Performance Analysis of DiffServ Multicast (DSMCast) for Heterogeneous Multicast Receivers

by

Amee Bhatt

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Technology

in

Information and Communication Technology

to

Dhirubhai Ambani Institute of Information and Communication Technology

April 2004

DA-IICT

Declaration

This is to certify that

(i) The thesis comprises my original work towards the degree of Master of Technology in Information and Communication Technology at DA-IICT and has not been submitted elsewhere for a degree,

(ii) Due acknowledgement has been made in the text to all other material used.

Signature of Student

Certificate

This is to certify that the thesis work entitled **Study and Performance Analysis of *DSMCast* for heterogeneous multicast receivers in Differentiated Services Domain** has been carried out by Amee Bhatt (200211034) for the degree of Master of Technology in Information and Communication Technology at this Institute under my/our supervision.

Thesis Supervisor(s)

Acknowledgements

I express my deepest gratitude to **Prof. N. D. Jotwani** for his able guidance and valuable suggestions all through my research work. He has been indeed a source of motivation and encouragement that has led to the successful completion of the thesis.

I am also greatly thankful to **Dr. Aaron Striegel**, who has proposed DSMCast, for his cooperativeness and prompt reciprocation to clarify any conceptual doubts through mails. He has helped me a lot to verify my simulation results.

I am deeply indebted to **my husband Shardul and my parents** for all the moral support and strength they have rendered.

Last but not the least, I shall thank **God Almighty** without whom I believe any success is in complete.

Contents

Abstract

Internet is a large infrastructure that provides means for global communication. One of the biggest challenges faced in its growth and maintenance is the provision for the best Quality of Service (QoS) to the end-user applications. There exist some mechanisms that have evolved to achieve QoS in IP networks and amongst them Differentiated Services (DiffServ) is a prevalent one. With the increased usage of multimedia applications such as video and audio conferencing, which primarily use multicast mode of communication, one obvious question that arises is whether suitable QoS in terms of resource assurance and service differentiation can be guaranteed to them. This thesis answers the question affirmatively and discusses a recently proposed approach: DSMCast, which attempts to offer QoS for multicast applications in DiffServ domain.

The main objective of the thesis is to analyze how DSMCast provides support for heterogeneous QoS receivers participating in a multicast session. The performance analysis of DSMCast indicates that it can support the heterogeneous QoS requirements for the multicast receivers in a better way than traditional IP multicast. The simulation results bring out an important implication due to the nearness of receivers who have low QoS demands to those who have high QoS demands.

List of Acronyms

QoS	Quality of Service
IntServ	Integrated Services
RSVP	Resource ReSerVation Protocol
DiffServ	Differentiated Services
LAN	Local Area Network
WAN	Wide Area Network
DVMRP	Distance Vector Multicast Routing Protocol
MOSPF	Multicast Open Shortest Path First
PIM	Protocol Independent Multicast
DM	Dense Mode
SM	Sparse Mode
CBT	Core Based Tree
MPR	Multi Path Routing
QMRP	QoS Multicast Routing Protocol
DS	DiffServ Domain
TCA	Traffic Conditioning Agreement
SLA	Service Level Agreement
BB	Bandwidth Broker
DSCP	DiffServ Code Point
PHB	Per Hop Behavior
AF	Assured Forwarding
EF	Expedited Forwarding
BA	Behavior Aggregate
MF	Multi Field Classification
IPV4	Internet Protocol version 4
IPV6	Internet Protocol version 6
RED	Random Early Detection
NRS	Neglected Reservation Subtree
TEH	Tree Encapsulation Header
GNE	Good Neighbor Effect

List of Tables

List of Figures

Chapter 1

Introduction

Quality of service (QoS) has become an important point of consideration as it describes how the Internet today should cater to the needs of the users and their applications. There exist mechanisms such as Integrated Services (IntServ) and Differentiated Services (DiffServ) that offer QoS in the IP networks. These mechanisms mainly suggest how a service provider can properly allocate resources amongst the users and guarantee them adequate QoS depending on the resource availability and user affordability. One of the major challenges faced by these mechanisms is whether they can support multicast because multicasting is important as a mode of communication for the multimedia applications. Out of the two, as IntServ is inherently receiver-driven i.e. the reservation of resources are done on per-flow basis according to the demands of the receivers, it is more suitable for multicast applications. However, IntServ suffers from a serious drawback of scalability. The sca*lability problem* basically means that the complexity of the network grows much faster than the number of nodes in the network and as it exceeds a certain size limit; it becomes impossible to manage such a network. This is what happens with IntServ – as the number of users increases, the number of flows increases and it becomes extremely difficult to accommodate all of them. This motivated researchers to use DiffServ for multicasting, which is more scalable than IntServ mainly because it is based on service differentiation on per-packet basis. Even though it is a viable solution to combine the two technologies, the integration of DiffServ and Multicasting uncovered some inherent conflicts such as scalability of DiffServ vs. group state information at multicast routers which will increase with the number of nodes, sender-driven QoS mechanism of DiffServ vs. receiver-driven QoS mechanism of multicast and resource management due to multicast packet replication. Thus, this creates a need for an approach that can resolve these conflicts to provide QoS-enabled multicasting in DiffServ domain.

1.1 Problem description

In multicasting, different receivers may require different levels of QoS, which is termed as ***QoS Heterogeneity***. The traditional IP network does not make any provision for heterogeneous QoS for multicast receivers. Multicasting involves packet replication, which can lead to over positioning of the resources. Also, multicast will push the QoS to the receiver side allowing the routers along the path to adapt rather than making the sender aware. On the other hand, so far as DiffServ is concerned, it becomes more difficult to achieve heterogeneity, as it is a sender-driven architecture where QoS offered is based on how the packets are classified and selected when they enter the domain from the sender's side. The desired integration of DiffServ and Multicast must address this issue as it serves an important parameter for the service provider to estimate the overall network resource utilization.

1.2 Main Objective

With an objective to find a feasible solution for the problem of QoS heterogeneity, a novel approach – **DSMCast** is proposed in [1]. It is interesting to analyze how this approach successfully resolves the DiffServ and Multicast conflicts. The core idea of DSMCast is to take the state information from the every router and encapsulate the same in the packet header. It supports QoS heterogeneity by allowing the transformation of the QoS level as the packet reaches the downstream receivers. The main aim of the thesis is to validate and interpret the performance of DSMCast.

1.3 Related Work

Although many attempts have been made to integrate DiffServ and Multicast, the problem of QoS heterogeneity is addressed for the first time in [20]. In [20], the authors have proposed two levels of QoS provision: current QoS level and Limited Effort (LE). LE effort indicates that in absence of reservation of resources also, the receivers will be served by reduced quality. Although they were the first ones to address this issue, the solution for QoS heterogeneity proves to be inefficient as it involves lot of message passing between the network entities to change the current level of QoS. Moreover, the actual implementation of heterogeneity is left as the management issue.

1.4 Organization of Thesis

The next chapter describes the evolution of Best Effort (BE) networks to support QoS, basic QoS parameters, comparison between IntServ and DiffServ and more in depth discussion on the DiffServ mechanism

Chapter 3 brings out the importance of QoS for multicasting applications; basic concepts of multicast communication with discussion on some QoS enabled multicast protocols.

Chapter 4 explains DSMCast and how it attempts to solve the problem of QoS heterogeneity.

In Chapter 5, we present our simulation results obtained through the performance analysis of DSMCast in the presence of heterogeneous QoS receivers for different cases.

We conclude the thesis with chapter 6 and give some pointers for the future scope to resolve one important implication of the solution that DSMCast provides.

Chapter 2

Quality of Service

2.1 Introduction

IP networks provides what is called as “Best Effort Service” (BE) model wherein the network tries to deliver the data but gives no assurance to traffic flows. If network experiences congestion the packets are dropped as a result of queue overflows. Moreover, all packets *are treated equally*; any flow can be affected by the congestion. So, BE service is more suited for the applications that do not necessarily demand timely or predictable packet delivery. If guaranteed delivery is required, at best TCP is used as end-to-end mechanism since it can treat the traffic data with bursts in a better way by providing retransmissions. But it does not ensure the timely delivery of packets. Shortest path trees can be use to reduce the cost of transmission but that only establishes the connectivity and offers no guarantee about the actual availability of resources along a path at any given instant.

With the advent of multimedia applications such as video conferencing, teleconferencing, streaming audio and video etc, the Best-effort service does not stand to satisfy the resource requirements mainly due to following shortcomings:

- Unpredictable response by the routers in case of transient congestion
- Inability to provide priority service to different classes of traffic
- Inability to request end-to-end service quality
- Limited Mechanisms to allocate resource optimally

As BE service does not guarantee network performance, there is a need for a better service model that encompasses proper resource allocation and service differentiation. This need is precisely defined by the term “*Quality of Service*”: which will be the means to provide resource reservation for a *type* of network traffic over any of the prevalent technologies like ATM, IP, Frame Relay, SONET etc.

2.2 Definition

Quality of Service is the capability of the network to handle the traffic to meet the service needs of certain applications. QoS has two dimensions: one is by the network service providers who will aim to provide optimal resources and another is by the application users who will try to seize as much resources as possible. To strike a balance between the two, QoS requires deploying proper traffic handling mechanisms, which can identify the traffic based on service needs.

2.2.1 QoS Requirements

Most of the present day applications can be categorized as:

- Elastic Applications: The applications, which do not have any problem in the delivery as long as packets reach their destination. Applications over TCP fall into this category since TCP does all the hard work of ensuring that packets are delivered. No stringent demands on delays or bandwidth.
- Real Time Tolerant (RTT) Applications: The applications that need weak bounds on maximum delay. Occasional packet loss is acceptable. For e.g. Interactive and streaming media applications that use buffering to prevent jitter.
- Real Time Intolerant (RTI): The applications, which demand minimal latency and delay across the network, e.g. Video conferencing between two people.

As seen from the above classification, these applications demand specific QoS requirements, which can be termed as "*QoS related*" parameters. These parameters are:

Minimum Bandwidth: It is required that an application must receive a minimum amount of bandwidth (over a pre-specified interval) from the network.

Delay/Latency: The total delay experience by a packet has three components: transmission delay, propagation delay and queuing delay. Transmission delay is the time consumed in sending the packet onto the link. Propagation delay is the time taken by a packet to reach its destination. Queuing delay is the time consumed by the packet waiting in the queue. The delay is often specified as average-delay or the worst-case delay.

Delay jitter: Delay jitter is the maximum difference between the largest and smallest delays that the packet experiences.

Loss rate: The ratio of the lost packets over the total packets transmitted. Packet losses are mainly due to congestion.

The configuration of these parameters should be done keeping in mind the characteristics of the network onto which the applications transmit their data. The table 2.1 describes the network QoS requirements of different applications and media types:

Application	Data loss	Bandwidth [b/s]	Delay and Jitter sensitive
File transfer	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	Loss-tolerant	Elastic	No
Interactive audio/video	Loss-tolerant	Audio: 5K – 1M Video: 64K – 10 M	Yes
Video Conferencing	Loss-intolerant	More than 1M	Yes

Table 2.1: Network QoS requirements of different applications and media types [22]

2.2.2 QoS Architectures

To satisfy the QoS requirements for different applications, IETF work groups have formulated two main QoS architectures:

a) Integrated Services (IntServ)

IntServ aims to provide per-flow reservation where a flow means the packet stream with common source IP address, destination IP address, source port number and destination port number. Prior to the transmission of data, resource reservation is set up along the entire end-to-end path, which requires every router to maintain the state information. It uses Resource reSerVation Protocol (RSVP)[8] for stringent admission control and flow setup mechanism. The implementation of IntServ requires that each router must support flow identification and packet scheduling. IntServ supports two service models: the *guaranteed service* model [6] and the *controlled load* service model [7]. Former needs strict admission control and is mainly used for RTI applications where as latter guarantees average delay under lightly loaded conditions and is used for RTT applications.

b) Differentiated Services (DiffServ)

The basic idea of Differentiated Services is to achieve *service differentiation*. Users' traffic is divided into small number of "*forwarding classes*". The traffic profile for the conforming traffic is defined as per the user requirements. Nonconforming traffic can be dropped, delayed or offered different forwarding treatment. The information regarding the forwarding treatment is directly encoded in the packet header; this process is referred to as *marking* of the packet. The allocation of forwarding classes applies to *traffic aggregates* rather than individual flows.

On comparing the two mechanisms, DiffServ stands out to be more prevalent and preferable for two reasons:

i) Scalability: As the differentiation is based on per-packet basis i.e. on the aggregate traffic flows rather than individual flows, it easy for the routers to handle the increasing traffic. The existing traffic classes in DiffServ are able to accommodate almost all types of traffic and so the question of scalability in terms of classes does not arise.

ii) Core statelessness: The complexity of allocation of resources is pushed from the core (inside) router to the edge (boundary) of a DS domain so every router, unlike IntServ, does not need to maintain the state information. This results in faster routing of packets across the network.

A brief detail of the DiffServ architecture will serve to appreciate the above benefits in a better way. The following paragraphs present the basic framework and working within the DS domain.

2.3 Differentiated Services

2.3.1 Basic Framework

DiffServ divides the entire network into domains. The DiffServ (DS) domain as mentioned in [5], is a set of contiguous nodes, which operate, with a common set of service provisioning policies. Domain consists of edge nodes (boundary) and the core (interior) nodes. The boundary nodes connect a DS domain to other DS domains. Depending on the direction of the traffic flow, boundary nodes can be ingress (incoming) or egress (outgoing). The interior nodes are either connected to other interior nodes or the boundary nodes. The customer network or the host network is connected with the boundary nodes. A typical DS domain is shown in figure 2.1:

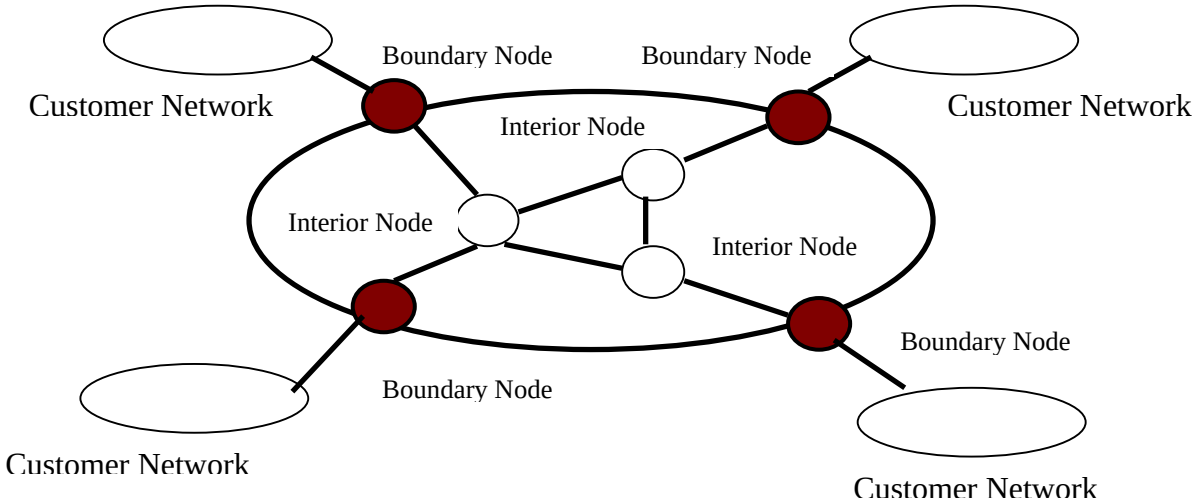

Figure 2.1 DS Domain

The boundary nodes in the DS domain are responsible for mapping the traffic profile of the incoming packets to one of the forwarding classes. They enforce the Traffic Conditioning Agreement (TCA), which is the part of Service Level Agreement (SLA) that entails the service parameters for traffic profiles and policing actions. These will include preferred traffic class, performance metrics like throughput, delay and drop precedence, actions to be taken on non-conformant packets and additional marking and shaping services provided by the service provider. Along with TCA, SLA also includes business related agreements as availability, security, monitoring, auditing, pricing and billing. A node known as Bandwidth Broker (BB) that act as an agent is responsible to administer the resource management in DS domain. It keeps track of the current

allocation of marked traffic and interpreting new requests based on the policies mentioned in the SLA. Each DS domain has one BB, which maintains the policy database and also performs intra-domain provisioning of resources. Two of the good references for BB architecture and implementation are [16,17]

When the traffic arrives at the ingress node of the DS domain, it classifies the packet and assigns a specific behavior aggregate. The DiffServ Code Point (DSCP) specifies each such aggregate. In the process of marking, DSCP is encoded in the packet by using first six bits of the Type-of-Service (TOS) field in the IP header. These bits together are known as DSField. The last two bits of TOS are currently unused (CU).

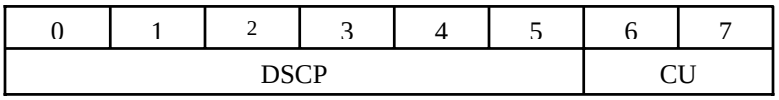

Figure 2.2 DSField

Each DSCP is associated with a forwarding treatment, which in DiffServ terminology is known as Per-Hop-Behavior (PHB). As defined in [5],

PHB is the externally observable forwarding behavior applied at a DS-compliant node to a DS behavior aggregate.

Under loaded conditions, when multiple behavior aggregates compete for the resources, the distinction is made based on the observable forwarding behavior i.e. delay, jitter and loss. Thus the differentiated services are recognized and constructed at very hop through different PHB's.

At the DS-complaint nodes, PHB can be defined in terms of

- Resources such as buffer and bandwidth
- Traffic parameters such as delay and loss
- Relative priority assigned to one PHB compared to other PHB's.

As per the standards, all the code points must be associated with some PHB i.e. there is either 1 to 1 or N to 1 mapping. In the absence of this mapping, code points are mapped to the default PHB. Default PHB is nothing but provision for the *best effort* service.

So far, two of the PHB's are standardized, they are

- Assured Forwarding (AF) [18]
- Expedited Forwarding (EF) [19]

2.3.1.1 Assured Forwarding

Assured Forwarding PHB offers different levels of forwarding assurances to the packets to meet the expected capacities of the in-profile traffic.

It consists of four forwarding classes, which are allocated with minimum amount of bandwidth and buffer. Each forwarding classes is calibrated into three-drop precedence: low, medium and high. If the traffic of the forwarding class exceeds its resources, the packets with the high drop precedence will be discarded prior to the ones with the low drop precedence. A code point is assigned to a combination of the forwarding class and drop precedence. A DS node may reorder packets of different AF class but cannot reorder the packets of same AF class with different drop priorities. The recommended code points for AF are shown in the table 2.2[5]:

AF Code Points				
Drop Precedence	Class 1	Class 2	Class 3	Class 4
Low (0)	001010 AF10	010010 AF20	011010 AF30	100010 AF40
Medium (1)	001100 AF11	010100 AF21	011100 AF31	100100 AF41
High (2)	001110 AF12	010110 AF22	011110 AF32	100110 AF42

Table 2.2 AF code point table

For example, traffic for applications such as video conferencing wherein the sources can change their rate based on the feedback received and also requires low latency can be marked with AF class.

2.3.1.2 Expedited Forwarding

Expedited Forwarding is meant provide low loss, low jitter and low delay services in the network. In order to have low delay, practically the queue length should not be very large and for queue to be small or for no queue formation, arrival rate of packets at any node should be less than the departure rate. As defined in [19],

Expedited Forwarding PHB is a traffic aggregate where the departure rate of the aggregate's packet from any DiffServ node must equal or exceed the configurable rate.

This PHB is also known as premium service as it is configured at any DiffServ node with highest priority i.e. the traffic having EF code point can preempt all other traffic. A special code point is recommended for EF: 101110

For example, traffic for applications such as VOIP requires low loss and low latency requirements for relatively constant rate traffic sources may be marked with EF code point.

2.3.2 Functioning at the DS node

Figure 2.3 shows the logical view within the edge and core router of the DS domain:

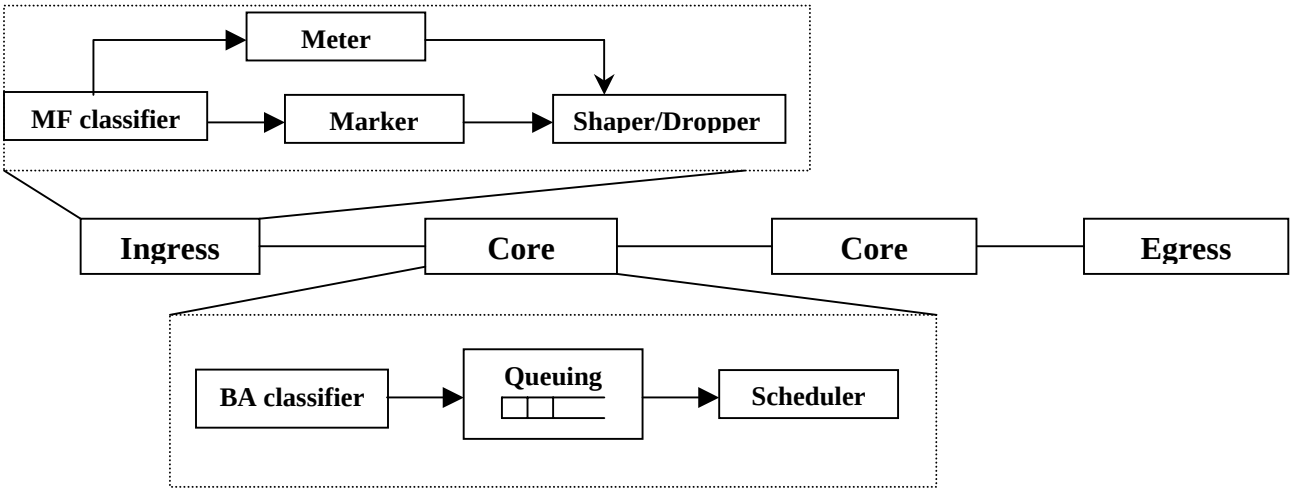

Figure 2.3 Logical view of the DS routers

2.3.2.1 Traffic Classification:

Traffic classifiers identify the packets and classify them based on the predefined rules. As shown in figure 2.3, there are two basic types of classifiers: multi-field classifier (MF) and behavior aggregate (BA). Multi-field classifier uses a combination of fields from the packet header for the classification. In case of IPV4, it uses source address, destination address, source port, destination port and protocol ID and in case of IPV6 it makes use of source address, destination address and flow label. MF classifier is used at the ingress router as the traffic coming from the upstream domain may come from a non-DS domain. And BA classifier is used at the core router as they forward the packet with the PHB indicated by the DSCP value.

2.3.2.2 Traffic Conditioning

After the traffic is classified, the in-profile packets are allowed in the network whereas the out-of-profile are subjected to further conditioning. Conditioners comprise of mainly four components: meter, marker, shaper and dropper.

Meter

It measures the characteristics of the incoming traffic stream and compares them to the ones specified in TCA. The packets in accordance to it are said to be *in-profile* and sent out and those that are not are *out-of-profile*. Meter passes the state information to the preceding blocks to perform appropriate action on in as well as out-of-profile packets.

Most commonly *token bucket* mechanism is used for metering. Tokens are added to the bucket with finite length, L, at a fixed rate, X, and are removed as the packet arrives with rate R, from the classifier. Appropriate setting of L, X and R can be used to enforce a traffic profile. When single rate is insufficient for enforcing the profile adequately, more than one bucket with different rate can be implemented.

Marker

Marker sets up the DS field in the packet header to encode the specific DSCP. This will add the packet to the particular traffic aggregate. According to the state information of the meter, the marker can mark all the packets with the one code point or mark packets with the set of code points that specify the same PHB for e.g. for AF group. Packets are re-marked in certain situations like when they violate the traffic profile at any administrative boundary or when two domains are using different DSCPs.

Shaper

Shaper delays the non-conformant packets to bring them in compliance with the traffic profile. Shaper has finite buffer space, so the packets may be discarded if there is insufficient buffer capacity.

Dropper

Dropper will discard some or all the packets, which are not conforming to the traffic profile. This is known as “policing”.

After the classification stage, the packets are fed into different queues based on their class of traffic. This is done by the queuing mechanism such as Random Early Detection (RED). To describe in brief, a queue is assigned a minimum and maximum threshold, and the queuing algorithm maintains a moving average of the queue depth. While the mean queue depth exceeds the maximum threshold, all arriving traffic is dropped. While the mean queue depth exceeds the minimum threshold but not the maximum threshold, a randomly selected subset of arriving traffic is marked or dropped.

DiffServ uses the variant of RED for en queuing the packets marked with AF classes. All the packet of AF1x class will be queued in one queue. And now to further calibrate each for drop precedence different minima and maxima are specified for different DSCPs for the same physical queue i.e. packets marked with AF10 and AF11 will have different values of minimum and maximum thresholds for the same queue. This kind of segregation within a single physical queue is often referred to as virtual queue.

Finally in the core routers, there is a scheduler that dispatches the packets to the next hop from different queues based on the priority of the class. Any scheduling mechanism such as WRR, WFQ etc can be used.

From the above discussion it is clear that DiffServ can efficiently provide QoS through service differentiation for unicast applications but the problem arises in case of multicast applications. The problem is that for a single multicast group DS domain can provide only one class of QoS but what if different receivers within the same group demand different QoS levels. As this thesis aims to focus on this problem of multicast receivers, it is necessary to clarify some fundamentals of multicasting. So, the next chapter discusses multicasting concepts required to understand the problem and later describes how multicasting can be made QoS enabled.

Chapter 3

Multicasting

3.1 Introduction

Most of the QoS sensitive applications such as multimedia applications are group-based applications. Along with scalable and efficient network support, these applications have stringent QoS requirements in terms of end-to-end delay, delay jitter, and loss. The QoS, as perceived by the receivers, in a multicast application is affected mainly by multicast routing protocols and the manner in which the multicast group members join/leave the group.

This chapter discusses some of the QoS-aware multicast routing protocols but before that it is important to understand the basic concepts regarding multicasting and different phases of a multicast group session.

3.2 Multicast Communication

In most general case, when a transmission takes place from one sender to one receiver, it is known as **unicast communication**. When the transmission is from one sender to all the nodes of the network, it is known as **broadcast communication**. However, when the transmission has to be from one sender to a set of nodes, unicast transmission will be inefficient, as it will need to send point-to-point messages to all the members, which will be a very expensive. And using broadcast communication will lead to wastage of resources, as many of nodes may not be interested in receiving. So sending from one sender to a *well-defined group*, which might be large, is known as **multicast communication**. The typical characteristics of a group include: scope of group membership, which may be within a LAN or WAN, can be transient or permanent with time and can have constant or varying membership. Based on these characteristics, the groups can be classified as:

- *Dense/Sparse groups*: those having members on most of the links or subnet of the network are dense and those having small number of members over widely separated links are sparse.
- *Open/Closed groups*: those in which the source is not the member of the group are open and those in which only the members are allowed to send are closed.

- *Permanent/Transient*: permanent groups last long where as transient change with time.
- *Static/Dynamic groups:* those in which the membership remains constant are static and those, which allow members to join/leave, are dynamic groups.

3.2.1 Phases of Multicast Communication

The multicast communication in a network can be described by the sequence of phases or steps as depicted in figure 3.1:

1. Multicast Group (session) Creation
2. Multicast Tree Construction
3. Data Transmission
4. End of Multicast session

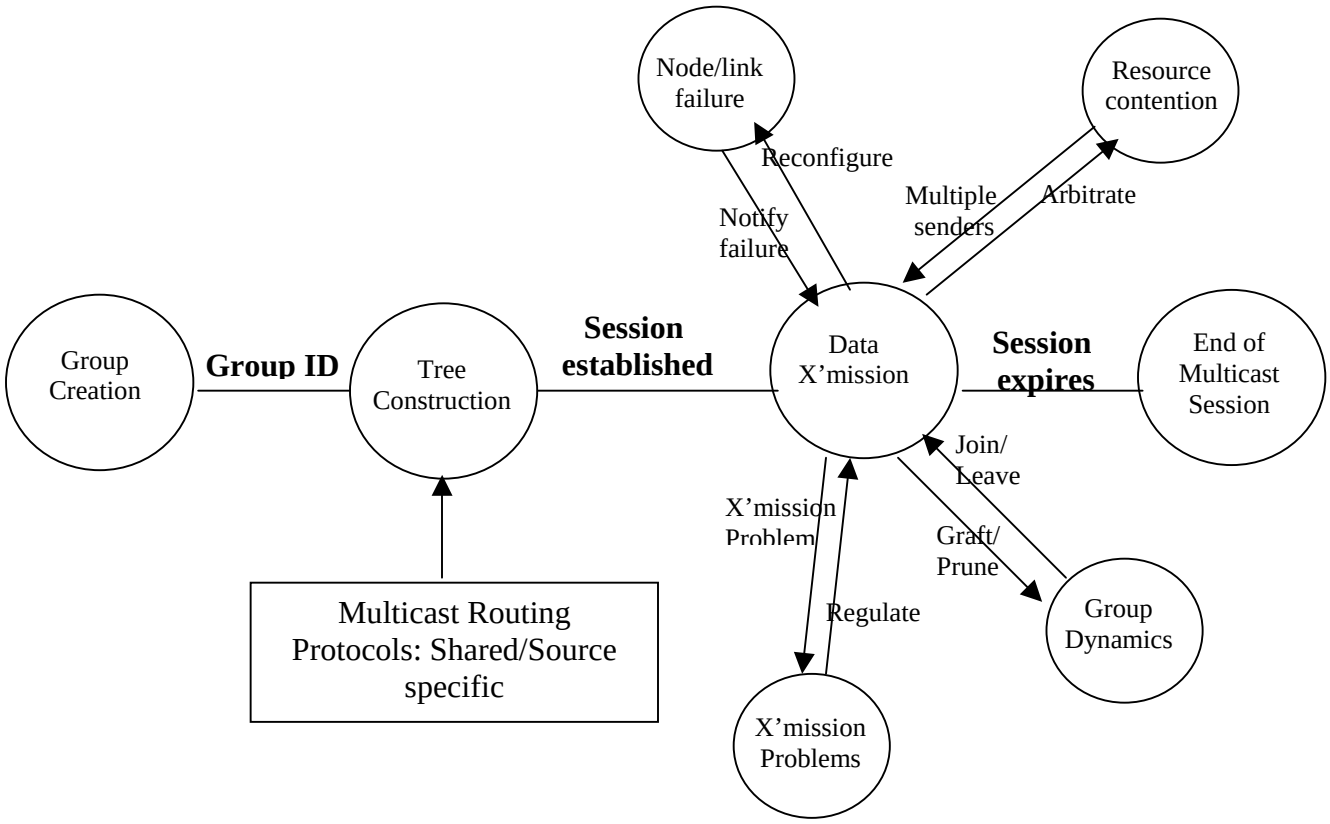

Figure 3.1 Phases of Multicast Session

1. Multicast Group (session) Creation

Formation of a multicast group is initiated by assigning a unique IP address (class D) to the entire group. Senders send the multicast packets with the destination field set to this address and receivers having this address receive the multicast packets.

2. Multicast Tree Construction

The most convenient way to maintain a multicast group is to form a tree structure as the source (in case of single source group) can be treated as root and it needs to forward only one packet down the tree. Also, tree structure shall enable parallel transmission of data and it will help to minimize the data replication, as packet will be replicated only at the branching point of the tree.

Practically, within LAN, routers determine the members of particular multicast group through Internet Group Management Protocol (IGMP). It has query and response message. The router periodically broadcasts a query on the LAN requesting the membership information. Each host, which is interested, sends back the report with group/groups it is interested in. For WAN, a simple solution is to *flood* the packets to all the routers i.e. if the router receives the packet for the first time, it forwards it to all the interfaces except the incoming one. But, flooding often runs into two problems that the router may end up getting duplicate packets and every router may receive sources' packets. To avoid duplication, identifiers can be used but this added cost for a permanent multicast group would be unacceptable. To overcome this problem another technique is used which is known as *reversed path forwarding.* In this technique, every router R will forward the packet to downstream router only if there is a shortest-path between the downstream router and the source that includes R. For the second problem, most of the tree construction algorithms provide the mechanism that a router can inform its parent that it is no longer interested in receiving multicast packets from a particular source. This process is referred as *Pruning.* Conversely, if the member again re-establishes the membership, it has to send a *graft* message to its parent, which will forward the request to the source.

Multicast protocols use the above techniques to construct a multicast tree along a path. These protocols mainly follow two approaches: **Source -specific** and **Shared Tree.**

Source-specific tree: The root of the tree is the source for the group. The branches form the spanning tree from the source to each of the receivers. As the tree is constructed by taking shortest path (in hops) as a metric, it is also known as Shortest Path Tree (SPT). Such a type of tree is denoted by (S, G), where S specifies the IP address of the source and G is the group address. Although the shortest path provides the minimal delay in transmission, it creates some scalability problem in large networks. Some of the common protocols that are based on this approach are:

DVMRP [12]: This is an extension of the basic distance vector algorithm for multicasting. It uses reversed path forwarding mechanism and later prunes the networks that do not have any group members.

MOSPF [13]: This protocol is the multicast extension of the basic link-state or open shortest path first algorithm. It builds the pruned shortest path tree when the first multicast packet is received. For the subsequent packets, shortest path calculation is cached for that source and the destination.

PIM-DM [11]: Protocol Independent Multicast protocol in Dense Mode (PIM-DM) implements similar flood and prune mechanism as DVMRP but it is independent of the underlying unicast routing protocol. It can use the routing information generated by any of the unicast protocols.

Shared-Tree: The multicast tree has the root or core node, which may not be the source. And the tree is constructed by spanning the branches from the core. The sources send the multicast packets to the core and then core forwards them to the other group members. The notation used for shared tree is (*, G), where * indicates any number of sources as they share a common tree. These protocols are highly suited for the sparse groups and large networks as the amount of link information is less at each router. But on the down side, the path from source to core may not always be optimal in terms of latency. Also the position of the core, which is also known as *rendezvous point*, is an important issue for the network designers. The protocols that follow this approach are:

CBT [9]: As described above, in Core Based Tree (CBT) every node in the network first sends the multicast packet to the core and then the core forwards it to the entire group. The join/leave request also takes place via the core.

PIM-SM [10]: It is another version of protocol independent multicast for sparse mode, which amalgamates both the approaches and facilitates the receivers to switch from shared tree to source-specific tree. Since this protocol is used in simulation studies, it is explained in more depth in Appendix.

3. Data Transmission

As seen in figure 3.1, there are some key run-time events that can happen during data transmission:

Group dynamics: the network should be able to keep track of members that join/leave the group. Under any circumstances, it is not affordable to transmit packets to members that have left the group or not delivering the packets to members that have newly joined.

Network link/node failure: multicast routing protocol that are based on unicast protocol like distance vector or link state can survive but those that are independent need to have their own mechanism.

Improper transmissions: controlling the flow of packets may be required if the receivers may be getting swamped by the multicast packets. Proper error checking needs to be implemented to avoid faulty packet transmission

Resource Contention: In case of many-to-many multicasting, many senders share the same multicast tree, which may lead to resource sharing problem.

4. End of Multicast Session

When a multicast session comes to an end, all the session-specific routing table entries is removed and the multicast address is released.

3.3 QoS Enabled Multicasting

In [4], the problem of dynamic QoS multicast routing can be rightly defined as

Given a new member M_{new}, find a path from M_{new} to an on-tree node that satisfies the QoS requirements of M_{new}

A QoS aware multicast routing must also enable a fair chance of a successful JOIN, should minimize the cost of joining path and time of JOIN and should be scalable to large networks. QoS multicast routing protocols can be classified by two basic approaches: Single-Path and Multi-Path. Single-path provides only a single path for the connection of the new member whereas Multi-Path provides many candidate paths.

Single-path routing is basically improvisation of the existing protocols like CBT and PIM to ensure QoS. These protocols consider delay and cost factors to setup the QoS path. But as these rely on the shortest path in term of number of hops may not always provide sufficient resources to support QoS.

Multi-path routing protocols on the other hand, provide many possible paths and from these the path, which satisfies the QoS requirements in the best possible way, is selected. Some of the recently proposed MPR protocols are:

Spanning Join: The new member broadcasts the JOIN request to all the on-tree nodes. The on-tree nodes reply to these messages and the new member chooses to join the best on-tree node to connect to.

QoSMIC [15]: Before joining, two search procedures are adopted for the new member to find out a suitable path: Local search and Multicast tree search. Local search is similar to

spanning join wherein only the small neighborhood is searched. If local search attempt is not successful in finding the suitable candidate path, new member resorts to tree search. It then contacts the designated manager node that is responsible for ordering a subset of on-tree nodes to establish path for them to the member. The new member then selects the best path amongst "offered" candidate paths.

QMRP [14] – The QMRP takes the benefit of both single-path mode and multicast mode. The protocol starts and continues with single-path mode until it reaches a node that had insufficient resources to satisfy the JOIN request. It will then switch over to multi-path mode.

However, these protocols do not satisfy all of the QoS requirements: spanning join and QoSMIC are not scalable to large networks and QMRP consumes considerable time to switch from single-path to MultiPath. More so these protocols do not effectively deal with one of the important issue of tree re-arrangement, which is an obvious consequence of group dynamics. It is essential that after a member join/leave it should not cause any service disruption and at the same time incur cost (quality) deterioration. For instance, in case of core-based trees if the selection of core is not adequate, the core may have to be migrated. This will result into the overhead of reconstructing the entire tree by migrating the members to the new core. More so, during this switch over it may happen that much of resources are wasted due to momentary over-allocation. To worsen the matter, one of the possible snags will occur if the core itself fails i.e. node failure. Thus it is required to have an approach that is equipped with mechanisms that enable fast recovery that minimizes service disorder.

None of these QoS aware multicast protocols talks about the issue of QoS heterogeneity amongst the multicast receivers, which is a much probable situation. More so, these protocols are more suitable for IntServ networks because it supports per-flow reservation and so the QoS state can be stored at every router along with the group information. As discussed earlier, DiffServ is more preferable to IntServ and hence a natural question that arises is how compatible DiffServ is with multicast. There are some issues that come up if an attempt is made to use DiffServ for multicast applications. What are these and how can they be resolved using DSMCast is very well elaborated in the next chapter.

Chapter 4
DSMCast: An approach

4.1 Inherent Conflicts of DiffServ and Multicasting

From the previous two chapters, we can now say that the integration of DiffServ and Multicasting will certainly prove useful for the next generation applications. However, from the implementation point of view, this is not a simple task as both these technologies have some inherent architectural conflicts [1, 20]:

4.1.1 Scalability in DiffServ Vs. per-group state information for multicasting

In multicasting, every router along the path maintains group state information, which will expand as the number of members in one group increases as well as number of groups over the Internet increases. More information will naturally require more storage and processing at each router. This attacks the fundamental concept of DiffServ, which intends to keep the core completely stateless that simply forwards data at very high-speed. As pointed out in [1], in spite of improving processing power of the CPUs, to maintain such huge amount of routing table entries at each node will be of high cost.

4.1.2 Sender Driven Vs. Receiver Driven QoS

The traffic enters the DS domain from the ingress router and is forwarded ahead, so it can be termed as “sender driven” mechanism. Whereas multicasting is “receiver-driven” as it mainly functions according to the receivers that join/leave the group. This conflict in mechanism leads to heterogeneous receiver problem wherein the receivers that join the multicast group may not join with same PHB requirement like other members of the receivers. In DiffServ, the QoS provision is made at the sender where it can either provide same QoS to all or none at all. To take care of the multicasting, different groups can be created to support different levels of QoS but this will incur high cost in terms of bandwidth. To enable heterogeneous user requirements within the single group, the packet that carries DSCP needs to be changed dynamically. This issue will be dealt in more depth towards the end of the chapter.

4.1.3 Resource management

Multicast packets may replicate one or more times within the network and if the packets replicate at the ingress node of the DS Domain, they may take multiple paths across the domain. This may lead into consumption of more resources than the unicast packets. In order to avoid this problem a plausible solution will be to reserve special code points and PHBs for multicast packets. Moreover, members of the multicast group often join and leave the group dynamically, which makes it difficult to predict the amount of resources that may be required by the multicast traffic originating from the upstream domain. Thus, the span of the multicast distribution tree within and outside the DS domain(s) needs to be specially considered. Multicast packets may exit the DS domain from multiple egress nodes and enter more than one downstream domain. So it becomes very important that the DS code point selected for the multicast traffic should be in compliance with code point assignment of the adjacent downstream domain(s). Any discrepancy in this may violate the peering SLA.

In [20], the authors introduce a specific problem of multicast in the DS domain, which is known as Neglected Reservation Sub tree (NRS) problem. It describes that whenever a member joins the multicast group in the DS domain at any node, the packets will be replicated at that node and so will its DSCP value. Accordingly, the new member will experience the same forwarding treatment. This will lead to unexpected resource consumption by the joining member, which was not estimated at the ingress router. The replicating node cannot determine whether a corresponding reservation exists for the replicated flow at the output link or not. Further, if the replicating node is egress node and the new member of the next DS domain joins with the same PHB as earlier members of the group, total amount of resources used for that aggregate will be greater than total amount reserved. So the policing component of the egress node will drop the packets till aggregate is in accordance of the traffic profile. These discarded packets can be of the originally reserved flow. And if the replicating node is the interior node where there is no policing component the resources of the lower priority class will be consumed.

Let us now learn how DSMCast proposed in [1] resolves the above conflicts.

4.2 DSMCast

DSMCast uses the *encapsulation-based method* to address the three conflicts between DiffServ and Multicast. Encapsulation-based method embeds the multicast tree information within the multicast packet header. The information is inserted in the packet at the ingress router of the DS domain, which travels through the multicast path(s) and is detached at the egress routers of the DS domain.

This method has its own advantages and disadvantages. The advantages are:

- Core router need not maintain the state, as the entire tree information is present in the packet itself. The approach makes the processing at the core nodes independent of the any of the underlying multicast routing protocols as PIM, MOSPF etc. as they are only responsible to forward them.
- It **can support the heterogeneity of receivers** by inserting and changing value of DSCP within the packet header instead of having separate trees for different QoS level.
- Deployment is restricted to a single DS domain and so the overhead of end-to-end support is eliminated thus providing operational transparency to the global Internet.

On the downside, disadvantages are

- The edge routers have to maintain the route table entries as the header is prepared and inserted at the edges only. Thus the admission control and policy at the edge routers have to be precisely implemented, as any improper allocation at the beginning will have its repercussions within the entire DS domain.

In spite of this disadvantage, it is interesting to study how the DSMCast Architecture addresses certain implemental challenges like developing the packet header that contains the information regarding the entire multicast tree, tree construction and maintenance, feature that accommodates the heterogeneous QoS requirements of the egress routers and design of specific join/leave protocol for multicast group dynamics within the domain.

4.2.1 DSMCast: Architecture

The DSMCast Architecture is divided into four main components:

- Transport mechanism
- Variable QoS Extension
- Tree Construction
- Join/Leave protocol

Following figure gives an overview of the DSMCast Architecture.

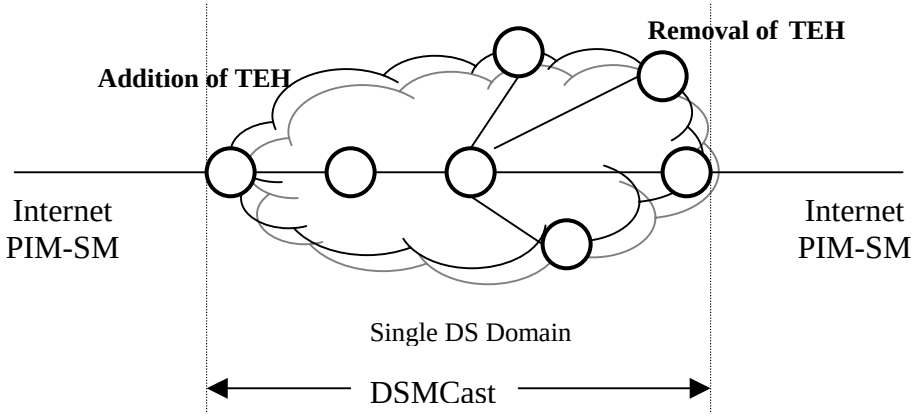

Figure 4.1: DSMCast Architecture

4.2.1.1 Transport Mechanism

The primary goal of DSMCast is to render the core router stateless and in order to achieve this all the replicating/routing information of the multicast packets is packed into the *Tree Encapsulation Header* (TEH). When the multicast ingress router arrives at the ingress router of the DS domain, this header is inserted for a specific multicast group between the IP header and the UDP header. As the packet is forwarded to the core routers, the header is processed by the hardware mechanism. The inspection of the header entails location of the tree information for the specific router, re-marking of the DSCP and the replication at the proper interface. In [1], it is stated that hardware implementation is preferable, as the start-of-the art processors will take minimal processing time. However, software, which enables simple pattern matching (information location), bit marking (DSCP, IP checksum) and packet replication is also a feasible option if the hardware implementation incurs high cost. Once the packet is replicated, it is

placed in the standard unicast queues of DiffServ and treated according to the PHB specified by the DSCP value. The DSCP may be remarked to indicate the heterogeneous QoS. As the packet exits the DS domain, TEH is detached from the packet.

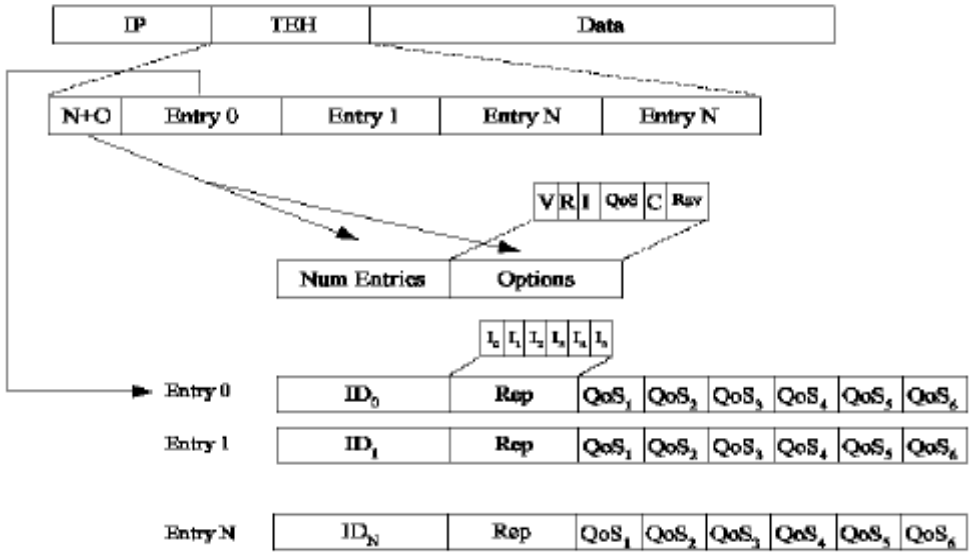

Figure 4.2: Structure of the TEH
©Striegel [1]

Tree Encapsulation Header (TEH) consists of three parts: Num Entries field, Options field and Replication/Routing entry field. Num Entries field contains the number of number of core router entries present in the third field TEH. Options field used for further configuring the TEH. The settings of the options can be understood from the table 4.1.

V	Variable QoS: 0:No Variation **1: Support for Heterogeneous QoS**
R	Replication and QoS indicates number of bits to use for replication and **QoS transformation**. 0: 5/2 1: 6/3
I	Identifier ID for the core routers
QoS	QoS transformation code for switching from one to another
C	Clear Replication 0: Should not replicate 1: should replicate
RSV	Reserved

Table 4.1: Description of the options field

Replication/Routing field consists of the entries of all the core routers with the information of the replicating interface and the corresponding QoS transformation that needs to be made. Last 8 bits of the IP address is used for uniquely identifying each router but this is just an assumption. Network administrator may wish to give his own formulated ID but in that case proper mapping between IP address and unique IP must be provided. During the processing of the header the core router shall identify its entry through the unique ID and the replication field will indicate how the packet should be replicated. **If the packet is replicated, QoS transformation code is assigned to change the DSCP and thereby the PHB. This is how it supports dynamic DSCPs.** A very elaborative example for the working of TEH is present in [1].

4.2.1.2 Variable QoS Extensions

To facilitate the heterogeneous QoS within the multicast tree, DS field needs to be changed as the packet gets replicated. The *Variable QoS* option adds the QoS transformation code that denotes to which the current DSCP value should be changed. Possibly one of the ways to implement will be to propagate the “best” PHB up the tree and then degrade it as per the requirement. Specific encoding scheme can be implemented to transform the QoS from the current PHB using 2/3 bits in the entry field of TEH. The implications of this transformation shall be discussed in more detail.

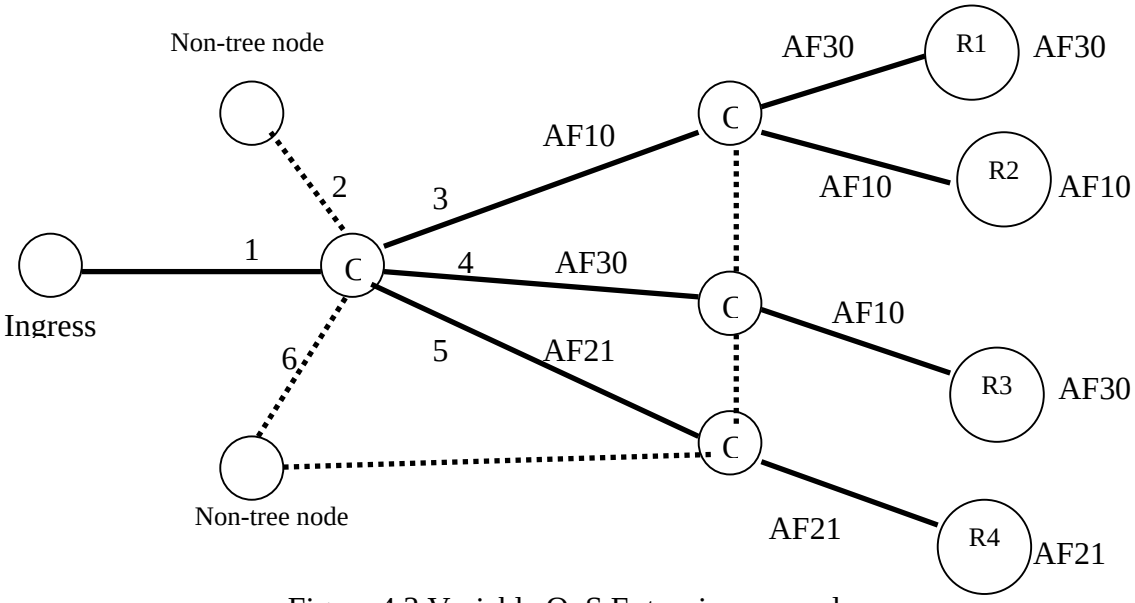

Figure 4.3 Variable QoS Extension example

As illustrated in figure 4.3, the best PHB – AF10 is propagated upstream and then it degrades by two delay classes to AF30 (with reference to the AF code point table). At interface 4 and 5, the PHB is also transformed from AF10 to AF30 and AF21 respectively.

4.2.1.3 Tree Construction

The basic assumption made for the DSMCast Architecture is that each edge router has the complete topological knowledge of the entire DS domain. This information can be given manually or obtain from the intra-domain protocols such as OSPF or IS-IS. With this information, the ingress router can construct the multicast distribution tree. One very important point to keep in mind is that the tree size is limited by the number of egress routers in the domain irrespective of the actual receivers across the global Internet. The advantage of making the ingress router to construct the tree is that ingress router can estimate maximum resource consumption by a multicast tree. It can then explicitly specify each link for replication and the PHB associated with it. Secondly, it is also beneficial for ingress router as it can negotiate the resources required by the whole tree instead of every router being concerned about it.

4.2.1.4 Egress Join/Leave Protocol

Any architecture cannot exclude the multicast group dynamics and so has not DSMCast. It has a special Join/Leave protocol that takes care of the members joining and leaving the group within the DS domain. As seen in chapter –3, there are many multicast routing protocols that provide join and leave mechanism but as DSMCast intends to be protocol-independent, a suitable Join/Leave protocol is proposed for the same. The basic problem that occurs routing a join/leave request is the location of the multicast group. The node intending to join a particular multicast group may not find the source of the multicast group within the same DS domain.

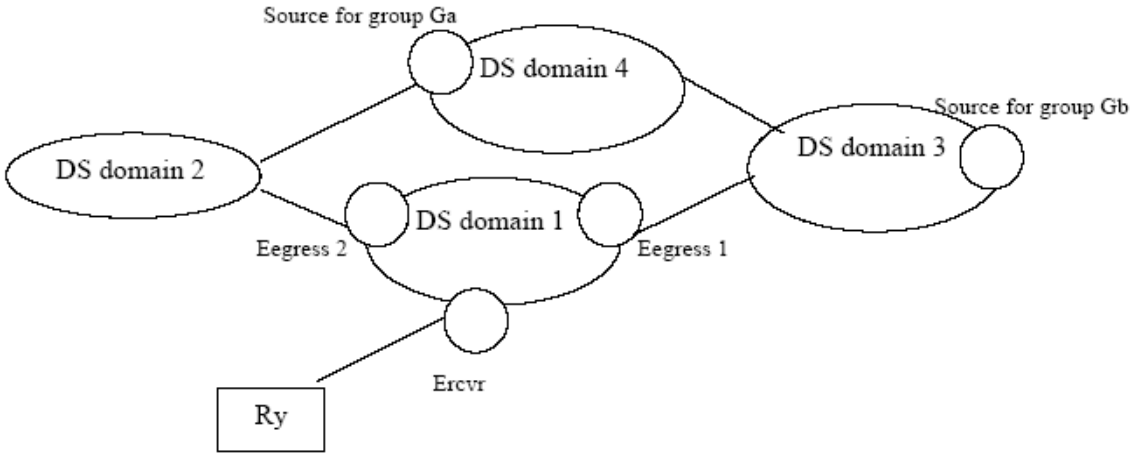

Figure 4.4: Join/Leave scenario

The join problem is precisely summarized in [1] in context of the figure 4.4.

For a join request to group G_x from the downstream node R_y that arrives at edge node (E_{Rcvr}), determine the appropriate edge node (E_{egress}) to which the multicast join request should be forwarded to and propagated outwards from the DS domain such that the QoS constraints of the join request are met

The type of the multicast tree mainly governs the determination of location of the multicast tree i.e. whether it is SSM (S, G) or Traditional IP multicast (*, G). Further, the location of the multicast tree may be in the same domain or outside the domain and so the proposed protocol addresses both the aspects: intra-domain and inter-domain scenarios.

Egress Join for SSM

In case of source-specific multicast, the matter is much simplified, as the location of the source is known. So, the new egress router has to tunnel *Join-Request* to the proper ingress router, which will be the source for the multicast tree it wishes to join. As the tunnel header will have a special identifier of the ingress router, it can intercept and process the packet.

In order to accommodate the sender-driven QoS, the new egress router can select its own PHB, to facilitate this DSMCast supports two types of Join-Request:

Absolute Join: The Join-Request from the new egress router includes its IP address and new PHB for traffic. On receiving the Join-Request, the ingress router if it is not in the same domain as egress will have to perform certain admission control tasks by contacting the Bandwidth Broker of the domain. When the new allocation of resources are assured, the ingress computes the new TEH and updates its shaping/policing to route the multicast packets to the new ingress router. The ingress router responds to the egress router with the *Join-Ack* response.

Dynamic Join: In this case, the new egress router first sends the *Bid-Request* towards the source of the multicast tree. The ingress router intercepts this request and in response sends a Bid-Probe message. As the packet across the domain, the core router feeds in the information about the network dynamics like average delay, average queue size, average loss rate and cost pertaining to each of the classes of traffic. For e.g. AF3x class is having delay of 5ms and 2% of loss. A timer mechanism can be implemented at the egress router to wait for the response to come. The new egress router will respond to Bid-Probe with the Join-Request and then follow the procedure of Absolute Join.

Egress join for Traditional IP multicast

Traditional IP multicast will include the two cases: many-to-many and shared trees. In both these cases, the source IP address is not available (*, G). So, the new egress router needs to locate the multicast group based on the address G. Egress router multicasts the Bid-Request to all the eligible edge routers via a pre-constructed static multicast tree. Similar to SSM, inter-domain routing to locate the multicast group may be necessary.

In response to this Bid-Request, all the edge routers respond with their Bid-Probe messages. It gathers the probe request till the timer expires and also evaluates the bids to find out the source it wants to establish the join. Based on the most suitable statistics obtained in the bid-probe, the egress selects the ingress routers (as it can join to more than one source) to join and proceeds further by sending the Join-Request.

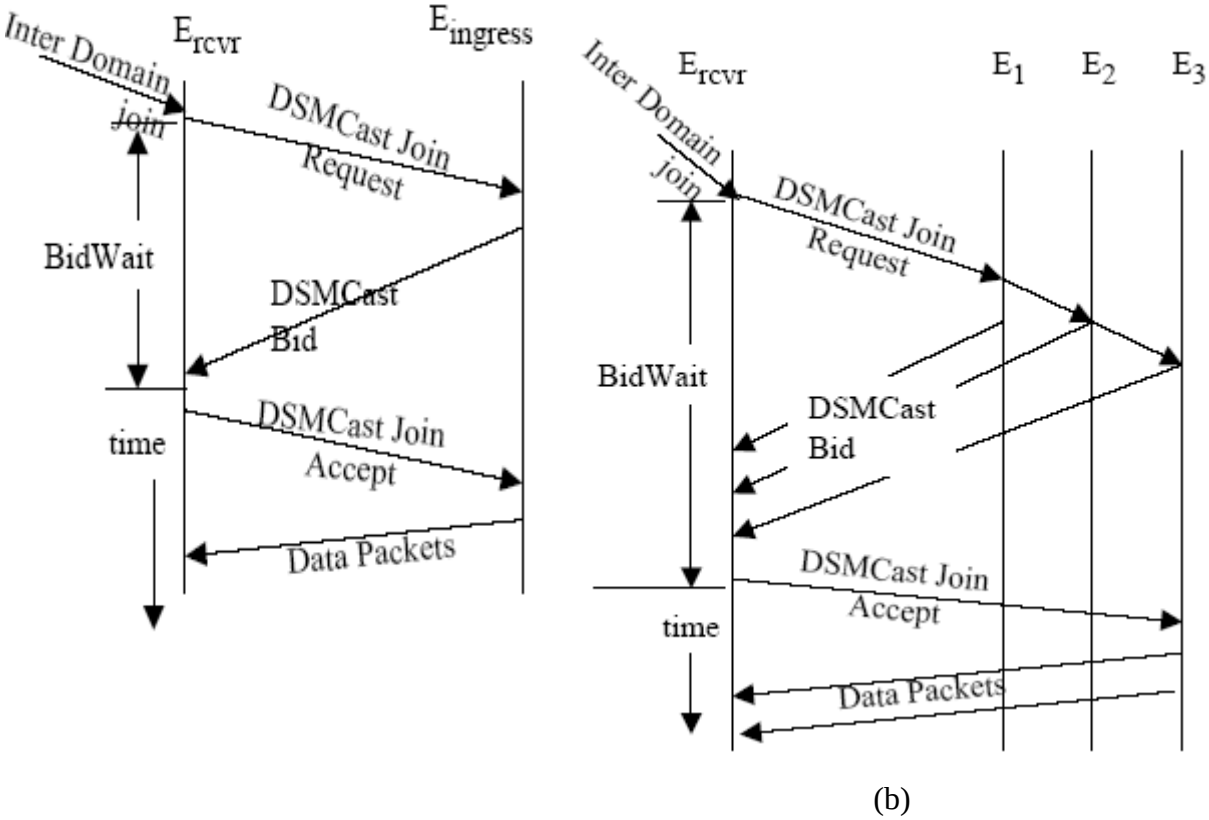

Figure 4.5 a) Egress Join –SSM b) Egress Join – Traditional IP multicast

As oppose to joining the group, leaving the group is simple. Since each egress router periodically refreshes their joining state to remain connected, for disjoining they need not refresh their state. When the ingress router does not receive the refresh message it assumes that the member have left the group.

4.3 QoS Heterogeneity

QoS Heterogeneity occurs when receivers participating in a multicast communication can have different requirements for the QoS. It is beneficial to incorporate the support of this heterogeneity, as the present day multicast applications will demand the network to adapt to its requirements rather than the applications adapting to the same. Example of applications adapting to achieve better quality will be by the use of different standards of compression and decompression algorithm but these options will be limited and not always optimized for all network scenarios. Another effect of heterogeneous QoS will be provision of extra bandwidth to accommodate multiple QoS level but this defies the very purpose of multicast that helps in reducing the overall bandwidth utilization.

The problem that arises to support heterogeneity for multicast applications in DiffServ domain can be better understood with an illustration:

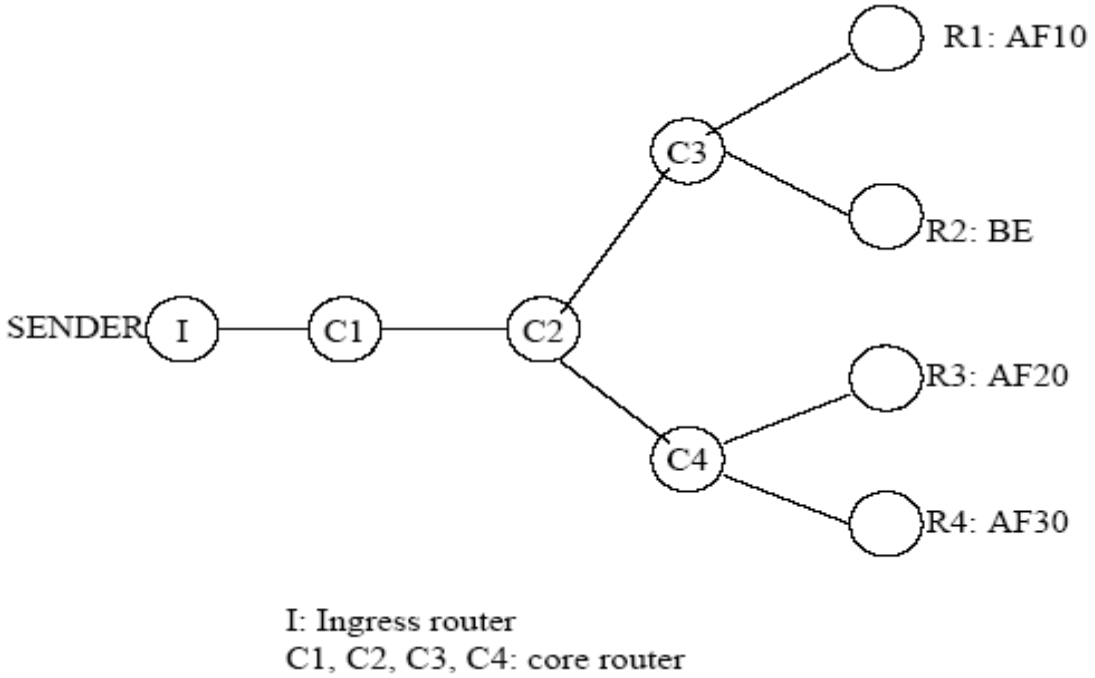

Figure 4.6: Heterogeneous QoS requirement

The above network is for multicast communication; say for example, audio conferencing. The receivers R1, R2, R3 and R4 are having different QoS requirements as shown in the figure. R1 wants AF10 (high priority class of assured forwarding) where as R2 can afford only the best effort service. But the path from ingress for both these is same till c3, so the difference needs to be resolved. It is also the case with the R3 and R4. Moreover,

DiffServ can provide only absolute PHB i.e. same PHB marking for the packet that goes from source to end. Due to this inflexibility of the DiffServ architecture, the problem of supporting heterogeneous QoS to different receivers is created.

The two viable solutions for this problem are: first and the most simple is, using the default DiffServ architecture to create separate multicast trees for each QoS level. But this may lead to heavy wastage of bandwidth resources with respect to the location of different multicast groups depending on the number of receivers. Effectively, this will also be more like doing multiple unicasts. Second solution is to provide highest level of QoS to all the receivers but this will result in overprovision of resources. More so, all the receivers may not be willing or afford to pay for the highest PHBs.

To reflect on the fact that if "best PHB" cannot be propagated upstream, going back to our example, consider the worst case, R3 joins first and so the end-to-end PHB for the tree is AF20. Now if the R1 and R2 join in succession, the priority problem occurs. R1 will actually receive the QoS of AF20-BE-AF10, as the priority is not propagated upwards. This is the case when traditional IP Multicast is done is DS domain and hence it is necessary to propagate "best PHB" up the tree.

4.3.1 DSMCast and QoS Heterogeneity

In view of the above mentioned problems, DSMCast approach attempts to solve it by enabling the QoS to change dynamically i.e. the "best PHB" must propagate and at the right point of the interface it should downgrade to the next receivers' expected level. From the above example, it would mean providing AF10 service till c3 and degrading to BE for R2. Similarly AF10 degrades to AF20 and AF30 from the c4 for R3 and R4 respectively.

The propagation of the "best PHB" is taken care by the ingress router with proper consensus of resources with the BB.

The use of dynamic DSCPs certainly has its own implications: *Good Neighbor Effect (GNE) and Prioritization.*

4.3.1.1 Good Neighbor Effect

Consider again the scenario depicted in figure 4.6, R1 and R2 are separated only by the last hop. Due to this, R2 gets the benefit of receiving near about AF10 service even though it is paying only for the BE service. Such a kind of scenario is known as the Good Neighbor Effect (GNE). The author coins the term *collusion attack* for such a situation wherein many of the downstream receivers can come together to pay minimal amount for the best QoS. Thus the use of dynamic DSCPs, which is actually meant to consume less bandwidth, may result in unfairness. Reverting back for the solution, use of separate trees for different PHBs can be thought of but it of course has the drawback of consuming more bandwidth. Hence, network provider is actually faced with a dilemma whether to consume less bandwidth to offer unfair service or to consume more bandwidth to achieve fairness. As a counter approach, as suggested in [1], some kind of "negative shaping" can be adopted which will be to add some delay or loss in order to degrade the service of the undeserving receiver. This solution does seem to be in paradox with the QoS principles but this may still prove to procure better fairness.

4.3.1.2 Prioritization

It may happen that the performance of PHBs may vary depending on the network load. For example, even though EF enjoys the highest priority service against AF but since it is rate-limited, it may perform worse than AF for lightly loaded networks. Similarly, AF11 has better scheduling delay than AF20 but has higher drop precedence. So under congestion, it can adversely affect the performance of AF20. Likewise, there are many such combinations of classes whose performance depends on the network conditions. The prioritization of PHB may differ from one DS domain to another depending on the network load. Though this problem is inherent to DiffServ, it will have its implications when the QoS state is transformed within the multicast group.

The effect of GNE can be very well understood from the simulation results, which are explained in the next chapter.

Chapter 5

Simulation

DSMCast is an approach, which enables multicasting in DiffServ domain. Simulation is carried out using NS-2 (Network Simulator version 2.26)[21]. Aaron [3] has provided the core routines for GenMCast. We will concentrate on the effect of **average per hop delay** experienced by each packet of different DiffServ classes in the domain.

5.1 Topology

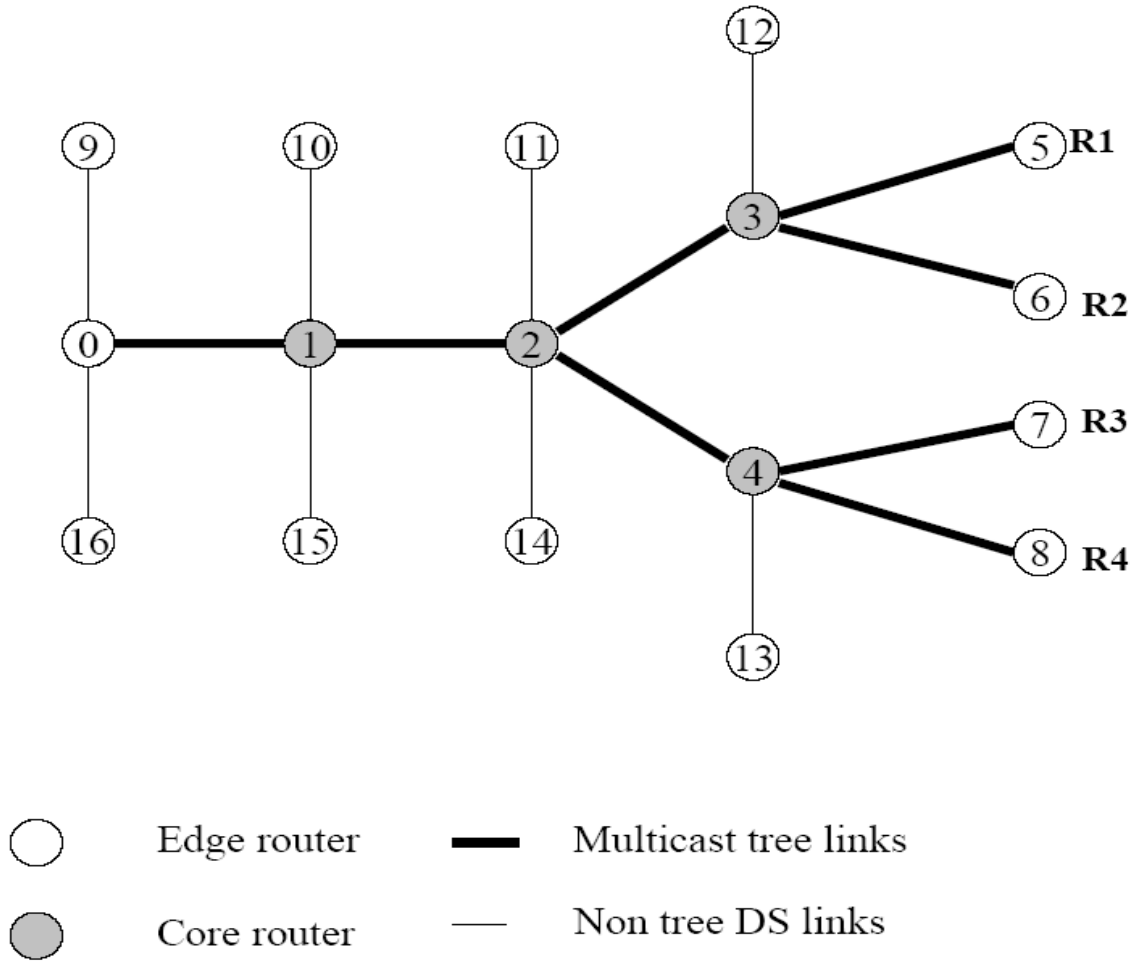

Figure 5.1 Simulation Topology

In this topology, multicast sender is connected to node 0, which is an ingress router. Four multicast receivers R1, R2, R3, R4 are connected to nodes 5,6,7,8 respectively. Each receiver is asking for different QoS, i.e. each one is joining with different AF class of DiffServ. Thus the network creates heterogeneous QoS requirements within a multicast group. The nodes 0,1,2,3,4,5,6,7,8 nodes form multicast tree. We need to inject background traffic into our network to simulate the real time situation. We know that a multicast capable router can be loaded with the unicast traffic at the same time. To enable this we have other nodes, which are contributing background traffic to the network. To understand the behavior of GenMCast with respect to average per hop delay, we can consider any topology. Selection of topology will not deviate the nature of the results obtained in the simulation.

5.2 Parameters for DSMCast

Parameters	**Value**
Link Bandwidth	10 Mbps
Link delay	5ms
DiffServ Configuration	
DSCP	AFxy
Uniform Case	x (25%,25%,25%,25%) y(33%,33%,34%)
Non uniform case	x (10%,20%,30%,40%) y(15%,30%,55%)
Scheduler	WRR weights = 24,18,12,6 and 1
Multicast Group traffic	UDP
Packet size	512 bytes
Rate	32 Kb
Background Traffic	50% TCP and 50% UDP
Rate	800 Kb
Packet size	1000 bytes

Table 5.1 Simulation Parameters of DSMCast

5.2.1 Traffic

We have two different categories of traffic encountered at core routers of the topology, multicast traffic and background traffic (unicast traffic).

5.2.1.1 Multicast

Multicast sender sends UDP/CBR traffic through ingress router at the rate 32 Kbps and each packet is of the size 512 bytes.

5.2.1.2 Background

Background traffic is contributed to the network/domain by different routers. These routers are nodes 9,10,11,14,15,16. Each router introduces equal amount of TCP and UDP traffic, which will be leaving the domain through nodes 12 and 13. This strategy is adopted so that all core routers are getting adequate amount of background traffic in the domain.

5.2.2 Link

We will see the behaviour of the domain for the range of bandwidth, 5 Mbps to 20 Mbps. Link delay of each link is 5ms for each bandwidth.

5.2.3 DiffServ Configuration

We need to configure each router to make it DiffServ enabled, as stated in section 2.3. Each router must have information about sharing of link bandwidth for DiffServ classes. We will see the effect of average per hop delay experienced by each packet of AF classes. First we will configure the domain in such a way so that each AF class is getting equal amount of bandwidth share. Since there are four major AF classes, each one will get 25% of bandwidth share. Since each AF class is having three drop precedence, each subclass must get 1/3 share of 25% allocated to each class. This is shown in figure 5.2. We will call this as a uniform distribution.

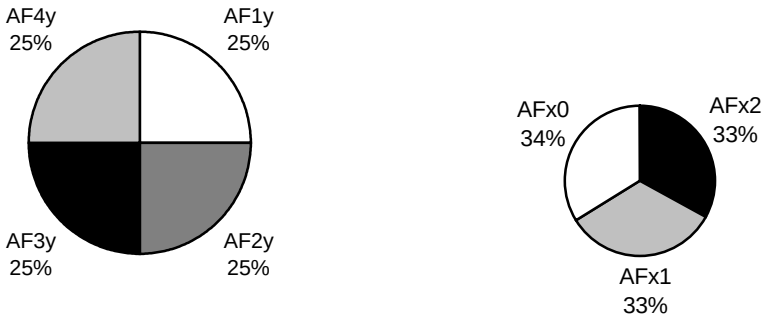

Figure 5.2 Uniform Distribution

We know that in DSMCast, we experience good neighbour effect, i.e. packets will propagate with the best DSCP as far as possible. This means if we have a receiver of AF10 class then ingress router will send the packets with DSCP of AF10. This leads us to see the effect by varying the Link distribution for DiffServ classes. As the AF10 class will be the most congested class, if AF10 receiver is present in the domain, we should see the behaviour by increasing the bandwidth share of AF10 class. This phase of simulation changes the bandwidth share scenario shown in figure 5.2. The modified link bandwidth distribution is shown in Figure 5.3. We will call this as a non-uniform distribution.

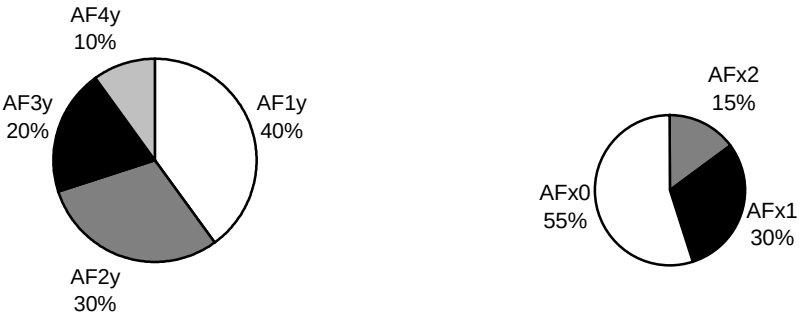

Figure 5.3 Non-uniform Distribution

5.3 Parameters for Traditional Multicast

We will see the effect of average per hop delay experienced by the packet of each DiffServ class in traditional IP multicast. It is not capable to provide support for heterogeneous QoS requirements of different receivers of the multicast group. We have created different multicast groups, one for each DiffServ class, and compared the results with DSMCast. Link bandwidth, bandwidth share for each class, background traffic, and multicast traffic parameters are same as stated above in section 5.2.

5.4 Results

In the first case, we will see the presence of GNE on average per hop delay with uniform distribution of link bandwidth as shown in figure 5.2, with the receivers joining in the following sequence. R1 is joining with AF40, R2 with AF10, R3 with AF30 and R4 with AF20. The link bandwidth is 10Mbps. As we can see in figure 5.1, AF40 should experience GNE because AF10 is its neighbor. The simulation as shown in table 5.2 validates this.

	Uniform (40-10-30-20)
AF10	13.03
AF20	14.24
AF30	14.96
AF40	13.33

Table 5.2

Now the same effect we will see for the non-uniform distribution, as shown in table 5.3.

	Non-Uniform (40-10-30-20) (ms)
AF10	12.02
AF20	14.13
AF30	14.79
AF40	12.32

Table 5.3

Here AF10 is having 15% more bandwidth share than uniform distribution. We know that DSMCast will allow the packets to propagate with the best DSCP available in the domain. Hence when AF10 is joining all packets will travel with AF10 up to node 2 in the domain. Since we have increased the bandwidth share of AF10 in non-uniform distribution, packets will experience less per hop delay compared to uniform distribution. Comparison of the results in Table 5.2 and Table 5.3 validates this.

Table 5.2 and 5.3 are stating the importance of joining order of the receivers. Here we can see that AF10 is a neighbor of AF40, hence as a result of GNE AF40 packets are experiencing almost the same delay as AF10 packets. This delay is even less than average per hop delay experienced by the packets of AF20 and AF30 classes. We will consider this joining order as the Worst Case joining order.

Next, we will apply all possible joining orders of the four receivers and find the average per hop delay for the packets of each QoS class for both the distributions. The result is summarized in table 5.4

	Non-Uniform (Avg. over all cases) (ms)	Uniform (Avg. over all cases) (ms)
AF10	12.40	12.69
AF20	13.18	13.62
AF30	14.93	15.19
AF40	15.49	15.42

Table 5.4

We will also carry out the simulation of traditional IP multicast with the parameters stated in section 5.3. We can see in table 5.5 that traditional multicast is having more per hop delay compared to DSMCast.

	Non-Uniform Worst Case (ms)	Non-Uniform Avg. Case (ms)	Traditional Multicast
AF10	12.02	12.40	21.75
AF20	14.13	13.18	22.64
AF30	14.79	14.93	23.12
AF40	12.32	15.49	25.44

Table 5.5

Now, we need to validate the presence of GNE over the range of link bandwidth, with uniform and non-uniform distribution of bandwidth for DiffServ classes. The nearness of the curve of AF40 to AF10 signifies the presence of GNE as shown in the graphs in figure 5.4 and 5.5. We can also see that per hop delay is less for non-uniform compared to the corresponding bandwidth of uniform distribution, as discussed earlier.

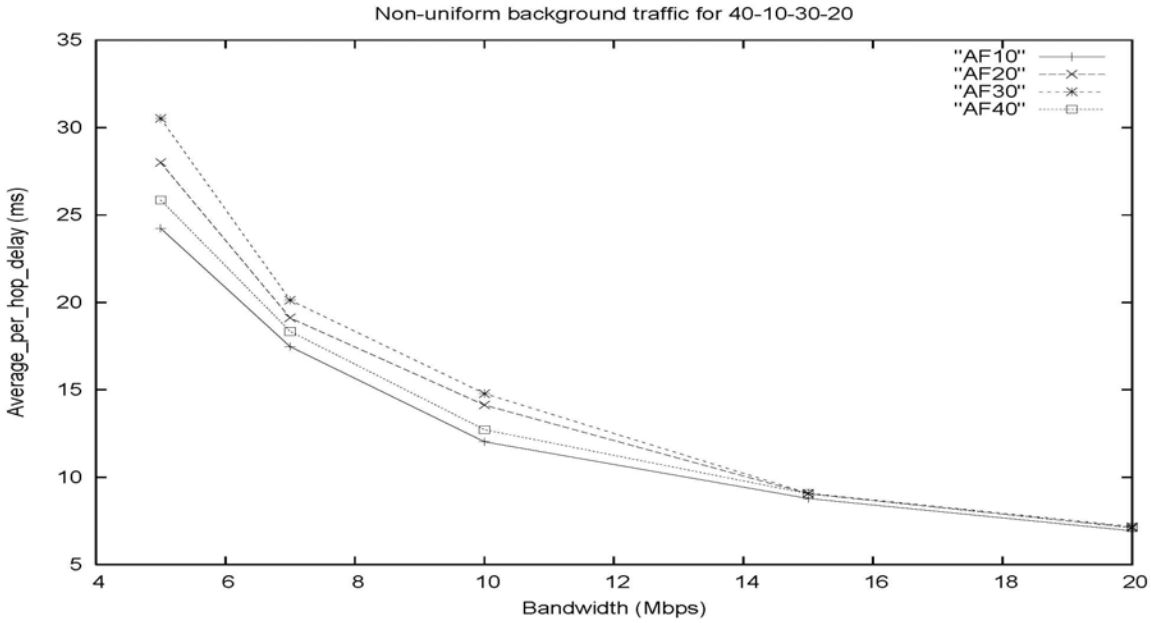

Figure 5.4

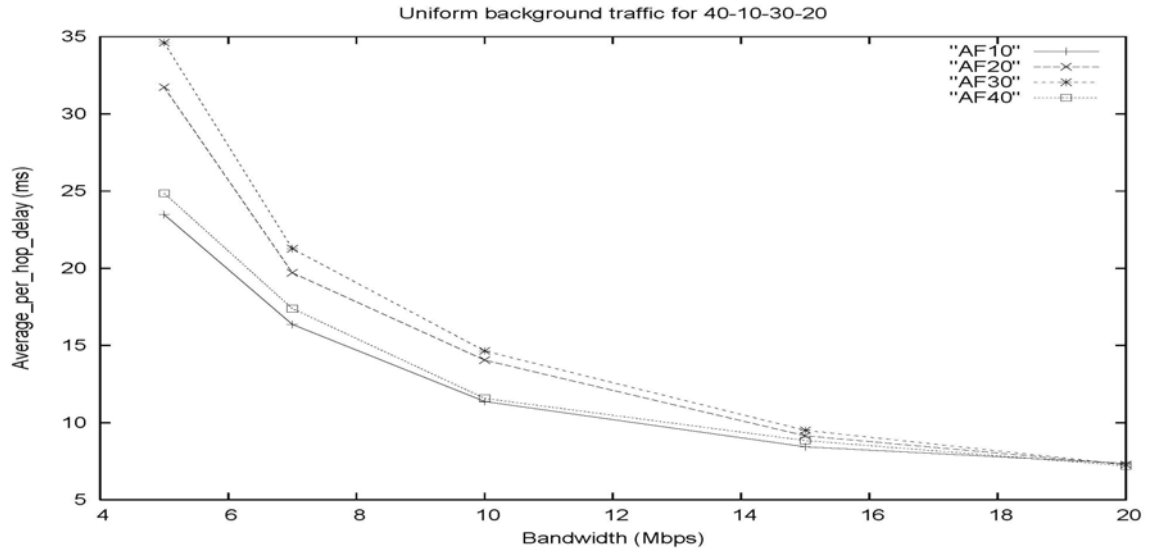

Figure 5.5

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

From the simulations conducted following points are verified and validated against the claims made in [2]:

1. DSMCast does offer better performance and support for QoS heterogeneity than traditional IP multicast due its ability to dynamically change the value of DSCP within the multicast packets itself and to propagate “best PHB” up the tree. DSMCast is also able to preserve the basic objective of multicasting to avoid wastage of bandwidth.

2. The clear evidence of Good Neighbor Effect can be considered as an important parameter to gauge the cost of providing support for heterogeneous receivers with proper traffic class differentiation. If the cost benefits are properly weighed ISPs can deploy DSMCast onto their routers to achieve better network efficiency

3. The performance of DSMCast in case of uniform and non-uniform case indicates that in a DiffServ domain it is important to properly configure the DiffServ class parameters depending upon the nature of the traffic. This can serve as an important pointer to the network administrator while configuring the network parameters.

6.2 Future Scope

DSMCast is one of the preliminary approaches to provide support for heterogeneous receivers; it naturally paves way for future research and considerations:

- Efficient software can be devised against the hardware implementation for reading and interpreting the encapsulated header that carries the multicast state information in order to reduce the cost of implementation of high-speed network processors.
- As referred in [1], the benefits that DSMCast offers for multicasting can also serve good for unicast applications. The problem of heterogeneous QoS in unicast applications can be defined as changing the DSCP along the path according to the resource availability rather than the present assignment of absolute QoS. For example, if the receiver demands EF service but along the way to due resource constraints, DSCP can change as EF-AF10-AF11-EF.

References

[1] A. Striegel, Scalable approaches for DiffServ multicasting, *Ph.D. Dissertation*, Dec. 2002.

[2] A. Striegel and G. Manimaran, Dynamic DSCPs for heterogeneous QoS in DiffServ multicasting, in *Proc. of GLOBECOM*, Taipei, Taiwan, Nov. 2002.

[3] A. Striegel, GenMCast: A generic multicast extension for ns-2, *ND CSE Technical Report*.

[4] A. Striegel and G. Manimaran, A survey of QoS Multicasting, IEEE communications Magazine, June 2002

[5] S. Blake, D. Black et. al., An Architecture for Differentiated Services, RFC 2475,December 1998.

[6] S. Shenker, C. Partridge, R. Guerin, Specification of Guaranteed Quality of Service, RFC 2212, September 1997.

[7] J. Wroclawski, Specification of the Controlled-Load Network Element Service, RFC2211, September 1997.

[8] J. Wroclawski, The Use of RSVP with IETF Integrated Services, RFC2210, September 1997.

[9] A. Ballardie, Core Based Trees (CBT version 2) Multicast Routing RFC 2189, September 1997

[10] D. Estrin, D. Farinacci et. al., Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification, RFC 2362. June 1998

[11] Andrew Adams, Jonathan Nicholas William Siadak, IETF draft: Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification (Revised), September 2003.

[12] D. Waitzman, C. Partridge, S. Deering, Distance Vector Multicast Routing Protocol, November 1998.

[13] J. Moy, Multicast Extensions to OSPF, RFC 1584, March 1994

[14] Shigang Chen, Klara Nahrstedt, and Yuval Shavitt,"QMRP: QoS-Aware Multicast Routing Protocol", IEEE trans. on communication, Vol-18, No. 12, December 2000.

[15] Shuqian Yan, Michalis Faloutsos, and Anindo Banerjea, QoSMIC: QoS Aware Multicast routing for the Internet. The design and evaluation of QoSMIC" IEEE/ACM trans. on Networking, Vol-10, No-10, February 2002.

[16] M. Ajmone Marsan et al. (Eds.): " End-to-End QoS supported on Bandwidth Broker" QoS-IP 2003, LNCS 2601, pp. 158–170, 2003. Springer-Verlag Berlin Heidelberg 2003.

[17] Deepak Sreekantan, Dhananjaya Rao An article: "Implementation of bandwidth broker system for Resource Management"

[18] J. Heinanen, F. Baker W. Weiss J. Wroclawski Assured Forwarding PHB Group, RFC 2597,June 1999.

[19] V. Jacobson, K. Nichols, K. Poduri, An Expedited Forwarding PHB, RFC 2598 June 1999.

[20] Roland Bless, Klaus Wehrle: Internet Draft: "IP Multicast in Differentiated Services Networks", <draft-bless-diffserv-multicast-07.txt>, August 2003.

[21] UCB/LBNL/VINT Network Simulator – ns (version 2.26), Available at www.mash.cs.berkeley.edu/ns/

[22] James F Kurose and Ross, "Computer Networking : A top down Approach" by PHI

APPENDIX A

Protocol Independent Multicast – Sparse Mode (PIM-SM)

PIM firsts constructs a shared tree with *rendezvous point* (RP) as the root. The selection of the RP is done from the set of routers according to the well-defined and complex rules* in the PIM protocol. All the routers in the domain are then assumed to be aware of RP by its unicast address. Consider the following example network:

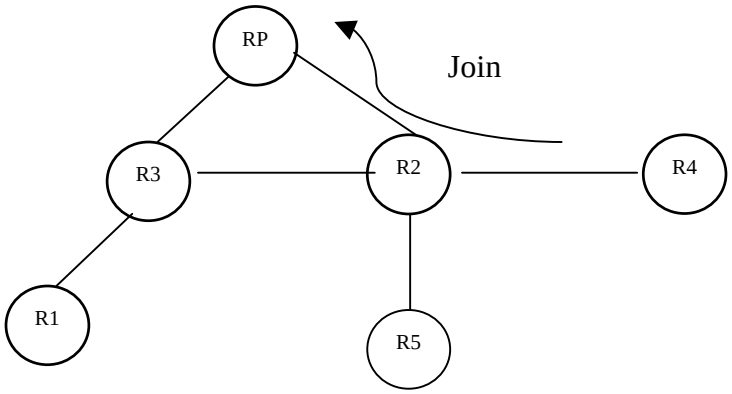

Figure A.1

The router R4 wants to join the group G. So, R4 will send the JOIN message as the unicast packet to the RP. The initial join message will be notified in the wildcard manner as (*, G) meaning to all senders. Before reaching RP, the join request will pass through many intermediate routers, say R2 in above figure. R2 will make the forwarding table entry for the interface from where the Join message comes from i.e. R4 to R2. R2 then determines the path towards RP and forwards the Join request to it. Thus a branch of the shared tree is created. Similarly, router R5 joins the group G.

Now for the multicast packets to be transmitted from the host: it sends the packet with the appropriate multicast address to the last-hop router of its network, which is known as the *designated router* (DR), say R1 in the above figure. DR encapsulates the multicast packet in the unicast packet and sends to the RP. RP understands that the

packet is meant for a particular group and so de-capsulate it and forwards it to R2. R2 then, forwards the packet to R4 and R5. The encapsulation and de-capsulation of packets does lead to some bandwidth inefficiency and processing cost. To optimize this aspect, the shared tree can be switched to source specific tree. If a particular source is identified to send packets at high rate, RP initiate a JOIN request specific to that source to establish the new state as (S, G). Within the intermediate routers (S, G) entry is created and the outgoing interface list is copied from (*,G), i.e., all local shared tree branches are replicated in the new shortest path tree. As a result a new source specific tree rotted at the source is created. In the above example, R1 is recognized as the source. Routers R4 and R5 have made the entry and so they will forward their JOIN request to R1 and completely by-pass RP.

APPENDIX B

Scripts

B.1 DSMCast Configuration for Uniform Distribution (10 Mbps)

File Name : GenMCast_Config_10Mb_U.txt

```
setDS-All set numQueues_ 5
setDS-All setNumPrec 3

setDS-All setSchedularMode WRR

setDS-All addQueueWeights 0 24
setDS-All addQueueWeights 1 18
setDS-All addQueueWeights 2 12
setDS-All addQueueWeights 3 6
setDS-All addQueueWeights 4 1

#BE Code point
setDS-All addPHBEntry 5 4 0

#AF
setDS-All addPHBEntry       10 0 0
setDS-All addPHBEntry       11 0 1
setDS-All addPHBEntry       12 0 2

setDS-All addPHBEntry       20 1 0
setDS-All addPHBEntry       21 1 1
setDS-All addPHBEntry       22 1 2

setDS-All addPHBEntry       30 2 0
setDS-All addPHBEntry       31 2 1
setDS-All addPHBEntry       32 2 2

setDS-All addPHBEntry       40 3 0
setDS-All addPHBEntry       41 3 1
setDS-All addPHBEntry       42 3 2

setDS-All configQ 0 0 22 40 0.01
setDS-All configQ 0 1 12 22 0.02
setDS-All configQ 0 2 5 12 0.02

setDS-All configQ 1 0 22 40 0.015
setDS-All configQ 1 1 12 22 0.02
setDS-All configQ 1 2 5 12 0.02

setDS-All configQ 2 0 22 40 0.015
setDS-All configQ 2 1 12 22 0.02
setDS-All configQ 2 2 5 12 0.02
```

```
setDS-All configQ 3 0 22 40 0.01
setDS-All configQ 3 1 12 22 0.02
setDS-All configQ 3 2 5 12 0.02

setDS-All configQ 4 0  100 100 0.00

setDS-All setMREDMode WRED 0
setDS-All setMREDMode WRED 1
setDS-All setMREDMode WRED 2
setDS-All setMREDMode WRED 3
setDS-All setMREDMode WRED 4


setDS-Edge addPolicyEntry -1 -1 TSW3CM 10 850000 875000

setDS-Edge addPolicerEntry TSW3CM 10 11 12

setDS-Edge addPolicyEntry -1 -1 TSW3CM 20 825000 830000

setDS-Edge addPolicerEntry TokenBucket 20 21 22

setDS-Edge addPolicyEntry -1 -1 TSW3CM 30 825000 830000

setDS-Edge addPolicerEntry TokenBucket 30 31 32

setDS-Edge addPolicyEntry -1 -1 TSW3CM 40 825000 830000

setDS-Edge addPolicerEntry TokenBucket 40 41 42

setDS-Edge addPolicyEntry -1 -1 TokenBucket 5 133333 0

setDS-Edge addPolicerEntry TokenBucket 5 5

setDS-All meanPktSize 512
```

B.2 Uniform Distribution Worst Case (10 Mbps)

```
set ns [new Simulator -multicast on]

$ns color 0 blue

$ns color 1 red

$ns color 2 green

$ns color 3 violet
$ns color 4 black
```

```
set f [open GenMCast_10Mb_WC_U.tr w]

$ns trace-all $f

set nf [open GenMCast_10Mb_WC_U.nam w]

$ns namtrace-all $nf

set packetSize 1000


for {set i 0} {$i < 17} {incr i} {

   set Node($i) [$ns node]
   puts "Node($i) set"

}

$ns simplex-link $Node(0) $Node(1) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(1) $Node(0) 10.0Mb 5ms dsRED/core


$ns duplex-link $Node(1) $Node(2) 10.0Mb 5ms dsRED/core

$ns duplex-link $Node(2) $Node(3) 10.0Mb 5ms dsRED/core

$ns duplex-link $Node(2) $Node(4) 10.0Mb 5ms dsRED/core

$ns simplex-link $Node(5) $Node(3) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(3) $Node(5) 10.0Mb 5ms dsRED/core


$ns simplex-link $Node(6) $Node(3) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(3) $Node(6) 10.0Mb 5ms dsRED/core

$ns simplex-link $Node(7) $Node(4) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(4) $Node(7) 10.0Mb 5ms dsRED/core

$ns simplex-link $Node(8) $Node(4) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(4) $Node(8) 10.0Mb 5ms dsRED/core
```

```
#Node 9 and Node 10 are non-tree DS nodes

$ns simplex-link $Node(9) $Node(0) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(0) $Node(9) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(10) $Node(1) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(1) $Node(10) 10.0Mb 5ms dsRED/core

$ns simplex-link $Node(11) $Node(2) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(2) $Node(11) 10.0Mb 5ms dsRED/core

$ns simplex-link $Node(12) $Node(3) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(3) $Node(12) 10.0Mb 5ms dsRED/core

$ns simplex-link $Node(13) $Node(4) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(4) $Node(13) 10.0Mb 5ms dsRED/core

$ns simplex-link $Node(14) $Node(2) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(2) $Node(14) 10.0Mb 5ms dsRED/core

$ns simplex-link $Node(15) $Node(1) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(1) $Node(15) 10.0Mb 5ms dsRED/core

$ns simplex-link $Node(16) $Node(0) 10.0Mb 5ms dsRED/edge

$ns simplex-link $Node(0) $Node(16) 10.0Mb 5ms dsRED/edge


#GenMCast Settings
#Protocol
$ns mrtproto GenMCast

#Master object
set GMCMaster [new Agent/GenMCast/Master]
```

```
$ns init-GenMCast $GMCMaster

#Data/Control blocks
for {set i 0} {$i < 9} {incr i} {

 set Group($i) [Node allocaddr]

 set Src($i) [new Application/Traffic/CBR]

 set SinkUDP($i) [new Agent/Null]

 set UDP($i) [new Agent/UDP]

 $Src($i) set packetSize_ 512

 $Src($i) set rate_ 32Kb

 $Src($i) attach-agent $UDP($i)

 $ns attach-agent $Node($i) $UDP($i)

 $ns attach-agent $Node($i) $SinkUDP($i)

 $UDP($i) set dst_addr_         $Group($i)

 $ns at 0.001 "$ns trace-annotate \"Node($i) with $Src($i) start\""

 $ns at 0.001 "$Src($i) start"

}

for {set i 0} {$i < 9} {incr i} {

 set GenMC_Node($i) [new Agent/GenMCast/Node/DiffServ]

 $GMCMaster attach $Node($i) $GenMC_Node($i)
}

$GenMC_Node(0) setNodeType Edge
```

```
$GenMC_Node(5) setNodeType Edge

$GenMC_Node(6) setNodeType Edge
$GenMC_Node(7) setNodeType Edge

$GenMC_Node(8) setNodeType Edge



for {set i 0} {$i < 9} {incr i} {

        $GenMC_Node($i) add-mcast-protocol DSMCast


        $GenMC_Node($i)  apply-protocol  All  init-linkage  $GMCMaster
$GenMC_Node($i)  $ns

        $GenMC_Node($i) apply-protocol All init-protocol



        # Routing protocol information


        $GenMC_Node($i) add-route-protocol GenLS


  $ns at 0.0 "$GenMC_Node($i) apply-route GenLS set-LS 0 100 3"


  $ns at 0.0 "$GenMC_Node($i) apply-route GenLS start"

}


for {set i 0} {$i < 9} {incr i} {
        $GenMC_Node($i) load-DS-cfg GenMCast_Config_10Mb_U.txt
}

#Background Traffic


for {set i 0} {$i < 1} {incr i} {
#for node 9
set udp912($i) [new Agent/UDP]
$ns attach-agent $Node(9) $udp912($i)

set cbr912($i) [new Application/Traffic/Exponential]
$cbr912($i) set rate_ 800k
```

```
$cbr912($i) set burst_time_ 10ms
$cbr912($i) set idle_time_ 40ms
$cbr912($i) set packetSize_ 1000

$cbr912($i) attach-agent $udp912($i)

set null912($i) [new Agent/Null]
$ns attach-agent $Node(12) $null912($i)
$ns connect $udp912($i) $null912($i)
$udp912($i) set fid_ 1

$ns at 0.01 "$cbr912($i) start"

set tcp912($i) [new Agent/TCP]
$ns attach-agent $Node(9) $tcp912($i)
$tcp912($i) set rate_ 800k
$tcp912($i) set packetSize_ 1000

set ftpsrc912($i) [new Application/FTP]
$ftpsrc912($i) attach-agent $tcp912($i)

set tcpsink912($i) [new Agent/TCPSink]
$ns attach-agent $Node(12) $tcpsink912($i)
$ns connect $tcp912($i) $tcpsink912($i)
$tcp912($i) set fid_ 2

$ns at 0.01 "$ftpsrc912($i) start"

set udp913($i) [new Agent/UDP]
$ns attach-agent $Node(9) $udp913($i)

set cbr913($i) [new Application/Traffic/Exponential]
$cbr913($i) set rate_ 800k
$cbr913($i) set burst_time_ 10ms
$cbr913($i) set idle_time_ 40ms
$cbr913($i) set packetSize_ 1000
$cbr913($i) attach-agent $udp913($i)

set null913($i) [new Agent/Null]
$ns attach-agent $Node(13) $null913($i)
$ns connect $udp913($i) $null913($i)
$udp913($i) set fid_ 1

$ns at 0.01 "$cbr913($i) start"

set tcp913($i) [new Agent/TCP]
$ns attach-agent $Node(9) $tcp913($i)
$tcp913($i) set rate_ 800k
$tcp913($i) set packetSize_ 1000
```

```
set ftpsrc913($i) [new Application/FTP]
$ftpsrc913($i) attach-agent $tcp913($i)

set tcpsink913($i) [new Agent/TCPSink]
$ns attach-agent $Node(13) $tcpsink913($i)
$ns connect $tcp913($i) $tcpsink913($i)
$tcp913($i) set fid_ 2

$ns at 0.01 "$ftpsrc913($i) start"

#for node 10

set udp1012($i) [new Agent/UDP]
$ns attach-agent $Node(10) $udp1012($i)

set cbr1012($i) [new Application/Traffic/Exponential]
$cbr1012($i) set rate_ 800k
$cbr1012($i) set burst_time_ 10ms
$cbr1012($i) set idle_time_ 40ms
$cbr1012($i) set packetSize_ 1000

$cbr1012($i) attach-agent $udp1012($i)

set null1012($i) [new Agent/Null]
$ns attach-agent $Node(12) $null1012($i)
$ns connect $udp1012($i) $null1012($i)
$udp1012($i) set fid_ 1

$ns at 0.01 "$cbr1012($i) start"

set tcp1012($i) [new Agent/TCP]
$ns attach-agent $Node(10) $tcp1012($i)
$tcp1012($i) set rate_ 800k
$tcp1012($i) set packetSize_ 1000

set ftpsrc1012($i) [new Application/FTP]
$ftpsrc1012($i) attach-agent $tcp1012($i)

set tcpsink1012($i) [new Agent/TCPSink]
$ns attach-agent $Node(12) $tcpsink1012($i)
$ns connect $tcp1012($i) $tcpsink1012($i)
$tcp1012($i) set fid_ 2

$ns at 0.01 "$ftpsrc1012($i) start"

set udp1013($i) [new Agent/UDP]
$ns attach-agent $Node(10) $udp1013($i)
```

```
set cbr1013($i) [new Application/Traffic/Exponential]
$cbr1013($i) set rate_ 800k
$cbr1013($i) set burst_time_ 10ms
$cbr1013($i) set idle_time_ 40ms
$cbr1013($i) set packetSize_ 1000
$cbr1013($i) attach-agent $udp1013($i)

set null1013($i) [new Agent/Null]
$ns attach-agent $Node(13) $null1013($i)
$ns connect $udp1013($i) $null1013($i)
$udp1013($i) set fid_ 1

$ns at 0.01 "$cbr1013($i) start"

set tcp1013($i) [new Agent/TCP]
$ns attach-agent $Node(10) $tcp1013($i)
$tcp1013($i) set rate_ 800k
$tcp1013($i) set packetSize_ 1000

set ftpsrc1013($i) [new Application/FTP]
$ftpsrc1013($i) attach-agent $tcp1013($i)

set tcpsink1013($i) [new Agent/TCPSink]
$ns attach-agent $Node(13) $tcpsink1013($i)
$ns connect $tcp1013($i) $tcpsink1013($i)
$tcp1013($i) set fid_ 2

$ns at 0.01 "$ftpsrc1013($i) start"


#for node 11

set udp1112($i) [new Agent/UDP]
$ns attach-agent $Node(11) $udp1112($i)

set cbr1112($i) [new Application/Traffic/Exponential]
$cbr1112($i) set rate_ 800k
$cbr1112($i) set burst_time_ 10ms
$cbr1112($i) set idle_time_ 40ms
$cbr1112($i) set packetSize_ 1000

$cbr1112($i) attach-agent $udp1112($i)

set null1112($i) [new Agent/Null]
$ns attach-agent $Node(12) $null1112($i)
$ns connect $udp1112($i) $null1112($i)
$udp1112($i) set fid_ 1

$ns at 0.01 "$cbr1112($i) start"
```

```
set tcp1112($i) [new Agent/TCP]
$ns attach-agent $Node(11) $tcp1112($i)
$tcp1112($i) set rate_ 800k
$tcp1112($i) set packetSize_ 1000

set ftpsrc1112($i) [new Application/FTP]
$ftpsrc1112($i) attach-agent $tcp1112($i)

set tcpsink1112($i) [new Agent/TCPSink]
$ns attach-agent $Node(12) $tcpsink1112($i)
$ns connect $tcp1112($i) $tcpsink1112($i)
$tcp1112($i) set fid_ 2

$ns at 0.01 "$ftpsrc1112($i) start"

set udp1113($i) [new Agent/UDP]
$ns attach-agent $Node(11) $udp1113($i)

set cbr1113($i) [new Application/Traffic/Exponential]
$cbr1113($i) set rate_ 800k
$cbr1113($i) set burst_time_ 10ms
$cbr1113($i) set idle_time_ 40ms
$cbr1113($i) set packetSize_ 1000
$cbr1113($i) attach-agent $udp1113($i)

set null1113($i) [new Agent/Null]
$ns attach-agent $Node(13) $null1113($i)
$ns connect $udp1113($i) $null1113($i)
$udp1113($i) set fid_ 1

$ns at 0.01 "$cbr1113($i) start"

set tcp1113($i) [new Agent/TCP]
$ns attach-agent $Node(11) $tcp1113($i)
$tcp1113($i) set rate_ 800k
$tcp1113($i) set packetSize_ 1000

set ftpsrc1113($i) [new Application/FTP]
$ftpsrc1113($i) attach-agent $tcp1113($i)

set tcpsink1113($i) [new Agent/TCPSink]
$ns attach-agent $Node(13) $tcpsink1113($i)
$ns connect $tcp1113($i) $tcpsink1113($i)
$tcp1113($i) set fid_ 2

$ns at 0.01 "$ftpsrc1113($i) start"

#for node 14
```

```
set udp1412($i) [new Agent/UDP]
$ns attach-agent $Node(14) $udp1412($i)

set cbr1412($i) [new Application/Traffic/Exponential]
$cbr1412($i) set rate_ 800k
$cbr1412($i) set burst_time_ 10ms
$cbr1412($i) set idle_time_ 40ms
$cbr1412($i) set packetSize_ 1000

$cbr1412($i) attach-agent $udp1412($i)

set null1412($i) [new Agent/Null]
$ns attach-agent $Node(12) $null1412($i)
$ns connect $udp1412($i) $null1412($i)
$udp1412($i) set fid_ 1

$ns at 0.01 "$cbr1412($i) start"

set tcp1412($i) [new Agent/TCP]
$ns attach-agent $Node(14) $tcp1412($i)
$tcp1412($i) set rate_ 800k
$tcp1412($i) set packetSize_ 1000

set ftpsrc1412($i) [new Application/FTP]
$ftpsrc1412($i) attach-agent $tcp1412($i)

set tcpsink1412($i) [new Agent/TCPSink]
$ns attach-agent $Node(12) $tcpsink1412($i)
$ns connect $tcp1412($i) $tcpsink1412($i)
$tcp1412($i) set fid_ 2

$ns at 0.01 "$ftpsrc1412($i) start"

set udp1413($i) [new Agent/UDP]
$ns attach-agent $Node(14) $udp1413($i)

set cbr1413($i) [new Application/Traffic/Exponential]
$cbr1413($i) set rate_ 800k
$cbr1413($i) set burst_time_ 10ms
$cbr1413($i) set idle_time_ 40ms
$cbr1413($i) set packetSize_ 1000
$cbr1413($i) attach-agent $udp1413($i)

set null1413($i) [new Agent/Null]
$ns attach-agent $Node(13) $null1413($i)
$ns connect $udp1413($i) $null1413($i)
$udp1413($i) set fid_ 1
```

```
$ns at 0.01 "$cbr1413($i) start"

set tcp1413($i) [new Agent/TCP]
$ns attach-agent $Node(14) $tcp1413($i)
$tcp1413($i) set rate_ 800k
$tcp1413($i) set packetSize_ 1000

set ftpsrc1413($i) [new Application/FTP]
$ftpsrc1413($i) attach-agent $tcp1413($i)

set tcpsink1413($i) [new Agent/TCPSink]
$ns attach-agent $Node(13) $tcpsink1413($i)
$ns connect $tcp1413($i) $tcpsink1413($i)
$tcp1413($i) set fid_ 2

$ns at 0.01 "$ftpsrc1413($i) start"

#for node 15

set udp1512($i) [new Agent/UDP]
$ns attach-agent $Node(15) $udp1512($i)

set cbr1512($i) [new Application/Traffic/Exponential]
$cbr1512($i) set rate_ 800k
$cbr1512($i) set burst_time_ 10ms
$cbr1512($i) set idle_time_ 40ms
$cbr1512($i) set packetSize_ 1000

$cbr1512($i) attach-agent $udp1512($i)

set null1512($i) [new Agent/Null]
$ns attach-agent $Node(12) $null1512($i)
$ns connect $udp1512($i) $null1512($i)
$udp1512($i) set fid_ 1

$ns at 0.01 "$cbr1512($i) start"

set tcp1512($i) [new Agent/TCP]
$ns attach-agent $Node(15) $tcp1512($i)
$tcp1512($i) set rate_ 800k
$tcp1512($i) set packetSize_ 1000

set ftpsrc1512($i) [new Application/FTP]
$ftpsrc1512($i) attach-agent $tcp1512($i)

set tcpsink1512($i) [new Agent/TCPSink]
$ns attach-agent $Node(12) $tcpsink1512($i)
$ns connect $tcp1512($i) $tcpsink1512($i)
$tcp1512($i) set fid_ 2
```

```
$ns at 0.01 "$ftpsrc1512($i) start"

set udp1513($i) [new Agent/UDP]
$ns attach-agent $Node(15) $udp1513($i)

set cbr1513($i) [new Application/Traffic/Exponential]
$cbr1513($i) set rate_ 800k
$cbr1513($i) set burst_time_ 10ms
$cbr1513($i) set idle_time_ 40ms
$cbr1513($i) set packetSize_ 1000
$cbr1513($i) attach-agent $udp1513($i)

set null1513($i) [new Agent/Null]
$ns attach-agent $Node(13) $null1513($i)
$ns connect $udp1513($i) $null1513($i)
$udp1513($i) set fid_ 1

$ns at 0.01 "$cbr1513($i) start"

set tcp1513($i) [new Agent/TCP]
$ns attach-agent $Node(15) $tcp1513($i)
$tcp1513($i) set rate_ 800k
$tcp1513($i) set packetSize_ 1000

set ftpsrc1513($i) [new Application/FTP]
$ftpsrc1513($i) attach-agent $tcp1513($i)

set tcpsink1513($i) [new Agent/TCPSink]
$ns attach-agent $Node(13) $tcpsink1513($i)
$ns connect $tcp1513($i) $tcpsink1513($i)
$tcp1513($i) set fid_ 2

$ns at 0.01 "$ftpsrc1513($i) start"

#for node 16

set udp1612($i) [new Agent/UDP]
$ns attach-agent $Node(16) $udp1612($i)

set cbr1612($i) [new Application/Traffic/Exponential]
$cbr1612($i) set rate_ 800k
$cbr1612($i) set burst_time_ 10ms
$cbr1612($i) set idle_time_ 40ms
$cbr1612($i) set packetSize_ 1000

$cbr1612($i) attach-agent $udp1612($i)

set null1612($i) [new Agent/Null]
```

```
$ns attach-agent $Node(12) $null1612($i)
$ns connect $udp1612($i) $null1612($i)
$udp1612($i) set fid_ 1

$ns at 0.01 "$cbr1612($i) start"

set tcp1612($i) [new Agent/TCP]
$ns attach-agent $Node(16) $tcp1612($i)
$tcp1612($i) set rate_ 800k
$tcp1612($i) set packetSize_ 1000

set ftpsrc1612($i) [new Application/FTP]
$ftpsrc1612($i) attach-agent $tcp1612($i)

set tcpsink1612($i) [new Agent/TCPSink]
$ns attach-agent $Node(12) $tcpsink1612($i)
$ns connect $tcp1612($i) $tcpsink1612($i)
$tcp1612($i) set fid_ 2

$ns at 0.01 "$ftpsrc1612($i) start"

set udp1613($i)  [new Agent/UDP]
$ns attach-agent $Node(16) $udp1613($i)

set cbr1613($i)  [new Application/Traffic/Exponential]
$cbr1613($i)  set rate_ 800k
$cbr1613($i)  set burst_time_ 10ms
$cbr1613($i)  set idle_time_ 40ms
$cbr1613($i)  set packetSize_ 1000
$cbr1613($i)  attach-agent $udp1613($i)

set null1613($i)  [new Agent/Null]
$ns attach-agent $Node(13) $null1613($i)
$ns connect $udp1613($i)  $null1613($i)
$udp1613($i)  set fid_ 1

$ns at 0.01 "$cbr1613($i)  start"

set tcp1613($i)  [new Agent/TCP]
$ns attach-agent $Node(16) $tcp1613($i)
$tcp1613($i)  set rate_ 800k
$tcp1613($i)  set packetSize_ 1000

set ftpsrc1613($i)  [new Application/FTP]
$ftpsrc1613($i)  attach-agent $tcp1613($i)

set tcpsink1613($i)  [new Agent/TCPSink]
$ns attach-agent $Node(13) $tcpsink1613($i)
$ns connect $tcp1613($i)  $tcpsink1613($i)
```

```
$tcp1613($i)  set fid_ 2

$ns at 0.01 "$ftpsrc1613($i)  start"
}



#DiffServ Configuration for all the non-tree nodes : 9,10,11,12 and 13

set qE9S1 [[$ns link $Node(9) $Node(0)] queue]
set qS1E9 [[$ns link $Node(0) $Node(9)] queue]

set qE10C1 [[$ns link $Node(10) $Node(1)] queue]
set qC1E10 [[$ns link $Node(1) $Node(10)] queue]

set qE11C [[$ns link $Node(11) $Node(2)] queue]
set qCE11 [[$ns link $Node(2) $Node(11)] queue]

set qE3E13 [[$ns link $Node(4) $Node(13)] queue]
set qE13E3 [[$ns link $Node(13) $Node(4)] queue]

set qE12E2 [[$ns link $Node(12) $Node(3)] queue]
set qE2E12 [[$ns link $Node(3) $Node(12)] queue]

set qE16S1 [[$ns link $Node(16) $Node(0)] queue]
set qS1E16 [[$ns link $Node(0) $Node(16)] queue]

set qE15C1 [[$ns link $Node(15) $Node(1)] queue]
set qC1E15 [[$ns link $Node(1) $Node(15)] queue]

set qE14C [[$ns link $Node(14) $Node(2)] queue]
set qCE14 [[$ns link $Node(2) $Node(14)] queue]

#Node(9) to Node(0)

$qE9S1 meanPktSize $packetSize
$qE9S1 set numQueues_ 5
$qE9S1 setNumPrec 3
$qE9S1 setSchedularMode WRR
$qE9S1 addQueueWeights 0 24

$qE9S1 addQueueWeights 1 18
$qE9S1 addQueueWeights 2 12
$qE9S1 addQueueWeights 3 6
$qE9S1 addQueueWeights 4 1

$qE9S1 addPolicyEntry -1 -1 TSW3CM 10 850000 900000
$qE9S1 addPolicerEntry TSW3CM 10 11 12
```

```
$qE9S1 addPHBEntry 10 0 0
$qE9S1 addPHBEntry 11 0 1
$qE9S1 addPHBEntry 12 0 2

$qE9S1 configQ 0 0 20 40 0.01
$qE9S1 configQ 0 1 10 20 0.02
$qE9S1 configQ 0 2  5 10 0.02

$qE9S1 configQ 4 0 100 100 0

#Node(0) to Node(9)

$qS1E9 meanPktSize $packetSize
$qS1E9 set numQueues_ 5
$qS1E9 setNumPrec 3
$qS1E9 setSchedularMode WRR
$qS1E9 addQueueWeights 0 24

$qS1E9 addQueueWeights 1 18
$qS1E9 addQueueWeights 2 12
$qS1E9 addQueueWeights 3 6
$qS1E9 addQueueWeights 4 1

$qS1E9 addPolicyEntry -1 -1 TSW3CM 10 850000 900000
$qS1E9 addPolicerEntry TSW3CM 10 11 12

$qS1E9 addPHBEntry 10 0 0
$qS1E9 addPHBEntry 11 0 1
$qS1E9 addPHBEntry 12 0 2

$qS1E9 configQ 0 0 20 40 0.01
$qS1E9 configQ 0 1 10 20 0.02
$qS1E9 configQ 0 2 5 10 0.02

$qS1E9 configQ 4 0 100 100 0

#Node(10) to Node(1)

$qE10C1 meanPktSize $packetSize
$qE10C1 set numQueues_ 5
$qE10C1 setNumPrec 3
$qE10C1 setSchedularMode WRR
$qE10C1 addQueueWeights 0 24

$qE10C1 addQueueWeights 1 18
$qE10C1 addQueueWeights 2 12
$qE10C1 addQueueWeights 3 6
$qE10C1 addQueueWeights 4 1
```

```
$qE10C1 addPolicyEntry -1 -1 TSW3CM 20 825000 847500
$qE10C1 addPolicerEntry TSW3CM 20 21 22

$qE10C1 addPHBEntry 20 1 0
$qE10C1 addPHBEntry 21 1 1
$qE10C1 addPHBEntry 22 1 2

$qE10C1 configQ 1 0 20 40 0.01
$qE10C1 configQ 1 1 10 20 0.02
$qE10C1 configQ 1 2 5 10 0.02

$qE10C1 configQ 4 0 100 100 0


#Node(1) to Node(10)

$qC1E10 meanPktSize $packetSize
$qC1E10 set numQueues_ 5
$qC1E10 setNumPrec 3
$qC1E10 setSchedularMode WRR
$qC1E10 addQueueWeights 0 24

$qC1E10 addQueueWeights 1 18
$qC1E10 addQueueWeights 2 12
$qC1E10 addQueueWeights 3 6
$qC1E10 addQueueWeights 4 1

$qC1E10 addPHBEntry 20 1 0
$qC1E10 addPHBEntry 21 1 1
$qC1E10 addPHBEntry 22 1 2

$qC1E10 configQ 1 0  20 40 0.01
$qC1E10 configQ 1 1 10 20 0.02
$qC1E10 configQ 1 2 5 10 0.02

$qC1E10 configQ 4 0 100 100 0

#Node(11) to Node(2)

$qE11C meanPktSize $packetSize
$qE11C set numQueues_ 5
$qE11C setNumPrec 3
$qE11C setSchedularMode WRR
$qE11C addQueueWeights 0 24

$qE11C addQueueWeights 1 18
$qE11C addQueueWeights 2 12
$qE11C addQueueWeights 3 6
$qE11C addQueueWeights 4 1
```

```
$qE11C addPolicyEntry -1 -1 TSW3CM 30 825000 847500
$qE11C addPolicerEntry TSW3CM 30 31 32

$qE11C addPHBEntry 30 2 0
$qE11C addPHBEntry 31 2 1
$qE11C addPHBEntry 32 2 2

$qE11C configQ 2 0 20 40 0.01
$qE11C configQ 2 1 10 20 0.02
$qE11C configQ 2 2 5 10 0.02

$qE11C configQ 4 0 100 100 0

#Node(2) to Node(11)

$qCE11 meanPktSize $packetSize
$qCE11 set numQueues_ 5
$qCE11 setNumPrec 3
$qCE11 setSchedularMode WRR
$qCE11 addQueueWeights 0 24

$qCE11 addQueueWeights 1 18
$qCE11 addQueueWeights 2 12
$qCE11 addQueueWeights 3 6
$qCE11 addQueueWeights 4 1

$qCE11 addPHBEntry 30 2 0
$qCE11 addPHBEntry 31 2 1
$qCE11 addPHBEntry 32 2 2

$qCE11 configQ 2 0 20 40 0.01
$qCE11 configQ 2 1 10 20 0.02
$qCE11 configQ 2 2 5 10 0.02

$qCE11 configQ 4 0 100 100 0

#Node(12) to Node(3)

$qE12E2 meanPktSize $packetSize
$qE12E2 set numQueues_ 5
$qE12E2 setNumPrec 3
$qE12E2 setSchedularMode WRR
$qE12E2 addQueueWeights 0 24

$qE12E2 addQueueWeights 1 18
$qE12E2 addQueueWeights 2 12
$qE12E2 addQueueWeights 3 6
$qE12E2 addQueueWeights 4 1
```

```
$qE12E2 addPolicyEntry -1 -1 TSW3CM 10 850000 900000
$qE12E2 addPolicerEntry TSW3CM 10 11 12

$qE12E2 addPHBEntry 10 0 0
$qE12E2 addPHBEntry 11 0 1
$qE12E2 addPHBEntry 12 0 2

$qE12E2 configQ 0 0 20 40 0.01
$qE12E2 configQ 0 1 10 20 0.02
$qE12E2 configQ 0 2 5 10 0.02


$qE12E2 addPolicyEntry -1 -1 TSW3CM 20 825000 847500
$qE12E2 addPolicerEntry TSW3CM 20 21 22

$qE12E2 addPHBEntry 20 1 0
$qE12E2 addPHBEntry 21 1 1
$qE12E2 addPHBEntry 22 1 2

$qE12E2 configQ 1 0  20 40 0.01
$qE12E2 configQ 1 1 10 20 0.02
$qE12E2 configQ 1 2 5 10 0.02

$qE12E2 addPolicyEntry -1 -1 TSW3CM 30 825000 847500
$qE12E2 addPolicerEntry TSW3CM 30 31 32

$qE12E2 addPHBEntry 30 2 0
$qE12E2 addPHBEntry 31 2 1
$qE12E2 addPHBEntry 32 2 2

$qE12E2 configQ 2 0 20 40 0.01
$qE12E2 configQ 2 1 10 20 0.02
$qE12E2 configQ 2 2 5 10 0.02

$qE12E2 addPolicyEntry -1 -1 TSW3CM 40 825000 830000
$qE12E2 addPolicerEntry TSW3CM 40 40 40

$qE12E2 addPHBEntry 40 3 0

$qE12E2 configQ 3 0 20 40 0.01

$qE12E2 configQ 4 0 100 100 0

#Node(3) to Node(12)

$qE2E12 meanPktSize $packetSize
$qE2E12 set numQueues_ 5
$qE2E12 setNumPrec 3
```

```
$qE2E12 setSchedularMode WRR
$qE2E12 addQueueWeights 0 24

$qE2E12 addQueueWeights 1 18
$qE2E12 addQueueWeights 2 12
$qE2E12 addQueueWeights 3 6
$qE2E12 addQueueWeights 4 1

$qE2E12 addPHBEntry 10 0 0
$qE2E12 addPHBEntry 11 0 1
$qE2E12 addPHBEntry 12 0 2

$qE2E12 configQ 0 0  20 40 0.01
$qE2E12 configQ 0 1 10 20 0.02
$qE2E12 configQ 0 2 5 10 0.02

$qE2E12 addPHBEntry 20 1 0
$qE2E12 addPHBEntry 21 1 1
$qE2E12 addPHBEntry 22 1 2

$qE2E12 configQ 1 0  20 40 0.01
$qE2E12 configQ 1 1 10 20 0.02
$qE2E12 configQ 1 2 5 10 0.02

$qE2E12 addPHBEntry 30 2 0
$qE2E12 addPHBEntry 31 2 1
$qE2E12 addPHBEntry 32 2 2

$qE2E12 configQ 2 0  20 40 0.01
$qE2E12 configQ 2 1 10 20 0.02
$qE2E12 configQ 2 2 5 10 0.02

$qE2E12 addPHBEntry 40 3 0

$qE2E12 configQ 3 0  20 40 0.01

$qE2E12 configQ 4 0 100 100 0

#Node(4) to Node(13)

$qE3E13 meanPktSize $packetSize
$qE3E13 set numQueues_ 5
$qE3E13 setNumPrec 3
$qE3E13 setSchedularMode WRR
$qE3E13 addQueueWeights 0 24

$qE3E13 addQueueWeights 1 18
$qE3E13 addQueueWeights 2 12
$qE3E13 addQueueWeights 3 6
```

```
$qE3E13 addQueueWeights 4 1

$qE3E13 addPHBEntry 10 0 0
$qE3E13 addPHBEntry 11 0 1
$qE3E13 addPHBEntry 12 0 2

$qE3E13 configQ 0 0 20 40 0.01
$qE3E13 configQ 0 1 10 20 0.02
$qE3E13 configQ 0 2 5 10 0.02

$qE3E13 addPHBEntry 20 1 0
$qE3E13 addPHBEntry 21 1 1
$qE3E13 addPHBEntry 22 1 2

$qE3E13 configQ 1 0 20 40 0.01
$qE3E13 configQ 1 1 10 20 0.02
$qE3E13 configQ 1 2 5 10 0.02

$qE3E13 addPHBEntry 30 2 0
$qE3E13 addPHBEntry 31 2 1
$qE3E13 addPHBEntry 32 2 2

$qE3E13 configQ 2 0  20 40 0.01
$qE3E13 configQ 2 1 10 20 0.02
$qE3E13 configQ 2 2 5 10 0.02

$qE3E13 addPHBEntry 40 3 0

$qE3E13 configQ 3 0  20 40 0.01

$qE3E13 configQ 4 0 100 100 0

#Node(13) to Node(4)

$qE13E3 meanPktSize $packetSize
$qE13E3 set numQueues_ 5
$qE13E3 setNumPrec 3
$qE13E3 setSchedularMode WRR
$qE13E3 addQueueWeights 0 24

$qE13E3 addQueueWeights 1 18
$qE13E3 addQueueWeights 2 12
$qE13E3 addQueueWeights 3 6
$qE13E3 addQueueWeights 4 1

$qE13E3 addPolicyEntry -1 -1 TSW3CM 10 850000 900000
$qE13E3 addPolicerEntry TSW3CM 10 11 12

$qE13E3 addPHBEntry 10 0 0
```

```
$qE13E3 addPHBEntry 11 0 1
$qE13E3 addPHBEntry 12 0 2

$qE13E3 configQ 0 0 20 40 0.01
$qE13E3 configQ 0 1 10 20 0.02
$qE13E3 configQ 0 2 5 10 0.02


$qE13E3 addPolicyEntry -1 -1 TSW3CM 20 825000 847500
$qE13E3 addPolicerEntry TSW3CM 20 21 22

$qE13E3 addPHBEntry 20 1 0
$qE13E3 addPHBEntry 21 1 1
$qE13E3 addPHBEntry 22 1 2

$qE13E3 configQ 1 0 20 40 0.01
$qE13E3 configQ 1 1 10 20 0.02
$qE13E3 configQ 1 2 5 10 0.02

$qE13E3 addPolicyEntry -1 -1 TSW3CM 30 825000 847500
$qE13E3 addPolicerEntry TSW3CM 30 31 32

$qE13E3 addPHBEntry 30 2 0
$qE13E3 addPHBEntry 31 2 1
$qE13E3 addPHBEntry 32 2 2

$qE13E3 configQ 2 0 20 40 0.01
$qE13E3 configQ 2 1 10 20 0.02
$qE13E3 configQ 2 2 5 10 0.02

$qE13E3 addPolicyEntry -1 -1 TSW3CM 40 825000 830000
$qE13E3 addPolicerEntry TSW3CM 40 40 40

$qE13E3 addPHBEntry 40 3 0

$qE13E3 configQ 3 0  20 40 0.01

$qE13E3 configQ 4 0 100 100 0

#Node(14) to Node(2)

$qE14C meanPktSize $packetSize
$qE14C set numQueues_ 5
$qE14C setNumPrec 3
$qE14C setSchedularMode WRR
$qE14C addQueueWeights 0 24

$qE14C addQueueWeights 1 18
$qE14C addQueueWeights 2 12
```

```
$qE14C addQueueWeights 3 6
$qE14C addQueueWeights 4 1

$qE14C addPolicyEntry -1 -1 TSW3CM 20 825000 847500
$qE14C addPolicerEntry TSW3CM 20 21 22

$qE14C addPHBEntry 20 1 0
$qE14C addPHBEntry 21 1 1
$qE14C addPHBEntry 22 1 2

$qE14C configQ 1 0 20 40 0.01
$qE14C configQ 1 1 10 20 0.02
$qE14C configQ 1 2 5 10 0.02

$qE14C configQ 4 0 100 100 0

#Node(2) to Node(14)

$qCE14 meanPktSize $packetSize
$qCE14 set numQueues_ 5
$qCE14 setNumPrec 3
$qCE14 setSchedularMode WRR
$qCE14 addQueueWeights 0 24

$qCE14 addQueueWeights 1 18
$qCE14 addQueueWeights 2 12
$qCE14 addQueueWeights 3 6
$qCE14 addQueueWeights 4 1

$qCE14 addPHBEntry 20 1 0
$qCE14 addPHBEntry 21 1 1
$qCE14 addPHBEntry 22 1 2

$qCE14 configQ 1 0  20 40 0.01
$qCE14 configQ 1 1 10 20 0.02
$qCE14 configQ 1 2 5 10 0.02

$qCE14 configQ 4 0 100 100 0

#Node(15) to Node(1)
$qE15C1 meanPktSize $packetSize
$qE15C1 set numQueues_ 5
$qE15C1 setNumPrec 3
$qE15C1 setSchedularMode WRR
$qE15C1 addQueueWeights 0 24

$qE15C1 addQueueWeights 1 18
$qE15C1 addQueueWeights 2 12
$qE15C1 addQueueWeights 3 6
```

```
$qE15C1 addQueueWeights 4 1

$qE15C1 addPolicyEntry -1 -1 TSW3CM 30 825000 847500
$qE15C1 addPolicerEntry TSW3CM 30 31 32

$qE15C1 addPHBEntry 30 2 0
$qE15C1 addPHBEntry 31 2 1
$qE15C1 addPHBEntry 32 2 2

$qE15C1 configQ 2 0 20 40 0.01
$qE15C1 configQ 2 1 10 20 0.02
$qE15C1 configQ 2 2 5 10 0.02

$qE15C1 configQ 4 0 100 100 0

#Node(1) to Node(15)

$qC1E15 meanPktSize $packetSize
$qC1E15 set numQueues_ 5
$qC1E15 setNumPrec 3
$qC1E15 setSchedularMode WRR
$qC1E15 addQueueWeights 0 24

$qC1E15 addQueueWeights 1 18
$qC1E15 addQueueWeights 2 12
$qC1E15 addQueueWeights 3 6
$qC1E15 addQueueWeights 4 1

$qC1E15 addPHBEntry 30 2 0
$qC1E15 addPHBEntry 31 2 1
$qC1E15 addPHBEntry 32 2 2

$qC1E15 configQ 2 0 20 40 0.01
$qC1E15 configQ 2 1 10 20 0.02
$qC1E15 configQ 2 2 5 10 0.02

$qC1E15 configQ 4 0 100 100 0

#Node(16) to Node(0)

$qE16S1 meanPktSize $packetSize
$qE16S1 set numQueues_ 5
$qE16S1 setNumPrec 3
$qE16S1 setSchedularMode WRR
$qE16S1 addQueueWeights 0 24

$qE16S1 addQueueWeights 1 18
$qE16S1 addQueueWeights 2 12
$qE16S1 addQueueWeights 3 6
```

```
$qE16S1 addQueueWeights 4 1

$qE16S1 addPolicyEntry -1 -1 TSW3CM 40 825000 830000
$qE16S1 addPolicerEntry TSW3CM 40 40 40

$qE16S1 addPHBEntry 40 3 0

$qE16S1 configQ 3 0  20 40 0.01

$qE16S1 configQ 4 0 100 100 0

#Node(0) to Node(16)

$qS1E16 meanPktSize $packetSize
$qS1E16 set numQueues_ 5
$qS1E16 setNumPrec 3
$qS1E16 setSchedularMode WRR
$qS1E16 addQueueWeights 0 24

$qS1E16 addQueueWeights 1 18
$qS1E16 addQueueWeights 2 12
$qS1E16 addQueueWeights 3 6
$qS1E16 addQueueWeights 4 1

$qS1E16 addPolicyEntry -1 -1 TSW3CM 40 825000 830000
$qS1E16 addPolicerEntry TSW3CM 40 40 40

$qS1E16 addPHBEntry 40 3 0

$qS1E16 configQ 3 0  20 40 0.01

$qS1E16 configQ 4 0 100 100 0

puts "spawning some packets..."

$ns at 0.002 "$GMCMaster join-group-QoS $Node(5) $Group(0) SSM [$Node(0) id]
DiffServ 40"
$ns at 0.03 "$GMCMaster join-group-QoS $Node(7) $Group(0) SSM [$Node(0) id]
DiffServ 30"
$ns at 0.01 "$GMCMaster join-group-QoS $Node(6) $Group(0) SSM [$Node(0) id]
DiffServ 10"
$ns at 0.05 "$GMCMaster join-group-QoS $Node(8) $Group(0) SSM [$Node(0) id]
DiffServ 20"

for {set j 0.01} {$j < 1.0} {set j [expr $j+0.04]} {
      puts "in the loop gathering stats....."

         #$ns at $j "$GMCMaster gather-stats-protocol DSMCast DSMCast-Router 0"
```

```
        #$ns at $j "$GMCMaster add-stats-file-text Test.csv t=$j"
        #$ns at $j "$GMCMaster dump-stats-file 0 Test.csv Router"
        #$ns at $j "$GMCMaster add-stats-file-blank Test.csv"
        #$ns at $j "$GMCMaster reset-stats 0"

        $ns at $j "$GMCMaster gather-stats-protocol DSMCast DSMCast-Group 1"
        $ns at $j "$GMCMaster dump-stats-file 1 Testdyn.csv Group"
        $ns at $j "$GMCMaster add-stats-file-newline Testdyn.csv"
        $ns at $j "$GMCMaster reset-stats 1"
}

$ns at 1.0 "$GMCMaster gather-stats-protocol DSMCast DSMCast-Router 0"
$ns at 1.0 "$GMCMaster add-stats-file-blank Testdyn.csv"
$ns at 1.0 "$GMCMaster add-stats-file-text Testdyn.csv Router"
$ns at 1.0 "$GMCMaster dump-stats-file-title 0 Testdyn.csv"

$ns at 1.0 "$GMCMaster gather-stats-protocol DSMCast DSMCast-Group 0"
$ns at 1.0 "$GMCMaster add-stats-file-blank Testdyn.csv"
$ns at 1.0 "$GMCMaster add-stats-file-text Testdyn.csv Group"
$ns at 1.0 "$GMCMaster dump-stats-file-title 1 Testdyn.csv"


$GMCMaster set-status-tick 1 1
$ns at 2.0 "finish"
proc finish {} {
```

```
        global ns f nf

        $ns flush-trace

        close $f

        close $nf

        puts "running nam..."

        #exec nam GenMCast_10Mb_WC_U.nam &

        exit 0

}
$ns run
```

B.3 Uniform Distribution Average case (10 Mbps)

In average case, above script is executed for different joining order. Following part of the code will differ.

```
$ns at 0.002 "$GMCMaster join-group-QoS $Node(5) $Group(0) SSM [$Node(0) id] DiffServ [lindex $argv 0]"
$ns at 0.01 "$GMCMaster join-group-QoS $Node(8) $Group(0) SSM [$Node(0) id] DiffServ [lindex $argv 1]"
$ns at 0.03 "$GMCMaster join-group-QoS $Node(7) $Group(0) SSM [$Node(0) id] DiffServ [lindex $argv 2]"
$ns at 0.05 "$GMCMaster join-group-QoS $Node(6) $Group(0) SSM [$Node(0) id] DiffServ [lindex $argv 3]"
```

B.4 DSMCast Configuration for Non-Uniform Distribution

File Name : GenMCast_Config_10Mb_NU.txt

```
setDS-All set numQueues_ 5
setDS-All setNumPrec 3

setDS-All setSchedularMode WRR

setDS-All addQueueWeights 0 24
setDS-All addQueueWeights 1 18
setDS-All addQueueWeights 2 12
setDS-All addQueueWeights 3 6
setDS-All addQueueWeights 4 1
```

```
#BE Code point
setDS-All addPHBEntry 5 4 0

#AF
setDS-All addPHBEntry       10 0 0
setDS-All addPHBEntry       11 0 1
setDS-All addPHBEntry       12 0 2

setDS-All addPHBEntry       20 1 0
setDS-All addPHBEntry       21 1 1
setDS-All addPHBEntry       22 1 2

setDS-All addPHBEntry       30 2 0
setDS-All addPHBEntry       31 2 1
setDS-All addPHBEntry       32 2 2

setDS-All addPHBEntry       40 3 0
setDS-All addPHBEntry       41 3 1
setDS-All addPHBEntry       42 3 2

setDS-All configQ 0 0 22 40 0.02
setDS-All configQ 0 1 12 22 0.05
setDS-All configQ 0 2  6 12 0.10

setDS-All configQ 1 0 22 40 0.02
setDS-All configQ 1 1 12 22 0.05
setDS-All configQ 1 2 6 12 0.10

setDS-All configQ 2 0 22 40 0.02
setDS-All configQ 2 1 12 22 0.05
setDS-All configQ 2 2  6 12 0.10

setDS-All configQ 3 0 22 40 0.02
setDS-All configQ 3 1 12 22 0.05
setDS-All configQ 3 2  6 12 0.10

setDS-All configQ 4 0  100 100 0.00

setDS-All setMREDMode WRED 0
setDS-All setMREDMode WRED 1
setDS-All setMREDMode WRED 2
setDS-All setMREDMode WRED 3
setDS-All setMREDMode WRED 4


setDS-Edge addPolicyEntry -1 -1 TSW3CM 10 2200000 3000000

setDS-Edge addPolicerEntry TSW3CM 10 11 12
```

setDS-Edge addPolicyEntry -1 -1 TSW3CM 20 1650000 2000000

setDS-Edge addPolicerEntry TokenBucket 20 21 22

setDS-Edge addPolicyEntry -1 -1 TSW3CM 30 1100000 1500000

setDS-Edge addPolicerEntry TokenBucket 30 31 32

setDS-Edge addPolicyEntry -1 -1 TSW3CM 40 550000 600000

setDS-Edge addPolicerEntry TokenBucket 40 41 42

setDS-Edge addPolicyEntry -1 -1 TokenBucket 5 133333 0

setDS-Edge addPolicerEntry TokenBucket 5 5

setDS-All meanPktSize 512

B.5 Non-Uniform Worst Case (10 Mbps)

Following line will differ from B.2

```
for {set i 0} {$i < 9} {incr i} {
        $GenMC_Node($i) load-DS-cfg GenMCast_Config_10Mb_NU.txt
}
```

B.6 Non-Uniform Average Case (10 Mbps)

Following line will differ from B.3

```
for {set i 0} {$i < 9} {incr i} {
        $GenMC_Node($i) load-DS-cfg GenMCast_Config_10Mb_NU.txt
}
```