Master of Science in Electrical Engineering with emphasis on Telecommunication Systems

January- 2019

Performance Evaluation of Multicast Behavior in Congested Networks

Urmila Jyothula

Faculty of Computing

Blekinge Institute of Technology

SE 37179 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering with Emphasis in Telecommunication Systems.

Contact Information:
Author(s): Urmila Jyothula
E-mail:
urjy16@student.bth.se
urmila7842@gmail.com

External advisor:
Stefan Bernbo CEO,
Compuverde AB
E-mail: stefan@compuverde.com

University advisor:
Dr. Dragos Ilie
Assistant Professor of Telecommunication Systems
Department of Communication Systems
E-mail: dragos.ilie@gmail.com
Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Compuverde's software-defined storage product uses multicast for the communication between servers in a cluster. The product makes use of IP UDP multicast for sending status messages between the servers that form the storage cluster. The storage clusters capacity and performance scale linearly to the number of servers in the cluster. The problem is, the multicast traffic also increases with the number of nodes. All nodes in the cluster send packets to the remaining nodes in the cluster. In this document, we present an evaluation of IP multicast behavior in a network congested with traffic similar to that produced by Compuverde's product. IP multicast is a method of sending Internet Protocol (IP) datagrams to a group of interested receivers in a single transmission. In order to provide efficient, timely, and world wide many-to-many distribution of data, and as such data may become the broadcast medium of choice in the future, IP multicasting is used. The main benefit of IP Multicast is that it reduces the bandwidth consumption when data from a sender must reach multiple receivers. We are interested in studying the effects on the network when we send multicast packets at a rate close to the operational limit of the switch. To be able to study this behavior at larger scale Compuverde provided a cluster with 48 servers all connected to the same switch. Also, we compare the behavior of IPv4 multicast traffic to that IPv6 multicast traffic.

Aims and Objectives: The aim of the thesis is mainly to focus on IP multicast and compare the IPv4 multicast performance results to that of IPv6 multicast. To achieve the aim, an analysis of IP multicast (IPv4, IPv6) will be done. To compare both the IPv4 multicast and IPv6 multicast results. C++ tool is developed for generating both IPv4 and IPv6 multicast traffic is developed on Linux, and its design and implementation will be documented here. Additionally, this report will include a detailed study on the pattern of dropped packets when traffic rate approaches operational limit and other related impairments on QoS metrics (e.g., CPU utilization, Throughput). CPU utilization means should measure both the CPU utilization of a switch while sending IPv4 and IPv6 packets to different servers and also CPU utilization of server while sending IPv4 and IPv6 packets to different servers.

Methods: The method is to develop a tool that will generate a multicast load towards servers in a cluster. The data sent as multicast packets will consist of information that will make it possible to detect packet loss on the receiving servers if the network gets congested. The first version of the tool will use existing socket classes that are based on the IPv4 protocol and is programmed in C++. The tool will be able to run in two modes at the same time: client mode and server mode. The server part of the tool will subscribe to a predefined multicast address and receive incoming multicast packets. The client part of the tool will send data packets to the same predefined multicast address at a configurable rate that will increase over time. The load should start with a small number of servers in the cluster, and then gradually increase the number of servers, until a maximum of 48 servers is reached. The rate at which multicast packets are sent should also be increased until the switch gets overloaded and starts to drop packets. The pattern of how packets are dropped should be observed. For example, if it is bigger chunks of packets that get dropped or if it is every second packet that gets dropped. The second version of the tool will support IPv6 multicast. The second round of tests should be performed in a way that makes them comparable to the results from the IPv4 tests, so it is possible to conclude if one protocol performs better or is more reliable.

Result: The maximum number of IPv4 packets a switch can handle is 140 packets per second. The maximum number of IPv6 packets a switch can handle is six packets per second. The CPU utilization is more while multicasting the IPv4 packets than while multicasting IPv6 packets by using a switch, 95 Nodes.

Conclusion: The experiments indicate that IPv4 multicast, as implemented by the used switch, is a more efficient protocol than IPv6 multicast while sending packets a high data rates. The CPU utilization is higher while sending IPv6 multicast than when sending IPv4 multicast.

Keywords: Packet Loss, IP multicast, CPU Utilization.

Acknowledgment

For all their constant technical and intellectual support, I am thankful to my project supervisor from Compuverde, Stefan Bernbo as well as to Dragos llie the assigned supervisor and examiner from Blekinge Institute of Technology for his great supervision. I am very much thankful to the examiner Kurt Tutchku for his responsible examination, and management for the thesis course. I also owe high gratitude to my parents for all their inspiration, care, and encouragement.

CONTENTS

LIST OF ABBREVIATIONS

IPv4 - Internet Protocol version 4

IPv6 - Internet protocol version 6

IGMP - Internet group management protocol

DVMRP - Distance Vector Multicast Routing Protocol

MOSPF - Multicast Open Shortest Path First

MADCAP - Multicast Address Dynamic Client Allocation Protocol

PGM - Pragmatic General Multicast

SDP - Session Description Protocol

OSPF - Open Shortest Path First

MAC - Medium Access Control

TOS - Type of Service

MLD - Multicast Listener Discovery

ND - Neighbor Discovery

IHL - International humanitarian law

URL - Uniform Resource Locator

TOS - Type of service

ARP - Address Resolution Protocol

TCP - Transfer control protocol

ICMP - Internet Control Message Protocol

UDP - User Datagram Protocol

GBE - Gigabit Ethernet

RS - Router Solicitation

HTTP - HyperText transfer protocol

SMTP - Simple Mail Transfer Protocol

DHCP - Dynamic Host Configuration Protocol

List of Table

List of Figures

1. Introduction

This document presents a study on Performance Evaluation of Multicast behavior in congested networks. The document is organized as follows. Chapter 1 gives the background, aim, objectives, research questions, related work involved in this thesis. Chapter 2 deals with details about IPv4 and IPv6 multicast. The 2nd chapter describes related work. Chapter 3 deals with the methodology and testbed used to study the multicast behavior in a congested network. Chapter 4 deals with the results obtained from the measurement and the analysis. Chapter 5 deals with the conclusion part of the thesis and future work that can be done.

1.1 Background

Compuverde's software-defined storage product uses multicast for the communication between servers in a cluster.

The product makes use of IP UDP multicast for sending status messages between the servers that form the storage cluster. The storage clusters capacity and performance scale linearly to the number of servers in the cluster. The problem is that the multicast traffic also increases with the number of nodes. All nodes send packets to all the remaining nodes in the cluster. In this document, we present a report on the evaluation of IP multicast behavior in a network congested with traffic similar to that produced by Compuverde's product. The main benefit of IP Multicast is that it reduces the bandwidth consumption when data from a sender must reach multiple receivers.

Our interest is in studying the effects on the network when we send multicast packets at a rate close to the operational limit of the switch. To be able to study this behavior at larger scale Compuverde's will provide a cluster with 48 servers all connected to the same switch.

1.2 Aim and Objectives

Our aim of the thesis is mainly to focus on IP multicast and compare the IPv4 multicast performance results to that of IPv6 multicast. Also, a C++ tool for generating multicast traffic is developed on Linux.

Objectives:

- A detailed study on IP multicast (IPv4, IPv6).
- A detailed study on the design and efficient implementation of a multicast traffic generating tool.
- An overview of the switch that will be used in the project.
- A detailed study on the pattern of dropped packets when traffic rate approaches operational limit and other related impairments on QoS metrics (e.g., CPU Utilization, Throughput).

1.3 Research Questions

The research work mainly focuses on the evolution of multicast behavior in a congested network.

The below are the research questions which are related to this research.

1. What is a good way for a server to detect that multicast packet loss has occurred due to an overloaded network or switch?

2. What is the pattern of dropped packets when the switch gets overloaded? For example, does the switch drop every second packet or does it drop chunks of packets?

3. Is IPv6 a more efficient protocol than IPv4 when it comes to sending multicast packages at a very high rate?

1.4 Related Work

This chapter provides the information of previous researches related to the IPv4 multicast and IPv6 multicast.

The below are related works that have been coming across before starting the implementation.
Oliver Hermanns[1] gave a clear understanding of IP multicast concepts like multicast architecture, multicast measurements and also about multicast performance aspects like throughput, delay.

Shaung Yang[2] determined that the use of network coding has been shown to improve throughput in multicast switches, but not without costs of computational complexity and delays. In their paper, they investigate the network coding algorithms in a switch with multicast traffic. Their papers explained the packet buffers and packet loss detections. They had many simulations to find the efficiency of Haste and implemented a framework to emulate input-queued switches using asynchronous network sockets. Their framework can process actual UDP traffic using Haste with online network coding and shows definitive conclusion that Haste is useful for practical use, and in multicast switches. They have studied and given a clear scenario about the performance of a switch with network coding. Though network coding can improve the throughput, the conventional network coding algorithm failed in two aspects: large packet delays and high computational complexity. They are convinced that the design of Haste constitutes one major step closer towards

practical implementations of network coding in network switches. The number of incoming packets per unit time should be chosen small even through requires at least billions of multiplications per second.

Marco Ajmone Marsan[3] presented a formal description of the multicast traffic scheduling problem in input-queued packet switches with internal multicast capabilities. An optimal multicast scheduling discipline, called max-scalar, for the transfer of multicast packets through the switch, was defined, and a proof of its optimality, based upon the use of Lyapunov functions, was provided. From the definition of the optimal scheduling discipline, the formal characterization of the capacity region, i.e., of the set of input loads that an ideal IQ switch can transfer to output ports, was obtained. In particular, they proved that for large-size switches, when using per-flow FIFO queueing architectures, no finite speedup guarantees 100% throughput under admissible multicast traffic patterns.

Shaif S.Shihab[4] explains the IPv4 multicast and IP6 multicast. Multicast technology appeared approximately twenty years ago. Although right, from its conception it attracted big interest for a long time but did not become the accredited technology — however, the recent huge growth in network traffic and the creation of new applications like YouTube. Internet Protocol version 4 (IPv4) is one of the key fundamentals of the Internet, which is currently serving up to four billion hosts over various networks. Despite this, the IPv4 protocol has still been successfully functioned well since 1981. It gives a clear picture of comparative analysis between IPv4 multicast and IPv6 multicast. It explained the performance metrics of IPv4 and IPv6.

Jay Kumar Sundararajan[5] consider the problem of serving multicast flows in a crossbar switch. They showed that linear network coding across packets of flow could support the traffic patterns, which will not be allowed if network coding were not allowed. So, network coding will lead to a larger rate region in a multicast crossbar switch. They signify a traffic pattern which needs a switch speedup if coding were not allowed, whereas, with coding, the speedup requirement can be removed. Along with the throughput benefits, coding makes the characterization of the rate region simpler. They provide a graph-theoretic design of the rate region with the fanout splitting and the intra-flow coding, regarding the stable set polytope of the "enhanced conflict graph" of the traffic pattern. A design like it is not known in the case of fanout splitting without coding. They show that computing the offline schedule can be reduced to certain graph coloring problems.

Zongpeng Li[6] have studied how to compute and achieve optimal throughput in data networks, in the general case of undirected communication links. They have been pleasantly surprised at how results from network coding can facilitate the design of efficient solutions to this fundamental problem that was previously viewed as very hard. They also show the counterintuitive conclusion that the most significant benefit of network coding is not to achieve higher optimal throughput but to make it feasible to achieve such optimality

in polynomial time. They show that such efficient algorithms may be designed for multiple communication sessions of a variety of types, and the more realistic model of overlay networks. Simulation studies also suggest that overlay multicast techniques may approach optimal multicast throughput quite well.

2. IP Multicast

Multicasting is a transmission of the datagram to multiple numbers of receivers known as a multicast group through a single transmission of the network. IP protocol consists of three methods for packets transmission. IP unicasting, IP broadcasting, IP multicasting. IP unicasting transmits a single datagram to one receiver. IP broadcasting transmits a single datagram to all receivers. IP multicasting transmits a single datagram to multiple receivers through a single transmission. IP multicast is, in fact, point-to-multipoint communication. It is used for media streaming and other applications on the internet and private networks. Basing on IP protocol versions 4 and version 6. IP multicast is available both as IPv4 multicast and IPv6 multicast.

2.1 Multicast in IPv4 Networks

The IPv4 multicast has various components such as router, host, multicast address, multicast group, and MBone. This IPv4 multicast consists of the source address and destination as we have seen in figure1. IPv4 multicast uses various types of protocols such as IGMP, DVMRP, MOSPF, PIM, MADCAP, PGM.

IGMP protocol is used in IPv4 network to uphold a host group membership on the local subnet.

DVMRP is a routing protocol which is used to split the data between routers to provide the transmission of IP multicast packets among networks.

MOSPF is an extension for OSPF protocol that provides mutual operation between unicast and multicast routers.

PIM is an internet protocol which provides one too many and many to many data over the internet.

MADCAP is a protocol which is used to allow the hosts to request multicast address allocation services from allocation servers of the multicast address.

PGM is a suitable quality multicast protocol which provides a reliable sequence of packets to multiple numbers of receivers.

IPv4 multicast is dependent on two components, one is the host component, and another one is the route component. The host component is used in a single subnet whereas the route component is required when listeners are spread over multiple subnets. In a Level 2

multicast-enabled internet, any host can transmit IP multicast datagrams to any multicast group address, and receive IP multicast datagrams from any multicast group address. In a Level 1 multicast-enabled internet, any host can send multicast datagrams, but it is unable to receive them.

The difference between Level 1 and Level 2 is not important nowadays as most hosts support Level 2 multicast.

The header of a multicast IP datagrams consists of own IP address as a source address and the obtained IP address as a destination address and a suitable TTL value that can be seen in figure 1. The destination IP address is typically obtained from the source IP address and it is simply called as receiver information. TTL value measures the number of subnets the datagram can travel. The TTL value ranges from 0 to 255. In the local subnet IPv4 multicast address ranges from 224.0.0.0 to 224.0.0.255 (224.0.0.0/24).

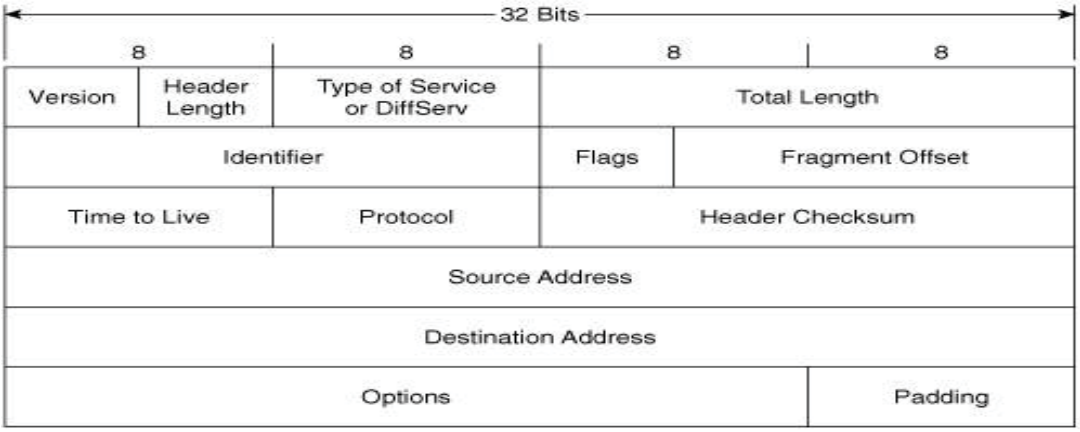


Figure 1: View of IPv4 Header Format

To receive IP multicast datagrams, a host informs IP to send the multicast datagrams for a specific group address, registers the proper multicast MAC address with the network adapter, and informs the local router to join the multicast group. To determine which particular multicast group address to be used, the host can obtain an address from the application or derive the address from a universal resource locator (URL). On the MBone, multicasts are generated with the Session Description Protocol (SDP) which contains all the data to receive a multicast IP datagram, like the multicast name and description, the times it needs to be active, the media type it uses, as well as the IP addresses, ports, and protocol it uses. The data is transmitted to a well-known IP address and port where a host running the session directory tool receives it.

The datagrams receiving can be stopped by a host for a specific multicast group address when a corresponding MAC-level address is removed from a table of interesting destination MAC address. This occurs when a multicast enables application informs sockets to stop

receiving datagram for a specific group address. This IPv4 multicasting is used for efficient bandwidth utilization.

The multicast router manages multicast group membership that transmits the IP datagrams to a subnet that have at least one group member. The multicasting routers will send and receive the IGMP group membership messages. The multicast routers receive the incoming IGMP Host Membership Report messages from all other hosts on local subnets. Later the multicast routers process these messages by placing entries that keep track of group membership in the IP multicast forwarding table.

The multicast router will communicate group membership to other multicast routers. Router process will receive all multicast traffic and it forward all multicast traffic.

2.1.1 IPv4 Header Format

The IPv4 packet header contains 14 fields, in which 13 fields essential. The 14th field is optional named called options. IPv4 header contains twenty bytes of data.

The version is a first header filled in an IPv4 packet is the four-bit version field. This four-bit version field is set to binary 0100 to indicate version 4 (IPv4) or binary 0110 to indicate version 6 (IPv6).

Header Length is the second field, and it is a 4 bits field is the Internet Header Length (IHL) indicating the number of 32-bit words in the header. The minimum value for the header length field is 5, which is a length of 5×32 = 160 bits = 20 bytes.

Types of service (TOS) used to transmit the information to provide quality of service features. New technologies are arriving that require real-time data streaming and therefore make use of the DSCP field.

Total length is a 16-bit field determines the entire datagram size, including header and data, in bytes.

Identification is a field identification field, and this identification field is mostly used for uniquely identifying fragments of an original IP datagram.

The flag is a three-bit field used to control the fragments.

Fragment offset is measurable in a block of eight-bytes, is 13 bits long and specifies the offset of a particular kind of fragment relative to the beginning of the original unfragmented IP datagram.

Time to live is an 8-bit field represents the maximum time the datagram is allowed to remain in the internet system.

The protocol is a field which defines the protocol used in the data portion of the IP datagram.
Header checksum is a 16-bit field useful for checking the errors in the header.

The source address is IPv4 address indicates to send the packets.

Destination address which is an IPv4 address indicates to receive the packets.
Options is an additional address field which may follow the destination address field[7].

2.1.2 ICMP4 Protocol

The protocol called Internet Control Message Protocol (ICMP) is an integral part of the IP stack. ICMP messages are sent in various situations like for example when a datagram is not able to reach its destination when the gateway does not have the buffering capacity to pass a datagram, and when the gateway is directed by the host to send the traffic on a smaller route. The Internet Protocol is not designed to set reliable. The control messages are used to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees for a datagram to be delivered to the destination and their loss does not necessarily generate ICMP error messages. The higher-level protocols that will use IP must implement their procedures of reliability if reliable communication is required [8].

2.1.3 Types of address prefixes in IPv4

Addressing Types	Purpose
Loopback IP address	Mostly useful for testing requirements such as client-server architecture on a single machine. Other than that client-server architecture on a single machine, if a host machine may successfully ping 127.0.0.1 IP address or any other IP from loopback range, implies that the TCP/IP software stack on the machine is successfully loaded and working.

Private IP address	This type of address is required for the Internet at large to conserve the globally unique address space by not using it where the global uniqueness is not required. It has ranges that start at 10.0.0 to 10.225.225.225 for Class A private networks, 172.16.0.0 to 172.31.225.225 for the Class B private addresses, Class C private network starts from 192.168.0.0 to 192.168.225.225.
Public IP address	An address that is assigned to a computing device to allow direct access over the Internet. A web server, email server and any server device directly accessible from the Internet are a candidate for a public IP address. The public IP address ranges from 1 to 223.225.225.225
Unicast-Prefix-Based IPv4 Multicast Addresses	A multicast address with the prefix 234/8 indicates that the address is a Unicast-Based Multicast (UBM) address. This multicast addresses at the same time as unicast prefixes, network operators will be able to identify their multicast addresses without needing to run an inter-domain allocation protocol.

Table: 1 Different type of addressing prefixes in IPv4

From the above Table 1 we can observe that IPv4 consists of some addressing prefixes.

2.1.4 ARP (ADDRESS RESOLUTION PROTOCOL)

Address Resolution Protocol (ARP) is an important protocol in the TCP/IP suite. The ARP is used to resolve an IPv4 address (32-bit Logical Address) to the physical address (48 bit MAC Address).

The following are present in the fields in the Address Resolution Protocol (ARP) Message Format.

- Hardware Type field in the ARP Message specifies the type of hardware used for the local network ARP message.
- Protocol type is a protocol is assigned a number used in this field. IPv4 is 2048 (0x0800 in hexadecimal).
- Hardware address length in the ARP is the message length in bytes of a hardware MAC address.
- Protocol Address Length is a Length in bytes of a logical address (IPv4 Address).
- Opcode field in the ARP Message specifies the nature of the ARP message.
- The sender Hardware address is a layer2 MAC address of the device sending the message.
- Sender protocol address is an IPv4 address of the device sending the message.
- Target protocol address is an IPv4 address of the intended receiver.
- The target hardware address is a MAC address of the intended receiver.

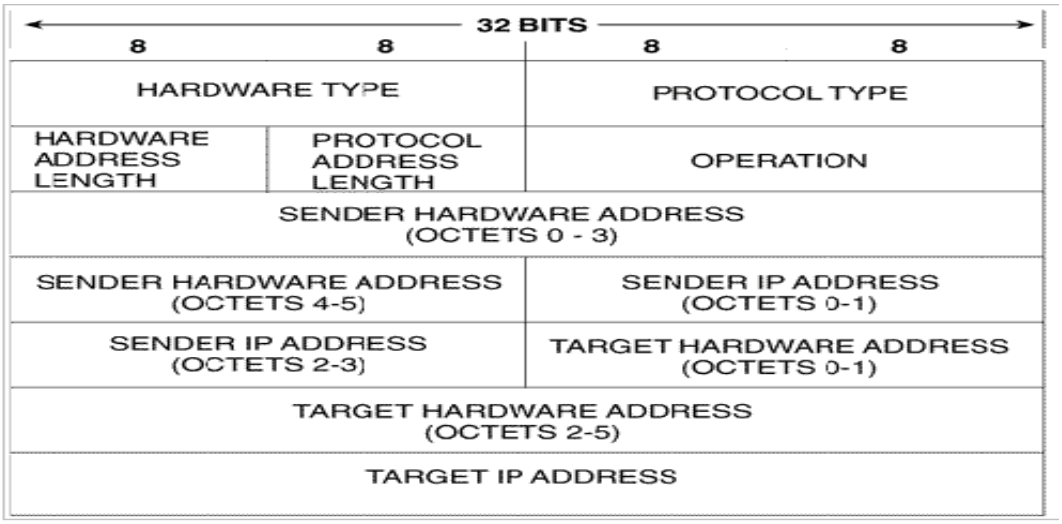


Figure 2: ARP Message format

2.1.5 IGMPv1 Protocol

There are four types of messages are present in the IGMP protocol. The following table describes the IGMP messages.

Message type	Description

Membership Query	Used by IGMPv1 and IGMPv2 multicast routers to locate the multicast groups in use by hosts on the local network
V1 membership report	This report identifies the message as an IGMPv1 Membership Report. The Internet Group Management Protocol (IGMP) will be used by hosts to report their multicast group memberships to any interesting immediately neighboring multicast routers. It describes only the use of IGMP between hosts and routers to find the correct group membership.
v2 Membership Report	This report identifies this message as an IGMPv2 Membership Report.
v2 Leave Report	Used by IGMPv2 hosts to announce that they are leaving a multicast group[9].

Table 2: IGMP Message types

The Internet Group Management Protocol is known as IGMP(v1). It is utilized by IP hosts to report their host group memberships to any immediately-neighboring multicast routers.

Multicast routers are used in IGMP to learn which groups have members on each of their attached physical networks. A router of multicast keeps a list of multicast group memberships for each attached network and a timer for each membership. "Multicast group memberships" consists of at least one member of the multicast group on a given attached network, but not a bunch of all of the members[3].

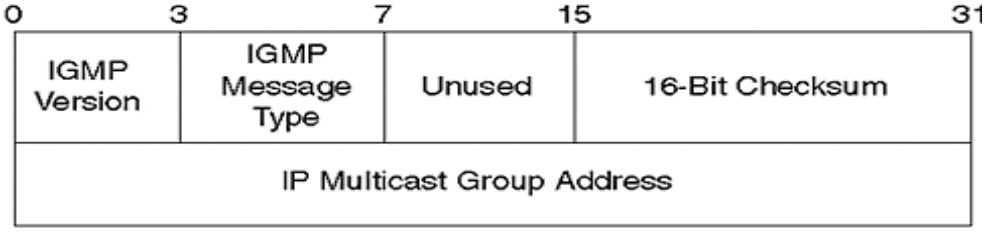


Figure 3: IGMPv1 Header format

The IGMPv1 header consists of version, type, unused, IGMP checksum, and Group address fields.

The version is a 4-bit field, and it is set to 1.

Unused is an 8-bit field, and it is cleared to zero when the IGMP packet is sent and ignored when received.

The checksum is the 16-bit one's complement of the one's complement sum of the 8-byte IGMP message. While calculating the checksum, the checksum should first should set to 0. When the data packet is sent, the checksum is calculated and inserted in this field. When the data packet are received, the checksum is again calculated and will be verified against the checksum field. If the two checksums are not equal then, an error has occurred.

Group Address is 32 bits. In a Host Membership Query message, the group address field is zeroed when sent, ignored when received. In a Host Membership Report message, the group address field holds the IP host group address of the group being reported.

2.1.6 IGMPv2

IGMP allows group membership termination to be reported fast to the routing protocol which is vital for high-bandwidth multicast groups and subnets with highly volatile group membership.

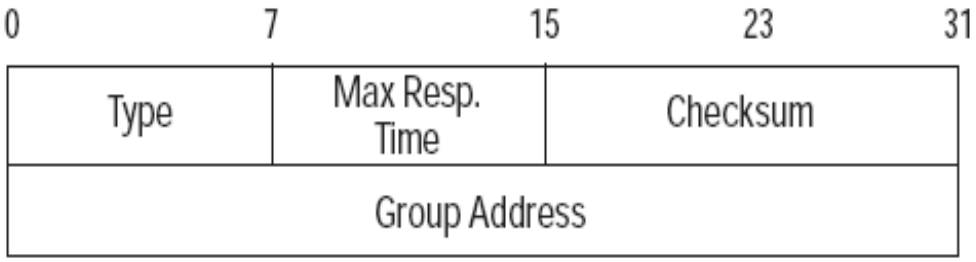


Figure 4: IGMP version2 Header format

IGMPv2 packet format consists of a type, IGMP checksum, Group address which has the same functionality as IGMPv1 with an exception, the max response time field is present in IGMPv2 not present in IGMPv1.

Max response time is an 8 bit, which is useful only in Membership Query messages and indicates the maximum allowed time before sending a responding report in units of 1/10 second.

2.1.7 IGMPv3

IGMP Version 3 supports "source filtering" which is, the ability of a system to report interest in receiving packets only from the specific set of source addresses, as required to support Source-Specific Multicast [SSM], or from all sources addresses except for few, sent to a particular multicast address. Version 3 is designed to be interoperable along with the Versions 1 and 2. MLD version 2 [MLDv2] implements the functionality of IGMP version 3.

The message format of IGMPv3 consists of fields like Type, Max response code, checksum, Group address which have the same purpose like IGMPv2, Reserved, S Flag (Suppress Router-Side Processing), QRV, QQIC, No. of sources, Source address are the different fields from IGMPv2.

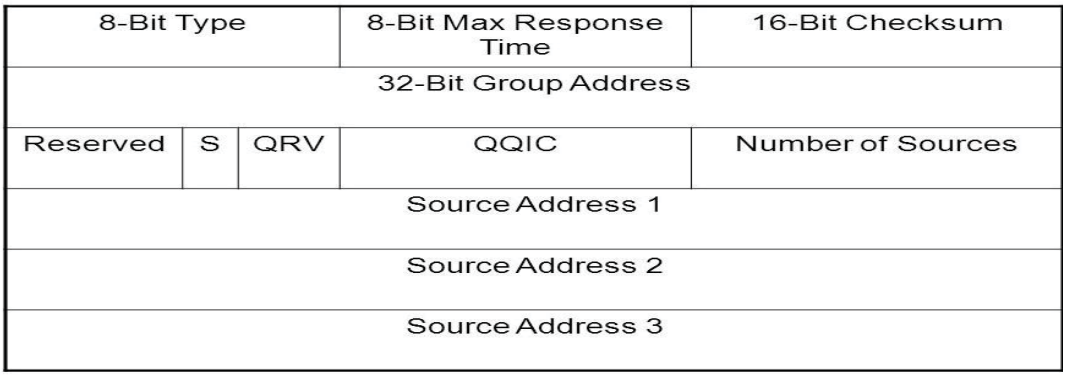


Figure 5: IGMPv3 Message format

Resv (Reserved) is set to zero on transmission and ignored on reception.

S Flag (Suppress Router-Side Processing) is set to one; which specifies the destination multicast routers that, they need to suppress the normal timer updates they perform upon listening to a Query.

QRV (Querier's Robustness Variable) a non zero field contains the [Robustness Variable] value used by the querier (the sender of the Query). If the querier's [Robustness Variable] exceeds 7, the maximum value of the QRV field, the QRV is set to zero.

QQIC (Querier's Query Interval Code) is a field which determines [Query Interval] used by the querier, i.e., the sender of the Query. Querier's Query Interval (QQI), is represented in the units of seconds and is obtained from the Querier's Query Interval Code.

The number of Sources (N) is a field which specifies how many source addresses are present in the Query. It is zero in a Group-Specific Query or a General Query, and non-zero in a Group-and-Source-Specific Query[10].

The Difference between IGMPv1, IGMPv2, IGMPv3 protocol versions are mentioned below.

Feature	IGMPv1	IGMPv2	IGMPv3
1st Octet value for Query message	0x11	0x11	0x11
Group address for General query	0.0.0.0	0.0.0.0	0.0.0.0
Destination address for General query	224.0.0.1	224.0.0.1	224.0.0.1
Default Query Interval	60 sec	125 sec	125 sec
1st octet value for Report	0x12	0x16	0x22
Group address for the report	Joining multicast group address	Joining multicast group address	Joining multicast group address and source address
Is report suppression mechanism available	Yes	Yes	No
Can max response time be configured	No , fixed at 10 sec	Yes,0 to 25.5 sec	Yes,0 to 53 min
Can a host send a leave group message	No	Yes	Yes
Destination address for leave group message	-	224.0.0.2	224.0.0.22
Can a Router send Group-specific query?	No	Yes	Yes
Can a Host send Source and group specific reports?	No	No	Yes
Can router end Source and Group specific Queries?	No	No	Yes
Rule for Electing a Querier?	None (depends on multicast routing protocol)	Router with the lowest IP address on the subnet	Router with the lowest IP address on the subnet
Compatible with other Versions of IGMP?	No	Yes, only with IGMP v1	Yes, with both IGMP v1 and v2

Table 3 : Shows the difference between IGMPv1, IGMPv2, IGMPv3[11]

2.2 Multicast in IPv6 Networks

The IPv6 multicast has various components like IPv4 multicast, such as host, router, multicast group, multicast address, and IPv6 protocol components are installed in windows operating systems is a series of interconnected protocols. This IPv6 multicast has various protocols such as IPv6, ICMPv6, Neighbor discovery (ND), multicast listener Discovery (MLD).

This IPv6 is a routable protocol, responsible for the addressing, routing, fragmenting of packets and sending host is responsible for it. Like IPv4, IPv6 is also a connectionless protocol. IPv6 is an unreliable datagram protocol, which is mainly responsible for addressing and routing packets between hosts. The Upper layer protocol like TCP is responsible for acknowledgment of packets delivered and the recovery of lost packets.

IPv6 header consists of source address, destination address, Next Header, Hop Limit as we can see in the below figure 2. An IPv4 router on receiving a packet that exceeds in size for the segment of the network to which the packet is passed, and fragmentation of the packet

is allowed, IPv4 split the original packet into smaller packets that were suitable on the downstream network segment. Unless the Don't Fragment (DF) flag in the packet header is set, the packet is dropped, and returns an ICMP "Packet too big" message to the sender.

In IPv6, only the host performs fragmentation. If an IPv6 router could not forward a packet because it exceeds in size, the router returns an ICMPv6 "Packet Too Big" message to the host and discards the packet[12].

The size of an IPv6 address is 128 bits which is four times larger than IPv4, but the address space is exponentially larger than in IPv4, i.e., 2^128/2^32=2^92=7.9E28 more address spaces than IPv4. The IPv6 multicast traffic operates in a similar way to IPv4. Arbitrary IPv6 nodes can listen for multicast traffic on arbitrary IPv6 multicast addresses. Multiple multicast addresses can listen simultaneously for traffic from IPv6 nodes. Nodes can join or leave a group of multicast at any time. Addresses of multicast cannot be used as source addresses or as intermediate destinations in a Routing header. Beyond the first 8 bits (set to 0xFF), multicast addresses include additional structure to identify their flags, scope, and multicast group.

In IPv6 multicast neighbor Discovery protocol uses ICMPv6 messages for managing the neighboring node interaction. Neighbor Discovery replaced ARP, ICMP Router Discovery, and ICMP Redirect protocol and introduced more functions. The functions of Neighbor Discovery are returned with the following messages such as Router Solicitation, Router Advertisement, Neighbor Solicitation, Neighbor Advertisement, Redirect. Neighbor discovery uses the following options such as source link-layer address option, target Link-Layer Address Option, Prefix Information Option, Redirected Header Option, and MTU Option.

The host sends the IPv6 packet through one or more IPv6 routers to the destination. These sections assume that the Hop-by-Hop Options, Destination Options, and Routing extension headers are not present.

The IPv6 devices are used to join and listen to multicast traffic on the IPv6 multicast addresses. Multiple interfaces are identified by using the IPv6 address. In IPv6 communication, the IPv6 datagram packets are sent to all interfaces, and those packets are identified by the multicast address[13].

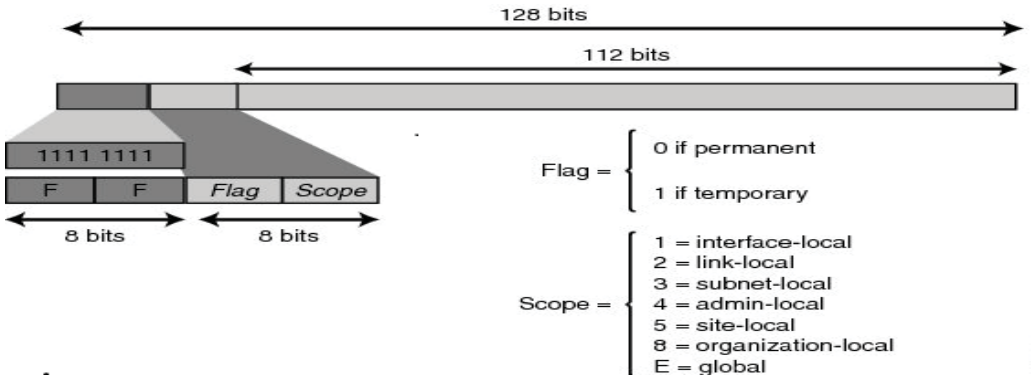


Figure 6: IPv6 multicast address format

In the IPv6 multicast addresses, the first eight bits are 1111 1111, and the prefix of an IPv6 multicast address is ff00::/8. Like IPv6 Link-Local addresses the IPv6 multicast address is also easy to identify because the IPv6 multicast addresses have left-most hexadecimal digits as "FF," and the size of that left hexadecimal digits is 8 bits. Beside the leftmost 8 bits, the four bits flag is called as a flag, scope. For the use in the future, the most significant bit in the 4 bits flags field is reserved. The other three flags are called R, P, and T.

The IPv6 multicast address has a 4-bit flag. Bit 0 is the left-most bit and is reserved for future use (thus lacking a name). Bit 1 bit is R flag also known as Rendezvous.
When R flag – 0, the multicast rendezvous point is not embedded within the multicast address.

R flag - 1, the multicast rendezvous point is embedded with the multicast address.
The 2nd bit in the 4bit flag field is P, and 'P' is known as Prefix.
When P flag – 0 The multicast address is not based on network prefix.

P flag – 1 The multicast address based on network prefix.
The third bit in the 4-bit flag field is 'T' is known as transient.
When T flag _ 0, the multicast address is a permanently assigned (well-known) multicast IPv6 address.

T flag – 1 The multicast address is a transient (Dynamically assigned) multicast address.
Following the flag field, there is a 4-bit Scope field. The scope is useful to show the scope delivery of IPv6 multicast traffic[13].

The following table 4 shows the values and the description of the scope field.

Values	Scope	Description
0	Reserved	Presently the scope is not in use
1	Interface local scope	The Interface-local scope indicates the

		single network interface and to carry the packets by using the loopback address.
2	Link-local scope	The link-local scope will represent the local link. The multicast address of FF02::2 has the limited traffic for local link scope. An IPv6 router will not proceed the multicast traffic destined to FF02::2 beyond the local link.
3	Subnet-local scope	The subnet-local scope is defined as the ranges subnets on multiple links.
4	Admin-local scope	An Administratively configured scope. This scope is defined as it must not be automatically copied from physical connections.
5	Site-local scope	The scope of Site-local multicast addresses is present inside the local physical network. The example of site-local scope is a small branch office. Site-local Multicast packets will not exceed the border router of IPv6 at the site.
8	Organization local scope	The scope of Organization-local addresses is present

		within various sites present inside an organization. Packets of an Organization-Local multicast will not cross the organization's border of an IPv6 router to reach the IPv6 Internet
E	Global scope	In the global scope, the Scope is the Global IPv6 internet
F	Reserved	Presently not under use

Table 4: List of Scope Field[13].

The below table 5 shows the essential IPv6 link-local scope multicast addresses and their description.

Important IPv6 Link-local scope Multicast Addresses	**Description**
FF02::1	An all nodes IPv6 multicast address.
FF02::2	An all routers IPv6 multicast address.
FF02::4	An all-Distance Vector Multicast Routing Protocol routers address. It is also as DVRMP router address.
FF02::5	Also known as OSPF IPv6 multicast address. In IPv4 the OSPF multicast address is 224.0.0.5
FF02::6	This address is an OSPFv3 Designated Router's and Non-Designated Router's IPv6 multicast address.
FF02::9	A RIPng IPv6 multicast address.
FF02::A	An EIGRP IPv6 multicast address.

FF02::D	An all PIM Routers address.
FF02::16	An all MLDv2-capable routers address.
FF02::1:2	DHCPv6 Servers and DHCPv6 Relay Agents address.
FF02::1:FF00:0000/104	A Solicited-Node IPv6 Multicast address.

Table 5: Important IPv6 Link-local scope Multicast Addresses[13]

2.2.1 IPv6 Header Format

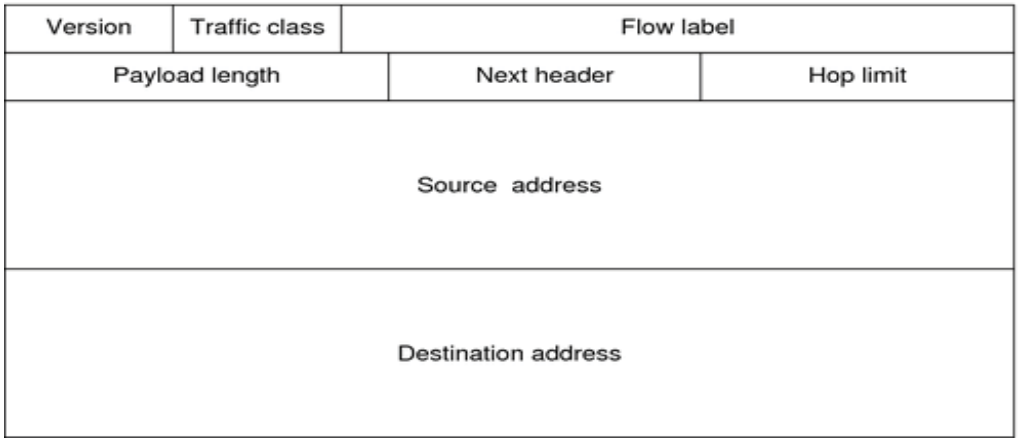


Figure 7: IPv6 Header Format

The IPv6 header is four times larger than the IPv4 header. IPv6 headers consist of one Fixed Header and zero or more Optional Headers.

The version is determining the version of an internet protocol, and it is a 4-bit field.

The traffic class is an 8-bit field split into two parts like the most significant part has 6bits field, and the least significant part is 2bits field.

The flow label is used to maintain the sequential flow of packets belongs to a communication. It is a 20-bit field.

Payload length is consisting of Extension Headers and Upper Layer data. It is a 16-bit field.

Next Header is an 8-bit field which is used to indicate either the type of Extension Header or if the Extension Header is not present then it indicates the Upper Layer PDU.

Hop limit is an 8-bit field is used to stop packet to loop in the network infinitely. This home limit is same as TTL which is present in IPv4.

The source address is a 128-bit field used for identifying the sender of the packet.

The destination address is a 128-bit field used for identifying the destination of the packet[14].

2.2.2 ICMPV6

The Internet Protocol version 6 (IPv6) utilizes the Internet Control Message Protocol (ICMP) like IPv4, with few exceptions. The protocol is called ICMPv6 and has an IPv6 Next Header value of 58.

IPv6 nodes use ICMPv6 to report errors while processing packets and perform other functions like diagnostics (Ex: ICMPv6 "ping"). ICMPv6 is an important part of IPv6, and every IPv6 node MUST fully implement the base protocol (all the messages and rules required by this specification).

An IPv6 header precedes each and every ICMPv6 message and contains zero or more IPv6 extension headers. A Next Header value of 58 identifies the ICMPv6 header in the immediately preceding header[15].

2.2.3 Types of Address Prefixes in IPv6

Addressing Types	Purpose
Loopback address	Mostly used for development and testing of network applications without even having network configurations and used by the developers.
Global Unicast address	Equivalent to IPv4 addresses from the public address space. Currently, IANA has assigned only 2000::/3 addresses the global pool and 2001::/16 address to various internet address registers[16].
Link-Local IPv6 Unicast Addresses	Will configure any interface using link-local prefix FE80::/10. This address is related only to a fixed physical layer.

Site-local unicast address	Similar to the private IP address in IPv4. It is used in organizations. The prefix of this address is FD00::/8.
Required anycast address	Assigned to interfaces set that typically belong to various nodes, and it should not use as a source address of an IPv6 packet.
Well-known multicast addresses	Has the prefix of ff00::/12. These addresses are reserved multicast addresses for assigned groups of devices. All node address, All router address, All OSPFv3 routers addresses, All EIGRP (IPv6) routers addresses are the well-known multicast addresses.
Solicited-Node Multicast Addresses	Multicast address of FF02:0:0:0:0:1:FFXX:XXXX. These addresses are measured as a function of a node's unicast and anycast addresses.

Table 6: Different types of addressing prefixes in IPv6

2.2.4 Differences between well-known addresses in IPv4 and IPv6

IPv4 Addressing Prefixes	IPv6 Addressing Prefixes
Uses 0.0.0.0 as unspecified address	Uses :: as unspecified addresses. In IPv6 notation two colons are used to "compress" set consecutive zeroes in the address. Thus, this is an all-zeroes address.
Uses 127.0.0.1 as loopback addresses	Uses ::1 as loopback address
Has broadcast addresses for all the devices.	Has no broadcast addresses, and it uses only multicast groups.

Multicast address space at 224.0.0.0/4.	Multicast address space at FF00::/8.
Supports global unique public address.	Supports global unique unicast address.

Table 7: Different types of addressing prefixes in IPv4 and IPv6

From the above Table 7, it can be observed that both IPv4 and IPv6 have similar well-known addresses. Both the Ipv4 and IPv6 support global addresses and both have Multicast address space.

2.2.5 Neighbor Discovery Protocol (NDP)

Neighbor Discovery Protocol (NDP) is an essential protocol in IPv6. The purpose of NDP is to identify the relationships between different neighboring devices in an IPv6 network. Many vital functions of IPv6 like resolving the MAC address of an IPv6 Address IPv4Router Discovery etc., are now performed using the Neighbor Discovery Protocol (NDP)[17]. Thus, NDP plays a role similar to the one ARP plays for IPv4.

NDP is based on ICMPv6 protocol. NDP uses ICMPv6 Type field values from 133 to 137. ND defines these mechanisms. ND's can be triggered by pings when using routers.

ICMPv6 Type	Name of the Message
133	Router Solicitation (RS)
134	Router Advertisement (RA)
135	Neighbor Solicitation (NS)
136	Neighbor Advertisement (NA)
137	Redirect Message

Table 8: ICMPv6 Messages defined for NDP

2.2.6 Mapping IPv6 addresses to Ethernet addresses

When IPv6 multicast packets are sent over an Ethernet link, at that point the destination, MAC address defines a 0x33-33-mm-mm-mm-mm, where the mm-mm-mm-mm will specifically outline the last 32 bits of the IPv6 multicast address. The underneath figure 4 demonstrates the mapping of an IPv6 multicast address to an Ethernet multicast address.

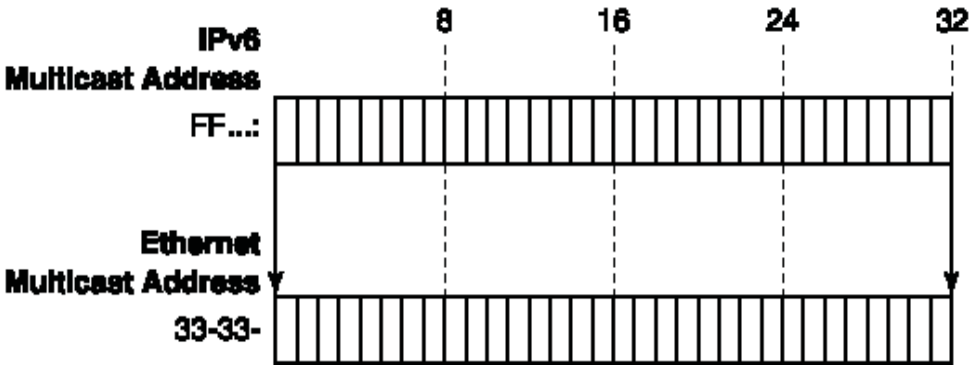


Figure 8: The mapping of IPv6 multicast addresses to Ethernet multicast addresses[18]

Ethernet network adapters consists a table containing interesting destination MAC addresses. If an Ethernet frame with an interesting destination MAC address is received, it is passed to the layers above it for additional processing. By default, this table contains the MAC-level broadcast address (0xFF-FF-FF-FF-FF-FF) and the unicast MAC address assigned to the adapter. For proficient delivery of multicast traffic, additional multicast destination addresses can be added or removed from the table. For every multicast address listened by the host, there is a corresponding entry in the table of interesting MAC addresses [18].

2.2.7 MLDv1(Multicast Listener Discovery)

MLDv1 is similar to IGMPv2.The Multicast Listener Discovery (MLDv1) is used to query IPv6 routers about the presence of multicast listeners. Multicast listeners are the nodes which are ready to receive packets of multicast to their attached links and also discover interesting multicast addresses close to the nodes. This information can be obtained by any multicast routing protocol that is being used by the router, to make sure that multicast packets are delivered to all links where there are interested receivers. MLDv1 protocol is known as an asymmetric protocol. MLDv1 is used to specify the various behaviors for multicast listeners and multicast routers to those multicast addresses to which a router a listening to. The router performs both listener and router part of the protocol, including responding to their messages. On another hand, the Listeners must perform the listener part of MLDv1 on all interfaces from which an application layer protocol or upper-layer protocol has requested to receive the multicast packets.

2.2.8 MLDv2

Multicast Listener Discovery Protocol Version 2 (MLDv2) is a protocol that is used to permit a host to notify its neighboring routers of its aim to receive IPv4 and IPv6 multicast data. Source-specific multicast (SSM) is a type of multicast in which a receiver is useful to mention both the network layer address of the source and the multicast destination address to receive the multicast transmission. The SSM-aware router and host is used for the

clarifying and modifying the behavior of both IGMPv3 and MLDv2 protocols on SSM-aware routers and hosts to accommodate source-specific multicast.

IGMPv3 has a potentiality for a host to selectively request or filter traffic from individual sources within a multicast group. MLDv2 offers the same functionality for IPv6 hosts. MLD version 2 helps to provide the similar source filtering functionality of IGMPv3 for IPv6. The term Group Management Protocol is used to denote both the IGMP and MLD protocols. The Source Filtering GMP or "SFGMP," terms are used to denote both the IGMPv3 and MLDv2 group management protocol[19].

3.Testbed and Methodology

The aim is to develop a tool that will transmit a multicast load towards servers in a cluster. The data sent as multicast packets will consist of information that will make it possible to detect packet loss on the receiving servers if the network gets congested.

- The first version of the tool will use existing socket classes that are based on the IPv4 protocol and are programmed in C++. The tool will be able to run in two modes at the same time: Client mode and Server mode. The Server part of the tool will subscribe to a predefined multicast address and receive incoming multicast packages[20]. The Client part of the tool will send data packets to the same predefined multicast address at a configurable rate that will gradually increase over time[2].
- The data packets to be sent will be designed in such a way that, the receiver (server) detects if a packet is lost in the transmission.
- The load will start with a small number of servers in the cluster, and will be gradually increased, until a maximum of 95 servers is reached.
- The rate at which multicast packets are sent is increased until the switch gets overloaded and starts to drop packets[2].
- The pattern of how packets are dropped is observed while gradually increasing the rate of multicast packets. For example, if it is chunks of packets that get dropped or if it is every *second, third,* or n^{th} packet that gets dropped.
- The second version of the tool will support IPv6 multicast. The second round of tests are to be performed in a similar way that makes them comparable to the results from the IPv4 tests to conclude if one protocol performs better or is more reliable.

.3.1 Testbed

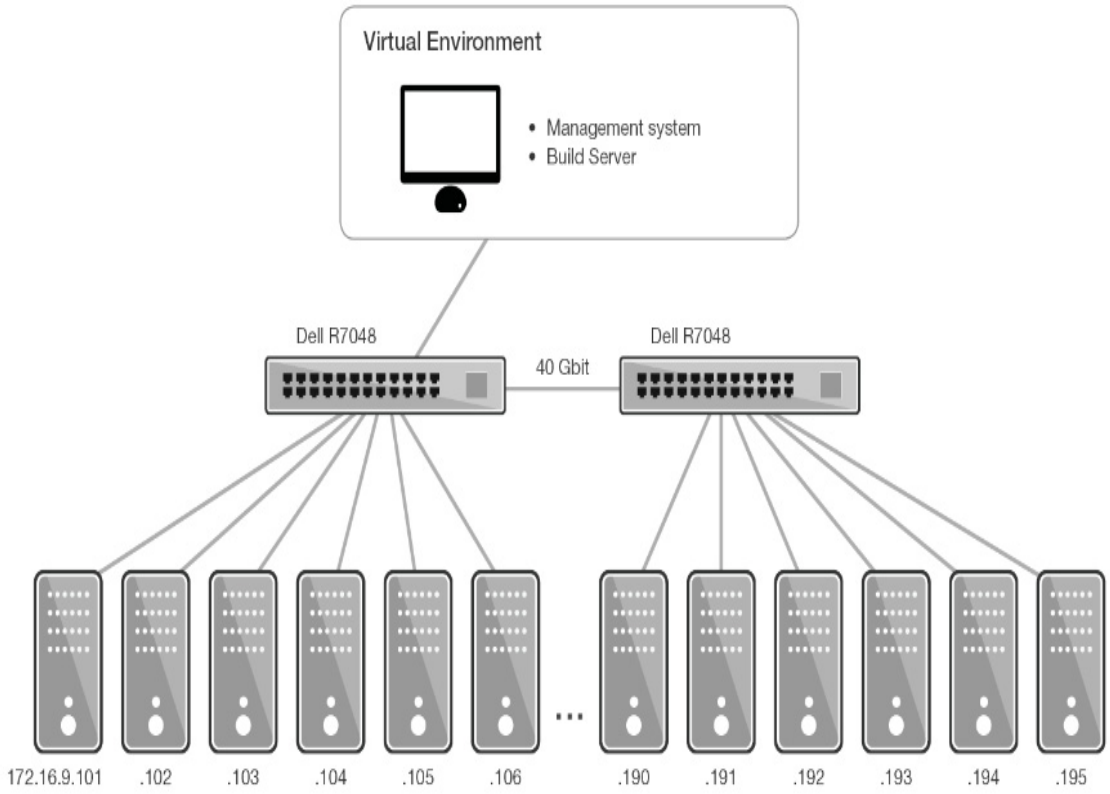


Figure 9: Testbed of IP multicast behavior in a Congested Networks

The testbed consists of one management system, which in turn consists of a virtual machine in it. The management system can be used to reach the testbed network, as the management system is isolated from the rest of the network infrastructure. There is no reason for the management system to be virtualized. It is available to use in the virtualized environment to reach the testbed network. This testbed consists of two switches (Dell 7048). The Dell 7048R switch has four 10Gbps ports on the rear side of the switch, so the link aggregation is 40Gbps. A 40GB link connects the two switches. The 95 nodes are connected to the two switches.

Node is a physical server, installed with the Compuverde software. These nodes are present below the switches as shown in the figure. Their purpose is to generate multicast traffic. A node consists of a Server and Client. The Server and Client part of the developed multicast tool are copied under /root directory on all servers. These 95 Nodes use the IPv4 addresses from 172.16.9.101 to 172.16.9.195. Typically, a node has sufficient storage drives like hard-disk drives (HDDs) and solid-state drives (SSDs), which are utilized as storage data and cache drives. These are present at the bottom part of the test bed. Ninety-five servers are physical servers which have CPU called Intel Atom D525, RAM 4GB and NIC is 1Gbit.

The Server and Client which are present inside the node are responsible for the transmission of multicast packets. Here the client will send packets to a specific multicast address, and the servers listening on the address will copy packets into their TCP stack. In this way, the IP multicasting occurs from clients to the servers.

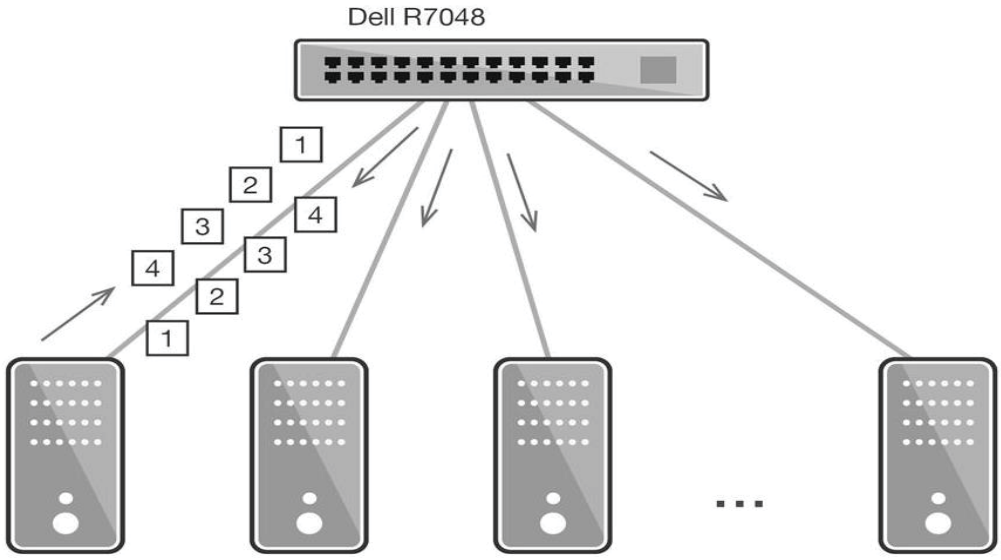


Figure 10: Shows how the Multicasting is going on using different servers and clients.

In the above figure 10, we can see how multicast traffic is generated. All servers (the server part of the multicast tool) subscribe to a specific multicast address. A client (the client part of the tool) sends packets to the multicast address. The switches distribute the packets to all servers. Each packet carries a sequence number, which enables the detection of packet loss or packet reordering.

3.1.1 Dell R7048 network switch

In this project, we are going to test a Dell networking and routing switch named as Dell R7048. Dell networking and routing switch consist of 48 Gigabit Ethernet (GBE) which are available for high-speed switching.

Dell switching network is designed to offer secure, fixed-port Gigabit Ethernet switching solutions which deliver full wire-speed switching performance.

The use of IGMP snooping or MLD snooping is to manage port-based multicast group membership and to function multicast efficiently.

3.1.2 IGMP Snooping

IGMP snooping is a technique of listening to Internet Group Management Protocol (IGMP) network traffic. The function allows a network switch to listen to the IGMP conversation between routers and hosts. By listening to these conversations, the switch creates a map of link with its suitable IP multicast streams. Multicasts will be filtered from the links which don't need them and then controls which ports can receive specific multicast traffic.

A switch by default transmits the multicast traffic to all the ports in a broadcast domain. Hence, multicast will cause overload on host devices by forcing them to process packets they have not requested. When purposefully overloading the host devices, this will lead to the denial-of-service attack. IGMP snooping is designed in such a way to prevent hosts on a local network from receiving traffic for a multicast group, which they have not requested. It allows the switches to avoid multicast traffic from links which do not contain a multicast listener (an IGMP client). IGMP snooping permits a switch to only transmit multicast traffic to the links that have requested them[21].

3.1.3 MLD Snooping

In IP version 4 (IPv4), Layer 2 switches use the Internet Group Management Protocol (IGMP) snooping to control the overloading of the multicast traffic by configuring Layer 2 interfaces, so the multicast traffic is only transmitted to the interfaces related to the IP multicast devices. Multicast Listener Discovery (MLD) Snooping provides the same functions for the IPv6 multicast routers, that IGMP Snooping provides to the IPv4 multicast routers.

The device has two versions of MLD Snooping namely, MLDv1 and MLDv2. MLDv1 Snooping finds MLDv1 control packets and setups traffic bridging based on IPv6 destination multicast addresses. MLDv1 is equivalent to IGMPv2. MLDv2 Snooping uses control packets to set up traffic forwarding based on source IPv6 address and destination IPv6 multicast address. MLDv2 is equivalent to IGMPv3. Like IGMP snooping, MLD frames are snooped as they are sent by the switch from stations to the upper multicast router[22].

4. C++ testing tools

The C++ tool is a socket programming tool which consists of two parts. One is a Client, and the other one is a Server. One tool is used to send the IPv4 multicast from different clients to different servers, and the other tool is used to send the IPv6 multicast packets. The reason for choosing C++ for tool development is because C++ is the language with the least CPU overhead compared to the other languages and has no garbage collection which

could have affected the results in the tests. C++ was chosen to minimize the risks of interfering with the results of the tests.

4.1 IPv4 client tool implementation

Step 1: The IPv4 client tools require multiple command line arguments. Those arguments are a local client IP address, Packets per sec (No. of packets sent per second), duration in a sec (Duration is the number of seconds that client continues to send packets, regardless of the packet send rate). The duration and packets per sec values can be changed when we want to change the speed of sending packets, duration of time in second. IPv4 packets (Datagrams) are sent by using AF_INET socket family.

```
/* Create a datagram socket on which to send. */
int sd = socket(AF_INET, SOCK_DGRAM, 0);
```

Step 2: The datagram socket will be used to send the IPv4 packets to servers. Initialize the group addr structure with a group address of 224.3.2.2 and with a port.no 4321. The sockaddr_in structure specifies about the IP address and Port number. 4321 is a protocol port which is used to send the UDP packets. We are sending the packets to 224.3.2.2 multicast group address by using the port.no 4321.

Step 3: After that, we have to initialize the number of packets to be sent to the servers. This is the product of duration in sec and packets per sec.

Step 4: The size of the IPv4 packet which we are sending is 60 bytes. The 60 bytes of the packet includes UDP header, IP header, Ethernet header, Payload. The Payload size is 8 bytes, where 4 bytes are used for the packet number and the remaining 4 bytes are used to store and IP address. All clients that are running the client tool uses the same packet number sequencing when sending packets. The sequence of packet numbers are stored as 4 bytes of packet number integer. The clients local IP-address is inserted in the packet payload as a way of distinguishing the packets on the receiving server side to be able to keep track of if packets are dropped.

Step5: The insertion of IPv4 traffic into the network is done by using sendto() function to send packets to the destination multicast address in an unconnected socket mode.

```
(sendto(sd, &dataBuffer, 8, 0, (struct sockaddr*)&groupSock, sizeof(groupSock))
```

Step 6: The details about the speed of outgoing packets and duration time, the real duration time of a packet sent can be seen in client output as shown in the figure below. A sleep interval between the packets calculated based on desired client transfer rate.

nanoToSleep = 1000000000 / packetsPerSec

To be able to include the time it took to send the actual packet in the sleep time, a sleep method is used that sleeps on the free clock instead of relative time.

clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &deadline, NULL).

```
2018-02-05 14:58:26 [client6 172.16.9.101] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:04:20 [client6 172.16.9.102] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:02:21 [client6 172.16.9.103] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:01:55 [client6 172.16.9.104] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:01:05 [client6 172.16.9.105] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:01:25 [client6 172.16.9.106] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:00:57 [client6 172.16.9.107] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:00:49 [client6 172.16.9.108] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:03:02 [client6 172.16.9.109] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:01:26 [client6 172.16.9.110] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
2018-02-05 15:02:15 [client6 172.16.9.111] PacketsPerSec=1 Duration=100s NumPacketsSent=100 RealDuration=100.0s RealPacketsPerSec=1
```

Figure 11: Shows how many packets (Speed) are sending using clients to different servers.

By observing the above figure 11, the sent the packets at a speed of 1 packet per sec with a duration time of 100s. Duration time is the number of seconds that the client continues to send packets, regardless of the packet send rate(Speed). The NumPacketsSent is the product of PacketsPerSec and Duration. So, the number of packets sent to the server by a client is 100 packets.

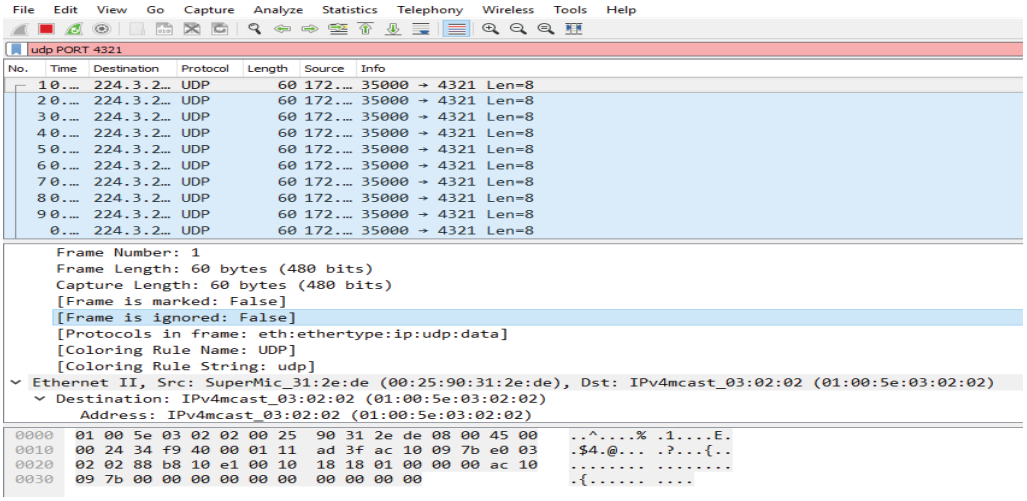


Figure 12: IPv4 Packets captured using Wireshark

The above figure 12 shows the packet size of an IPv4 packet as 60 bytes. Observe that the packet type is UDP, the protocol port used for communication between the client and servers is 4321. The multicast group address is 224.3.2.2 (where the packets are sent).

4.2 IPv4 server tool implementation

Step 1: Server requires two arguments, Server and Server local IP address respectively. The AF_INET is an internet family address with a UDP transport (SOCK_DGRAM) which helps to receive the packets used in this socket. Setsockopt() will set the SO_REUSEADDR to allow multiple applications to receive the copies of datagrams to the multicast group. After that bind to the proper port number with the IP address specified as INADDR_ANY.

Step 2: INADDR_ANY specified to receive datagrams that are addressed to the multicast group. After that joining the multicast group, 224.3.2.2 on the local 10.0.0.1 interface is done on the server side of IPv4 tool. Note that this IP_ADD_MEMBERSHIP option must be called for each local interface over which the multicast datagrams are to be received.

Step 3: If the packets are received successfully without any invalid packets, then the server local IP address number, date and time of the packets received and number of packets received as a server-side output can be observed in the figure below. The below statement represents packets received successfully.

fprintf(stderr, "%02d-%02d-%02d %02d:%02d:%02d [server %s] Number of packets received: %u - %s\r\n", tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min, tm.tm_sec, ipAddress, numPacketsReceived, textBuffer);

```
2018-01-10 16:09:56 [server 172.16.9.101] Number of packets received: 9400
2018-01-10 16:15:01 [server 172.16.9.102] Number of packets received: 9400
2018-01-10 16:13:23 [server 172.16.9.103] Number of packets received: 9400
2018-01-10 16:13:01 [server 172.16.9.104] Number of packets received: 9400
2018-01-10 16:12:14 [server 172.16.9.105] Number of packets received: 9400
2018-01-10 16:12:23 [server 172.16.9.106] Number of packets received: 9400
2018-01-10 16:12:13 [server 172.16.9.107] Number of packets received: 9400
```

Figure 13: Shows how the packets received by the servers from the client

Step 4: This is the window which can be observed in figure 14, figure 15 below. This will show if there are any packet losses and changes in the order of packets on the servers. To detect the packet reordering a 32 packet buffer is used. There is no specific reason for choosing 32 packet buffer size. If a packet is delayed more than 32 packets, it is considered dropped[2].

The below code prints the contents of the packet buffer after the dropped packets are detected. The argument should be the packet buffer and it is the buffer that holds the packets. Reordering means that a packet arrives out of sequence. The loss means the packets does not arrive at all.

void PrintPacketBuffer(unsigned int *the packet buffer, unsigned int thePacketBufferPos)
{

```
unsigned int packetBufferPos = thePacketBufferPos;
for (int i = 0; i < 32; i++) {
        fprintf(stderr, "%d ", thePacketBuffer[packetBufferPos]);
        packetBufferPos = (packetBufferPos == 31) ? 0 : packetBufferPos + 1;
} }
```

```
2018-01-10 16:13:34 [server 172.16.9.111] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 14
2018-01-10 16:12:43 [server 172.16.9.112] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 14
2018-01-10 16:11:44 [server 172.16.9.113] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 14
2018-01-10 16:13:25 [server 172.16.9.114] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 14
2018-01-10 16:13:06 [server 172.16.9.115] Invalid packet number: 17258 (expected 17257) 1 packets dropped from 172.16.9.115 | 17249 17250 17251 17252 17253 17254 17255
2018-01-10 16:12:23 [server 172.16.9.116] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 14
2018-01-10 16:12:46 [server 172.16.9.117] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 14
2018-01-10 16:11:43 [server 172.16.9.118] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 14
2018-01-10 16:11:25 [server 172.16.9.119] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 14
2018-01-10 16:11:34 [server 172.16.9.120] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 14
```

Figure 14: Shows how the packets loss while receiving packets on the servers.

```
172.16.9.189] Invalid packet number: 19976 (expected 19975) 1 packets dropped from 172.16.9.147 | 19967 19968 19969 19970 19971 19972 19973 19974 19976 19977 19978 1997
172.16.9.190] Invalid packet number: 27487 (expected 27486) 1 packets dropped from 172.16.9.112 | 27478 27479 27480 27481 27482 27483 27484 27485 27487 27489 27488 2749
172.16.9.191] Invalid packet number: 7882 (expected 7874) 8 packets dropped from 172.16.9.172 | 7866 7867 7868 7869 7870 7871 7872 7873 7882 7883 7884 7885 7886 7887 78
172.16.9.192] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 142 143 144 145 146 147 148 14
172.16.9.193] Invalid packet number: 30709 (expected 30708) 1 packets dropped from 172.16.9.135 | 30700 30701 30702 30703 30704 30705 30706 30707 30709 30711 30712 3071
172.16.9.194] Invalid packet number: 139 (expected 138) 1 packets dropped from 172.16.9.191 | 130 131 132 133 134 135 136 137 139 140 141 142 143 144 145 146 147 148 14
```

Figure 15: Shows both the packets loss in and packet reordering while receiving packets to servers.

The packet reodering is detected by the help of the following line of code will detect if the index number (Called “Packet number” in the code) of the received packet is not greater than the previously received packet.

```
if (invalidPacketReceived == false && previousPacketNumber[ipIndex] != (packetNumber
- 1) && packetNumber != 1)
```

If this “if-statement” is TRUE then the code prints what index number that was expected.

In the above figure 15, observe the sequence of packets received and the dropped packets. Notice that eight packets are dropped in the form of chunks highlighted on the above figure 18. Also, there is packet reordering, i.e., missing in the packet sequence about that reordering highlighted in above figure 15.

4.3 IPv6 client tool implementation

Step 1: The client part of the tool has three arguments the client local IP address, packets per sec (speed of sending packets in 1 second), a duration time (time is taken for sending in packets). These values are changed according to speed and duration in seconds.

Step 2: In this tool PF_INET6 is an IP_protocol family used. FF02:0:0:0:0:0:0:8 is the IPv6 multicast group used in the tool and 4321 is a port protocol used to send the IPv6 packets to different servers. The *setsockopt()* function is used to set socket options. *IPPROTO_IPV6* is a request applied to IPv6 protocol layer.

Step 3: PacktetsToSend (total number of packets sending to clients) is the product of duration in Sec (time taken for packets to send) and PacketsPerSec (number of packets sent in 1sec). The size of IPv6 packet is 70bytes. The IPv6 packet consists of the payload of 8 bytes. The payload in the packet is divided into IPadressinteger(4bytes) and packet number(4bytes). All clients that are running the client tool uses the same packet number sequencing when sending packets. The client's local IP-address is inserted in the packet buffer for 4 bytes as a way of distinguishing the packets on the receiving server side, to be able to keep track of if packets are dropped.

Step 4: The insertion of IPv6 traffic into the network is done by using the sendto() function which is responsible for the IPv6 mode to send packets to the destination multicast address in an unconnected socket mode.

sendto(sd, &dataBuffer, 8, 0, multicastAddr->ai_addr, multicastAddr->ai_addrlen)

Step 5: CLOCK_MONOTONIC is used in the tool used to update the information regarding the speed of sending packets by placing a sleep interval between the packets were calculated based on the desired speed of packets sent. To be able to include the time it took to send the actual packet in the sleep time, a sleep method is used to sleep on the free clock instead of relative time.

4.4 IPv6 Server tool implementation

Step 1: The server part of a tool passes an argument like server local IP address. Server local IP address acts as the second argument. The PF_INET6 is an internet family address with a UDP transport (SOCK_DGRAM) which helps to receive the packets used in the socket.

Step 2: Setsockopt() will set the SO_REUSEADDR to allow multiple applications to receive the copies of datagrams to the multicast group. After that binding to the proper port number with the IP address FF02:0:0:0:0:0:0:8 specified as INADDR_ANY. INADDR_ANY IP address is binded to a port specified to receive datagrams that are addressed to the multicast group. After that joining of the multicast group on the scope, id interface is done on the server side of IPv6 tool. Note that this IP_ADD_MEMBERSHIP option must be called for each local interface over which the multicast datagrams are to be received.

Step 3: If the IPv6 packets are received successfully to the servers then the server local IP address number, date and time of the packets received and number of IPv6 packets received as an output can be observed in figure 14 like as shown in the IPv4 client. The data buffer is responsible for the packet receiving.

Step 4: The packet loss and packet reordering are observed with the help of a data buffer. When the packets can be received in a different order than they were sent, then it is considered that packet reordering buffer of the 32 latest packets is used. If a packet is delayed more than 32 packets, it is considered dropped like IPv4 tool server. The below code prints the contents of the packet buffer after a dropped packet are detected. The argument should be the packet buffer, and it is the buffer that holds the packets. Reordering means that a packet arrives out of sequence. The loss means the packets do not arrive at all.

```
unsigned int StorePacket(unsigned int *thePacketBuffer, unsigned int thePacketBufferPos,
unsigned int thePacketIndex)
{
    thePacketBuffer[thePacketBufferPos] = thePacketIndex;

    return (thePacketBufferPos == 31) ? 0 : thePacketBufferPos + 1;
}
```

The packet reodering is detected by the help of the following line of code will detect if the index number (Called “Packet number” in the code) of the received packet is not greater than the previously received packet.

```
                    if (invalidPacketReceived == false &&
previousPacketNumber[ipIndex] != (packetNumber - 1) && packetNumber !=
                                  1)
```

If this “if-statement” is TRUE then the code prints what index number that was expected.

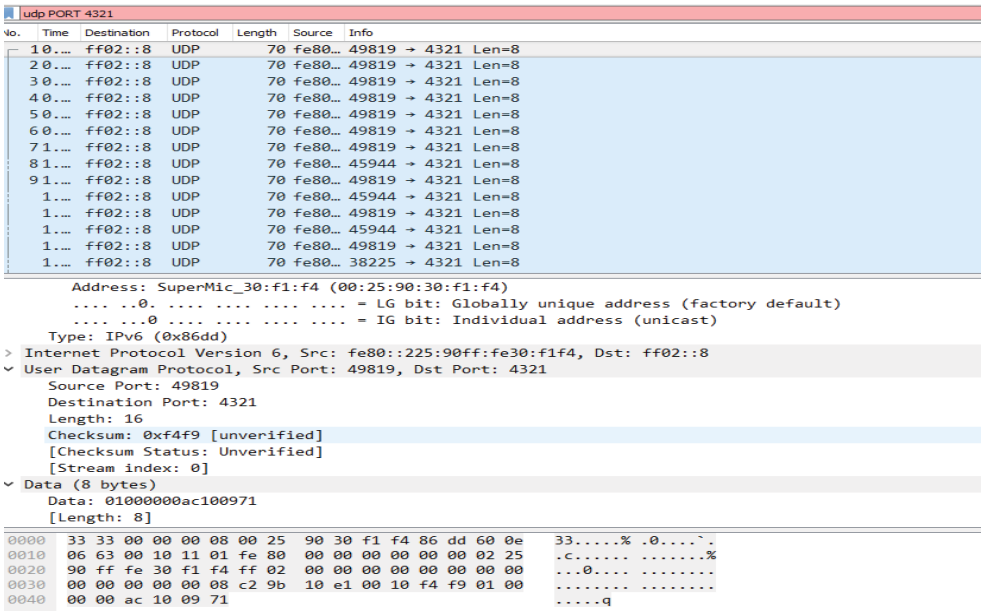


Figure 16: Shows IPv6 packet size and that is captured using Wireshark

4.5 Tool Validation

A sleep interval between the packets was calculated based on the desired client transfer rate. The packet rate which sent by client tool is also checked by Wireshark using AP mirrored wire interface. If a tool is generating the multicasting traffic or not that can be identified by using Wireshark. If a client part of tool is sending multicast traffic and server part of a tool is receiving the multicast traffic is checked by using Wireshark. Wireshark is also used to look the generated multicast traffic. So Wireshark is used to look at the generated multicast traffic.

The server side shows all packet drops and also packet reordering (Packets out of sequence).

Both the IPv4 and IPv6 tools are verified similarly.

5. Measurement Results and Analysis

5.1 Switch Throughput for IPv4 Multicast

In IPv4 multicast, the throughput of a switch can be identified by running the automation scripts. The first step of the testing process should run the scripts which will upload the server and client codes. After that, check clients and servers using the check server and

check client scripts to find how many clients are sending the packets and how many servers are ready to receive the packets. To clear the previously obtained results, clear_all_logs scripts are used. After clearing all the previous results, start the server. After running the start_server script, later run the client script to start the client. After starting the server and the client, wait for some time because it will take some time to send and receive the packets between the clients and the attentive servers. Later download the server and client results which are obtained by running the download server and download client scripts as discussed previously in other section.

Packets /sec	Packets/Sec(Per Switch)	Packets/Sec(2 switches)	Valid/Invalid Packets
0	0	0	Valid
1	4559	9118	Valid
5	22795	45590	Valid
10	45590	91180	Valid
50	227950	455900	Valid
100	455900	911800	Valid
200	911800	1823600	Valid
300	1367700	2735400	Valid
400	1823600	3647200	Valid
450	2051550	4103100	Valid
460	2097140	4194280	Invalid
500	2279500	4559000	Invalid
750	3419250	6838500	Invalid
1000	4559000	9118000	Invalid

Table 9: Shows the valid/invalid IPv4 packets transmitted through Switch.

During the initial stages of testing, while sending packets with a rate of 1 packet per second, it was observed that there are no invalid packets present in server results (server-side), as shown in Table 8. An Invalid packet consists of both reordered packets and lost packets. In the next test iteration, the rate was increased to 5 packets per second. Even during this case, there are no invalid packets in the servers, which means there is no packet loss or packet reordering is observed at server side.

However, Later the entire experiment is done with 95 servers with the help of two switches. Later packets are sent with a speed of 100, 200, 300, 400 packets per second, then there are

no invalid packets while sending the packets at a different speed from different clients to the different interested servers.

By Increasing the speed of packets sent, to a speed of 500 packets per second, there is a presence of invalid packets on the server. A presence of packet loss and also packet reordering. After that send packets with a speed of 450 packets per second then there is a valid packet observed in the same servers. Again, sending the packets with a speed of 460 packets per second, there are invalid packets present in the servers. Same invalid packet has been dropped in the multiple servers.

After that send packets with a speed of 750 and 1000 packets per second, which results in more invalid packets, i.e., more packets loss and packet reordering are observed on the servers. So, we can say that while sending packets at a speed of 450 packets per second, 95 times to 95 servers by using two switches. The maximum number of packets two switches can handle is 4103100. A switch contains 48 ports, of which 47 ports are connected to servers, and one port is connected to the second switch. So maximum packets a switch can handle is 2051550 packets/sec which means 985.7Mbps~1Gbps.

Threshold discovery is made for a switch to identify a maximum number of packets a switch can handle without packet loss. When 450 packets per second are sent by using 95 servers, the two switches can handle a rate of 4103100~2Gbps. After testing observed that packet numbering (numbering on the packets) is an excellent way to detect the multicast packet loss and packet reordering on a server, due to an overloaded network or switch-like as we have seen in above figure 17.

The IPv4 packets are dropped both in the form of chunks and also the packets are dropped by second. At low speed, the packets are dropped in the form of drop by second and at high speed, the packets are dropped in the form of chunks. The below graph and table show about the patterns of dropped packets at a certain speed.

Speed	Packets Dropped
0	0
1	0
5	0
10	0
50	0
100	0
200	0
300	0
400	0
450	0
460	1

500	1
750	2
1000	4
1250	5
1500	5
1750	6
2000	7

Table 10: Shows the pattern of dropped IPv4 packets when speed increases

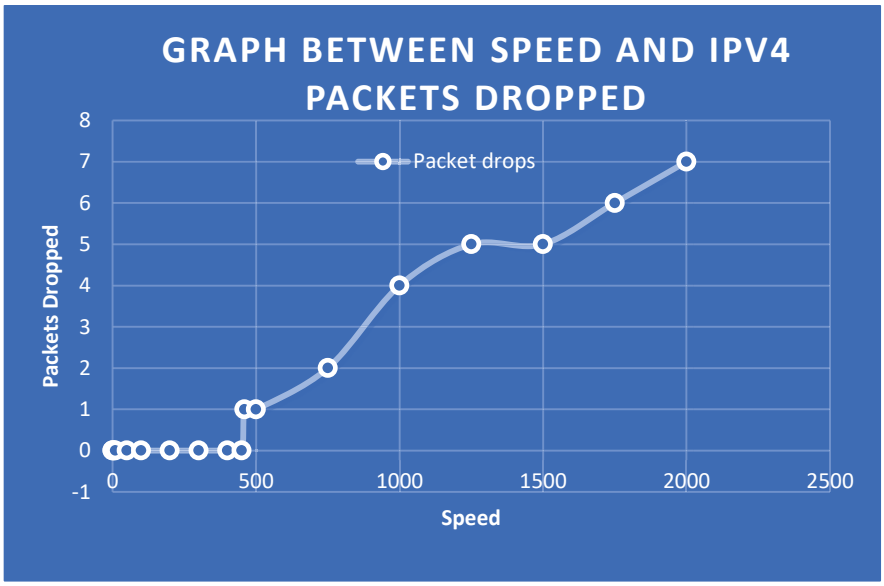


Figure 17: Shows a graph between the pattern of dropped IPv4 packets and speed increases

5.2 Switch Throughput for IPv6 Multicast

Like IPv4 multicast throughput for IPv6 multicast can also be identified by running the test scripts. Before starting the tests, find the maximum number of packets a switch can handle by sending the packets with a speed of 1 packet per second from a client with a duration time of 30 seconds to servers. If not found any invalid packets by sending with a speed of 1 packet per second from the client, then later send packets with a speed of 10 packets per second from a client with a duration time of 30 seconds to servers.

When observing the server and client logs, if there are no invalid packets on the server side, it means switch can handle the 10 packets per second when packets are sent with a duration time of 30 seconds. After that using client sending packets with increasing speed of 60 packets per second results in packet loss. Later sending the packets with a speed of 40,50 packets per second then did not result in invalid packets. Later sending packets with a speed of 51 packets per second to all 95 servers, resulted in invalid packets which are reported in

the server log file. So, we can say that a switch can handle a speed of 51 packets per second and a duration time of 30 packets per second.

After that sending packets with a speed of 51 packets per second with a duration time of 30 seconds from clients for all the interest servers. This resulted in some invalid packets in server side. Invalid packets are found in the server logs when clients from 172.16.9.101 to 172.16.9.153 send packets to servers. But invalid packets are found in server logs from 172.16.9.154 to 172.16.9.195, when these clients send packets to different servers. These servers used IPv6 link-local addresses. This means that a switch can handle a maximum speed of 50 packets per second when the packets are sent with a duration time of 30 seconds.

After that sent with a speed of 50 packets per second using clients to all the interested servers. Then the results obtained are, there are no invalid packets present in the server logs from 172.16.9.101 to 172.16.9.148 clients. When the 172.16.9.149 client sends packets to different servers, then found some errors in the file. From 172.16.9.150 to 172.116.9.155 clients send the packets to all the servers, there are no invalid packets in the server logs. When 172.16.9.156 to 172.16.9.195 clients send the packet to servers, then many invalid packets are present in the server logs. So, we can say that the maximum number of packets a switch can handle is 110450 packets per second.

Sending packets with a speed of 100 packets per second with a duration time of 30 seconds then many invalid packets are present in the server logs.

Speed(packet/sec)	Packets/sec (Switch)	Packets/sec (2 Switches)	VALID/INVALID PACKETS
1	4559	9118	No invalid packets
10	45590	91180	No invalid packets
20	91180	182360	No invalid packets
30	136770	273540	No invalid packets
40	182360	364720	No invalid packets
50	227950	455900	No invalid packets
51	232509	465018	Invalid packets
60	273540	547080	Invalid packets
70	319130	638260	Invalid packets
80	364720	729440	Invalid packets
90	410310	820620	Invalid packets
100	455900	911800	Invalid packets

Table 11: Shows the valid/invalid IPv6 packets transmitted through Switch

The above table represents the number of invalid packets and valid packets a switch can handle by using servers and clients. Traffic (450pkt/s) from 47 clients sent to the multicast address. Each such packet is sent to 47 servers (each listening on a different port) and one packet is sent over port 48 to reach servers connected to the next switch (switch 2). So,

each packet to the multicast address is replicated 48 times. There will be 47*450 pkt/s arriving in port 48 (from the 47 clients on switch 2). Each of these packets will be forwarded to the 47 servers on switch 1. Therefore, the total number of packets carried by switch 1 is 2051550 packets/sec which means 985.7 Mbps/sec ~ 1Gbps. The total amount of traffic carried by both the switches without any packet loss is 455900 packets/sec.

The IPv6 packets are dropped both in the form of chunks and also the packets are dropped by second. At low speed the packets are dropped in the form of drop by second and at high speed the packets are dropped in the form of chunks. The below graph and table show about the patterns of dropped packets at certain speed.

Speed (Packets/Sec)	Packets Dropped Sequence
1	0
10	0
20	0
30	0
40	0
50	0
60	1
70	1
80	1
90	1
100	2
250	3
500	4
750	6
1000	8

Table 12: Shows the pattern of dropped IPv6 packets when speed increases

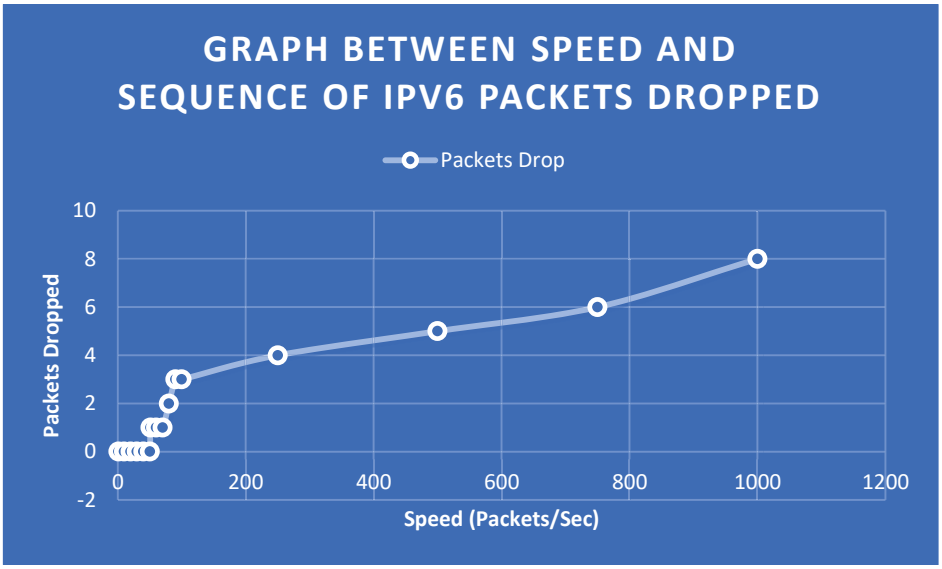


Figure 18 : Shows a graph between the pattern of dropped IPv6 packets and speed increases

5.3 CPU Utilization on Servers for IPv4 multicast

The reason for choosing SAR (System Activity Reporter) is an important tool that helps system administrators to get an overview of the server machine with the status of different important metrics at different points of time. The CPU utilization of a switch in IPv4 multicast is measured by the help of using a Linux command, i.e.;
_ Sar -u 2

The above command will display the CPU usage for every 2 seconds.

```
[root@ATOM9-102 ~]# sar -u 2
Linux 3.18.3.1 (ATOM9-102)      09/04/2017      _x86_64_        (4 CPU)

05:29:55 PM      CPU      %user      %nice    %system    %iowait    %steal      %idle
05:29:57 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:29:59 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:01 PM      all       0.12       0.00       0.12       0.00       0.00      99.75
05:30:03 PM      all       0.75       0.00       0.75       0.00       0.00      98.50
05:30:05 PM      all       0.75       0.00       0.12       0.00       0.00      99.13
05:30:07 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:09 PM      all       2.00       0.00       1.00       0.00       0.00      97.00
05:30:11 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:13 PM      all       0.00       0.00       0.12       0.00       0.00      99.88
05:30:15 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:17 PM      all       0.00       0.00       0.12       0.00       0.00      99.88
05:30:19 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:21 PM      all       0.12       0.00       0.00       0.00       0.00      99.88
05:30:23 PM      all       0.00       0.00       0.12       0.00       0.00      99.88
05:30:25 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:27 PM      all       0.00       0.00       0.12       0.00       0.00      99.88
05:30:29 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:31 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:33 PM      all       0.00       0.00       0.12       0.00       0.00      99.88
05:30:35 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:37 PM      all       0.00       0.00       0.00       0.00       0.00     100.00
05:30:39 PM      all       0.00       0.00       0.12       0.00       0.00      99.88
05:30:41 PM      all       0.12       0.00       0.00       0.00       0.00      99.88
05:30:43 PM      all       0.00       0.00       0.12       0.00       0.00      99.88
```

Figure: 19 Shows the CPU utilization of a severs while sending IPv4 packets

The above command which is used to measure the CPU utilization of servers. The following parameters are shown while running this command like as shown in figure 13.

%user is the CPU utilization percentage that will take place while executing at the user level.**% nice** is the CPU utilization percentage that will take place while executing at the user level with placing priority as nice.**% system** is also a CPU utilization percentage that will take place while executing at the kernel system level.**%iowait** is time percentage the CPU is idle during while system had an outstanding disk I/O request.**% Idle** is also a time percentage the CPU is idle while the system did not have an outstanding disk I/O request. So here considered the %user, %nice, %system CPU utilisation values and added them for CPU utilization of a server.

To measure the CPU utilization automation scripts like sever upload script, client upload script, clear all the results script, start the server, start the client, download server result script and download client result script should be used. The advantage of this scripts is discussed earlier. To measure the CPU utilization first we have to upload the server and client using the upload server script, upload client script into a virtual environment. After that, clear the previous results which are downloaded by the editor using the clear all logs script. Run the Linux command which is used to measure the CPU utilization simultaneously while running the start server script and start client script. The start server script is used to start the server and start client script is used to start the client. Measure the CPU utilization values after starting the server and client, until the CPU utilization values are getting lower. When measuring the CPU utilization of a server after starting the client and the server wait for sometime until the server gives the lower CPU utilization values.

After getting the lower values in CPU utilization, stop running the command then consider the average value. Before uploading the client, change the speed of the client. Considering the speed, measure the different CPU utilization values.

The below Table 13 shows CPU utilization values of a server at a different speed for 11 times repeatedly measured server CPU utilization Mean value, standard deviation values, 95% confidence interval values are calculated and there is no specific reason for measuring 11 times. CPU utilization for all the other servers is equivalent to the results below.

Speed(packets/sec)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	Mean	sd	95%CI
0	2.2	2.1	2.2	2.3	2.1	2.2	2.1	2.2	2.2	2.2	2.3	2.3	0.89527	0.529
1	2.3	2.3	2.4	2.3	2.3	2.3	2.3	2.2	2.3	2.3	2.4	2.30909	0.05394	0.0319
5	2.5	2.4	2.5	2.5	2.6	2.5	2.4	2.5	2.5	2.5	2.6	2.5	0.06325	0.0374
10	2.6	2.5	2.6	2.6	2.7	2.6	2.5	2.6	2.6	2.6	2.7	2.6	0.06325	0.0374
50	2.9	2.8	2.9	2.8	2.9	2.8	2.9	2.9	2.9	2.8	2.9	2.86364	0.05045	0.0298
100	3.2	3.2	3.1	3.2	3.2	3.3	3.2	3.3	3.2	3.2	3.4	3.2272	0.07862	0.0465
200	4.2	4.1	4.2	4.2	4.2	4.3	4.2	4.2	4.1	4.2	4.3	4.2	0.06325	0.0374
300	5.3	5.4	5.2	5.3	5.2	5.3	5.3	5.4	5.3	5.2	5.3	5.29091	0.07006	0.0414
400	8.6	8.5	8.5	8.6	8.7	8.6	8.6	8.7	8.6	8.5	8.6	8.59091	0.07006	0.0414
450	8.9	9	8.9	9.1	8.9	8.9	8.8	8.9	9	8.9	8.9	8.92727	0.07862	0.0465
460	9.7	9.8	9.7	9.7	9.6	9.7	9.6	9.7	9.8	9.7	9.7	9.7	0.06325	0.0374
500	11.6	11.4	11.7	11.6	11.7	11.6	11.6	11.5	11.6	11.6	11.7	11.6	0.08944	0.0529
750	16.8	16.9	16.8	16.9	16.8	16.7	16.8	16.7	16.9	16.7	16.9	16.8091	0.08312	0.0491
1000	23.4	23.5	23.4	23.3	23.6	23.4	23.4	23.5	23.4	23.6	23.4	23.4455	0.09342	0.0552

Table 13: Shows measured CPU utilization of Server and Speed while sending IPv4 packets.

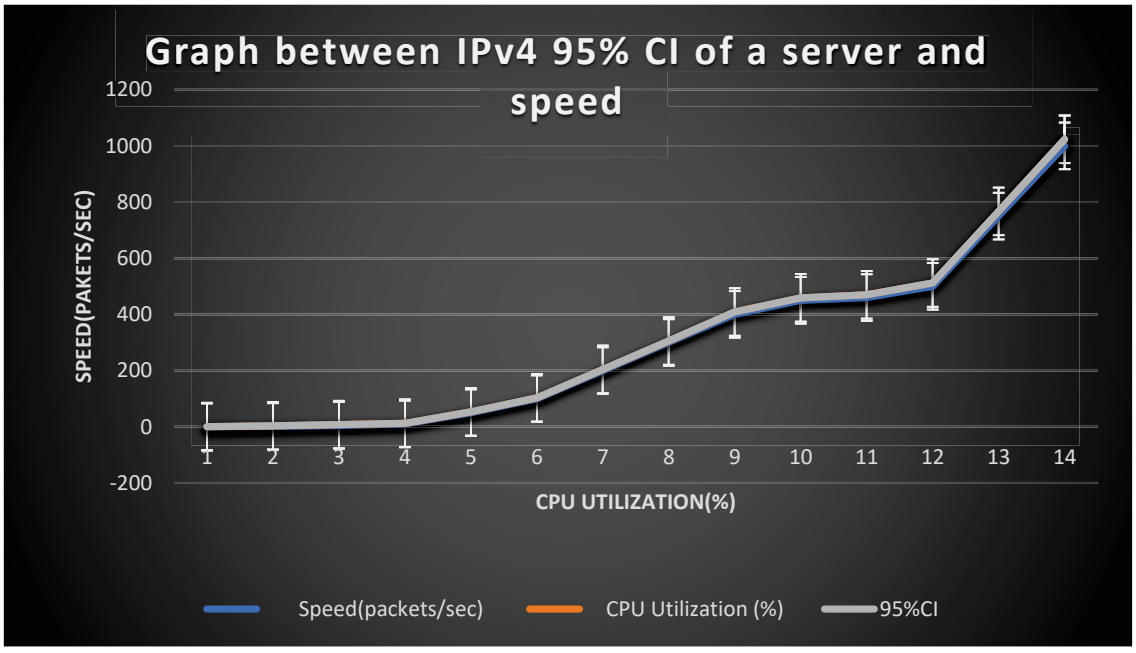
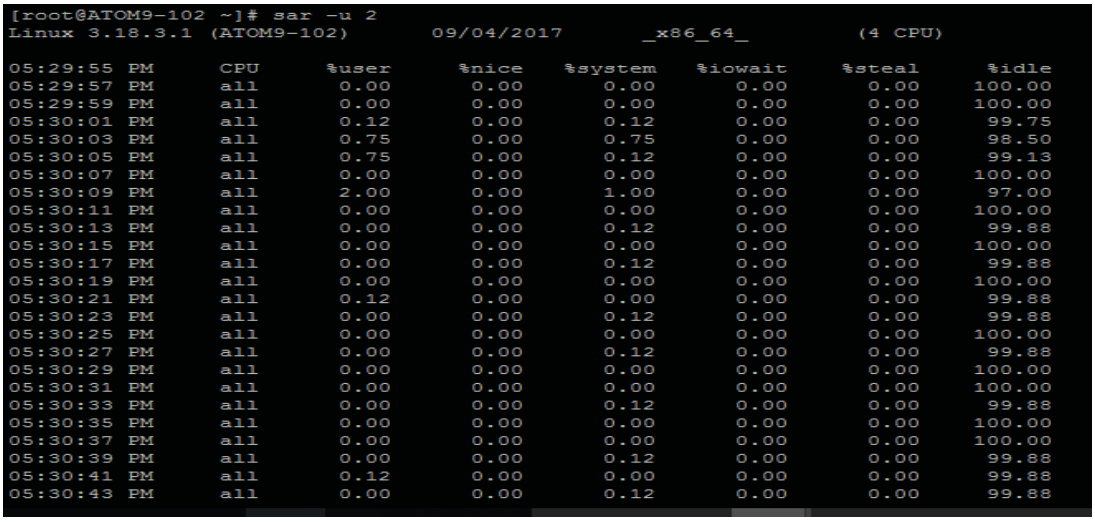


Figure 20: Graph between CPU utilization of Server and Speed while sending IPv4 packets.

5.4 CPU Utilization on servers for IPv6 Multicast

As IPv4 multicast the CPU utilization of a switch in IPv6 multicast is measured by the help of using a Linux command, i.e.;

_ Sar -u 2



```
[root@ATOM9-102 ~]# sar -u 2
Linux 3.18.3.1 (ATOM9-102)      09/04/2017      _x86_64_        (4 CPU)

05:29:55 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
05:29:57 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:29:59 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:01 PM     all      0.12      0.00      0.12      0.00      0.00     99.75
05:30:03 PM     all      0.75      0.00      0.75      0.00      0.00     98.50
05:30:05 PM     all      0.75      0.00      0.12      0.00      0.00     99.13
05:30:07 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:09 PM     all      2.00      0.00      1.00      0.00      0.00     97.00
05:30:11 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:13 PM     all      0.00      0.00      0.12      0.00      0.00     99.88
05:30:15 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:17 PM     all      0.00      0.00      0.12      0.00      0.00     99.88
05:30:19 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:21 PM     all      0.12      0.00      0.00      0.00      0.00     99.88
05:30:23 PM     all      0.00      0.00      0.12      0.00      0.00     99.88
05:30:25 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:27 PM     all      0.00      0.00      0.12      0.00      0.00     99.88
05:30:29 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:31 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:33 PM     all      0.00      0.00      0.12      0.00      0.00     99.88
05:30:35 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:37 PM     all      0.00      0.00      0.00      0.00      0.00    100.00
05:30:39 PM     all      0.00      0.00      0.12      0.00      0.00     99.88
05:30:41 PM     all      0.12      0.00      0.00      0.00      0.00     99.88
05:30:43 PM     all      0.00      0.00      0.12      0.00      0.00     99.88
```

Figure 21: This is the command which is used to measure the CPU utilization.

SAR stands for System Activity Report, as its name suggests sar command is used to collect, report & save CPU, Memory, I/O usage in Unix like operating system.

To measure the CPU utilization Test scripts (Bash scripts) should be used.

To measure the CPU utilization first run the upload server script and upload client script to upload the server code and upload the client code respectively. Clear the previous results which are downloaded and viewed using the clear all logs script. Run the Linux command to measure the CPU utilization of server simultaneously while running the start server script and start client script. This start server script is used to start the start the server for receiving the packets from clients and start client script is used to start the client for sending the packets to attentive servers. Measure this CPU utilization after starting the server and client until the CPU utilization values are getting lower.

To find the average value of CPU utilization, wait for some time (to transfer the packets from clients to servers it takes some time) after starting the client and after starting the server until it gives the lower values. After getting the lower values in CPU utilization, stop running the command then consider the average value. The average value is by sending the packets using the client program to the servers. Simultaneously run the sar command while sending the packets to the interested servers. The command should stop running after sending the packets. While running the command, we can see the values related to CPU

utilization. After stopping the command, will result in average value. This process has been repeated 11 times to account variability. The 95% confidence interval and standard deviation can be seen in table 14.

Before uploading the client, we have to change the speed of outgoing packets by using the client. Measure the CPU utilization of servers at different speeds.

In IPv6 multicast, the following parameters are shown while running this command like as shown in figure 25. **%user** is the CPU utilization percentage that will take place while executing at the user level.**% nice** is the CPU utilization percentage that will take place while executing at the user level with placing priority as nice.**% system** is also a CPU utilization percentage that will take place while executing at the kernel system level.**%iowait** is time percentage the CPU is idle during while system had an outstanding disk I/O request.**% Idle** is also a time percentage the CPU is idle while the system did not have an outstanding disk I/O request. %iowait is always remained zero while measuring CPU utilization So here considered the %user, %nice, %system CPU utilization values and added them for CPU utilization of a server.

The below table 14 shows the different CPU utilization values of the server at a different speed.

Speed	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU	CPU	CPU	mean	SD	95% CI
0	1.2	1.22	1.21	1.2	1.23	1.22	1.21	1.2	1.21	1.2	1.2	1.20909	0.01044	0.00617
10	2.73	2.72	2.71	2.72	2.71	2.72	2.71	2.71	2.72	2.71	2.71	2.71545	0.00688	0.00406
20	3.16	3.15	3.15	3.14	3.14	3.15	3.14	3.14	3.14	3.15	3.14	3.14545	0.00688	0.00406
30	3.97	3.97	3.95	3.98	3.99	3.98	3.99	3.95	3.99	3.98	3.99	3.97636	0.01502	0.00887
40	4.32	4.33	4.35	4.35	4.35	4.34	4.35	4.34	4.33	4.35	4.34	4.24091	0.01044	0.00617
50	5.45	5.46	5.45	5.44	5.45	5.44	5.45	5.46	5.45	5.46	5.45	5.5091	0.00701	0.00414
51	5.48	5.47	5.47	5.48	5.47	5.47	5.48	5.47	5.48	5.47	5.47	5.47364	0.00505	0.00298
60	5.88	5.82	5.86	5.88	5.87	5.87	5.83	5.86	5.85	5.87	5.87	5.86	0.01949	0.0115
70	6.37	6.36	6.36	6.37	6.36	6.35	6.36	6.35	6.36	6.36	6.36	6.36	0.00632	0.00374
80	7.65	7.65	7.65	7.65	7.65	7.66	7.65	7.64	7.65	7.66	7.65	7.65091	0.00539	0.00319
90	8.1	8.2	8.2	8.1	8.1	8.1	8.1	8	8.1	8.1	8.1	8.10909	0.05394	0.0319
100	9.35	9.33	9.35	9.36	9.32	9.34	9.34	9.34	9.35	9.33	9.34	9.34091	0.01136	0.00671

Table 14: shows the different CPU utilization values of a server when IPv6 packets are sent at a different speed.

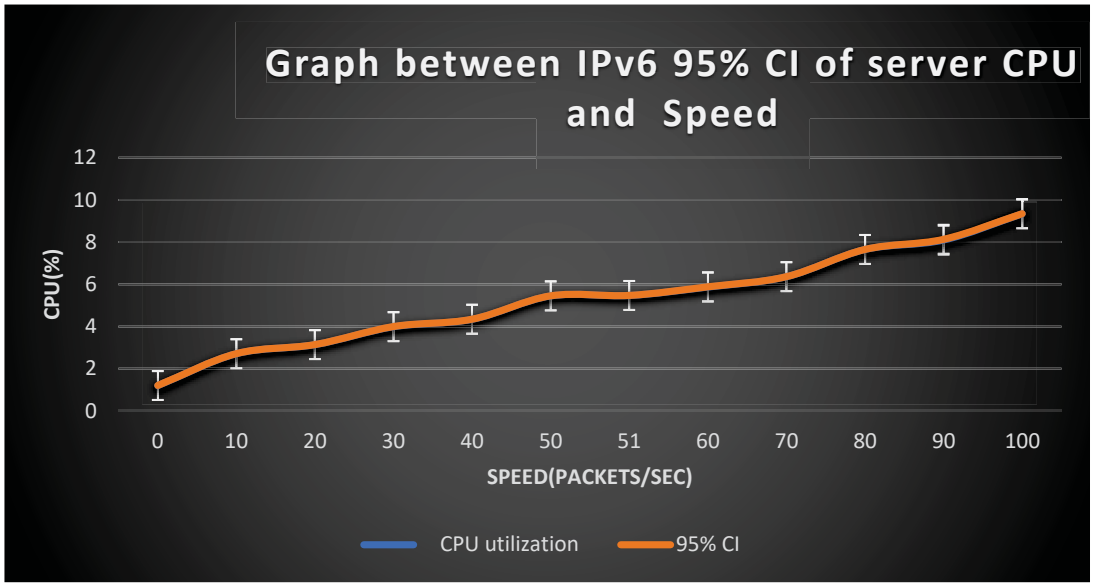


Figure 22: Graph Between CPU utilization of server and Speed while sending IPv6 Packets

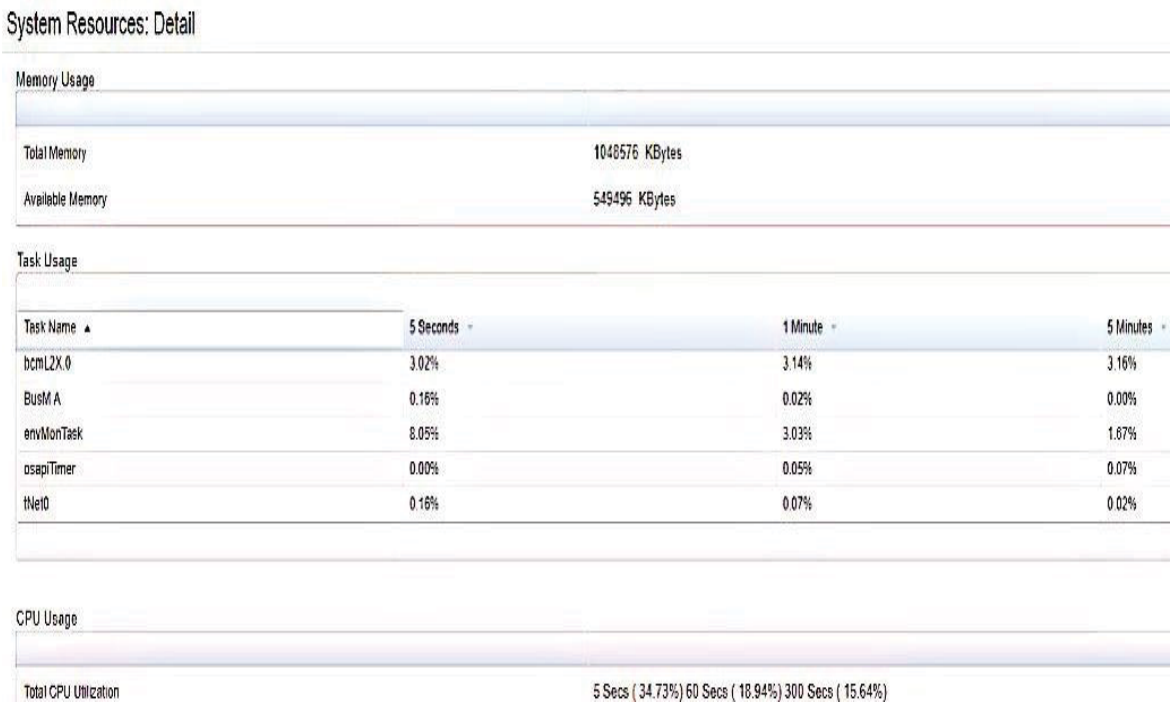
From the above Figure 22, observed that when the speed of packets sent by using a client increases, then the CPU utilization is also increased. However, as the transfer rate increases the CPU utilization should increase linearly with the speed of packet sent.

5.5 CPU Utilization on switch for IPv4 multicast

The CPU utilization of a switch is measured by using the Dell switch open switch administrator with the help of switch IP address. To measure CPU utilization of a switch, login into the Dell open manage switch administrator by using the switch IP address. Later click system option after that click on general and in that general click on system resources.

System ⟶ General ⟶ System Resources

After opening the system resources, we can see task usage and CPU usage and memory usage as as shown in figure 23.



System Resources: Detail

Memory Usage

Total Memory	1048576 KBytes
Available Memory	549496 KBytes

Task Usage

Task Name ▲	5 Seconds	1 Minute	5 Minutes
bcmL2X.0	3.02%	3.14%	3.16%
BusM A	0.16%	0.02%	0.00%
envMonTask	8.05%	3.03%	1.67%
osapiTimer	0.00%	0.05%	0.07%
tNet0	0.16%	0.07%	0.02%

CPU Usage

Total CPU Utilization	5 Secs (34.73%) 60 Secs (18.94%) 300 Secs (15.64%)

Figure 23: Shows the details about the CPU utilization of switch

The CPU utilization of switch is measured while sending the IPv4 packets at different rates, by starting the clients to send the packets to the servers. In that switch administrator, we can see the percentage of CPU utilization of a switch in system resources of Dell switch administrator. At some point, while measuring packets sent to servers, when the CPU utilization is high, then consider that CPU utilization for 60 seconds. The CPU utilization of the switch is increased when the speed of outgoing packets is increased like as shown in graph and CPU utilization values of the switch while sending packets at different rates can be shown by using a table 15.

Speed (packets/sec)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	Mean	SD	95% CI
0	9.2	9.3	9	9.4	9.2	9.2	9.3	9.2	9.1	9.3	9.2	9.218182	0.107872	0.0637
1	11.7	11.8	11.7	11.6	11.8	11.7	11.7	11.6	11.5	11.7	11.7	11.68182	0.087386	0.0516
5	17.1	17.3	17.2	17.2	17.1	17.3	17.1	17.1	17.3	17	17.1	17.16364	0.102691	0.0607
10	19.1	19.1	19.3	19.2	19.1	19.1	18.9	19.2	19.1	19.3	19.1	19.13636	0.112006	0.0662
50	20.3	20.4	20.4	20.2	20.1	20.3	20.4	20.2	20.3	20.2	20.3	20.28182	0.098165	0.058
100	23	23.2	23.1	23	23.6	23.4	23.2	23	23	23.4	23	23.17273	0.210195	0.124
200	24.5	24.4	24.7	24.6	24.5	24.5	24.6	24.4	24.5	24.5	24.4	24.50909	0.094388	0.0558
300	26.3	26.4	26.2	26.1	26.3	26.2	26.2	26.3	26.4	26.3	26.3	26.27273	0.090453	0.0535
400	26.9	26.8	26.9	26.7	26.8	26.8	26.9	26.9	26.8	26.7	26.9	26.82727	0.078625	0.0465
450	27.8	27.4	27.2	27.8	27.7	27.6	27.8	27.8	27.7	27.8	27.8	27.67273	0.200454	0.118
460	28.8	28.6	28.7	28.9	30.1	28.8	28.9	28.8	28.8	28.7	28.6	28.88182	0.416697	0.246
500	29.2	29.1	29.3	29.2	29.2	29.4	29.2	29.3	29.2	29.3	29.1	29.22727	0.090453	0.0535
750	30.1	30	30.2	30.1	30.2	30.1	30.3	30.1	30.2	30.3	30.2	30.16364	0.092442	0.0456
1000	32.7	32.4	32.6	32.7	32.7	32.8	32.7	32.5	32.7	32.7	32.6	32.64545	0.112815	0.0667

Table 15: Representing Speed and CPU utilization of Switch

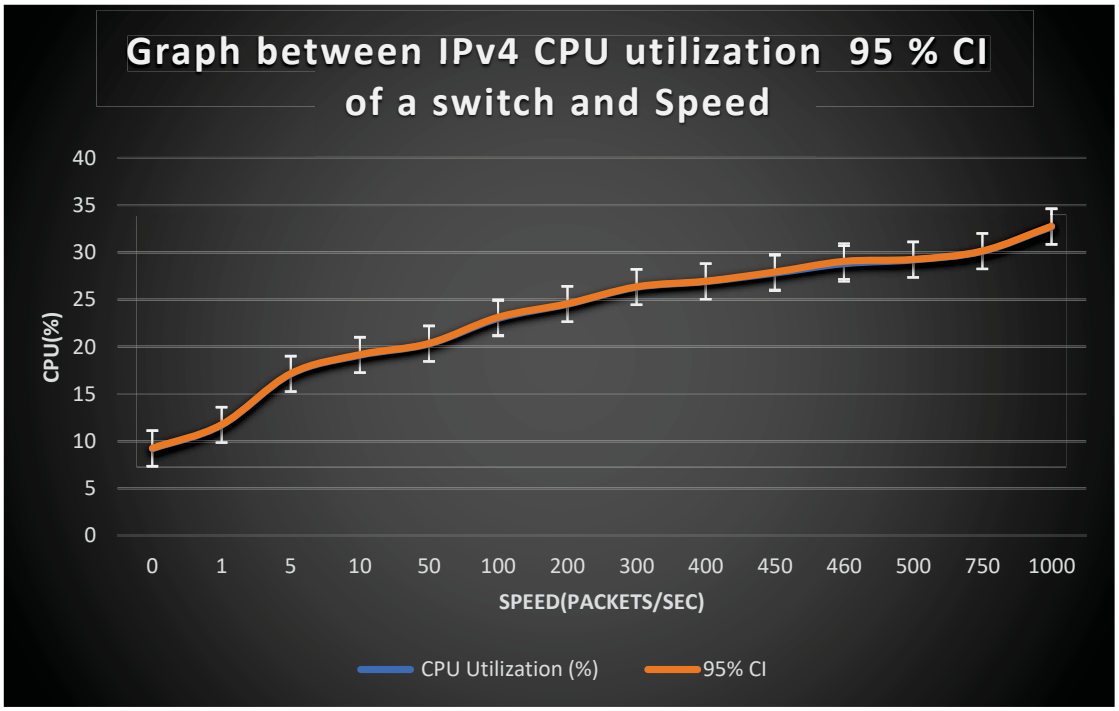


Figure: 24 Graph Between IPv4 CPU utilization of switch and Speed

5.6 CPU Utilization of switch for IPv6 multicast

The CPU utilization of a switch is measured by using the Dell switch open switch administrator. To measure CPU utilization of a switch, Login into the Dell open manage switch administrator as shown in the figure. Later click system option after that click on general and in that general click on system resources.

System ⟶ General ⟶ System Resources

After opening the system resources, see task usage and CPU usage and memory usage like as shown in the figure 25.

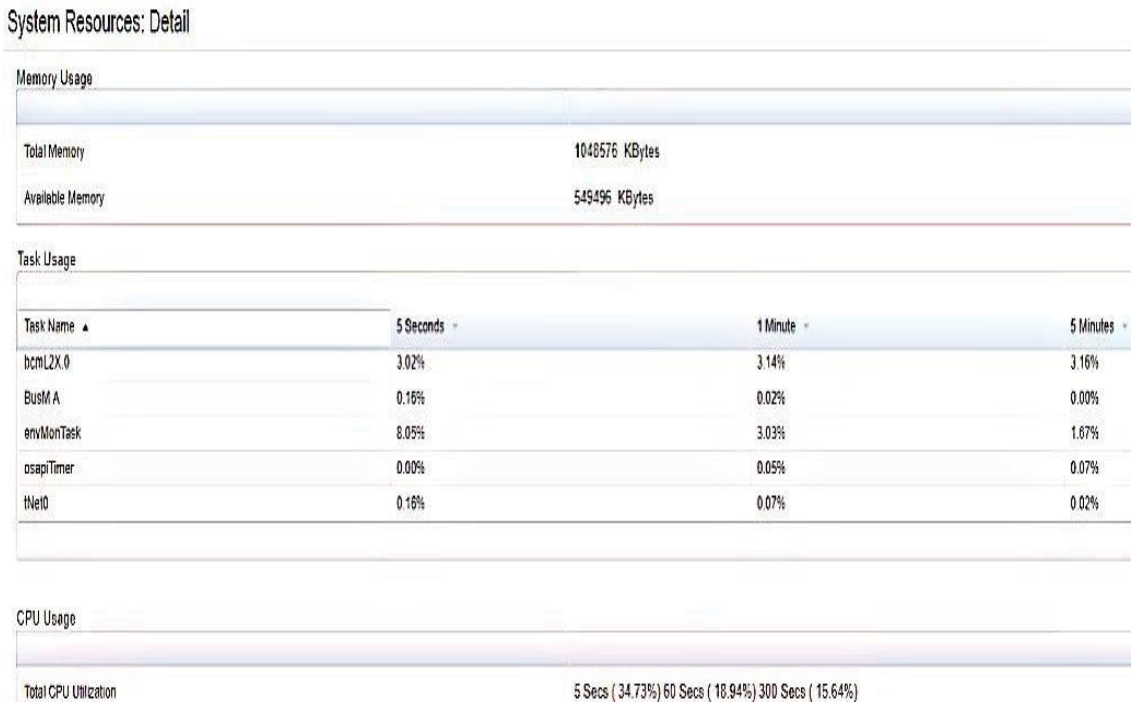


Figure 25: Shows the details about the CPU utilization of switch

The CPU utilization of switch is measured simultaneously while sending the IPv6 packets at different speeds to the attentive servers by using 95 clients. To find the CPU utilization of a switch start the clients and servers in order to send the IPv6 packets from clients to the interested receivers. The CPU utilization of the switch is increased when the speed of sending packets is increased like as shown in the graph and CPU utilization values of the switch while sending packets at different speed can be shown by using a table. In the below table the CPU utilization of switch is measured 11 times.

Speed(packets/sec)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	CPU (%)	Mean	SD	CI 95%
0	39.6	39.4	39.5	39.6	39.2	39.5	39.6	39.4	39.5	39.6	39.6	39.5	0.12649	0.0747
10	40.5	40.5	40.8	40.4	40.5	40.7	40.5	40.8	40.6	40.6	40.7	40.6	0.13416	0.0793
20	44.8	44.4	44.3	44.8	44.8	44.6	44.6	44.4	44.3	44.8	44.8	44.6	0.21448	0.127
30	53.6	53.2	53.7	53.6	53.6	53.3	53.4	53.6	53.5	53.6	53.3	53.4909	0.16404	0.0969
40	58.1	58.6	58.9	58.2	58.1	58.1	58.5	58.2	58.4	58.5	58.5	58.3727	0.25726	0.152
50	61	61.2	61.1	61.1	61	61	61.4	61.8	61	61.4	61.4	61.2182	0.2562	0.151
51	63.5	63.7	63.3	63.5	63.5	63.4	63.6	63.6	63.2	63.4	63.5	63.4727	0.14206	0.084
60	67.3	67.3	67.4	67.8	67.2	67.4	67.5	67.3	67.5	67.3	67.3	67.3909	0.16404	0.0969
70	70.9	70.7	70.9	70.8	70.8	70.2	70.7	70.9	70.9	70.7	70.9	70.7636	0.20627	0.122
80	73.8	73.3	73.5	73.8	73.8	73.2	73.9	73.1	73.8	73.6	73.8	73.6	0.28284	0.167
90	77.8	77.2	77.3	77.8	77.4	77.4	77.2	77.4	77.8	77.6	77.8	77.5182	0.24827	0.147
100	80.4	80.5	80.6	80.7	80.7	80.4	80.4	80.3	80.1	80.7	80.4	80.4727	0.19022	0.112

Table 16: Showing 11 times measured CPU utilization of Switch at a different speed.

The CPU utilization of switch is measured for different speeds from 0 to 100 packets per second.

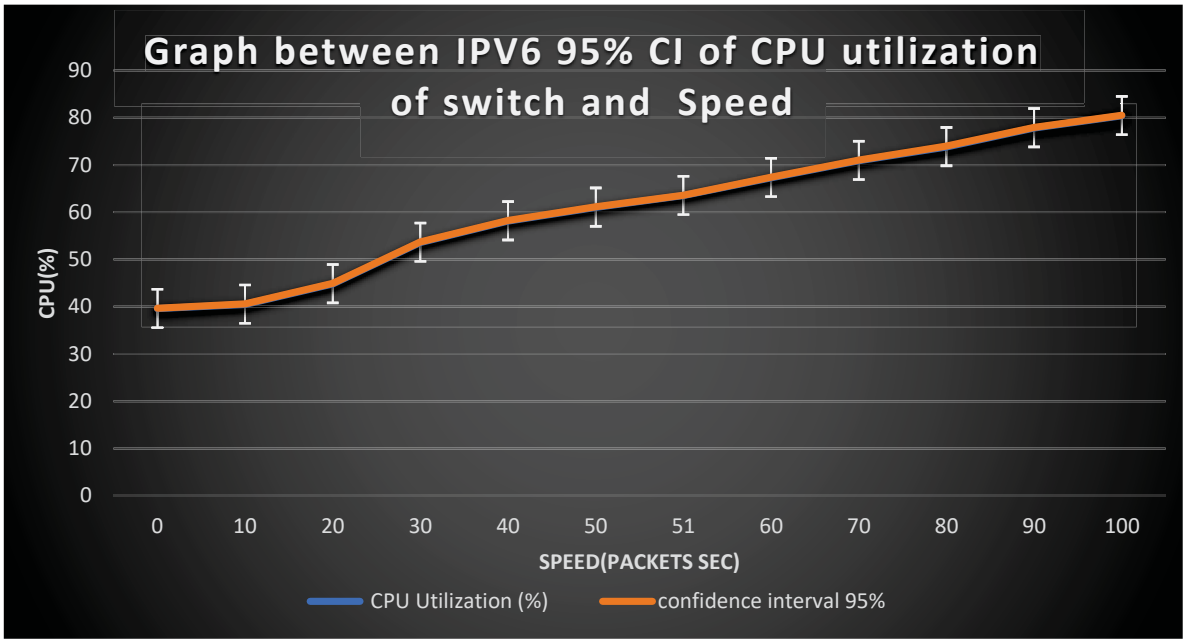


Figure 26: Shows the graph between IPv6 CPU of switch and Speed

6.Conclusion and Future work

The research work mainly focuses on the evolution of multicast behavior in a congested network.

The below are the research questions which are related to this research.

1. What is a good way for a server to detect that multicast packet loss has occurred due to an overloaded network or switch?

The sending the packets in the numbered form is the good way for a server to detect the packet loss has occurred due to an overloaded network or switch. The numbered form of packets while sending packets to servers is shown in figure 10. The both the IPv4 and IPv6 packets are detected in the servers when the packets are sent in the numbered form.

2. What is the pattern of dropped packets when the switch gets overloaded? For example, does the switch drop every second packet or does it drop chunks of packets?

The pattern of the dropped packets is, both in the form of every second packet and in the form of chunks. The packets are dropped in the form of every second when we send the packets at low speed. The packets are dropped in the forms of chunks when sent with high speed. The table 10, table 12, figure 17, figure 18 gives the information about the pattern of packets drops when speed increases.

3. Is IPv6 a more efficient protocol than Ipv4 when it comes to sending multicast packages at a very high rate?

This experiment could not conclude on IPv6 was a more efficient protocol than IPv4. The reason for concluding IPv4 as more efficient because of excessive CPU utilization in Linux when IPv6 was used. Even though that the latest available Linux kernel version was used, the CPU utilization was abnormally high. But did not observed which CPU utilization component is high while measuring the CPU utilization.

Comparison between IPv4 multicast results with IPv6 multicast results

By using the two switches, 95 servers, and 95 clients are used for generating IPv4 multicast packets traffic and also IPv6 multicast packets traffic to interested servers. The maximum number of IPv4 packets a switch can handle efficiently without any packet loss is 2051550 IPv4 packets per seconds = 2051550*60bytes = 123,1 Mbytes/s = 985,7Mbps ~ 1Gbps.

Like IPv4, multicast IPv6 multicast (sending of IPv6 packets to interested receivers) also multicasting is done by using two switches, different servers, and client. The maximum number of IPv6 packets a switch can send efficiently without any packet loss is 227950 packets per second = 227950*60bytes = 13.677Mbytes/s = 109,416Mbps.

The CPU utilization of a switch while sending the IPv6 packets from clients to servers by using a switch is more when compared to IPv4 packets this we can observe by seeing on the graph figure 26 and figure 24. Even though that the latest Linux kernel available at the time was used, the CPU utilization was abnormally high. So, by observing the both IPv4 multicast and IPv6 multicast results can conclude that IPv4 is an efficient protocol compared with IPv6 protocol in this test.

Future Work

This study tries to observe the pattern of packet loss when the packets are transmitted with high speed and also introduce a better way to find the packet loss and by comparing the both IPv4 and IPv6 results by considering the CPU utilization and Packet loss for only one kind of switch called DellR7048. The future work can be extended by comparing the IPv4 multicast and IPv6 multicast results with order kind of switches like Cisco, Juniper and also measure more QoS metrics like Jitter, latency.

7.References

[1] O. Hermanns and M. Schuba, *Performance Investigations of the IP Multicast Architecture*. 1996.

[2] S. Yang, X. Wang, and B. Li, "Haste: Practical Online Network Coding in a Multicast Switch," in *2010 Proceedings IEEE INFOCOM*, San Diego, CA, USA, 2010, pp. 1–5.

[3] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Multicast traffic in input-queued switches: Optimal scheduling and maximum throughput," *IEEEACM Trans. Netw.*, vol. 11, no. 3, pp. 465–477, Jun. 2003.

[4] S. Saad and I. Mohamad, "PIM-SM based Multicast Comparison for IPv4 verses IPv6 using GNS3 and JPERF," *Iraqi J. Sci.*, vol. 58, pp. 140–151, Jan. 2017.

[5] J. K. Sundararajan, M. Medard, M. Kim, A. Eryilmaz, D. Shah, and R. Koetter, "Network Coding in a Multicast Switch," p. 9.

[6] Z. Li, B. Li, and L. C. Lau, "On Achieving Maximum Multicast Throughput in Undirected Networks," *IeeeAcm Trans Netw.*, 2006.

[7] "IPv4 Header," *AIT*, 26-Jan-2012. .

[8] J. Postel, "Internet Control Message Protocol." [Online]. Available: https://tools.ietf.org/html/rfc792. [Accessed: 14-Feb-2017].

[9] W. C. F. <fenner@parc.xerox.com>, "Internet Group Management Protocol, Version 2." [Online]. Available: https://tools.ietf.org/html/rfc2236. [Accessed: 12-Mar-2017].

[10] I. Kouvelas, B. Cain, B. Fenner, S. Deering, and A. Thyagarajan, "Internet Group Management Protocol, Version 3." [Online]. Available: https://tools.ietf.org/html/rfc3376. [Accessed: 04-Dec-2018].

[11] "Comparison of IGMPv1, IGMPv2 and IGMPv3 | IP With Ease | IP With Ease." [Online]. Available: https://ipwithease.com/comparison-of-igmpv1-igmpv2-and-igmpv3/. [Accessed: 04-Dec-2018].

[12] "How IPv6 Works: IPv6." [Online]. Available: https://technet.microsoft.com/en-us/library/cc781672(v=ws.10).aspx. [Accessed: 14-Feb-2017].

[13] "Multicast IPv6 Address Format, Prefix, Flags and Scope." [Online]. Available: http://www.omnisecu.com/tcpip/ipv6/multicast-ipv6-addresses.php. [Accessed: 10-Mar-2017].

[14] tutorialspoint.com, "IPv6 Headers," *www.tutorialspoint.com*. [Online]. Available: https://www.tutorialspoint.com/ipv6/ipv6_headers.htm. [Accessed: 09-Mar-2017].

[15] A. Conta and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification." [Online]. Available: https://tools.ietf.org/html/rfc4443. [Accessed: 04-Dec-2018].

[16] upravnik, "IPv6 global unicast addresses."

[17] “NDP (Neighbour Discovery Protocol), functions of NDP, Neighbour Solicitation and Advertisement, Router Solicitation and Advertisement.” [Online]. Available: http://www.omnisecu.com/tcpip/ipv6/ndp-neighbour-discovery-protocol-functions-of-ndp.php. [Accessed: 11-Mar-2017].
[18] “Mapping IPv6 Multicast Addresses to Ethernet Addresses - Understanding Ipv6.” [Online]. Available: http://flylib.com/books/en/2.223.1.44/1/. [Accessed: 10-Mar-2017].
[19] “RFC 4604: Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast - The RFC Archive.” [Online]. Available: http://www.rfc-archive.org/getrfc.php?rfc=4604. [Accessed: 27-Nov-2017].
[20] “Sockets-notes.pdf.” .
[21] “IGMP snooping,” *Wikipedia*. 15-Nov-2018.
[22] “Defining Mld Snooping - D-link DGS-3100-48 User Manual [Page 102].” [Online]. Available:https://www.manualslib.com/manual/419247/D-Link-Dgs-3100-48.html?page=102. [Accessed: 05-Dec-2018].

Appendix A

In order to create a project by clicking on New on the File menu, and then click on the project.
• In the Visual Studio C++ project pane, Click on Win32 and then click on theWin32 Console Application.
• Assign a name to the particular project.
• By default, the solution which contains the project has the same name as the project, but we can provide a different name completely. We can also type a different project location.
• Click on OK for creating the project.
• In the Win32 Application Wizard, click on nex at that time choose an Empty Project, and then click Finish.
• If the Solution Explorer is not displayed, on the View menu, click on Solution Explorer.
• Add a new project source file as follows.
• Right-click the Source Files folder of solution explorer, point to Add, and then click on a New Item.
• In the Code node, click C++ File (.cpp), type a name for the file, and then click Add.

• In order to create a project by clicking on New on the File menu, and then click on the project.
• In the Visual Studio C++ project pane, Click on Win32 and then click on theWin32 Console Application.
• Assign a name to the particular project.
• By default, the solution which contains the project has the same name as the project, but we can provide a different name completely. We can also type a different project location.
• Click on OK for creating the project.
• In the Win32 Application Wizard, click on nex at that time choose an Empty Project, and then click Finish.
• If the Solution Explorer is not displayed, on the View menu, click on Solution Explorer.
• Add a new project source file as follows.
• Right-click the Source Files folder of solution explorer, point to Add, and then click on a New Item.
• In the Code node, click C++ File (.cpp), type a name for the file, and then click Add.
• The .cpp file seems within the Source Files folder in Solution Explorer, and therefore the file is opened within the Visual Studio editor.
• Type a valid C++ program in the editor file that uses the standard library or copy one of the sample programs into the files.
• If we use the sample program, mention by using directive called namespace std;. This directive will activate the program to use cut and end without requiring fully qualified names known as std::court and std::end.

Server

A server is a program or machine, which will wait for incoming requests.

Client

A client is a program or machine which sends the requests to servers.

Socket programming is a field of programming which is used to enable two or more hosts to communicate with each other. Socket programming usually applicable to the underlying communication protocols like TCP/UDP and raw sockets like ICMP. These protocols have a small communication overhead when compared to underlying protocols such as HTTP/DHCP/SMTP etc.

To execute the CPP tool centos is used and these centos are connected to putty.

In this socket programming tool, the server is used to receive the datagrams (IPv4 packets) from clients. The port 4321 which is used to send the packets from the client to servers. It is a protocol port which is used to send and receive the packets in this socket programming tool which consists of one multicast group to send and receive the packets. The size of the IPv4 ethernet UDP packet is 46 bytes.

Appendix B

The implementation of testing is done by the developed C++ tool and which is uploaded to using upload scripts. The testing is done by using bash scripts like upload client, upload server, check client, check server, clear log, start the client, start server, download result client, download result server.

The upload client script which is used to upload the client code and the upload server script which is used to upload the server code into the network. When we are running the upload client and upload server scripts, the code is uploaded to every server and client.

The \bin\pscp.exe is used to secure transferring off the server files to the network. In upload

Server script \bin\pscp.exe is used for secure transferring of files into the network.

Check client script is used to check the number of clients is used for multicasting the packets using different clients and different Servers. After running this check client script, we can see the number of clients ready for multicasting of packets.

@echo off

start ..\bin\md5sum_client.cmd

```
timeout 20

..\bin\md5sum_client_2.cmd > ..\md5sum\client.txt
```

Check server script is used to manage number of servers is interested in receiving the multicast packets using the switch. After running this check server script, we can see some servers are participating in the multicast test.

```
@echo off

start ..\bin\md5sum_server.cmd

timeout 20

..\bin\md5sum_server_2.cmd > ..\md5sum\server.txt
```

In this check server script ..\bin\md5sum is used for managing the multiple servers which are used to receive the packets from clients.

Clear_all_logs.cmd script is used to clear the output results about the server and client those are the output results of previous testing results.

```
for /L %%i in (101,1,194) do call :execute 172.16.9.%%i "rm -f /root/*172.16.9.*.log"

goto :eof

:execute

echo -- %1 -- 1>&2

echo -- %1 --

start "%1" "..\bin\plink.exe" -pw password root@%*

echo


goto :EOF
```

172.16.9.%%i "rm -f /root/*172.16.9.*.log" is a command used to remove all the client results and server results. 172.16.9. %%i represents the last number in IP address. It represents an IP address between 172.16.9.1 to 172.16.9.194.

The start_client.cmd script which is used to start the clients and a start_server.cmd script which is used to start the server scripts.

```
@echo off

for /L %%i in (101,1,195) do call :execute 172.16.9.%%i "/root/server.sh"
```

goto :eof

:execute

echo -- %1 -- 1>&2

echo -- %1 --

start "%1" "..\bin\plink.exe" -pw password root@%*

echo.

goto :EOF

for /L %%i in (101,1,195) do call :execute 172.16.9.%%i "/root/server.sh"

is used to start all the 95 servers and we can see the server IP address range from 101 from 195.

In start client script for /L %%i in (101,1,195) do call :execute 172.16.9.%%i "/root/client.sh" is a command used to start all the 95 servers i.e, from 172.16.9.101 to 172.16.9.195.

This Download_results_server.cmd script is used to see the output results of the server which shows all the server details.

@ECHO OFF

call ..\bin\get_server_logs.cmd > ..\results\server_.txt

call ..\bin\get_server_logs2.cmd

call del ..\results\server_.txt

\bin\get_server_logs is used to download the server results.

This Download_results_client.cmd script is used to see the output results of the client which shows whether there are any invalid packet numbers are present or not.

In this script, \bin\get_client_logs is used to download the server clients.