data_size <0 )
        {
            printf("Recvfrom error , failed to get packets\n");
            return 1;
        }
        //Now process the packet
        ProcessPacket(buffer , data_size);
    }
    close(sock_raw);
    printf("Finished");
    return 0;
}
```

```
void ProcessPacket(unsigned char* buffer, int size)
{
    //Get the IP Header part of this packet , excluding the ethernet header
    struct iphdr *iph = (struct iphdr*)(buffer + sizeof(struct ethhdr));
    ++total;
    switch (iph->protocol) //Check the Protocol and do accordingly...
    {
        case 1:  //ICMP Protocol
            ++icmp;
            print_icmp_packet( buffer , size);
            break;

        case 2:  //IGMP Protocol
            ++igmp;
            break;

        case 6:  //TCP Protocol
            ++tcp;
```

```
            print_tcp_packet(buffer , size);
            break;

        case 17: //UDP Protocol
            ++udp;
            print_udp_packet(buffer , size);
            break;

        default: //Some Other Protocol like ARP etc.
            ++others;
            break;
    }
    printf("TCP : %d   UDP : %d   ICMP : %d   IGMP : %d   Others : %d   Total : %d\r", tcp , udp , icmp , igmp , others , total);
}

void print_ethernet_header(unsigned char* Buffer, int Size)
{
    struct ethhdr *eth = (struct ethhdr *)Buffer;
```

```
    fprintf(logfile , "\n");

    fprintf(logfile , "Ethernet Header\n");

    fprintf(logfile , "   |-Destination Address : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth->h_dest[0] , eth->h_dest[1] , eth->h_dest[2] , eth->h_dest[3] , eth->h_dest[4] , eth->h_dest[5] );

    fprintf(logfile , "   |-Source Address      : %.2X-%.2X-%.2X-%.2X-%.2X-%.2X \n", eth->h_source[0] , eth->h_source[1] , eth->h_source[2] , eth->h_source[3] , eth->h_source[4] , eth->h_source[5] );

    fprintf(logfile , "   |-Protocol            : %u \n",(unsigned short)eth->h_proto);

}



void print_ip_header(unsigned char* Buffer, int Size)

{

    print_ethernet_header(Buffer , Size);



    unsigned short iphdrlen;



    struct iphdr *iph = (struct iphdr *)(Buffer  + sizeof(struct ethhdr) );

    iphdrlen =iph->ihl*4;
```

```
memset(&source, 0, sizeof(source));

source.sin_addr.s_addr = iph->saddr;


memset(&dest, 0, sizeof(dest));

dest.sin_addr.s_addr = iph->daddr;


fprintf(logfile , "\n");

fprintf(logfile , "IP Header\n");

fprintf(logfile , "   |-IP Version        : %d\n",(unsigned int)iph->version);

fprintf(logfile , "   |-IP Header Length  : %d DWORDS or %d Bytes\n",(unsigned int)iph->ihl,((unsigned int)(iph->ihl))*4);

fprintf(logfile , "   |-Type Of Service   : %d\n",(unsigned int)iph->tos);

fprintf(logfile , "   |-IP Total Length   : %d  Bytes(Size of Packet)\n",ntohs(iph->tot_len));

fprintf(logfile , "   |-Identification    : %d\n",ntohs(iph->id));

//fprintf(logfile , "   |-Reserved ZERO Field   : %d\n",(unsigned int)iphdr->ip_reserved_zero);

//fprintf(logfile , "   |-Dont Fragment Field   : %d\n",(unsigned int)iphdr->ip_dont_fragment);

//fprintf(logfile , "   |-More Fragment Field   : %d\n",(unsigned int)iphdr->ip_more_fragment);
```

```
    fprintf(logfile , "   |-TTL      : %d\n",(unsigned int)iph->ttl);

    fprintf(logfile , "   |-Protocol : %d\n",(unsigned int)iph->protocol);

    fprintf(logfile , "   |-Checksum : %d\n",ntohs(iph->check));

    fprintf(logfile , "   |-Source IP        : %s\n",inet_ntoa(source.sin_addr));

    fprintf(logfile , "   |-Destination IP   : %s\n",inet_ntoa(dest.sin_addr));

}


void print_tcp_packet(unsigned char* Buffer, int Size)

{

    unsigned short iphdrlen;


    struct iphdr *iph = (struct iphdr *)( Buffer  + sizeof(struct ethhdr) );

    iphdrlen = iph->ihl*4;


    struct tcphdr *tcph=(struct tcphdr*)(Buffer + iphdrlen + sizeof(struct ethhdr));


    int header_size =  sizeof(struct ethhdr) + iphdrlen + tcph->doff*4;
```

```
fprintf(logfile , "\n\n***********************TCP Packet*************************\n");


print_ip_header(Buffer,Size);


fprintf(logfile , "\n");

fprintf(logfile , "TCP Header\n");

fprintf(logfile , "   |-Source Port      : %u\n",ntohs(tcph->source));

fprintf(logfile , "   |-Destination Port : %u\n",ntohs(tcph->dest));

fprintf(logfile , "   |-Sequence Number    : %u\n",ntohl(tcph->seq));

fprintf(logfile , "   |-Acknowledge Number : %u\n",ntohl(tcph->ack_seq));

fprintf(logfile , "   |-Header Length      : %d DWORDS or %d BYTES\n" ,(unsigned int)tcph->doff,(unsigned int)tcph->doff*4);

//fprintf(logfile , "   |-CWR Flag : %d\n",(unsigned int)tcph->cwr);

//fprintf(logfile , "   |-ECN Flag : %d\n",(unsigned int)tcph->ece);

fprintf(logfile , "   |-Urgent Flag          : %d\n",(unsigned int)tcph->urg);

fprintf(logfile , "   |-Acknowledgement Flag : %d\n",(unsigned int)tcph->ack);

fprintf(logfile , "   |-Push Flag            : %d\n",(unsigned int)tcph->psh);

fprintf(logfile , "   |-Reset Flag           : %d\n",(unsigned int)tcph->rst);
```

```
fprintf(logfile , "   |-Synchronise Flag     : %d\n",(unsigned int)tcph->syn);

fprintf(logfile , "   |-Finish Flag          : %d\n",(unsigned int)tcph->fin);

fprintf(logfile , "   |-Window         : %d\n",ntohs(tcph->window));

fprintf(logfile , "   |-Checksum       : %d\n",ntohs(tcph->check));

fprintf(logfile , "   |-Urgent Pointer : %d\n",tcph->urg_ptr);

fprintf(logfile , "\n");

fprintf(logfile , "                    DATA Dump                    ");

fprintf(logfile , "\n");



fprintf(logfile , "IP Header\n");

PrintData(Buffer,iphdrlen);



fprintf(logfile , "TCP Header\n");

PrintData(Buffer+iphdrlen,tcph->doff*4);



fprintf(logfile , "Data Payload\n");

PrintData(Buffer + header_size , Size - header_size );
```

```
    fprintf(logfile , "\n###########################################################");
}

void print_udp_packet(unsigned char *Buffer , int Size)
{

    unsigned short iphdrlen;

    struct iphdr *iph = (struct iphdr *)(Buffer +  sizeof(struct ethhdr));
    iphdrlen = iph->ihl*4;

    struct udphdr *udph = (struct udphdr*)(Buffer + iphdrlen  + sizeof(struct ethhdr));

    int header_size =  sizeof(struct ethhdr) + iphdrlen + sizeof udph;

    fprintf(logfile , "\n\n***********************UDP
Packet*************************\n");

    print_ip_header(Buffer,Size);
```

```
fprintf(logfile , "\nUDP Header\n");

fprintf(logfile , "   |-Source Port      : %d\n" , ntohs(udph->source));

fprintf(logfile , "   |-Destination Port : %d\n" , ntohs(udph->dest));

fprintf(logfile , "   |-UDP Length       : %d\n" , ntohs(udph->len));

fprintf(logfile , "   |-UDP Checksum     : %d\n" , ntohs(udph->check));


fprintf(logfile , "\n");

fprintf(logfile , "IP Header\n");

PrintData(Buffer , iphdrlen);


fprintf(logfile , "UDP Header\n");

PrintData(Buffer+iphdrlen , sizeof udph);


fprintf(logfile , "Data Payload\n");


//Move the pointer ahead and reduce the size of string

PrintData(Buffer + header_size , Size - header_size);
```

```
    fprintf(logfile , "\n###########################################################################");

}


void print_icmp_packet(unsigned char* Buffer , int Size)

{

    unsigned short iphdrlen;


    struct iphdr *iph = (struct iphdr *)(Buffer  + sizeof(struct ethhdr));

    iphdrlen = iph->ihl * 4;


    struct icmphdr *icmph = (struct icmphdr *)(Buffer + iphdrlen  + sizeof(struct ethhdr));


    int header_size =  sizeof(struct ethhdr) + iphdrlen + sizeof icmph;


    fprintf(logfile , "\n\n**********************ICMP
Packet**************************\n");


    print_ip_header(Buffer , Size);
```

```
fprintf(logfile , "\n");


fprintf(logfile , "ICMP Header\n");

fprintf(logfile , "   |-Type : %d",(unsigned int)(icmph->type));


if((unsigned int)(icmph->type) == 11)

{

   fprintf(logfile , "  (TTL Expired)\n");

}

else if((unsigned int)(icmph->type) == ICMP_ECHOREPLY)

{

   fprintf(logfile , "  (ICMP Echo Reply)\n");

}


fprintf(logfile , "   |-Code : %d\n",(unsigned int)(icmph->code));

fprintf(logfile , "   |-Checksum : %d\n",ntohs(icmph->checksum));

//fprintf(logfile , "   |-ID       : %d\n",ntohs(icmph->id));
```

```
    //fprintf(logfile , "   |-Sequence : %d\n",ntohs(icmph->sequence));
    fprintf(logfile , "\n");

    fprintf(logfile , "IP Header\n");
    PrintData(Buffer,iphdrlen);

    fprintf(logfile , "UDP Header\n");
    PrintData(Buffer + iphdrlen , sizeof icmph);

    fprintf(logfile , "Data Payload\n");

    //Move the pointer ahead and reduce the size of string
    PrintData(Buffer + header_size , (Size - header_size) );

    fprintf(logfile , "\n####");
}

void PrintData (unsigned char* data , int Size)
```

```
{
    int i , j;
    for(i=0 ; i < Size ; i++)
    {
        if( i!=0 && i%16==0)   //if one line of hex printing is complete...
        {
            fprintf(logfile , "         ");
            for(j=i-16 ; j<i ; j++)
            {
                if(data[j]>=32 && data[j]<=128)
                    fprintf(logfile , "%c",(unsigned char)data[j]); //if its a number or alphabet

                else fprintf(logfile , "."); //otherwise print a dot
            }
            fprintf(logfile , "\n");
        }

        if(i%16==0) fprintf(logfile , "   ");
```

```
    fprintf(logfile , " %02X",(unsigned int)data[i]);


if( i==Size-1)  //print the last spaces
{
    for(j=0;j<15-i%16;j++)
    {
      fprintf(logfile , "   "); //extra spaces
    }

    fprintf(logfile , "         ");

    for(j=i-i%16 ; j<=i ; j++)
    {
       if(data[j]>=32 && data[j]<=128)
       {
         fprintf(logfile , "%c",(unsigned char)data[j]);
       }
       else
```

```
    {
      fprintf(logfile , ".");
    }
}

fprintf(logfile ,  "\n" );
```

Applications

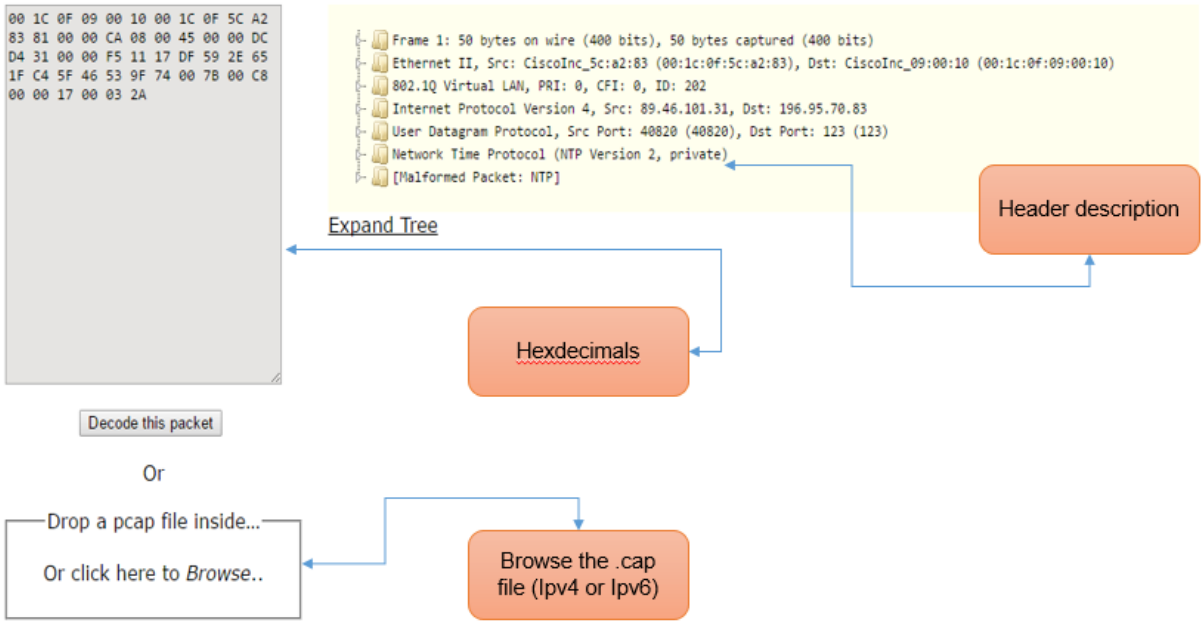
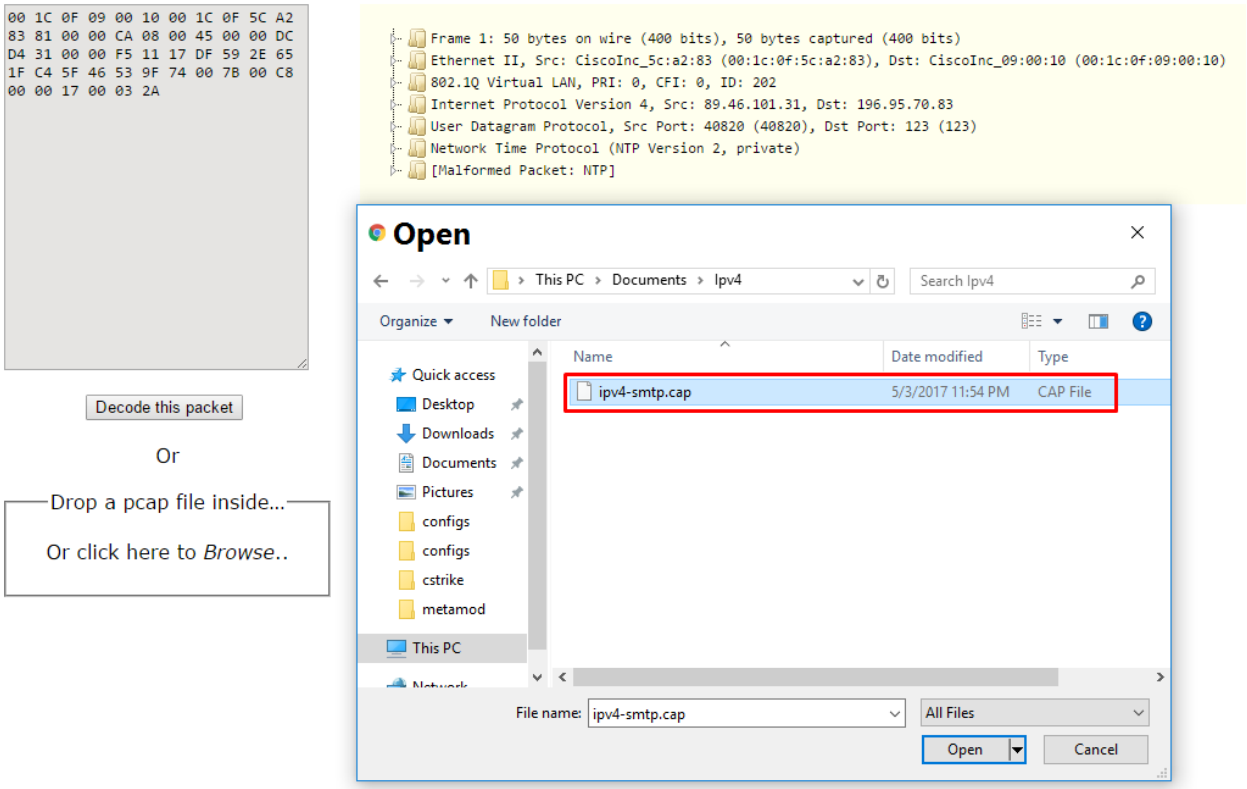
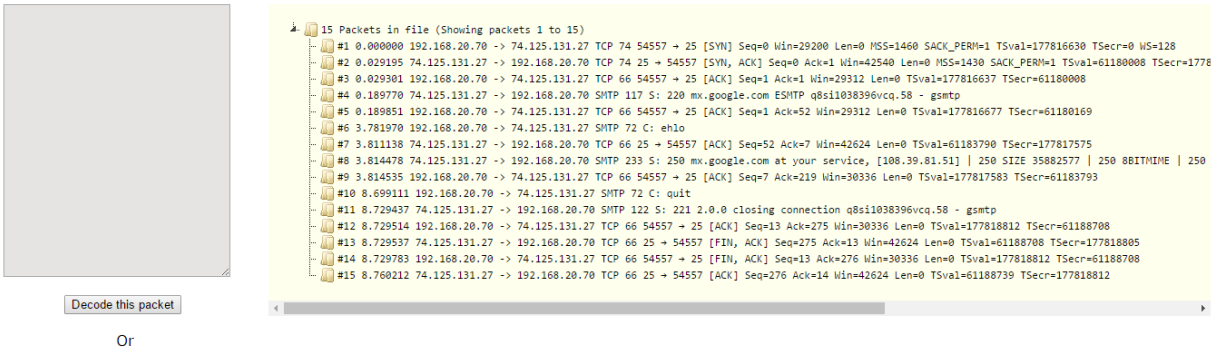


Figure 34: Translation mechanism

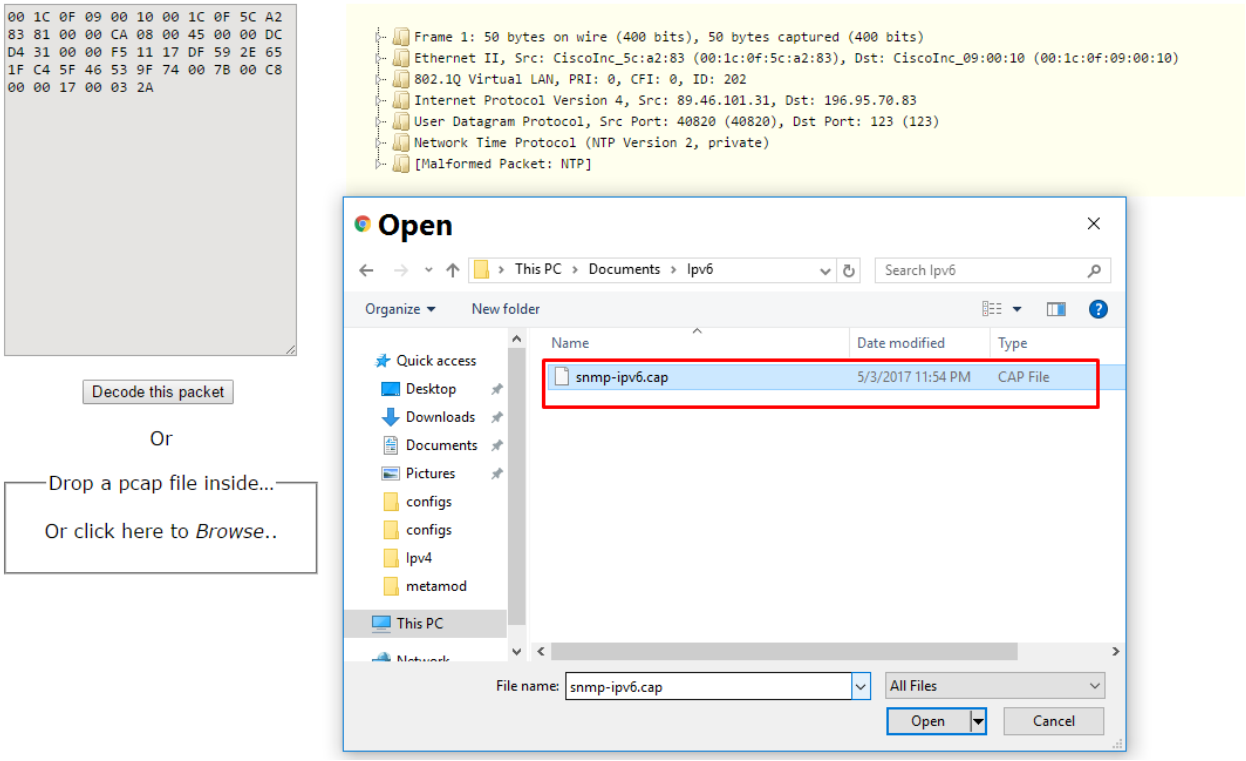
5.9.1 Browse an Ipv4 .cap file



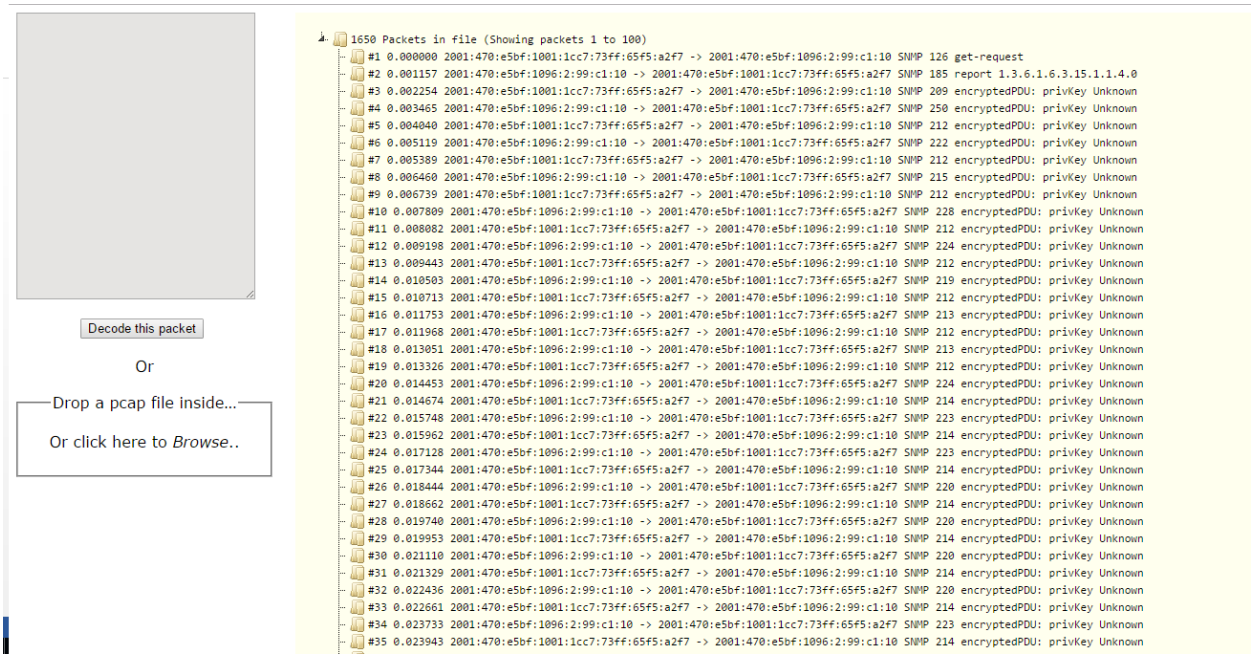
Results:



5.9.2 Browse an Ipv6 .cap file



Results:



5.10 Converting from IPv4 to IPv6

Firstly before starting I will assume everyone knows the following:

- **Binary** is a Base-2 numbering system, as it has only 0,1
- **Decimal** is a Base-10 numbering system, as it has 0,1,2,3,4,5,6,7,8,9
- **Hexadecimal** is a Base-16 numbering system, as it has 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

I also assume you know the hex values in decimal:

A	=	**10**
B	=	**11**
C	=	**12**
D	=	**13**
E	=	**14**
F	=	**15**

Two more things I would like to mention before explaining the conversion.

An **IPv4 address**: example 192.168.99.1

- Each Octet (8 bits) “between the dot-thing” denote 1 byte

An **IPv6 address**: example 2001:0db8:85a3:0000:0000:8a2e:0370:7334

- Two Tuples (1 Tuple = 4 bits = 1 Hex character) denotes 1 byte

Then converting is easy. Let’s take the following IPv4 address: 192.168.99.1 and convert it to Hex.

Step1

Divide the first octet (192) by 16 (since Hex is a Base-16)
IE : 192/16 = 12 times exactly with 0 left over
– 12 in Hex is represented as C
– 0 (zero) in Hex is, you guessed it, 0
Thus 192 in HEX is C0

Step2

Repeat step 1 with the second octet (168),
IE : 168/16 = 10 times with 8 left over because 10*6 = 160,
– 10 in HEX is A
– 8 in HEX is 8
Thus 168 in HEX is A8

Step3

Repetition rules!!! Third octet (99)
IE : 99/16 = 6 times with 3 left over
– 6 in HEX is 6
– 3 in HEX is 3
Thus 99 in HEX is 63

Step4

Last octet
IE : 1/16 = 0 times with 1 left over
– 0 in HEX is, yeah it is 0
– 1 in HEX is 1

Thus 1 in **HEX** is 01

So the IPv4 address of 192.168.99.1, represented in the IPv6 address portion would be C0A8:6301**.**
So when using IPv6 6-to-4 Tunnels, on the one endpoint of the tunnel, with the IPv4 address of 192.168.99.1, the complete IPv6 address would be 2002:C0A8:6301::1/64

5. 11 Converting back from IPv6 to IPv4

Now to convert the same portion of the IPv6address 2002:C0A8:6301::1/64 back to IPv4, the reverse method would apply.

Let me point one more thing about Base-16 out to understand why I'm doing what I am below:

$16^0 = 1$

$16^1 = 16$

Taking the portion C0A8:6301, first divide the address into 2 Tuple-groupings (2 Hex Characters) = C0 A8 63 01

Step1

Take C0 and multiply the first character 'C' by 16 and the second character '0' by 1.
Add the two decimal values together to get the IPv4 decimal value
IE: **((C=12)*16) + (0*1) = 192**

Step2

Repeat the same process with A8,
IE: **((A=10)*16) + (8*1) = 168**

Step3

Repeat the same process with 63,
IE: **(6*16) + (3*1) = 99**

Step4

Repeat the same process with 01,

IE: **(0*16) + (1*1) = 1**

This will give you an **IPv4 address of 192.168.99.1**

Chapter 6: Conclusions and Future Work

6.1 Conclusion

Based on the above overview of all mechanisms and current practices in researched enterprise networks, nearly all deployments of IPv6 in enterprise networks apply dual stack mechanism as it gives us a way to know more about IPv6 as well as to improve practical experience with a new address family, which plays an important role in the success of transition implementation. Therefore, in this thesis, we choose the dual stack mechanism to build a model for large enterprise networks.

6.2 Summary of three methods

Below is the summary table containing the advantages and disadvantages for three main methods above.

	Advantages	Disadvantages
Tunneling	- Configure tunnel endpoints only - Simple deployment - No additional management	- Face another problem of NATs - Take more time and CPU power - Harder to troubleshooting and network management - Have single points of failure - Vulnerable to security attacks
Translation	- The router is used as a translation communicator - Solve network interoperability problems	- Limitations similar to IPv4 NAT - Reduction in the overall value and utility of the network. - Harder to control on a larger scale - Complexity increases in IP addresses
Dual stack	- Easy to implement - Low cost - Greatest flexibility - Already supported in all OSs and devices	- Two routing tables - Additional memory and CPU power - Two firewall sets of policies

6.3 Future Work

Everyone one in the modern business world uses Internet and its rapid growth is the main reason being the importance of Internet Protocol address. Every device must have a unique IP address connect to the Internet in order to exchange data with another. Everyone uses IP addresses but only few understand them as they cannot be seen and are highly intangible. There is a great change happening as people are moving to adopt Ipv6 and providers are trying to save their customers by moving to Ipv6. If providers do not do so, they will soon be out of business as the Ipv4 will be no longer in use. Providers should understand ipv6 and ensure they are prepared for the future, ipv4 and ipv6 are not compatible and devices using one or the other cannot communicate directly. Either providers should upgrade their devices or change to ipv6 by using transition techniques. One of the popular techniques is dual stacking as it supports both Ipv4 and ipv6. Ipv6 is the only long-term solution to the depletion of ipv4 address pool. The depletion ipv4 address pool affects routing security. While distributing ipv4 addresses there was always pressure to preserve the free pool of ipv4by making smaller allotment and aggregation. The astounding amount of available ipv6 address means that conservation is no longer pressing and large piles of blocks can be given out. Without ipv6 there will be limitation is use of technology in the coming years such as smart phones, which are gaining popularity in the recent years. With the increase of population and users of Internet, it is hard to settle with Ipv4. Every Internet Provider ad company is moving to ipv6. Most of the countries, like China, The USA and India etc... Have already moved their services to IPV6. For the continuous growth of INTERNET and spreading through the world deployment of Ipv6 is necessary.

6.3 Limitation and Further Study

The scope of this thesis mainly discusses the most applicable transition method from IPv6 to IPv4 for enterprises with large network. The presentation of the method includes literature review, advantages, and configurations as well as simplified model of the method. As this thesis aims at large network, large enterprises with big network traffic may find it more useful than small and medium sized network. There are some transition procedures which may not be suitable for small and medium sized networks due to their complexity. Therefore, this thesis is most applicable and limited to large network. Currently all routers are working on ipv4 protocol, in the future work all companies around the world should work on the firmware to be able to implement both Ipv4 and Ipv6 on the same Router (I.e Cisco, TP-link, Link-6,etc..).

Appendix A

Installing Ubuntu on VMWare Workstation

1. Download the Ubuntu ISO image from the Ubuntu official website.
2. Create a new virtual machine and configure it using the *New Virtual Machine Wizard*.
3. Open the virtual machine settings editor (**VM** > **Settings**).
4. Select **Use ISO Image** and enter the path and filename for the image file or click **Browse** to navigate to the file.
5. Click **OK** to save the configuration and close the virtual machine settings editor.
6. Power on the virtual machine to start installing Ubuntu.
7. The installer is done automatically. When the installer is finished, the virtual machine is up and running.

Appendix B

How to setup Wirehshark Tool in Linux

Step 1. Add the offical PPA
sudo add-apt-repository ppa:wireshark-dev/stable

Step 2. update the repositoy
sudo apt-get update

Step 3. Install Wireshark 2.2.3
sudo apt-get install wireshark

During the installation,it will require to confirm security about allowing non-superuser to execute Wireshark. Just confirm YES if you want to. If you check on NO, you must run Wireshark with sudo. Later, if you want to change this,

Step 4.
Sudo dpkg-reconfigure wireshark-common

Appendix C

How to enable Ipv4 and Ipv6 on both machine.

Step 1. : Turn on networking in this file.

#Cat /etc/sysconfig/network

Step 2. Set default IPv6 router IP and server IP address in this file.

#Cat /etc/sysconfig/network-scripts/ifcfg-eth0

Step 3. Open /etc/sysconfig/network file, enter:

#Nano or / Vi/etc/sysconfig.network

Append following line:

NETWORKING_IPV6=yes

Step 4.Open /etc/sysconfig/network-scripts/ifcfg-eth0 (1st network config file)

Append following config directives for Ipv6

```
IPV6INIT=yes
IPV6ADDR=<IPv6-IP-Address>
IPV6_DEFAULTGW=<IPv6-IP-Gateway-Address>
```

Step 5. Save and close the file. Restart networking:

#Service network restart

References

[1] What is IPv6? [ONLINE] Available at : http://IPv4.opus1.com/IPV6/whatisIPV6.html.

[Accessed: 03 March 2015].

[2] Router-Switch. 2013. What is OSI Model & the Overall Explanation of ISO 7 Layers.

[ONLINE] Available at: http://IPv4.cisco1900router.com/what-is-ios-model-the-overall-explanation-of-ios-7-layers.html. [Accessed: 04 March 2015].

[3] omnisecu. TCP/IP. [ONLINE] Available at: http://www.omnisecu.com/tcpip/tcpip-model.php.[Accessed: 07 July 2015].

[4] Kurose, J &Ross, K. 2010. Computer Networking. 5th ed. USA: Pearson

[5] Eazynotes. Comparasion between OSI and TCP/IP model. [ONLINE] Available at:

http://www.eazynotes.com/notes/computer-networks/slides/comparison-between-osi-tcpip-model.pdf. [Accessed: 04 July 2015].

[6] tutorialslodge. 2014. OSIandTCP/IPModel.[ONLINE]Available at: http://tutorialslodge.com/osi-tcpip-model/. [Accessed: 10 July 2015].

[7] tcpipguide. Internet Protocol Version 4 (IP, IPv4). [ONLINE] Available at:

http://IPv4.tcpipguide.com/free/t_InternetProtocolVersion4IPIPV4.html. [Accessed: 20 May 2015].

[8] Ipv4 Addressing. 2015. Ipv4 Addressing. [ONLINE] Available at:

https://technet.microsoft.com/en-us/library/dd379547(v=ws.10).aspx. [Accessed: 02 July 2015].

[9] IBM. Comparison of IPv4 and IPv6. [ONLINE] Available at: https://IPv4-01.ibm.com/support/knowledgecenter/ssw_ibm_i_72/rzai2/rzai2compIPV4IPV6.html. [Accessed: 08 April 2015].

[10] Graziani. R. 2012. IPv6 Fundamentals: A Straightforward Approach to Understanding IPv6.

[11] training.apnic.net. 1800. IPv4 to IPv6 Transition. [ONLINE] Available at: https://training.apnic.net/docs/eIP603_Transition.pdf. [Accessed: 12 July 2015].

[12] Sellers, C. 2009. IPv6 Transition Mechanisms and Strategies. [ONLINE] Available at: http://IPv4.rmv6tf.org/wp-content/uploads/2012/11/Chuck-Sellers-090421-IPV6-Transition-Mechanisms-Sellers1.pdf. [Accessed: 29 June 2015].

[13] Das, K. Network Address Translation (NAT) Pros & Cons. Network Address Translation (NAT) Pros & Cons, [Online]. Available at: http://IPv6.com/articles/nat/NAT-Pros-and-Cons.html [Accessed: 29 July 2015].

[14] Cisco. NAT-PT for IPv6. [ONLINE] Available at: http://IPv4.cisco.com/c/en/us/td/docs/ios-xml/ios/ipaddr_nat/configuration/15-mt/nat-15-mt-book/ip6-natpt.html. [Accessed 28 July 2015].
TURKU UNIVERSITY OF APPLIED SCIENCES THESIS | Bhuwan Chhetri52

[15] IP Addresses. 2013. IP Addresses. [ONLINE] Available at: http://www.itproportal.com/2013/07/02/ip-addresses-how-they-shaped-the-past-of-the-internet-and-what-they-will-influence-its-future/. [Accessed: 09 August 2015].

[16] IETF. 2010. Problem Statements of IPv6 Transition of ISP. [ONLINE] Available at: https://tools.ietf.org/html/draft-lee-v4v6tran-problem-02. [Accessed: 04 August 2015].

[17] Internet Engineering Task Force. 2011. Broadband Service Provider Use Case . [ONLINE] Available at: http://www.watersprings.org/pub/id/draft-tian-v4v6tran-broadband-sp-usecase-00.txt. [Accessed 11 July 2015].